Featured Scents

Lima dos Santos, Edilton; Schobbens, Pierre-Yves; Perrouin, Gilles

Published in:
2022 IEEE 19th International Conference on Software Architecture Companion, ICSA-C 2022

DOI:
10.1109/icsa-c54293.2022.00026

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

Featured Scents: Towards Assessing Architectural Smells for Self-Adaptive Systems at Runtime

Edilton Lima dos Santos
PReCISE, NaDI,
Faculty of Computer Science,
University of Namur
Namur, Belgium
edilton.limados@unamur.be

Pierre-Yves Schobbens
PReCISE, NaDI,
Faculty of Computer Science,
University of Namur
Namur, Belgium
pierre-yves.schobbens@unamur.be

Gilles Perrouin
PReCISE, NaDI,
Faculty of Computer Science,
University of Namur
Namur, Belgium
gilles.perrouin@unamur.be

***Abstract*—Self-adaptive systems (SAS) change their behavior and structure at runtime to answer the changes in their environment. Such systems combine different architectural fragments or solutions via feature binding/unbinding at runtime. Moreover, this combination may negatively impact the system's architectural qualities, exhibiting architectural bad smells (ABS). These issues are challenging to detect in the code due to the combinatorial explosion of interactions amongst features. Since SAS does not document these features in their source code, design time smell detection ignores them and risks reporting smells that are different than those observed at runtime. This paper assesses this risk to understand how ABS occurs at runtime for different feature combinations. We look for cyclic dependency and hub-like ABS in various runtime adaptations of two SAS, Adasim and mRubis. Our results indicate that architectural smells are feature-dependent and that their number is highly variable from one adaptation to the other. Some ABS appear in all runtime adaptations, some in only a few. We discuss the reasons behind these architectural smells for each system and motivate the need for targeted ABS analyses in SAS.**

***Index Terms*—Architectural Smells, Self-Adaptive Systems, Arcan, Behavioral Maps.**

I. Introduction

Nowadays, more and more self-adaptive systems (SAS) are required to run without interruptions in heterogeneous environments. SAS vary their behavior through the (de)activation of *features* depending on environmental changes and reconfiguration plans and goals to answer such requirements. Such a reconfiguration combines architectural fragments or different solutions at runtime that may negatively impact their architectural qualities. Thus, Architectural Bad Smells (ABS) may emerge, implying reductions in system maintainability [1], [2]. ABS result from architectural design decisions that negatively impact system lifecycle properties (e.g., testability and maintainability) [3].

Variability management is key for SAS as well as for software product lines (SPL), where one derives a family of variants based on *core* (present in all variants) and *optional* (present in some variants) features [4]. Many studies target ABS in single systems or [1]–[3], [5]–[7]. However, there are less studies focusing on SAS or dynamic software product lines (DSPL) [8]–[10]. In particular, these studies do not discuss the impact of runtime variability on smell detection and evolution as the SAS adapt. To assess runtime architectural qualities of SAS, we developed a framework that instruments SAS to monitor runtime adaptations and captures them in *behavioral maps* (BM) [10].

This poster paper presents a preliminary comparison between design time and runtime smell detection for SAS. To perform this comparison, we selected two ABS detection tools, Arcan [11] and our BM framework [10]. We motivate the choice of the former tool because it was applied on SAS at design time [8]. The BM framework is the only approach to focus on runtime ABS. Both tools focus on SAS written in Java. We selected Adasim [12] and mRubis [13] for their public availability and the diversity of adaptation mechanisms these SAS use. As for smells, we considered Cyclic Dependency (CD) and Hub-Like Dependency (HL) [14] as they are supported by both Arcan and BM framework tools.

Our results show important differences between smells occurrences at design time and runtime for Adasim, and smells appearing at runtime not found at design time for mRubis. ABS occurrences also vary along SAS reconfigurations. Our results suggest that runtime ABS assessment is required to fully grasp SAS architectural qualities. In summary, this paper provides the following contributions: i) A first empirical comparison of architectural bad smells for SAS detected at design time and at runtime; ii) Our analysis based on 40 runtime adaptations of Adasim and 16 runtime adaptations of mRubis, demonstrates that runtime variability affects the type and occurrence of smells found. The results and scripts to process behavioral maps are also available here: https://doi.org/10.5281/zenodo.5814028.

The remainder of the paper is as follows. Section II introduces research questions. We describe our results in Section III. Finally, Section V wraps up the paper.

II. Study Design

A. Research Questions

This empirical study aims at investigating differences between smells one detects at design time and smells occurring at runtime. To understand these differences, we formulate the following research questions:

RQ1. Are smells found at design time also found at runtime? This involves: *i)* running different configurations of self-adaptive systems, *ii)* measure the type and number of occurrence of each smell, and *iii)* compare these results with smells detected at design time.

RQ2. How does the number of architectural bad smells evolves during the reconfiguration process at runtime? We aim at qualifying the variations of smell occurrences at runtime, and this is related to the SAS variability.

B. Systems under Study

We have chosen Adasim [12] and mRubis [13] systems, both written in the Java programming language and available in a reference repository for SAS[1]. Besides the programming language, the motivation of these choices also relies on the adaptive mechanisms. The former system uses the parameter-based routing algorithm to trigger their adaptation process, and the last uses a mix of adaptation mechanisms (MAPE-K [15], Event-Condition-Action (ECA), and State-based feedback loop) depending on the instantiated version. This is relevant to study the impact of the variability on ABS occurrences.

Adasim is a simulator for the Automated Traffic Routing Problem (ATRP), implemented as an agent-based system [8], [12]. The system is composed of six kinds of abstract entities: i) a map; ii) vehicles; iii) agents that make routing decisions and collect and store information; iv) sensors that allow agents to observe the environment; v) measurement uncertainty filters that control the noise and other sources of uncertainty in the sensor measurements; and vi) data privacy policies that allow vehicles and streets to restrict part or all information about themselves from sensors [12]. The self-adaptive mechanisms are utilized to deal with the scalability problems and the unexpected changes in the environment, for instance, an accident or a closed street.

mRUBiS is a marketplace on which users sell or auction items [13] based on RUBiS [16], a popular case study to evaluate control-theoretic adaptation. mRubis comprises 18 components and can arbitrarily host many shops. These shops manage items, users, auctions/purchases, inventory, and user ratings, authenticate users, and persist and retrieve data from the database.

C. Architectural Bad Smells

The architectural smells considered in our study are:

Cyclic Dependency (CD): CD occurs when two or more components depend on each other directly or indirectly [14].

Hub-Like Dependency (HL): HL arises when a component has (outgoing and ingoing) dependencies with a large number of other abstractions (*e.g.,* other components) [8], [14].

D. Experimental Setup

We ran Arcan on the Jar files available on GitHub for Adasim[2] and mRubis[3]. We instantiated the BM framework on both systems, a result of a two weeks work from the first author.

[1]https://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/

[2]https://github.com/brunyuriy/adasim

[3]https://github.com/thomas-vogel/mRUBiS

TABLE I: ABS identified by Arcan and Behavioral Map.

	Arcan		Behavioral Map		
Feature Name	**CD**	**HL**	**CD**	**HL**	**Feature Type**
TrafficSimulator	Yes		Yes		Core
RoadSegment	Yes	Yes	Yes	Yes	Core
Vehicle	Yes		Yes	Yes	Core
VehicleManager	Yes		Yes		Core
RoadVehicleQueue	Yes		Yes		Core
AdasimMap			Yes		Core
QLearningRoutingAlgorithm			Yes		Optional
SimulationXMLBuilder				Yes	Core

TABLE II: ABS identified by the BM in adaptation 1 and 2 of the Adasim - QLearningRoutingAlgorithm.

		Adaptation 1		Adaptation 2	
Feature Name	**Feature Type**	**CD**	**HL**	**CD**	**HL**
TrafficSimulator	Core	Yes		Yes	
RoadSegment	Core	Yes	Yes (13)	Yes	Yes (12)
Vehicle	Core	Yes	Yes (14)	Yes	Yes (13)
VehicleManager	Core	Yes		Yes	
RoadVehicleQueue	Core	Yes		Yes	
AdasimMap	Core	Yes		Yes	
QLearningRoutingAlgorithm	Optional	Yes			
SimulationXMLBuilder	Core		Yes (9)		Yes (9)

III. Results

A. Adasim Results

Adasim is a parameter-based routing algorithm adaptive mechanism, and we identified two adaptation modes according to the algorithms initiating the reconfiguration process: QLearningRoutingAlgorithm and AdaptiveRoutingAlgorithm.

Adasim QLearningRoutingAlgorithm: Table I presents the ABS identified by Arcan and BM. For the last tool, we show only the smells identified in the first adaption loop. Additionally, the table shows the feature type affected for each ABS. The Arcan analysis identified ABS only in the core features. Thus, the TrafficSimulator, RoadSegment, Vehicle, VehicleManager, and RoadVehicleQueue are CD, and RoadSegment is HL.

BM found the same features identified by Arcan involved in ABS and identified three more additional features, as we depicted Table I. Thus, the core feature AdasimMap and the optional feature QLearningRoutingAlgorithm were identified as a CD. Also, the core feature SimulationXMLBuilder was identified as HL. These ABS were identified in the first adaptation loop executed by Adasim.

We analyzed Adasim during 13 self-adaptations and found that the number of detected ABS only changed between the first and second adaptations. Further adaptations did not affect further the architecture. Table II presents in detail the features involved in ABS during the two first adaptations. The QLearningRoutingAlgorithm is an optional feature involved in CD

only in adaptation 1 with the feature RoadSegment, and Vehicle. Nevertheless, the absence of the QLearningRoutingAlgorithm (in adaptation 2) reduces the numbers of dependency in the features RoadsSegment and Vehicle involved in HL, see Table II. This situation occurred because RoadSegment and Vehicle are not sharing QLearningRoutingAlgorithm in the second adaptation.

Adasim AdaptiveRoutingAlgorithm. In this mode, we monitored the system during 27 self-adaptations, involving 20 features. Similarly, we observed differences between the two first adaptations. Table III presents the ABS identified during adaptations one and two. We observe that the number of CD identified increase or decrease depending on the number of optional features required in each adaptation process. This situation also impacts the number of HL identified in each adaptation, mainly because the features identified as CD and HL concentrated on the core features. Also, there is a strong dependency amongst them at runtime. Thus, we detected that the Vehicle feature identified as HL in Adaptation 1 was not identified in Adaptation 2. Such a situation occurred because the optional features AdaptiveRoutingAlgorithm, QLearningRoutingAlgorithm, and LookaheadShortestPathRoutingAlgorithm are not used in adaptation 2. Consequently, the BM identified in adaptation 2 the RoadSegment feature as a new HL, as identified by Arcan.

B. mRubis Results

The mRubis system is divided into self-healing and self-optimization versions.

mRubis self-optimization: Table IV details the ABS identified by Arcan and the BM framework. Arcan analysis identified the `SimulatorUtil` as HL and the classes `ModelParameterPage` and `ModelParameterPage$1` as CD in the mRubis self-optimization version. The `SimulatorUtil` is a class part of the framework used to implement the mRubis simulator, but the Arcan tool identified the class as a mRubis implementation. Also, the classes `ModelParameterPage` and `ModelParameterPage$1` are optional features responsible for implementing the graphical interface.

TABLE III: ABS identified by the BM in adaptation 1 and 2 of the Adasim AdaptiveRoutingAlgorithm.

		Adaptation 1		Adaptation 2	
Feature Name	**Feature Type**	**CD**	**HL**	**CD**	**HL**
TrafficSimulator	Core	Yes		Yes	
RoadSegment	Core	Yes		Yes	Yes (13)
Vehicle	Core	Yes	Yes (17)	Yes	
VehicleManager	Core	Yes		Yes	
RoadVehicleQueue	Core	Yes		Yes	
AdasimMap	Core	Yes		Yes	
AdaptiveRouting Algorithm	Optional	Yes			
QLearningRouting Algorithm	Optional	Yes			
LookaheadShortest PathRoutingAlgorithm	Optional	Yes			
SimulationXMLBuilder	Core		Yes (11)		Yes (11)

TABLE IV: Architectural Bad Smells identified by Arcan and Behavioral Map in mRubis Self-Optimization.

	Arcan		Behavioral Map		
Feature Name	**CD**	**HL**	**CD**	**HL**	**Feature Type**
SimulatorUtil		Yes			
ModelParameterPage	Yes				Optional
ModelParameterPage$1	Yes				Optional
SelfOptimizationConfig				Yes	Core
MRubisModelQuery				Yes	Core
EventBasedMapeFeedbackLoop				Yes	Core

TABLE V: Architectural Bad Smells identified by Arcan and Behavioral Map in mRubis Self-Healing MAPE-K loop.

	Arcan		Behavioral Map		
Feature Name	**CD**	**HL**	**CD**	**HL**	**Feature Type**
SimulatorUtil		Yes			
ModelParameterPage	Yes				Optional
ModelParameterPage$1	Yes				Optional
StateBasedMapeFeedbackLoop				Yes	Core

However, the BM identified the SelfOptimizationConfig, MRubisModelQuery, and EventBasedMapeFeedbackLoop as HL in four adaptation loops. Thus, these features are core used in all configurations of mRubis self-optimization. We observed in the SelfOptimizationConfig feature a decrease in the numbers of dependencies used in the second adaptation. This situation occurred because the feature is responsible for adding the validators and other parameters related to self-optimization. However, the number of validators used at runtime decreases, impacting the dependencies identified. The MRubisModelQuery and EventBasedMapeFeedbackLoop maintain the same numbers of dependencies in all adaptations. Also, the BM framework did not identify other types of ABS during the adaptation loop.

mRubis self-healing: The Arcan identified the same ABS identified in the mRubis self-optimization version because the classes the `SimulatorUtil`, `ModelParameterPage` and `ModelParameterPage$1` are used in the mRubis build independent of the version. However, the BM does not identify ABS in the self-healing version with adaptation mechanism ECA and SBFL after four reconfiguration processes at runtime. The difference between the results presented by Arcan and BM is triggered by the way how ABSs are identified. Table V presents in detail the ABS identified by Arcan and BM for self-healing version with adaptation mechanism MAPE-K. The BM identified one instance of HL in the core feature StateBasedMapeFeedbackLoop in four adaptations loops. The feature is the main entry point to other features as Monitor, Action, Plan, Execute, SelfHealingConfig, SelfHealingScenario, and MRubisSelfHealingUtilityFunction.

C. Synthesis

Answering RQ1. For both systems we observed significant differences between design time and runtime smell detection. Some smells are only present at runtime and conversely some smells only appear at design time. By answering no to RQ1, we motivate the needs for further assessment of runtime smells for SAS.

Answering RQ2. We also observed a variation in the occurrences of smells found between the adaptations. For instance, in Adasim AdaptiveRoutingAlgorithm, the BM found 9 CD and 3 HL in the first adaptation, but in the second, the BM found 6 CD smells. We could explain this variation by the activation and deactivation of certain runtime features.

IV. Threats to Validity

Internal Validity: The absence of a feature model and feature annotations in the source code may hamper the variability identification process. To mitigate this threat, we used the Eclipse IDE tool to verify the feature implementation and to debug the systems' source code to check the execution of each feature identified using the process. We also analyzed execution logs to ensure that our identification of features was correct.

External Validity: Our results may not generalize to all SAS, since we selected only two systems in our studies. Further, it is impossible to run all the possible systems adaptations or to estimate their number. However, the selected systems have different architectural models, adaptation mechanisms and application domains. Diversity in the selected systems contributes to mitigate this threat. We recall that our goal was not to obtain statistical evidence on the differences between runtime and design time architectural bad smells but rather to reveal and explain their existence. A more quantitative assessment is left for future work.

V. Conclusion

In this paper, we made the case for assessing architectural bad smells (ABS) for self-adaptive systems (SAS) at runtime, hypothesizing differences between design time and runtime analyses. To reveal and explain these differences, we selected two SAS (Adasim and mRubis), two ABS detection tools (Arcan and the Behavioral Map framework) and performed design time and runtime smell detection on a number of systems reconfigurations. Our results showed that there are indeed differences between design time and runtime detection. While we could have assumed that all smells found at runtime would be a subset found by at design time, we also found occurrences of smells only found at runtime. We identified the root causes for this seemingly surprising finding, including polymorphism that affects the precision of design time analysis. These differences are of interest for SAS architects in order to more precisely put their maintenance efforts and assess the architectural qualities of a given runtime adaptation. However, instrumenting such SAS for runtime ABS identification requires expertise and time, especially since core and variable features are not documented. This the main lesson learned of our study besides our results.

There is room for future work. First, we would like to reduce the cost of engineering involved in analyzing SAS at runtime, which is a current impediment of large-scale analyses. In particular, we will design a dedicated ABS tool operating at the bytecode level, easing runtime analyses. Second, we will generalize our findings by assessing more SAS.

Acknowledgment

Edilton Lima dos Santos is funded by a CERUNA grant from the University of Namur. Gilles Perrouin is an FNRS Research Associate.

References

[1] M. Lippert and S. Roock, *Refactoring in large software projects: performing complex restructurings successfully*. John Wiley & Sons, 2006.

[2] H. S. de Andrade, E. Almeida, and I. Crnkovic, "Architectural bad smells in software product lines: An exploratory study," in *Proceedings of the WICSA 2014 Companion Volume*, 2014, pp. 1–6.

[3] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Toward a catalogue of architectural bad smells," in *International conference on the quality of software architectures*. Springer, 2009, pp. 146–162.

[4] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[5] F. A. Fontana, P. Avgeriou, I. Pigazzini, and R. Roveda, "A study on architectural smells prediction," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019, pp. 333–337.

[6] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying architectural bad smells," in *13th European Conference on Software Maintenance and Reengineering*. IEEE, 2009, pp. 255–258.

[7] H. Mumtaz, P. Singh, and K. Blincoe, "A systematic mapping study on architectural smells detection," *Journal of Systems and Software*, 2020.

[8] C. Raibulet, F. Arcelli Fontana, and S. Carettoni, "A preliminary analysis of self-adaptive systems according to different issues," *Software Quality Journal*, pp. 1–31, 2020.

[9] M. A. Serikawa, A. d. S. Landi, B. R. Siqueira, R. S. Costa, F. C. Ferrari, R. Menotti, and V. V. De Camargo, "Towards the characterization of monitor smells in adaptive systems," in *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*, IEEE. IEEE, 2016, pp. 51–60.

[10] E. L. dos Santos, S. Fortz, G. Perrouin, and P.-Y. Schobbens, "A vision to identify architectural smells in self-adaptive systems using behavioral maps," in *15th European Conference on Software Architecture (ECSA 2021)*. CEUR Workshop Proceedings, 2021, p. 1.

[11] F. A. Fontana, I. Pigazzini, R. Roveda, D. Tamburri, M. Zanoni, and E. Di Nitto, "Arcan: A tool for architectural smells detection," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 282–285.

[12] J. Wuttke, Y. Brun, A. Gorla, and J. Ramaswamy, "Traffic routing for evaluating self-adaptation," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2012, pp. 27–32.

[13] T. Vogel, "mrubis: An exemplar for model-based architectural self-healing and self-optimization," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, 2018, pp. 101–107.

[14] U. Azadi, F. A. Fontana, and D. Taibi, "Architectural smells detected by tools: a catalogue proposal," in *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 2019, pp. 88–97.

[15] IBM, "An architectural blueprint for autonomic computing," *IBM White Paper*, vol. 31, pp. 1–6, 2006.

[16] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A systematic survey on the design of self-adaptive software systems using control engineering approaches," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE, 2012, pp. 33–42.