A Study on the Taxonomy of Service Antipatterns

Francis Palma*† and Naouel Moha†
*Ptidej Team, DGIGL, École Polytechnique de Montréal, Canada. Email: francis.palma@polymtl.ca
†Département d'informatique, Université du Québec à Montréal, Canada. Email: moha.naouel@uqam.ca

***Abstract*—Antipatterns in Service-based Systems (SBSs)—*service antipatterns*—represent "bad" solutions to recurring design problems. In opposition to *design patterns*, which are good solutions, antipatterns should be avoided by the engineers. Antipatterns may also be introduced due to diverse changes performed against new user requirements and execution contexts. Service antipatterns may degrade the quality of design and may hinder the future maintenance and evolution of SBSs. The detection of service antipatterns is important to improve the design quality of SBSs and to ease their maintenance. A better understanding of service antipatterns is a must prerequisite to perform their detection. This paper presents a taxonomy of service antipatterns in Web services and SCA (Service Component Architecture), the two common SBSs implementation technologies. The presented taxonomy will facilitate engineers their understanding on service antipatterns. Other substantial benefits of the presented taxonomy include: (1) assisting in the specification and detection of service antipatterns, (2) revealing the relationships among various groups of service antipatterns, (3) grouping together antipatterns that are fundamentally related, and (4) providing an overview of various system-level design problems ensemble.**

Keywords*—*Antipatterns, Taxonomy, Service-based systems

I. Introduction

Service Oriented Architecture (SOA), as an architectural style, is now broadly adopted in the industry because it allows the development of low-cost, flexible, and scalable distributed systems by composing autonomous, reusable, and platform-independent software units—*services*—which can be consumed over the Internet [1]. In practice, SOA can be realised using various technologies and architectural styles including SCA (Service Component Architecture) [2] and Web services. Google Maps, Amazon, eBay, PayPal, and FedEx are well-known examples of service-based systems (SBSs).

However, the emergence of SBSs raises several software engineering challenges. Indeed, like any other complex software systems, SBSs must evolve to fit new user requirements and to cope with new execution contexts, such as changes in technologies or protocols. All these changes, over the time, may degrade the design and quality of service (QoS) of SBSs and may often result in the appearance of common poor solutions to recurring problems—*antipatterns*—by opposition to *design patterns*, *i.e.*, good solutions to recurring design problems.

Multi Service [3] and *Tiny Service* [3] are two common antipatterns in SBSs. *Multi Service* is an antipattern that represents a service implementing a multitude of operations related to different business and technical abstractions. Such a service is not easily reusable because of the low cohesion among its operations and is often unavailable to end-users because it is overloaded [3]. Conversely, *Tiny Service* is a service with just a few operations, which only implements part of an abstraction. Such service often requires several coupled services to be used together, leading to higher development complexity and reduced flexibility [3]. Therefore, the detection of such antipatterns is an important activity to assess the design and QoS of SBSs and to ease their maintenance and evolution.

To perform the detection of service antipatterns, their in-depth understanding and relationships among them is the first and crucial step. To ease their understanding, this paper as its main contribution proposes the first classification of service antipatterns. In the following sections, we define what we mean by service antipattern and propose a taxonomy of service antipatterns. The presented taxonomy, we believe, will facilitate engineers their understanding of service antipatterns. The taxonomy will also (1) assist in the specification and detection of service antipatterns, (2) reveal the relationships among various groups of service antipatterns, (3) group together antipatterns that are fundamentally related, and (4) provide an overview of various system-level design problems.

In the remainder of this paper, Section II highlights existing contributions in the literature. Section III presents the catalogue and taxonomy of service antipatterns with their detailed classifications. Finally, Section IV concludes the paper.

II. Related Work

In object-oriented (OO) systems, a code (or design) smell is a symptom in the source code (or program) indicating auxiliary problems. As Fowler mentioned: "*a code smell is a surface indication that usually corresponds to a deeper problem in the system*" [4]. A number of contributions exist in the literature that classified object-oriented (OO) code- and design-level smells, including [5]–[7].

Mäntylä *et al.* [6] performed an empirical study on OO code smells and presented a subjective taxonomy that categorises similar smells and showed that the taxonomy for the smells facilitates explaining the identified correlations between the subjective evaluations of the smells. Moha *et al.* [5] presented the classification of ten OO smells and proposed an approach for the specification and automated detection of OO smells in the source code. Such classification was carried out after a thorough domain analysis of OO design and code smells. However, the authors' main interest in [5] was the automatic detection of OO smells. Ganesh *et al.* [7] proposed an approach to classify and catalog 31 new OO design smells based on how they break the key OO design principles [8]. The significant contribution of this work was the empirical validation of design smells with real system architects. However, all these works addressed OO code and–or design smells.

In the service domain, service smell holds the same notion, instead, at the service-level, which is coarser-grained and at

higher-level of abstraction than OO classes. However, to the best of our knowledge, no such classifications were performed in the service domain. In this paper, we proposed a preliminary taxonomy of service smells and service antipatterns in Web service and SCA (Service Component Architecture) technology [2]. Web service and SCA are two well-known SBSs implementation technologies relying on services and components as their building blocks, respectively. In the next section, we introduce the concepts of service smell and service antipattern and provide their classifications.

III. TERMINOLOGY

A clear understanding of the different types of service antipatterns and a classification of those antipatterns is necessary before considering their detection. This section clarifies what we mean by service antipattern and proposes a first classification of various categories of antipatterns. Andrew Koenig coined the term *antipattern* as "*a common solution to a recurring problem that is usually ineffective and highly counterproductive*" [9]. A service antipattern carries highly similar notion, but applicable to service-based systems (SBSs). A service antipattern is not an erroneous design or implementation, rather it hinders the maintenance and evolution, or even degrades the design and quality of service of SBSs.

A. Catalogue

In this section, we provide a catalogue of different service antipatterns from the literature that are commonly found in Web services and SCA (Service Component Architecture). Figures 1 and 2 list 20 common service antipatterns in Web services and SCA. In the following section, we provide the classification of those service antipatterns.

► **Ambiguous Name** is an antipattern where the developers use the names of interface elements (*e.g.*, port-types, operations, and messages) that are very short or long, include too general terms, or even show the improper use of verbs, etc., which are not semantically and syntactically sound [10].
► **Chatty Web Service** is an antipattern where a high number of operations are required to complete one abstraction where the operations are typically attribute-level setters or getters. Due to many interactions required, the overall performance suffers with a high response time [3].
► **CRUDy Interface** is an antipattern where the design encourages services the RPC-like behavior by creating CRUD-type operations, *e.g.*, create_X(), read_Y(), etc., which again becomes chatty [11].
► **Data Web Service** typically contains accessor operations, *i.e.*, *getters/setters*, which only perform simple information retrieval or data access operations. This service usually deals with very small messages of primitive types and has high data cohesion.
► **Duplicated Web Service** corresponds to a set of highly similar Web services where identical operations with the same names and–or message parameters might exist.
► **Fine Grained Web Service** is a small Web service with few operations implementing only a part of an abstraction and very often requires several coupled Web services to complete an abstraction, resulting in higher development complexity [3].
► **God Object Web Service** corresponds to a Web service that contains a large number of very low cohesive operations related to different business abstractions. This service may reflect low availability and result in high response time [3].
► **Low Cohesive Operations** is an antipattern where developers place very low cohesive operations in a single port-type, which may not be semantically related [10].
► **Maybe It's Not RPC** is an antipattern where the Web service mainly provides CRUD operations with a large number of parameters causing poor system performance since the clients wait for the synchronous responses [3].
► **Redundant PortTypes** is an antipattern where multiple port-types are duplicated with the similar set of operations dealing with the same messages [12].

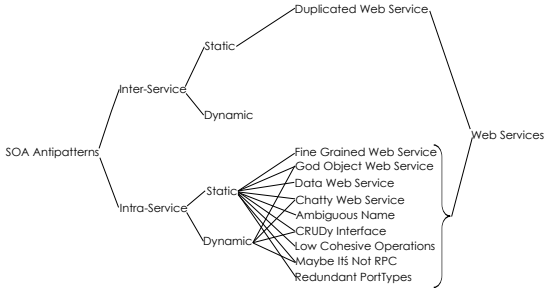
Figure 1: Ten antipatterns commonly found in Web services.

B. Service Antipatterns Classifications

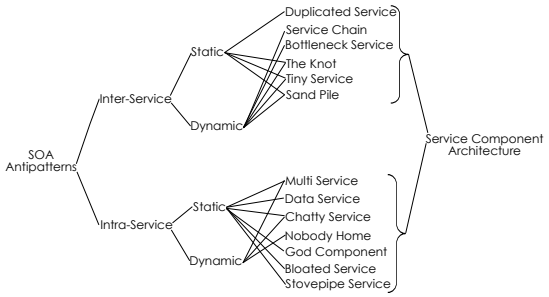
It is important to have a consistent classification to avoid redundancies in definitions and to ease the comprehension of service antipatterns. Figure 3 shows the classifications of service antipatterns in Web services (see Figure 3a) and SCA (see Figure 3b). We classify service antipatterns based on the two following categories:

► **Multi Service** implements a multitude of operations related to different business abstractions by aggregating too many low cohesive operations into one service, which may often unavailable to end-users and induces a high response time [3].
► **Tiny Service** is a small service with a very few operations, which often requires several coupled services to be used together to fulfill an abstraction, and causes higher development complexity, reduced usability [3].
► **Sand Pile** appears when a service is composed by multiple smaller services sharing common data, thus has high data cohesion [13].
► **Chatty Service** corresponds to a set of services that exchange a lot of small data of primitive types, resulting a high number of operation invocations [3].
► **Knot** is a set of very low cohesive but tightly coupled services. The availability of these services may be low while their response times become high [14].
► **Bottleneck Service** is highly used by the clients. Being overloaded, its response time can be high. Moreover, its availability may be low due to the traffic.
► **Data Service** corresponds to a service that, with high data cohesion, contains mainly accessor methods, *i.e.*, getters/setters with small and primitive parameters.
► **God Component** encapsulates a multitude of services and represents high responsibility enclosed by many operations with many different types of parameters to exchange. It may have a high coupling with the communicating services [3].
► **Bloated Service** has one large interface and lots of parameters and performs heterogeneous operations with low cohesion among them [15].
► **Stovepipe Service** has a large number of private or protected methods that primarily focus on utility functions, *i.e.*, logging or notifications rather than focusing on main operational goals [3].

Figure 2: Ten antipatterns commonly found in SCA systems.



(a) Classifications of service antipatterns in Web service



(b) Classifications of service antipatterns in SCA

Figure 3: Classifications of service antipatterns.

1) Existential: In this category, service antipatterns are classified based on the types of their distribution within the SBS constituents, *e.g.*, intra-service or inter-service.

- *Intra-service* category gathers antipatterns that are found only within a service (or component in the SCA) and independent of other services (or components) in the system, *e.g.*, Multi Service [3], Chatty Service [3], and CRUDy Interface [11] (Figure 3).

- *Inter-service* category gathers antipatterns that are spread among several services (or components in the SCA), *e.g.*, Service Chain [16], Knot [14], and Sand Pile [13] (Figure 3).

2) Analytical: Based on their natures of analysis, service antipatterns can be classified into three groups. Antipatterns that require (i) only static analysis, (ii) only dynamic analysis, and (iii) both static and dynamic analyses.

- *Static service antipatterns* require only static analysis for their detection; and thus only their structural properties are enough to detect them within the services (or components in the SCA), *e.g.*, Duplicated Service [17], Low Cohesive Operation [10], and Stovepipe Service [3] require only static analysis.

- *Dynamic service antipatterns* require dynamic analysis for their detection; and thus only their structural properties cannot help engineers to detect them within the services (or components in the SCA), *i.e.*, services performances are also observed. For example, Service Chain [16], Bottleneck Service [16], and Nobody Home [18] require dynamic analysis to be detected within the systems.

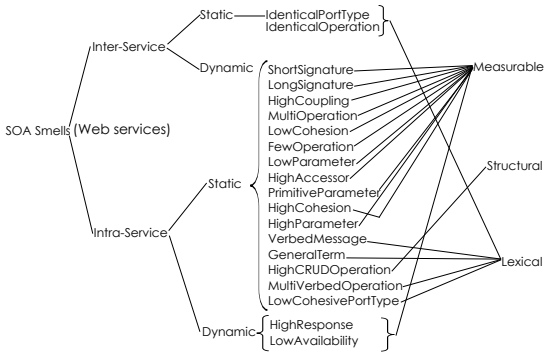
- *Compound service antipatterns* require both static and dynamic analyses to be detected, *e.g.*, Chatty Web Service [3], Maybe It's Not RPC [3], Knot [14], and Sand Pile [13].

Service smells are indicators of the presence of service antipatterns in SBSs and are lower-level design problems than antipatterns. We also classify service smells for Web services and SCA in Figures 4a, 4b. We identify two categories, namely inter-service and intra-service, and sub-categorise them as static and dynamic. Moreover, service smells can be classified as the *measurable*, *structural*, or *lexical* depending on the types of properties required to be analysed. Such classification is also performed in the object-oriented (OO) literature [19]. Measurable smells are mainly expressed with the measure of internal attributes of service interfaces. Lexical smells are related to the vocabulary or lexicons used to name service interface elements, *i.e.*, signatures of messages, operations, and so on. Structural smells deal with measuring the relationships (1) among services, *i.e.*, service containment or composition, or (2) among interfaces within a service.

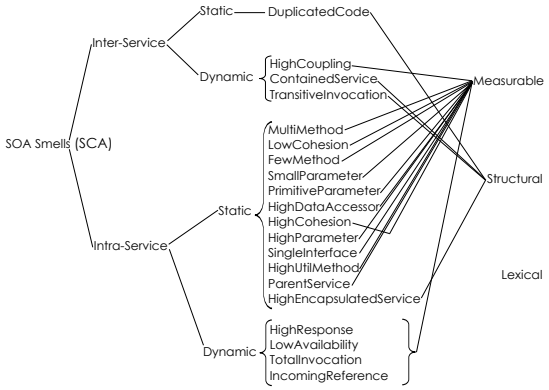
C. Taxonomy of Service Antipatterns

We use the vocabulary of service domain to manually organise service smells with respect to one another and build a taxonomy that puts all service smells on a single map and highlights their relationships. The map organises and combines service smells and service antipatterns, and their related measurable or observable properties using set operators, *e.g.*, intersection (`INTER`) or union (`UNION`).

Figure 5 summarises the classification as a taxonomy in the form of a map. It is similar to Pattern Map by Gamma *et al.* [8] for OO patterns. Due to space limit, we only show seven service antipatterns and their relations in the SCA domain. This taxonomy describes the structural relationships between service smells and service antipatterns and their measurable, structural, and lexical properties (ovals in white). It also describes the structural relationships (edges) between service antipatterns (dark grey hexagons) and service smells (grey ovals). It gives an overview of all key concepts that characterise a service antipattern. It also makes explicit the relationships



(a) Classifications of service smells in Web service.



(b) Classifications of service smells in SCA.

Figure 4: Classifications of service smells.

between service smells and service antipatterns. The map in Figure 5 is useful to prevent misinterpretation by clarifying and classifying smells and antipatterns based on their key concepts.

To illustrate the Figure 5 with an example, the *Multi Service* antipattern combines four service smells via intersection (`INTER`) operator, namely *LowAvailability*, *HighResponse*, *LowCohesion*, and *MultiMethod*. Moreover, the *Bottleneck Service* antipattern combines two service smells, namely the *LowPerformance* and *HighCoupling* where the *LowPerformance* service smell is the combination of *LowAvailability* and *HighResponse* service smells. Therefore, *Multi Service* and *Bottleneck Service* share similar behavioral properties. Similarly, *Bottleneck Service* and *Tiny Service* share similar structural properties, *i.e.*, *HighCoupling*.

The relationships reported above play an important role in the specification and detection of service antipatterns in SBSs. In the next section, we briefly highlight the specification approach for service antipatterns, which is facilitated by the taxonomy presented above.

D. Specification of Service Antipatterns

To specify service antipatterns, we performed a thorough domain analysis of antipatterns for Web service and SCA. We investigated their definitions and descriptions in the literature [3], [10], [12], [13], [15]. We identified a set of key properties related to each antipattern, *i.e.*, static and dynamic properties related to service design. Static properties are properties that apply to the static descriptions of SBSs, such as WSDL for

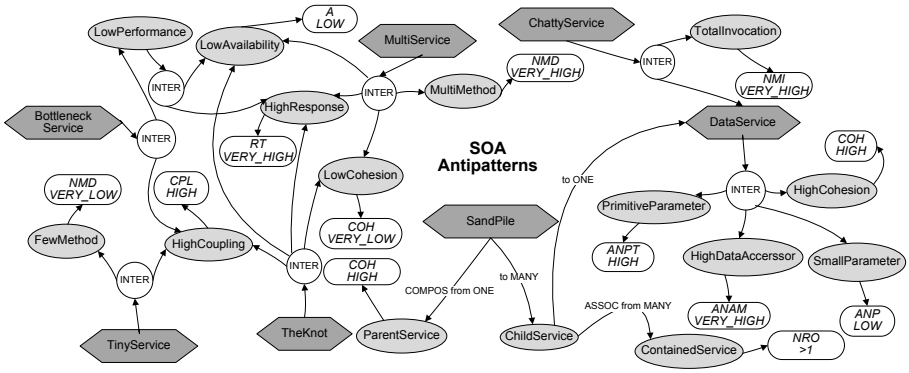


Figure 5: The taxonomy of service antipatterns in SCA technology (ovals in white are measurable properties, dark grey hexagons are service antipatterns, grey ovals are service smells, and directed edges are relationships).

Web Services and SCDL for SCA, whereas dynamic properties are related to the dynamic behavior of SBSs as observed during their execution. We used these relevant key properties as *metrics* to specify the rule cards relying on rules. Table I provides a global classification of our defined SOA metrics, which we can use in the taxonomy (Figure 5).

Table I: The list of 27 service metrics to specify the service antipatterns ('WS' - Web service, 'S' - *Static*, 'D' - *Dynamic*).

Metrics	Full Names	S|D	Structural|Behavioral	WS|SCA|Both
A	Availability of a Service	D	Behavioral	Both
ALS	Average Length of Signatures	S	Structural	WS
ANP	Average Number of Parameters in Operations	S	Structural	Both
ANPT	Average Number of Primitive Type Parameters	S	Structural	Both
ANIO	Average Number of Identical Operations	S	Structural	Both
ANAO	Average Number of Accessor Operations	S	Structural	Both
ARIP	Average Ratio of Identical Port-Types	S	Structural	WS
ARIO	Average Ratio of Identical Operations	S	Structural	WS
ARIM	Average Ratio of Identical Messages	S	Structural	WS
COH	Service Cohesion	S	Structural	Both
CPL	Service Coupling	S	Structural	Both
NCO	Number of CRUD Operations	S	Structural	WS
NOD	Number of Operations Declared	S	Structural	Both
NOPT	Number of Operations in Port-Types	S	Structural	WS
NI	Number of Interfaces	S	Structural	SCA
NIR	Number of Incoming References	S	Structural	SCA
NMI	Number of Method Invocations	D	Behavioral	SCA
NOR	Number of Outgoing References	S	Structural	SCA
NPT	Number of Port-Types	S	Structural	WS
NTMI	Number of Transitive Methods Invoked	D	Behavioral	SCA
NSE	Number of Services Encapsulated	S	Structural	SCA
NUM	Number of Utility Methods	S	Structural	SCA
NVMS	Number of Verbs in Message Signatures	S	Structural	WS
NVOS	Number of Verbs in Operation Signatures	S	Structural	WS
RGTS	Ratio of General Terms in Signatures	S	Structural	WS
RT	Response Time of a Service	D	Behavioral	Both
TNP	Total Number of Parameters	S	Structural	SCA

IV. Conclusion and Future Work

The detection of service antipatterns is important to assess the design and QoS of service-based systems (SBSs) and thus ease their maintenance and evolution. The absence of taxonomy specific to service antipatterns in the literature inspired us to contribute with this work. In this paper, we presented the detailed classifications of service antipatterns grounded in the service domain, which may deepen the understanding of relationships among service smells and service antipatterns. As future work, we plan to analyse the taxonomy of other SBSs technologies and architectural styles like RESTful services.

References

[1] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, August 2005.

[2] M. Edwards, *Service Component Architecture (SCA)*. USA: OASIS, April 2011. [Online]. Available: http://oasis-opencsa.org/sca

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

[4] M. J. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[5] N. Moha, Y.-G. Guéhéneuc, A.-F. L. Meur, L. Duchien, and A. Tiberghien, "From a Domain Analysis to the Specification and Detection of Code and Design Smells," *Formal Aspect of Computing*, vol. 22, no. 3-4, pp. 345–361, May 2010.

[6] M. Mäntylä, J. Vanhanen, and C. Lassenius, "A Taxonomy and an Initial Empirical Study of Bad Smells in Code," in *Proceedings of the International Conference on Software Maintenance*, ser. ICSM '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 381–384.

[7] S. G. Ganesh, T. Sharma, and G. Suryanarayana, "Towards a Principle-based Classification of Structural Design Smells," *Journal of Object Technology*, pp. 1–29, 2013.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[9] A. Koenig, "Patterns and antipatterns," *The patterns handbook: techniques, strategies, and applications*, pp. 383–390, 1998.

[10] J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo, "Automatically Detecting Opportunities for Web Service Descriptions Improvement," W. Cellary and E. Estevez, Eds. Springer Berlin Heidelberg, 2010, vol. 341, pp. 139–150.

[11] J. Evdemon, "Principles of Service Design: Service Patterns and Anti-Patterns," August 2005.

[12] A. Heß, E. Johnston, and N. Kushmerick, "ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services," in *In Proceedings of International Semantic Web Conference*, 2004.

[13] J. Král and M. Žemlička, "Crucial Service-Oriented Antipatterns," in *In Proceedings of the International Conference on Software Engineering Advances*, vol. 2, no. 1. International Academy, Research and Industry Association (IARIA), 2008, pp. 160–171.

[14] A. Rotem-Gal-Oz, E. Bruno, and U. Dahan, *SOA Patterns*. Manning Publications Co., 2012.

[15] T. Modi, "SOA Management: SOA Antipatterns," August 2006.

[16] F. Palma, M. Nayrolles, N. Moha, Y.-G. Guéhéneuc, B. Baudry, and J.-M. Jézéquel, "SOA Antipatterns: An Approach for their Specification and Detection," *International Journal of Cooperative Information Systems*, vol. 22, no. 04, 2013.

[17] L. Cherbakov, M. Ibrahim, and J. Ang, "SOA Antipatterns: The Obstacles to the Adoption and Successful Realization of Service-Oriented Architecture," January 2006.

[18] S. Jones, "SOA Anti-patterns," www.infoq.com/articles/SOA-anti-patterns, June 2006.

[19] 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 Transaction on Software Engineering*, vol. 36, no. 1, pp. 20–36, January 2010.