# Detection of SOA Antipatterns

Francis Palma[1,2]

[1] Ptidej Team, DGIGL, École Polytechnique de Montréal, Canada
[2] Département d'informatique, Université du Québec à Montréal, Canada
francis.palma@polymtl.ca

**Abstract.** Like any other large and complex systems, user requirements may change for Service Based Systems (SBSs), as well as their execution contexts, in the form of evolution and maintenance. Consequently, these changes may cause degradation of design, and Quality of Service (QoS), resulting to the bad practiced solutions, commonly known as *Antipatterns*. Therefore, detecting SOA (Service Oriented Architecture) antipatterns deserves an extra importance for assessing the design and QoS of SBSs. Also, this detection may facilitate the future evolution and maintenance. Despite of its importance, there are no methods and techniques for detecting SOA antipatterns within SBSs. The subject of my PhD thesis is to propose a novel and innovative approach, supported by a framework for specifying and detecting SOA antipatterns. My contributions are: (1) an approach for SOA antipatterns detection, (2) a framework supporting analysis and detection for SOA antipatterns in SBSs, and finally (3) a concrete empirical evidence to show the effectiveness of the proposed approach and framework.

**Keywords:** SOA Antipatterns, Service Based Systems, Detection, Quality of Service, Design, Software Evolution and Maintenance.

## 1 Introduction

The wide acceptance of SOA [2] is mainly due to its flexibility, scalability and inexpensive development efforts. SBSs developed with such architectural style, are composed of loosely-coupled and platform independent reusable units, i.e., *service* easily accessible over Internet. Amazon and eBay are two examples of SBSs. In fact, the emergence of SBSs is unable to avoid some common software engineering challenges, e.g., evolution, to fit new user requirements or changes in execution contexts. All these changes may degrade the quality of design and QoS of SBSs, thus may cause the presence of common bad practiced solutions, i.e., *antipatterns* - oppose to *design patterns*, that are good solutions to recurring problems. *Multi Service* and *Tiny Service* are two common and recurring antipatterns in SBSs, and it is shown, in particular, *Tiny Service* is the root cause of many SOA failures [5]. *Multi Service* is an SOA antipatterns that corresponds to a service implementing a multitude of methods related to different business and technical abstractions. Such a service has low reusability and is often unavailable to the end-users [1]. Conversely, *Tiny Service* is a small service with just a few methods, which only implements part of an abstraction. Such

A. Ghose et al. (Eds.): ICSOC 2012, LNCS 7759, pp. 412–418, 2013.
© Springer-Verlag Berlin Heidelberg 2013

service often requires several coupled services to be used together, resulting in higher development complexity and reduced usability [1].

The remainder of this paper is organized as follows. The next few Sections discuss more details on the problem. Section 2 summarizes the related work. Section 3 introduces the proposed approach and the framework, in short. Section 4 identifies key research assumptions and presents some preliminary results. Finally, Section 5 concludes and sketches the future work.

**Context: Assessment of Design and QoS, Maintenance and Evolution of SBSs -** The 'bad' solutions to the recurring problems, i.e., *antipatterns* are the cause of low maintainability and may hinder the evolution of the system (Klimas et al., 1996). Like Object Oriented (OO) systems, SBSs also faces challenges with low maintainability and evolvability due to *antipatterns*. Thus, it becomes difficult for engineers to make changes, i.e., adding new or modifying existing functionalities.

**Problem: No Approach and Framework for the Detection of SOA Antipatterns -** Promoting the adoption of SOA refers to more usage of SBSs. SBSs operate in Internet-based dynamic environment and its smooth operations depend on many factors including quality design and good QoS. These made the detection of SOA antipatterns challenging but still an open problem. There are a number of contributions in the literature for the detection and correction of OO antipatterns. However, there is no concrete methods and techniques for such detection in SBSs and to assess the QoS. Specification of SOA antipatterns is the primary step for this purpose, and another open problem.

**Motivation:** Solving the problem of detection of SOA antipatterns, and assessing design and QoS in SBSs, help to ease maintenance and evolution of SBSs. Software maintenance is one of the longest and important activity, it requires more expenses and resources with time [3]. In 1980, Lientz and Swanson showed that software maintenance requires 60% to 80% of total budget in its whole lifespan. The detection of the SOA antipatterns within SBSs is also a part of maintenance. Detection of such antipatterns may also help improving QoS. In summary, by proposing an approach for detecting SOA antipatterns, we can contribute to better maintenance and evolution of SBSs.

Therefore, with the above context, problem definition and motivation, my PhD thesis will mainly focus on proposing a novel and innovative approach, named as SODA (Service Oriented Detection for Antipatterns), for specifying and detecting SOA antipatterns. SODA is supported by a underlying framework, named as SOFA (Service Oriented Framework for Antipatterns). I also intend to provide a concrete empirical evidence supporting my research hypotheses.

**Research Challenges:** Key research challenges are:

(a) *Specifying SOA Antipatterns*: The current literature for SOA antipatterns is not matured enough and there are limited number of journals, proceedings and books on SOA antipatterns. Researchers and practitioners mostly depend on

online resources, i.e, open forums, blogs, shared resources. Therefore, available references for specifying SOA antipatterns are not adequate.

(b) *Repository of SBSs*: The research on the detection of SOA antipatterns still did not gain much attention despite of its importance. And, there is no repository of SBSs to perform experiments, as the community did not make the effort to provide it. We can find some Web Service search engines, i.e., *Seekda*[1], *Woogle*[2] but no complete systems. Researchers are still in the lack of freely available SBSs as testbed.

(c) *Handling Dynamic Environment of SOA*: SBSs are developed adopting SOA design principles, and the execution environment of SBSs is dynamic involving many operating factors, e.g., implementation technologies, inter-operability, QoS, critical design criteria. Thus, handling dynamic environment of SOA for analyzing SBSs is another major research challenge.

**Major Impacts:** The major impact of this research work is threefold: (a) Industrial practitioners will find a way to handle SOA antipatterns. Many applications are now Web-based and there exists no accepted approach within the industrial community. Successful familiarization of a novel approach will encourage them to develop advanced tools, for the detection of SOA antipatterns, (b) Academic researchers will commence new research directions and analysis paths towards the detection of SOA antipatterns, till now which is not considered with a greater importance, and finally, (c) A fruitful collaboration between industrial practitioners and academic fellows will ensure SBSs with good QoS, cheaper maintenance and easier evolution, with the increased adoption of SOA.

## 2 Related Work

Much analysis with service orientation and commercial aspect of SOA exist in the literature, but the architectural-design aspect is still not highlighted properly [9]. Design quality is essential for building well maintainable and evolvable SBSs. Design patterns and antipatterns are two major ways for expressing design issues and related solutions. Nonetheless, unlike OO antipatterns, there is no detection methods and techniques for SOA antipatterns in SBSs.

For SOA, a few number of books and papers dealt with antipatterns, and most of the references are Web sites, forums and blogs where practitioners share their experiences in SOA design and development. The book by Dudney *et al.* [1] is the first book to introduce a number of architectural, design and implementation antipatterns for the systems based on J2EE technologies. But, most of them cannot be detected automatically. Also, Král *et al.* [4] specified briefly seven SOA antipatterns, but did not discuss their detection. Several methods and tools exist for the detection [7] of antipatterns in OO systems. But, all these OO detection methods and tools are not applicable to SOA, because of its granularity difference in their building blocks. Moreover, the more dynamic nature of SOA-based SBSs

[1] `http://webservices.seekda.com/`
[2] `http://db.cs.washington.edu/woogle.html`

brings in several issues to confront, that do not exist in OO systems. There are a couple of works that deal with the detection of antipatterns: Wong *et al.* [10] used a genetic algorithm for detecting software faults and anomalous behavior; and, Parsons *et al.* [8] performed the detection of performance antipatterns in component-based enterprise systems, using a rule-based approach with static and dynamic analysis. Though there is no significant contributions in the literature to support the detection of SOA antipatterns, all the above OO methods and techniques can form a baseline to formulate a generic approach for detecting SOA antipatterns.

## 3 Proposed Methodology

**The Approach: SODA -** Figure 1 shows the proposed approach, SODA, for detecting SOA antipatterns in SBSs. The steps include:

*Step 1: Specification* - Includes performing a domain analysis by studying definitions and specifications from the literature to identify relevant static and dynamic properties (a.k.a. *metrics*), to specify SOA antipatterns. Then, using these properties as the basis for the vocabulary to define domain specific language (DSL), and finally formalizing the rule cards. A rule card is the representation of a SOA antipattern at a high-level of abstraction. Figure 2 shows an example of rule cards for *Multi Service* and *Tiny Service*.

*Step 2: Generation* - With the rule cards of SOA antipatterns, I generate detection algorithms automatically using a simple *template*-based technique.

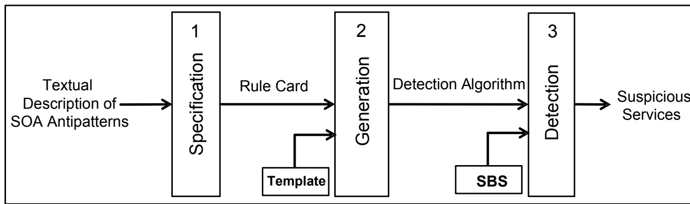


**Fig. 1.** Proposed SODA Approach

```
1 RULE_CARD: MultiService {
2  RULE: MultiService {INTER MultiMethod
3  HighResponse LowAvailability LowCohesion};
4  RULE: MultiMethod {NMD VERY_HIGH};
5  RULE: HighResponse {RT VERY_HIGH};
6  RULE: LowAvailability {A LOW};
7  RULE: LowCohesion {COH LOW};
8 };
```

(a) Multi Service

```
1 RULE_CARD: TinyService {
2  RULE: TinyService {INTER FewMethod
3                          HighCoupling};
4  RULE: FewMethod {NMD VERY_LOW};
5  RULE: HighCoupling {CPL HIGH};
6 };
```

(b) Tiny Service

**Fig. 2.** Example Rule Cards for Multi Service and Tiny Service

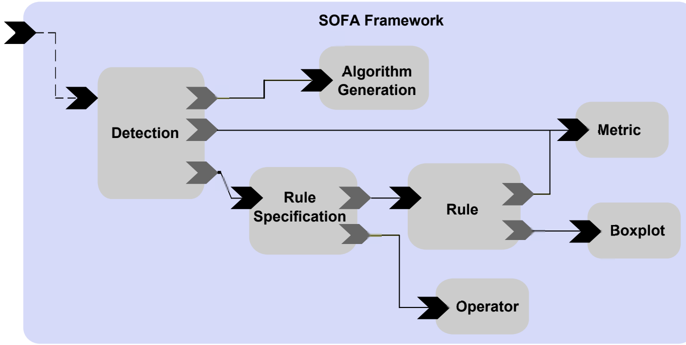


**Fig. 3.** SOFA: Underlying Framework for the SODA Approach (black arrows represent service provided by the component, grey arrows dependency on referenced component)

*Step 3: Detection* - For the detection of SOA antipatterns, an underlying framework, SOFA is introduced. All the computations of static and dynamic metrics, i.e., identified relevant properties, and related analysis are performed in SOFA.

**The Framework: SOFA -** Figure 3 shows the underlying framework, SOFA, that supports the specification and detection of SOA antipatterns in SBSs. SOFA has seven modules, programmatically each of which represents a component providing a stand-alone service. The components include: (1) *Detection Component*, representing the main detection engine that initiates and controls the overall detection process, (2) *Metric Component* that provides the computation of all metrics from the metric suite, both static and dynamic ones, (3) *Rule Specification Component* is responsible for specifying rule cards using the *Rule Component* and *Operator Component*, (4) *Algorithm Generation Component* generates the detection algorithms automatically from the specified rules, (5) *Rule Component* represents all the singleton rules that are composed of metrics, and depends on *Metric Component* to get required metrics, (6) *Operator Component* provides all the boolean and comparison operators, and finally, (7) *Boxplot Component* provides the means for computing boundary values and setting threshold values. The SOFA itself is a *service*-based framework.

# 4 Preliminary Experiments and Results

## 4.1 Key Research Hypotheses

Here, I define key research assumptions for the experimental purpose. The initial set of assumptions includes:

**Table 1.** Primary Results for the Detection of the Four SOA Antipatterns in the Original and Evolved Version of the *Home-Automation* System *(S: Static, D: Dynamic)*

| Antipattern Name | Suspicious Services | Version | Analysis | Metrics | Detection Time |
|---|---|---|---|---|---|
| Tiny Service | [MediatorDelegate] | evolved | *S* | NOR: 4<br>CPL: 0.440<br>NMD: 1 | 0.194s |
| Multi Service | [IMediator] | original | *S, D* | COH: 0.027<br>NMD: 13<br>RT: 132ms | 0.462s |
| Duplicated Service | [Communication-Service]<br>[IMediator] | original | *S* | ANIM: 25% | 0.215s |
| Bottleneck Service | [IMediator]<br>[PatientDAO] | original | *S, D* | NIR: 7<br>NOR: 7<br>CPL: 1.0<br>RT: 40ms | 0.246s |

*A1. Generality: The DSL allows the specification of many different SOA antipatterns, from simple to more complex ones.*

*A2. Accuracy: The generated detection algorithms have a recall of 100%, i.e., all existing antipatterns are detected, and a precision greater than 75%, i.e., more than three-quarters of detected antipatterns are true positive.*

*A3. Extensibility: The DSL and the proposed framework, SOFA is extensible for adding new SOA metrics and SOA antipatterns to detect.*

*A4. Performance: The computation time required for the detection of antipatterns using the generated algorithms is reasonably very low, i.e., in the order of few seconds.*

*A5. Technology: The proposed approach, SODA supports any technologies, i.e., capable of handling any SBSs implemented with diverse technologies, i.e., SOAP-RPC, SCA, REST, Web services, WCF etc.*

**Preliminary Results:** With the aim of supporting previous hypotheses, I conducted some preliminary experiments, showing the results in Table 4.1. Column 1 shows the SOA antipatterns to detect, Column 2 shows the identified suspicious service(s), Column 3 and 4 show the version of the system and analyses methods I applied. Column 5 shows all the metric values for the service(s), and finally, Column 6 shows the required detection times in Table 4.1. The detection were performed on an SBS, *Home-Automation*. Here, I briefly discuss the detection result for *Multi Service*. In Table 4.1, `IMediator` is detected as *Multi Service* because of its very high number of methods (*i.e.,* NMD equal 13) and its low cohesion (*i.e.,* COH equal 0.027). These values are evaluated by the *Box-Plot* service as high and low in comparison with the values of other services of *Home-Automation*.

More details on the detection results are available in our ICSOC '12 paper [6]. Textual descriptions of these four antipatterns with their corresponding rule cards, an elaborative presentation on *Home-Automation*, and more materials are available on http://sofa.uqam.ca.

## 5 Conclusion and Future Work

Due to the importance of the detection of SOA antipatterns in SBSs, I proposed a novel and innovative approach, SODA, supported by an underlying framework, SOFA. The framework supports the specification and detection of SOA antipatterns using a DSL and generated detection algorithms. In a preliminary experiment, I detected four common SOA antipatterns in *Home-Automation* and I reported suspicious service(s). As future work, I intend to support all the research hypotheses, stated in Section 4.1. In particular, I intend to perform thorough experiments with larger and more complex SBSs using different experimental and control groups. The rectification of the detected SOA antipatterns is also in the future plan.

## References

1. Dudney, B., Asbury, S., Krozak, J.K., Wittkopf, K.: J2EE AntiPatterns. John Wiley & Sons Inc. (2003)
2. Erl, T.: Service Oriented Architecture: Concepts, Technology and Design (2005)
3. Hanna, M.: Maintenance Burden Begging for a Remedy. Datamation, 53–63 (1993)
4. Král, J., Žemlička, M.: Crucial Service-Oriented Antipatterns, vol. 2, pp. 160–171. International Academy, Research and Industry Association, IARIA (2008)
5. Kral, J., Zemlicka, M.: Popular SOA Antipatterns. In: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Computation World, pp. 271–276 (2009)
6. Moha, N., Palma, F., Nayrolles, M., Conseil, B.J., Guéhéneuc, Y.-G., Baudry, B., Jézéquel, J.-M.: Specification and Detection of SOA Antipatterns. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC 2012. LNCS, vol. 7636, pp. 1–16. Springer, Heidelberg (2012)
7. Munro, M.J.: Product Metrics for Automatic Identification of "Bad Smell" Design Problems in Java Source-Code. In: Proceedings of the 11th International Software Metrics Symposium. IEEE Computer Society Press (September 2005)
8. Parsons, T., Murphy, J.: Detecting Performance Antipatterns in Component Based Enterprise Systems. Journal of Object Technology 7(3), 55–90 (2008)
9. Rotem-Gal-Oz, A., Bruno, E., Dahan, U.: SOA Patterns. Manning Publications Co. (2012) (to be published in Summer 2012)
10. Wong, S., Aaron, M., Segall, J., Lynch, K., Mancoridis, S.: Reverse Engineering Utility Functions Using Genetic Programming to Detect Anomalous Behavior in Software. In: Proceedings of the 2010 17th Working Conference on Reverse Engineering, pp. 41–149. IEEE Computer Society, Washington, DC (2010)