A Tool to Improve the Detection of Antipatterns in SOA Systems by Mining Execution Traces

Mathieu Nayrolles, Naouel Moha and Petko Valtchev
LATECE Team, Département d'informatique, Université du Québec à Montréal, Canada
mathieu.nayrolles@gmail.com, {moha.naouel,valtchev.petko}@uqam.ca

***Abstract*—Service Based Systems (SBSs), like other software systems, evolve due to changes in both user requirements and execution contexts. This continuous evolution could easily deteriorate the design and reduce the Quality of Service (QoS) of SBSs and may introduce poor design solutions, commonly known as SOA antipatterns. SOMAD is a tool that aims to ensure the QoS of SBSs. SOMAD mines strong associations between sequences of service/method calls from the execution traces of a SBS to (1) specify and detect SOA antipatterns. This tool also provides a (2) visualization of the targeted system and can (3) help software architects and engineers to evolve their systems. Experiments show that SOMAD has a precision higher than 87.5% and a 100% recall on small to large scale SBSs and clearly outperforms the only other tool named SODA.**

***Index Terms*—SOA Antipatterns, Mining Execution Traces, Sequential Association Rules, Service Oriented Architecture.**

I. Introduction

Service Based Systems (SBSs) are composed of ready-made services that are accessed through the Internet [1]. Services are autonomous, interoperable, and reusable software units that can be implemented using a wide range of technologies like Web Services, REST (REpresentational State Transfer), or SCA (Service Component Architecture, on the top of SOA). Most of the world biggest computational platforms: Amazon, Paypal, and eBay, for example, represent large scale SBSs. Such systems are complex–they may generate massive flows of communication between services–and highly dynamic: services appear, disappear or get modified. The constant evolution in an SBS can easily deteriorate the overall architecture of the system and thus make architectural defects, known as SOA antipatterns [2], appear. An antipattern is the opposite of a design pattern: while design patterns should be followed to create more maintainable and reusable systems, antipatterns should be avoided since they have a negative impact, e.g., hinder the maintenance and reusability of SBSs.

In this paper, we present an innovative tool named SOMAD (Service Oriented Mining for Antipattern Detection). This tool relies on execution traces produced by SBSs and is able to eliminate the non-relevant data using data mining techniques and then, detect antipatterns. SOMAD discovers antipatterns by first extracting associations between services as expressed in the execution traces of an SBS. To that end, we apply a specific variant of the association rule mining task based on sequences or episodes [3]: In our case, the sequences represent services or, alternatively, method calls. Further, we filter these generated association rules by means of a suite of dedicated metrics.

Multi-Service, *a.k.a* God Object corresponds to a service that implements a **multitude of methods** related to different business and technical abstractions. It aggregates too much into a single service, such a service is not easily reusable because of the **low cohesion** of its methods and is often unavailable to end-users because of its overload, which may induce a high response time [9].

Tiny Service is a small service with **few methods**, which only implements part of an abstraction. Such service often requires **several coupled services** to be used together, resulting in higher development complexity and reduced usability. In the extreme case, a Tiny Service will be limited to **one method**, resulting in many services that implement an overall set of requirements [9].

Table I: List of SOA Antipatterns [2]

The remainder of this paper is organized as follows: Section II discusses related works. Section III presents the underlying approach of our tool and its main features. Section IV introduces our validation studies and we provide some concluding remarks in Section V.

II. Related Works

With the goal of detecting OO code and design related issues, a number of tools have been introduced in the literature [4], [5], [6]. Nevertheless, the possibility to perform detection for SOA antipatterns has been rarely considered. Indeed, the only other–and therefore, state-of-the-art–available tool named SODA [7] was recently developed by our team. We contribute to the progress in this area by proposing an approach, called SOMAD [8], as a support for detecting SOA antipatterns in any SOA technologies and significantly improves SODA's precision.

III. Overview of SOMAD

We developed the tool SOMAD being inspired from the SOMAD approach [8]. Figure 1 represents the four main steps of SOMAD : (1) Specifying SOA antipatterns in the form of rule cards from their textual description (see Table I), (2) Generating detection algorithms from rule cards, (3) Mining association rules from execution traces and (4) Detecting SOA antipatterns. SOMAD can detect SOA antipatterns on any SBSs since it is based on execution traces that may come from any kind of technologies (Web Services, SCA, Rest).

Step 1. Specification SOA antipatterns: After a careful analysis of SOA antipattern textual description we create rule cards for six different antipatterns; Multi-Service, Tiny Service, Chatty Service, Knot Service, BottleNeck Service and Chain service. Rule cards are combination of height different metrics

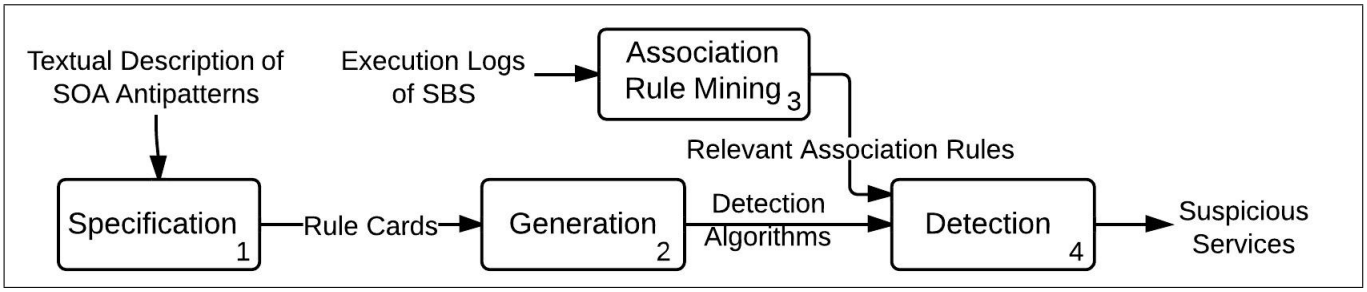

Figure 1: The SOMAD approach

dedicated to SOA antipattern detection. The definition of SOA antipatterns and metrics definition have been specified in [8].

Step 2. Generation of Detection Algorithms: This step consists in automatically generating detection algorithms by visiting models of rule cards specified in the previous step. This process is straightforward and enables the automatic generation of detection algorithms directly executable.

Step 3. Association Rules Mining. In the data-mining field, Association Rule Mining (ARM) is a well-established method for discovering co-occurences between attributes in the objects of a large data set [10]. Plain association rules have the form $X \longrightarrow Y$, where X and Y are sets of descriptors. While plain association rules could provide interesting information, we decide to use a variant named Sequential Association Rules to preserve the sequences of service/method calls. Applying this technique on execution traces files identify interesting relation between services like : *ServiceA*, *ServiceB* implies *ServiceC*; meaning that if services A and B appear, there is a high probability for C to appear on the same sequence of service calls.

Step 4. Detection of SOA antipatterns: This step consists in applying the detection algorithms generated in Step 3 on the sequential association rules mined in Step 4. At the end of this step, services in the SBS suspected to be involved in an antipattern are identified.

SOMAD have three main features supported by the approach described before:

- Specify SOA antipatterns using our metric catalogue.
- Detect SOA antipatterns in SBSs.
- Provides a visualization of the targeted system.

Another feature currently under development is to support the addition of new services using their description (for example Web Services Description Language), and recompute the SOA antipattern detection considering these additions.

IV. Experiments

As a validation study, we apply SOMAD on two independently developed SBSs, *Home Automation* and *FraSCAti* [11]. *Home Automation* is an SBS made of 13 services and selected for comparison with the outcome produced by SODA [7], the so far unique state-of-the-art tool for antipattern detection. Both tools were evaluated in terms of precision and recall, on one hand, and efficiency, on the other hand. We also apply SOMAD to *FraSCAti*, an SBS almost 10 times larger than *Home Automation*, which contains 91 components and 130 services. Table II displays our results in term of size (number of services), A.AP (Automatically Detected Antipatterns), M.AP (manually detected antipatterns), recall, precision and time for the detection. This experiments show that SOMAD has a precision up to 94.44% and a 100% recall. Moreover, for SOMAD the obtain precision are better than SODA by a fairly comfortable margin ranging from 2.6% from 16.67%.

Tool	System	Size	A.AP	M.AP	Recall	Precision
SOMAD	*HomeAut.*	13	12	10	100%	90.1%
SODA	*HomeAut.*	13	12	10	100%	87.5%
SOMAD	*FraSCAti*	130	8	7	100%	94.44%
SODA	*FraSCAti*	130	10	7	100%	77.77%

Table II: SOA Antipatterns Detection Results for HomeAutomation and FraSCAti.

V. Conclusion and Future Work

In this paper, we presented a tool named SOMAD that enables the detection SOA antipatterns detection in any SOA technologies and outperforms SODA in term of precision. Currently, we developing the support of services addition to master the evolution of SBSs. As a future work, we intend to improve our association rules mining algorithms and we envision SOMAD in the context of a large data center whereby the goal would be to optimize the data center communication. A video of SOMAD can be found at http://sofa.uqam.ca/somad.

References

[1] T. Erl, *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2006.

[2] N. Moha, F. Palma, M. Nayrolles *et al.*, "Specification and detection of soa antipatterns," in *ICSOC*, 2012.

[3] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, pp. 259–289, 1997.

[4] M. Fokaefs, N. Tsantalis, and A. Chatzigeorgiou, "JDeodorant: Identification and Removal of Feature Envy Bad Smells," in *ICSM*, Oct. 2007, pp. 519–520.

[5] R. Marinescu, "Detection Strategies: Metrics-based Rules for Detecting Design Flaws," in *IEEE ICSM*, 2004.

[6] N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. L. Meur, "Decor: A method for the specification and detection of code and design smells," *IEEE TSE*, vol. 36, no. 1, pp. 20–36, Jan. 2010.

[7] M. Nayrolles, F. Palma, and Moha, "Soda : A tool for automatic detection of soa antipatterns," in *ICSOC - Tool Paper*, 2012.

[8] M. Nayrolles, N. Moha, and P. Valtchev, "Detection of soa antipatterns: Towards a non-intrusive and technology agnostic approach." in *Submitted to WCRE'13*, 2013.

[9] B. Dudney, S. Asbury, J. Krozak, and K. Wittkopf, *J2EE AntiPatterns*. John Wiley & Sons Inc, 2003.

[10] G. Piatetsky-Shapiro, "Discovery, analysis, and presentation of strong rules," *KDD*, pp. 229–238, 1991.

[11] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani, "A component-based middleware platform for reconfigurable service-oriented architectures," *SPE*, vol. 42, no. 5, pp. 559–583, 2012.