Software Architecture Reconstruction Research Support as Provided by HUSACCT

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

Wiebe Wiersema
HU University of Applied Sciences
Utrecht
The Netherlands
wiebe.wiersema@hu.nl

Jan Martijn van der Werf
Utrecht University
Utrecht
The Netherlands
j.m.e.m.vanderWerf@uu.nl

ABSTRACT

Software architecture reconstruction techniques may be used to understand and maintain software systems, especially in these cases where architectural documentation is outdated or missing. This paper presents the architecture reconstruction functionality of HUSACCT and describes how this functionality may be used and extended with algorithms in support of reconstruction research focusing on modular architectures. The tool provides a graphical user interface to select an algorithm, edit its parameters and to execute or reverse the algorithm. To study the results, browsers and diagrams are available. Furthermore, a user interface is provided to enhance the determination of the effectiveness of algorithms by means of the MoJoFM metric.

CCS CONCEPTS

• Software and its engineering~Software architectures;
• Software and its engineering~Software maintenance tools

KEYWORDS

Software Architecture; Module View; Architecture Reconstruction; Architecture Compliance

ACM Reference Format:
Leo Pruijt, Wiebe Wiersema, and Jan Martijn van der Werf. 2017. Software Architecture Reconstruction Research Support as Provided by HUSACCT. In Proceedings of ECSA '17, Canterbury, United Kingdom, September 11–15, 2017, 4 pages.
https://doi.org/10.1145/3129790.3129819

1 INTRODUCTION

Software architecture reconstruction (SAR) is a reverse engineering approach that aims at reconstructing viable architectural views of a software application, which may be used to understand and re-document the application [2]. In this paper, we focus on the reconstruction of the module view [1]. A *modular architecture* describes the modular elements, their form (properties and relationships) and the rationale [5], where properties and relationships express architectural rules.

HUSACCT [8] is a free-to-use, open source tool (husacct.github.io/HUSACCT) that was designed to support software architecture compliance checking (SACC); a means to prevent or detect architectural erosion by comparison of the intended architecture and the implemented architecture of a system. HUSACCT focuses on the support of a semantically rich modular architecture (SRMA) [7], a term that we use for an expressive modular architecture description, composed of semantically different types of modules (e.g., subsystem, layer component), which are constrained by different types of rules such as constraints related to layers, or basic dependency constraints (e.g., Is not allowed to use).

Since version 5.0, extensive SAR support is provided with algorithms and a SAR GUI. In [11], we reported on experiments with a layers reconstruction algorithm. In this paper, we introduce the SAR support of HUSACCT to the research community. The tool may be used to apply existing algorithms, and it facilitates the development of new reconstruction algorithms targeted at modular architectures.

This paper is outlined as follows. Section 2 introduces HUSACCT and the SRMACC metamodel. Section 3 presents the SAR functionality and a provided algorithm, while Section 4 focuses on research support, including algorithm development and effectiveness measurement. Section 5 discusses the provided support and related work, while Section 6 concludes this paper.

2 HUSACCT

HUSACCT distinguishes itself from other SACC-tools by the provision of extensive support of SRMAs. In this context, we focus SAR on the reconstruction of semantically different modules and rules, which are related to different architectural patterns [6]. The SAR features build on the existing functionality to define an intended architecture, analyze code, register the implemented architecture, and check the compliance.

2.1 SRMACC Metamodel

To enable the provision of SRMA support, we have developed the SRMACC metamodel [10], whereof the central part is included in Fig. 1. It includes concepts and associations relevant to understand our approach. A *SoftwareArchitecture* may contain *Modules* of different *ModuleTypes*, where *AppliedRules*, each of a certain *RuleType*, may constrain the *Modules*.

ECSA '17, September 11–15, 2017, Canterbury, United Kingdom

ACM ISBN 978-1-4503-5217-8/17/09...$15.00
https://doi.org/10.1145/3129790.3129819

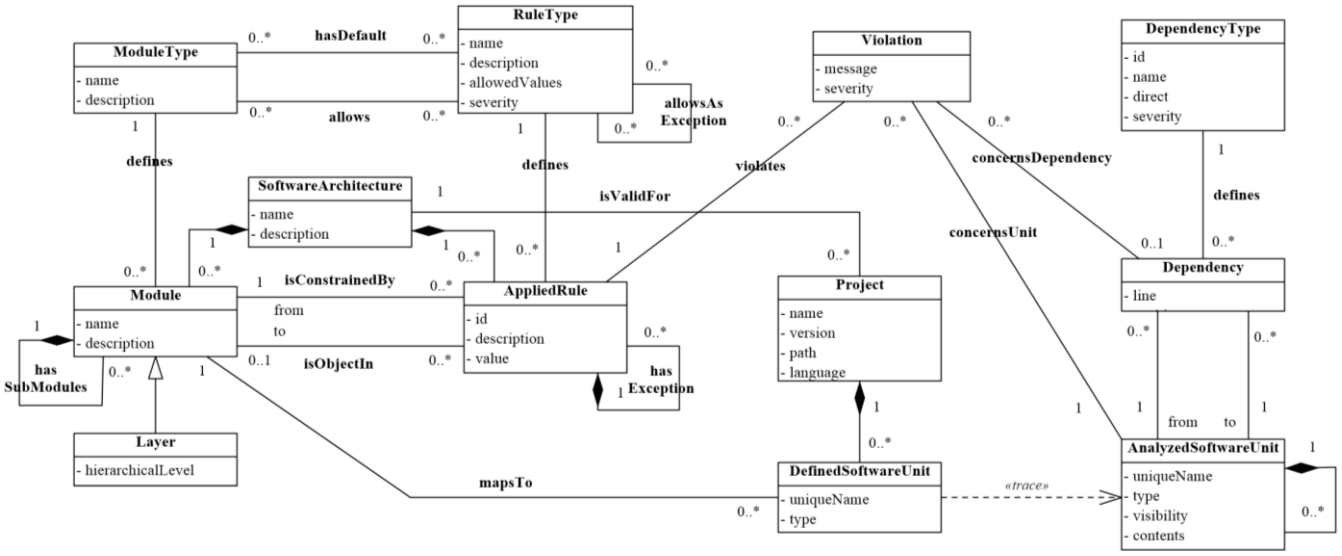

Figure 1: SRMACC Metamodel

The metamodel proves to be suitable for SAR as well. In fact, SAR can be considered as the reversed approach of SRMA-based SACC. For example in the case of layers, SAR tries to identify sets of units in the code that conform to the semantics of a layered architecture, while SACC checks the compliance of the code to the semantics of a defined layered architecture.

In terms of the SRMACC metamodel, SAR tries to identify sets of *AnalyzedSoftwareUnits*, packages and classes in the code, that conform to the default RuleTypes of ModuleType "Layer". In case sets are identified, a Module of ModuleType "Layer" is created for each set. The AnalyzedSoftwareUnits in a set are assigned to the Module by means of *DefinedSoftwareUnits*, which hold the data to trace the real units in analyzed code.

2.2 Definition of the Intended Architecture

Modules and rules can be added manually and programmatically to the intended architecture. Fig. 2 shows the GUI where the intended architecture is created and maintained manually. Modules can be added, edited and removed; software units can be assigned and removed; and rules can be added, edited or removed.

Currently HUSACCT provides extensive support for five common ModuleTypes (Subsystem, Layer, Component, Facade, External library) and eleven common RuleTypes. Relevant examples of extensive support in the context of SAR are the following: a) when a module of type Layer is created, two applied rules (back call ban and skip call ban) will be created, based on the default rule types associated to the module type of the module; b) when a module of type Component is created, a submodule of type Interface will be created as well.

3 SAR SUPPORT

SAR support is provided to the user by means of the GUI shown in Fig. 3. Currently, the SAR GUI provides three tabs. The first two tabs show a set of selectable approaches. After selection, the parameters and their set values are shown. The parameter values may be adjusted by means of the Edit Approach option. The first tab shows approaches for practical usage, sufficiently mature, while the second tab shows immature approaches. All these approaches have been developed at the HU University of Applied Sciences, though several layer identifying algorithms are based on work published by authors from other universities [11].

The Apply-button will start the selected approach. To allow an interactive process of reconstruction, the results may be reversed or edited, and several approaches may be applied consecutively to create a hierarchy of decompositions. The results will be visible in the view "Define intended architecture", shown in Fig. 2. Over there, a module may be edited (module name and type, assigned software units) or deleted. Furthermore, a module may be selected for another reconstruction iteration. If a module in the intended architecture is selected by the tool user, an algorithm may use this module as starting node; otherwise

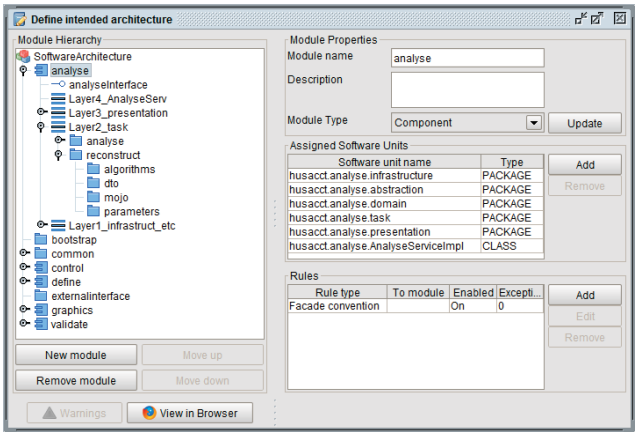

Figure 2: Intended architecture as defined in HUSACCT

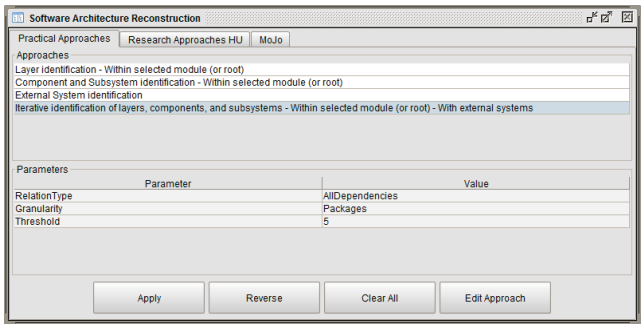

Figure 3: Reconstruct architecture user interface

the system's root will be used.

As example, we discuss an iterative algorithm, selected in Fig. 3, that makes use of the first two algorithms in the list. When it is started, first the input set of to-be-used software units is determined. In this case, we started with an empty intended architecture, so no module was selected. Instead, all software units in the root of the source code of HUSACCT_5.3 were used.

Next, the iterative algorithm will execute the Layer identification algorithm. If this algorithm identifies layers, based on relatively strict parameter values [11], it will add modules to the intended architecture; as submodules of the selected module or of the root module. If no layers are identified, the Component and Subsystem identification algorithm is executed, which may add modules in the same way. In the following iterations, the newly added modules are selected consecutively, and the procedure is repeated. This way a multi-level decomposition tree of modules may be reconstructed.

The module tree shown in Fig. 2. is the result of the iterative algorithm. Eight modules are added to the root of the architecture; five of type Component and three of type Subsystem. In the figure, component *analyse* is expanded, and its submodules are shown; one of type Interface and four of type Layer. Furthermore, six software units assigned to *analyse* are visible, as well as the rule of type Facade convention [10].

Fig. 4 shows the result in the form of an intended architecture

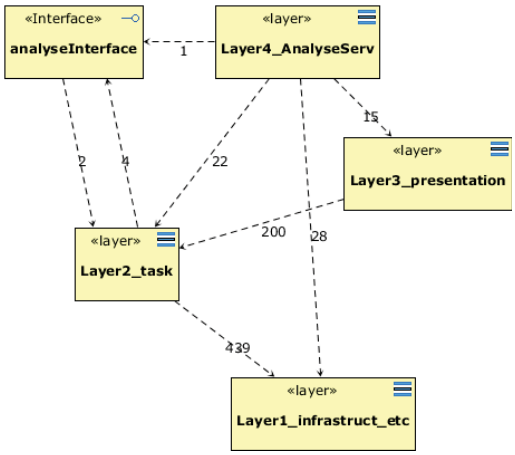

Figure 4: Intended architecture diagram with SAR result

diagram. The submodules of *analyse* with their module types are visible. A black, dashed arrow in the diagram represents dependencies, of which the number of dependencies is shown.

4 SAR RESEARCH SUPPORT

Researchers can make use of the SAR support, described in the previous section, to study the results of existing algorithms and parameters. Moreover, researchers can develop their own algorithms relatively easily. In addition, functionality is provided to measure the effectiveness of an algorithm.

4.1 Algorithm Development

New algorithms need to be programmed in Java within the context of HUSACCT, of which the source can be downloaded from github.com/HUSACCT/HUSACCT. The reconstruction algorithms are located in module *reconstruct* within the Task layer of component *analyse*, as visible in Fig. 2 where the submodules of *reconstruct* are visible as well. The SAR GUI is included in the Presentation layer. A general mechanism is implemented to easily include a new algorithm in the GUI, to support the control options within the GUI, and to work with parameters. If needed, an additional tab can be created within the GUI, to group a list of new algorithms.

Algorithms have to communicate with their environment. For this purpose, service interfaces of several components are available. Fig. 5 shows the context of module *reconstruct*. *AnalyseDomainService* provides data on request from the repository with code analysis results; the software units and their dependencies. The service interfaces of component *define* may be used to request data on the existing intended architecture and to add, edit or delete modules and rules. *ValidateService* may provide data on violations to a given rule.

4.2 Algorithm Effectiveness

To measure the effectiveness of algorithms, we make use of the MoJoFM metric, an effectiveness measure for software clustering algorithms based on MoJo distance as presented by Wen and Tzerpos [12]. We use their implementation MoJo 2.0 (downloaded April 2016 from: www.cs.yorku.ca/~bil/downloads).

Fig. 6 shows the MoJoFM GUI within HUSACCT. In the panel "Compare Architectures", two input files with decomposition structures have to be provided (in a specific format): a gold

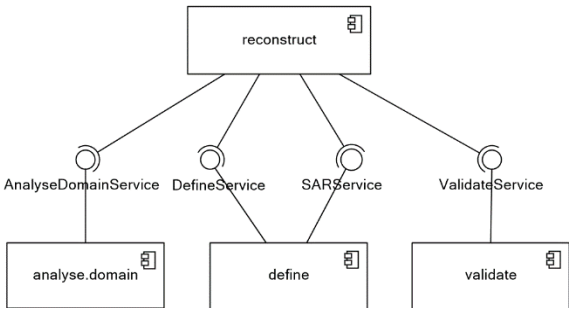

Figure 5: The context of the Reconstruct module

standard, and a to-be-compared structure. An algorithm that produces a decomposition structure deviating completely from the gold standard results in a MoJoFM value of 0%, while an algorithm that produces the same decomposition structure results in a MoJoFM value of 100%. The input files can be created by means of the provided functionality in the top panel.

For example, first we have created an intended architecture of HUSACCT manually, which acts as the gold standard, and we have exported the architecture in the required file format. For the reconstructed architecture in Fig. 2, we did the same. Comparison resulted in a MoJoFM value of 92.31%, as visible in Fig. 6. Please note that the MoJoFM metric does not take the module types into account, only the decomposition structure and the assignment of the software units to the modules.

4 DISCUSSION

In a previous study [7], we reported on the results of an SRMA-test on eight academic and commercial SACC-tools. However, none of these tools provide SAR support similar to HUSACCT. For example, Structure101 (structure101.com) reconstructs a useful view on the code. Software units are structured vertically and horizontally, based on the package structure and on dependencies between the packages. A difference with our approach is that Structure101 does not provide an intended architecture as in our approach, with different module and rule types. Rules are linked to software units directly. Furthermore, Structure101 does not provide a set of algorithms and a GUI that allows an interactive and incremental process of reconstruction.

The algorithms in this paper are used to illustrate the SAR environment and the SAR process; nothing more. Our layers reconstruction approach and the role of its parameters is described in [11]. The approach is partly based on published algorithms, especially those of Goldstein and Segal [3] and Laval et al. [4]. For an explanation of the algorithm and a discussion of related work, we refer to the paper itself. The other "practical approaches" have not yet been presented in published work. They are subject to ongoing research. Although the iterative algorithm, which is used in the examples in this paper, performs well in case of HUSACCT's own source code, it does not have to do so in case of other systems. Especially the identification of components and the facade pattern is problematic, since numerous variations in the source code exist.

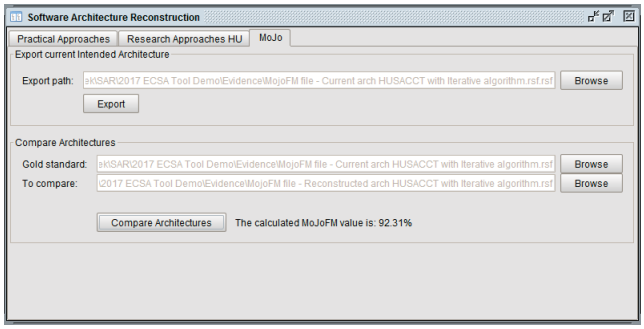

Figure 6: MoJoFM GUI within HUSACCT

Finally, not all dependencies in the code are detected and reported by HUSACCT; same as with other tools [9]. This may influence the results of an algorithm. Deficiencies may be present in HUSACCT itself or in the included open source lexer and parser. However, we have taken great care of the accuracy of dependency analysis. We are making use of large automated test sets to cover many types and subtypes of dependencies.

5 STATUS AND OUTLOOK

HUSACCT supports software architecture conformance checks (SACC), but since version 5 it provides extensive software architecture reconstruction (SAR) support as well. HUSACCT distinguishes itself from other tools in its extensive and configurable support of rich sets of module and rule types.

Currently, HUSACCT is in its sixth year of development and each year it is used in student courses on software architecture, and it is used to performed SACC and/or SAR on open source systems and professional systems. The SAR functionality is in its second year of development, and many improvements have been made to the GUI, the control mechanism, and the algorithms. The SAR user environment appears to be stable and several algorithms provide useful results in practical cases.

Future work will focus on improvement of existing algorithms, the development of new algorithms, and the effectiveness of these algorithms in practical cases.

REFERENCES

[1] Clements, P. et al. 2010. *Documenting Software Architectures: Views and Beyond*. Pearson Education.

[2] Ducasse, S. and Pollet, D. 2009. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *IEEE Transactions on Software Engineering*. 35, 4 (2009), 573–591.

[3] Goldstein, M. and Segall, I. 2015. Automatic and Continuous Software Architecture Validation. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. (2015), 59–68.

[4] Laval, J. et al. 2013. OZONE: Layer Identification in the presence of Cyclic Dependencies. *Science of Computer Programming*. 78, 8 (2013), 1055–1072.

[5] Perry, D.E. and Wolf, A.L. 1992. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*. 17, (1992), 40–52.

[6] Peters, J. et al. 2016. Architectural Pattern Definition for Semantically Rich Modular Architectures. *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)* (Venice, 2016), 256–261.

[7] Pruijt, L. et al. 2013. Architecture Compliance Checking of Semantically Rich Modular Architectures: A Comparison of Tool Support. *2013 IEEE International Conference on Software Maintenance* (2013), 220–229.

[8] Pruijt, L. et al. 2014. HUSACCT: Architecture Compliance Checking with Rich Sets of Module and Rule Types. *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering - ASE '14* (Sep. 2014), 851–854.

[9] Pruijt, L. et al. 2016. The Accuracy of Dependency Analysis in Static Architecture Compliance Checking. *Software: Practice and Experience*. (2016).

[10] Pruijt, L. and Brinkkemper, S. 2014. A metamodel for the support of semantically rich modular architectures in the context of static architecture compliance checking. *WICSA 2014 Companion Volume* (Apr. 2014), 1–8.

[11] Pruijt, L. and Wiersema, W. 2016. Dependency Related Parameters in the Reconstruction of a Layered Software Architecture. *Proceedings of the 10th European Conference on Software Architecture Workshops* (2016), 1–7.

[12] Wen, Z.W.Z. and Tzerpos, V. 2004. An effectiveness measure for software clustering algorithms. *12th IEEE International Workshop on Program Comprehension* (2004), 194–203.