Detection of Web Service Refactoring Opportunities

by

Taghreed Hassouna

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer and Information Science)
in The University of Michigan-Dearborn
2017

Master's Thesis Committee:

Assistant Professor Marouane Kessentini, Chair
Associate Professor Bruce Maxim
Professor William I. Grosky

DEDICATION

To My Family.

ACKNOWLEDGEMENTS

It is with a great joy that I reserve these few lines of gratitude and deep appreciation to all those who directly or indirectly contributed to the completion of this work:

First and foremost I offer my sincerest gratitude to Dr. Marouane Kessentini, who dedicated all his wonderful time to collaborate, support and lead me to the end of this piece of work. His advices, dedication, availability, relevant comments, corrections and committeemen led to the success of this work.

I also express my greatest thanks to SBSE members who supported me with valuable feedback and always kindly encouraged me to succeed this project.

I thank all the lecturers of the CIS master degree who have used their valuable time to transmit the knowledge that help in putting this work together me.

Finally, I wish to express my deep gratitude and thank my family who has consistently expressed its unconditional support and encouragement.

All those who contributed in one way or another, to make this work, can be found here, the crowning of their efforts.

Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

We propose, in this thesis, to consider the problem of Web service antipatterns detection as a multi-objective problem where examples of Web service antipatterns and well-designed code are used to generate detection rules. To this end, we use multi-objective genetic programming (MOGP) to find the best combination of metrics that maximizes the detection of Web service antipattern examples and minimizes the detection of well-designed Web service design examples.

We report the results of an empirical study using 8 different types of common Web service antipatterns. We compared our multi-objective formulation with random search, one existing mono-objective approach, and one state-of-the-art detection technique not based on heuristic search. Statistical analysis of the obtained results demonstrates that our approach is efficient in antipattern detection, on average, with a precision score of 94% and a recall score of 92%.

Keywords-Web service antipatterns, interface design, multi-objective optimization.

CHAPTER 1: INTRODUCTION

Web services are becoming the leading Service Oriented Architecture (SOA) technology used to build Service-based systems (SBSs) [1]. YouTube, Fedex, eBay, Google, FedEx, PayPal, and many other companies are leveraging these Web services in a reusable, distributed and portable fashion that can be invoked by the users [2]. SBSs evolve over time to meet new requirements or to fix bugs. Such continuous changes may have a negative impact on the quality of the services. Web services must be carefully designed and implemented to adequately fit in the required system's design whilst achieving good quality of services [3]. Indeed, there is no exact recipe to follow for proper service design. A set of guiding quality principles for service-oriented design exists, including such principles as service flexibility, operability, composability, and loose coupling. However, the design of services is strongly influenced by the context, environment and other decisions the service designers take, and such factors may lead to violations of quality principles. The presence of programming patterns associated with bad design and programming practices, known as antipatterns, are indications of such violations [2]. Furthermore, it is widely believed that such antipatterns lead to various maintenance and evolution problems including an increased bug rate, fragile design and inflexible code. Despite the extensive adoption of Web service technologies, very few studies has been proposed for the first step of the refactoring process which is the detection of antipatterns [4]. Indeed, the vast majority of existing work in Web services antipattern detection merely attempts to provide definitions and/or the key symptoms that characterize common antipatterns. Recent works [5] [6] rely on a declarative rule-based language to specify antipattern symptoms at a higher-level of abstraction using

combinations of quantitative (metrics), structural, and/or lexical information. However, in an exhaustive scenario, the number of possible antipatterns to be characterized manually and formulated with rules can be large. To make the situation worse, it is difficult to find a consensus to characterize and formulate such symptoms. For these reasons, the detection task is still mainly a manual, time-consuming and subjective process. To address the above-mentioned limitations, we propose in this thesis a multi-objective search-based approach for the generation of antipatterns detection rules from both bad and well-designed service examples. The process aims at finding the optimal combination of quality metrics, from an exhaustive list of possible metric combinations, that:

1. Maximizes the coverage of a set of antipattern examples collected from different systems.
2. Minimizes the detection of examples of good-design practices.

In fact, it is difficult to ensure that the used design defect examples cover all possible bad-design practices. Thus, we used good design practices as another objective to detect antipatterns that are not similar to the well-designed service examples and design defect examples. To this end, a multi-objective genetic programming (MOGP) [7] is used to generate the antipatterns detection rules that find trade-offs between the two above-mentioned objectives. MOGP is a powerful evolutionary metaheuristic which extends the generic model of learning to the space of programs [7].

To validate our proposal, we present an empirical evaluation of our approach on a benchmark of 415 Web services from ten different application domains and we considered 8 common Web service antipattern types. We compared multi-objective approach with random search, an existing mono-objective technique [6] [8] and a rule-based approach [5] not based on heuristic

search techniques. Statistical analysis demonstrates the efficiency of our approach in detecting Web service antipatterns, on average, with a precision score of 94% and a recall score of 92%. To the best of our knowledge, this is the first work to use multi-objective evolutionary algorithms for the detection of Web service antipatterns.

The remainder of this thesis is organized as follows. ***Chapter 2*** describes the necessary background details and presents the different challenges related to the detection of Web service antipattern; ***Chapter 3*** explains our multi-objective approach to address this problem. ***Chapter 4*** discusses the obtained experimental results. ***Chapter 5*** is dedicated to view surveys related work and finally we conclude and outline our future research directions in ***Chapter 6***.

CHAPTER 2: BACKGROUND AND PROBLEM STATEMENT

We first detail some required background information to understand the problem addressed in this work, then we present a motivating example to illustrate the limitations of existing studies. Finally, we present an overview of existing work.

2.1 BACKGROUND: WEB SERVICE ANTIPATTERNS

Antipatterns are symptoms of poor design and implementation practices that describe bad solutions to recurring design problems. They often lead to software which is hard to maintain and evolve [3], and may be introduced unintentionally during initial design or during software development due to bad design choices, poorly planned changes or time pressure. Different types of antipatterns presenting a variety of symptoms have been recently studied with the intent of improving their detection and suggesting improvements paths [1].

Common Web service antipatterns include:

- God object Web service (GOWS): implements a multitude of methods related to different business and technical abstractions in a single service.
- Fine grained Web service (FGWS): is a too fine-grained service whose overhead (communications, maintenance, and so on) outweighs its utility. This antipattern refers to a small Web service with few operations implementing only a part of an abstraction.

- Chatty Web service (CWS): represents an antipattern where a high number of operations, typically attribute-level setters or getters, are required to complete one abstraction.
- Data Web service (DWS): contains typically accessor operations, i.e., getters and setters. In a distributed environment, some Web services may only perform some simple information retrieval or data access operations.
- Ambiguous Web service (AWS): is an antipattern where developers use ambiguous or meaningless names for denoting the main elements of interface elements (e.g., port types, operations, messages).
- Redundant PortTypes (RPT): is an antipattern where multiple portTypes are duplicated with the similar set of operations
- CRUDy Interface (CI): is an antipattern where the design encourages services the RPC-like behavior by declaring create, read, update, and delete (CRUD) operations, e.g., createX(), readY(), etc. Interfaces designed in that way might be chatty because multiple operations need to be invoked to achieve one goal.
- Maybe It is Not RPC (MNR): is an antipattern where the Web service mainly provides CRUD-type operations for significant business entities. These operations will likely need to specify a significant number of parameters and/or complexity in those parameters

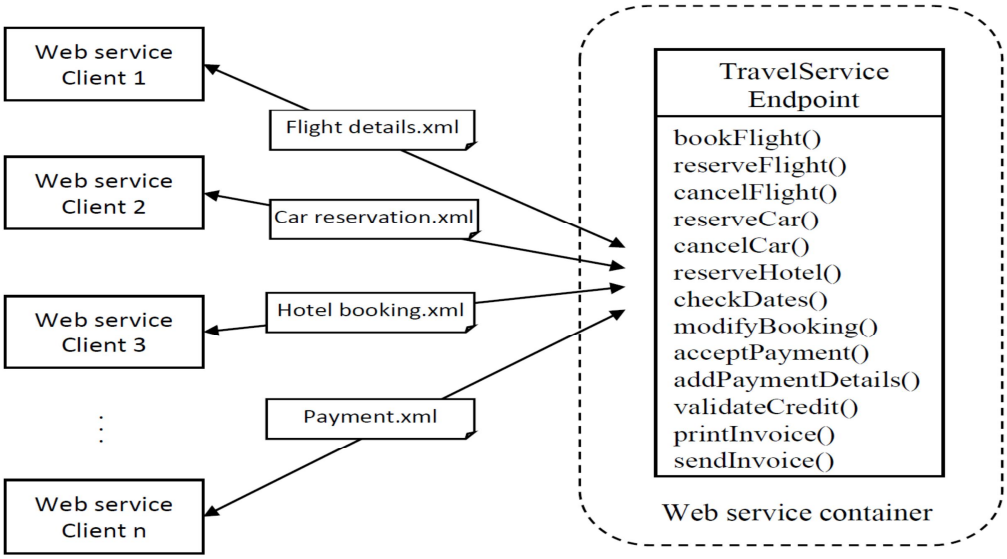

Figure 1: An example of god object Web service.

Figure 1 illustrates an example of a GOWS antipattern. One of the main symptoms of a GOWS is that it implements multiple core business and/or technical abstractions with un-cohesive operations. This is can detected at the service interface where public operations involve different entities or abstractions. In this example, it is clear that there are methods that operate on different core functionalities. For instance, the bookFlight() method is used to book a flight trip, while the reserveHotel() method attempts to reserve the specified hotel room. Overall, this GOWS supports the functionalities flight, car and hotel booking, payment, invoice services, and so on. Each of these is a significant core business abstraction, and typically will have many associated methods.

2.2 PROBLEM STATEMENT AND RELATED WORK

Several quality metrics can be used to capture the structural and semantic attributes of the Web services, and can be a reliable indicator of the quality of design [6]. These quality indicators can

then be used to quantitatively estimate and reflect the design signatures of Web Services architecture in terms of many metrics.

The antipatterns detection process usually involves finding the fragments of the design which violate these metrics. In this work, we used a set of static and dynamic Web service metrics as detailed in Table 1 and Table 2. Static metrics aim at measuring the structural properties of Web services in both the interface (WSDL) and code levels, whereas dynamic metrics aim at invoking the Web services and measuring different properties, e.g., response time.

Many metric combinations are possible, so the detection rules generation process is, by nature, a combinatorial optimization problem. The number of possible solutions quickly becomes huge as the number of metrics and possible threshold values increases. A deterministic search is not practical in such cases, and hence the use of heuristic search is warranted. The dimensions of the solution space are set by the metrics, their threshold values, and logical operations between them, e.g., union (metric1 OR metric2) and intersection (metric1 AND metric2). A solution is determined by assigning a threshold value to each metric.

The manual definition of rules to identify maybe difficult and can be time-consuming. The main issue with Web service antipattern detection is that there is no general consensus on how to decide if a particular design violates a quality heuristic. Indeed, there is a difference between detecting symptoms and asserting that the detected situation is an actual antipattern. Deciding which Web services are antipattern candidates heavily depends on the interpretation of each analyst. In some contexts, an apparent violation of a design principle may be consensually accepted as normal practice. For example, a translation Web service[1] may have in its interface

[1] http://www.webservicex.net/TranslateService.asmx?WSDL

only a single operation translate which is responsible for translating text from one language to another language. Although this service might be designed properly, from a strict antipattern definition, it may be considered as a fine-grained Web service.

Another inherent problem is related to the definition of threshold values when dealing with quantitative information. Indeed, there is no general agreement on extreme manifestations of Web service antipatterns [2]. That is, for each antipattern, rules that are expressed in terms of metrics need substantial calibration efforts to find the right threshold value for each metric, above which an antipattern is said to be detected.

To address or circumvent the above mentioned issues and challenges, we introduce a multi-objective heuristic-based approach to automatically detect Web service antipatterns as detailed in the next section.

Metric	Description	Metric level
NPT	Number of port types	Port type
NOD	Number of operations declared	Port type
NCO	Number of CRUD operations	Port type
NOPT	Average number of operations in port types	Port type
NPO	Average number of parameters in operations	Operation
NCT	Number of complex types	Type
NAOD	Number of accessor operations declared	Port type
NCTP	Number of complex type parameters	Type
COUP	Coupling	Port type
COH	Cohesion	Port type
NOM	Number of messages	Message
NST	Number of primitive types	Type
ALOS	Average length of operations signature	Operation
ALPS	Average length of port types signature	Port type
ALMS	Average length of message signature	Message
RPT	Ratio of primitive types over all defined types	Type
RAOD	Ratio of accessor operations declared	Port type
ANIPO	Average number of input parameters in operations	Operation
ANOPO	Average number of output parameters in operations	Operation
NPM	Average number of parts per message	Message
AMTO	Average number of meaningful terms in operation names	Operation
AMTM	Average number of meaningful terms in message names	Message
AMTP	Average number of meaningful terms in port type names	Type

Table 1: List of Web service interface metrics used.

Metric	Description	Metric level
WMC	Weighted methods per class	Class
DIT	Depth of Inheritance Tree	Class
NOC	Number of Children	Class
CBO	Coupling between object classes	Class
RFC	Response for a Class	Class
LCOM	Lack of cohesion in methods	Class
Ca	Afferent couplings	Class
Ce	Efferent couplings	Class
NPM	Number of Public Methods	Class
LCOM3	Lack of cohesion in methods	Class
LOC	Lines of Code	Class
DAM	Data Access Metric	Class
MOA	Measure of Aggregation	Class
MFA	Measure of Functional Abstraction	Class
CAM	Cohesion Among Methods of Class	Class
AMC	Average Method Complexity	Method
CC	The McCabe's cyclomatic complexity	Method

Table 2: List of Web service code metrics used.

CHAPTER 3: MULTI-OBJECTIVE WEB SERVICE ANTIPATTERNS DETECTION

We first present an overview of our multi-objective Genetic Programming approach. And then we provide the details of our problem formulation and the solution approach.

3.1 MULTI-OBJECTIVE GENETIC PROGRAMMING

Genetic Programming (GP) is a powerful evolutionary metaheuristic which extends the generic model of learning to the space of programs [7]. Differently to other evolutionary approaches, in GP, population individuals are themselves programs following a tree-like structure instead of fixed length linear string formed from a limited alphabet of symbols. GP can be seen as a process of program induction that allows automatically generating programs that solve a given task. Most exiting work on GP makes use of a single objective formulation of the optimization problem to solve using only one fitness function to evaluate the solution. Differently to single-objective optimization problems, the resolution of Multi-objective Optimization Problems (MOPs) yields a set of trade-off solutions called non-dominated solutions and their image in the objective space is called the Pareto front.

A high-level view of MOGP is depicted in ***Algorithm 1***. The algorithm starts by randomly creating an initial population P_0 of individuals encoded using a specific representation (line 1). Then, a child population Q_0 is generated from the population of parents P_0 (line 2) using genetic operators (crossover and mutation). Both populations are merged into an initial population R_0 of size N (line 5). Fast non-dominated-sort [7] is the technique used by MOGP to classify individual

solutions into different dominance levels (line 6). Indeed, the concept of non-dominance consists of comparing each solution x with every other solution in the solution x_1 is said to dominate another solution x_2, if x1 is no worse than x_2 in all objectives and x_1 is strictly better than x_2 in at least one objective". Formally, if we consider a set of objectives f_i , $i \in 1..n$, to maximize, a solution x_1 dominates x_2

$$if\ \forall i,\ f_i(x_2) \leqslant f_i(x_1) \text{ and } \exists j \mid f_j(x_2) < f_j(x_1)$$

Algorithm 1 High level pseudo code for MOGP

1. Create an initial population P_0
2. Generate an offspring population Q_0
3. t=0;
4. while stopping criteria not reached do
5. $R_t = P_t \cup Q_t$;
6. F = fast-non-dominated-sort (R_t);
7. $P_{t+1} = \emptyset$ and i=1;
8. while $| P_{t+1}| + |F_i| \leq N$ do
9. Apply crowding-distance-assignment(F_i);
10. $P_{t+1} = P_{t+1} \cup F_i$;
11. i = i+1;
12. end
13. Sort($F_i, \prec n$);
14. $P_{t+1} = P_{t+1} \cup F_i[1 : (N-| P_{t+1} |)]$;
15. Q_{t+1} = create-new-pop(P_{t+1});
16. t = t+1;
17. end

The whole population that contains N individuals (solutions) is sorted using the dominance principle into several fronts (line 6). Solutions on the first Pareto-front F0 get assigned dominance level of 0 Then, after taking these solutions out, fast-non-dominated-sort calculates the Paretofront F1 of the remaining population; solutions on this second front get assigned

dominance level of 1, and so on. The dominance level becomes the basis of selection of individual solutions for the next generation. Fronts are added successively until the parent population P_{t+1} is filled with N solutions (line 8). When NSGA-II has to cut off a front F_i and select a subset of individual solutions with the same dominance level, it relies on the crowding distance to make the selection (line 9). This parameter is used to promote diversity within the population. This front F_i to be split, is sorted in descending order (line 13), and the first (N-$|P_{t+1}|$) elements of Fi are chosen (line 14). Then a new population Q_{t+1} is created using selection, crossover and mutation (line 15). This process will be repeated until reaching the last iteration according to stop criteria (line 4).

3.2 MULTI-OBJECTIVE ALGORITHM ADAPTATION

3.2.1 PROBLEM FORMULATION

The Web service antipatterns detection problem involves searching for the best metric combinations among the set of candidate ones, which constitutes a huge search space. A solution of our antipatterns detection problem is a set of rules (metric combination with their thresholds values) where the goal of applying these rules is to detect design defects in a web service. We propose a multi-objective formulation of the Web service antipatterns rules generation problem. Consequently, we have two objective functions to be optimized: (1) maximizing the coverage of antipattern examples, and (2) minimizing the detection of good designpractice examples of Web services. The collected examples of well-designed Web services and antipatterns are taken as an input for our approach. Analytically speaking, the formulation of the multi-objective problem can be stated as follows:

$$\begin{cases} maxf_1(x) = \dfrac{\dfrac{|DCS(x)| \cap |ECS|}{|ECS|} + \dfrac{|DCS(x)| \cap |ECS|}{|DCS(x)|}}{2} \\ minf_2(x) = \dfrac{\dfrac{|DCS(x)| \cap |EGE|}{|EGE|} + \dfrac{|DCS(x)| \cap |EGE|}{|DCS(x)|}}{2} \end{cases}$$

where | *DCS(x)* | is the cardinality of the set of detected antipatterns by the metric combination *x,* | *ECS* | is the cardinality of the set of existing antipatterns, and | *EGE* | is the cardinality of the set of existing good examples. Once the bi-objective trade-off front is obtained, the developer can navigate through this front in order to select his/her preferred solution (metric combination).

The basic idea of the algorithm is to explore the search space by making a population of candidate solutions, also called individuals, and evolve this population towards an "optimal" solution for the detection of antipatterns. To evaluate the solutions, the fitness functions, as explained previously, are used. The best solutions (detection rules) will cover the maximum of anti-pattern examples and a minimum of good design examples of Web services.

In the initialization of the MOGP algorithm, our base of examples is split into ten subsets, each representing a different application domain, e.g., finance, travel, etc. One subset (WS) is the test dataset and the remaining subsets of good and bad design examples are the training datasets (the ground truth). Thus, MOGP is run to detect antipatterns in the selected subset (WS), which is not of course part of the training set.

The initial population for MOGP is a set of individuals (I) that stand for possible solutions representing detection rules (metrics combination). Then, the algorithm explores the search space and constructs new individuals by combining metrics to generate rules. In each iteration of the training process, antipatterns are iteratively evaluated using the generated rules. As described earlier, the process is driven by two fitness functions that calculates the quality of each candidate solution (detection rule) by comparing the list of detected antipatterns with the expected ones from the base of examples along with the percentage of covered well-designed examples. A new population of individuals is generated by iteratively selecting pairs of parent individuals from

population Pop and applying genetic operators to them (crossover and mutation). We include both the parent and child variants in the new population. We then apply the mutation operator, with a probability score, for both parent and child to ensure solution diversity; this produces the population for the next generation. Developers can use the best rules (solution) to detect potential antipatterns on any new Web service.

3.2.2 SOLUTION APPROACH

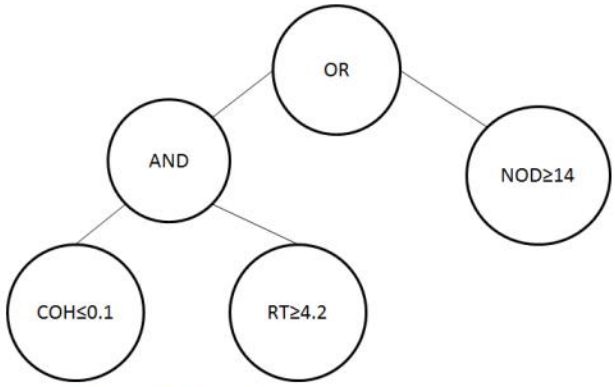

Figure 2: Solution representation example.

In the following, we describe the main steps of adaptation of the MOGP algorithm to our problem.

3.2.2.1 SOLUTION REPRESENTATION

Candidate solutions to the problem are antipattern detection rules. A solution is represented as a set of **IF−THEN** rules, each with the following structure: **IF** "Combination of metrics with their thresholds" **THEN** "antipattern type" The antecedent of the **IF** statement combines some metrics and their threshold values using logic operators (***AND, OR***). If these conditions are satisfied by a Web service, then it is determined to be of the antipattern type featuring in the **THEN** clause of

the rule. ***Figure 2*** provides an example. More formally, each candidate solution S is a sequence of detection rules where each rule is represented by a binary tree such that:

1) Each leaf node (terminal) L represents a metric (our metric suite described earlier) and its corresponding threshold, generated randomly.
2) Each internal node (function) N represents a logic operator, either *AND* or *OR*.

We will have as many rules as types of antipatterns to be detected. In this thesis, we focus on the detection of eight common types as defined in Section II-A.

3.2.2.2 EVALUATION FUNCTIONS

The solution is evaluated based on the two objective functions defined in the previous section. Since we are considering a bi-objective formulation, we use the concept of Pareto optimality to find a set of compromise (Pareto-optimal) solutions. The fitness of a particular solution in MOGP corresponds to a couple (Pareto Rank, Crowding distance). In fact, MOGP classifies the population individuals (of parents and children) into different layers, called non-dominated fronts. The output of MOGP is the last obtained parent population containing the best of the non-dominated solutions found. When plotted in the objective space, they form the Pareto-front from which the user will select his preferred antipatterns detection rules solution.

CHAPTER 4: EVALUATION

In order to evaluate our approach for Identification of web service refactoring opportunities as a multi-objective problem, we conducted a set of experiments based on real-world web services. The obtained results are subsequently statistically analyzed with the aim to compare our proposal with a variety of existing approaches. In this section, we present our research questions and then describe and discuss the obtained results.

4.1 RESEARCH QUESTIONS

We designed our experiments to answer the following research questions:

• **RQ1**: How does our multi-objective approach, MOGP, compare to random search and an existing mono-objective technique [8]?

• **RQ2**: To what extent can the proposed approach efficiently detect Web service antipatterns?

• **RQ3**: What types of Web service antipatterns does it detect correctly?

• **RQ4**: How does MOGP perform compared to existing Web service antipattern detection approach not based on heuristic search [5]?

4.2 EXPERIMENTAL DESIGN

To evaluate our approach, we collected a set of Web services using different Web service search engines including eil.cs.txstate.edu/ServiceXplorer, programmableweb.com, biocatalogue.org, webservices.seekda.com, taverna.org.uk and myexperiment.org. Table III

summarizes the collected services. Furthermore, our collected Web services are drawn from ten different application domains: financial, science, search, shipping, travel, weather, media, education, messaging and location. All services were manually inspected and validated to identify antipatterns based on guidelines from the literature [1] [2]. Furthermore, our dataset is available online [9] to encourage future research in the area of automated detection of Web service antipatterns.

Category	# services	# antipatterns	average NOD	average NOM	average NCT
Financial	94	67	29.52	57.31	19.01
Science	34	3	8.47	17.14	96.73
Search	37	13	8.35	18.94	26.13
Shipping	38	10	13.36	27.76	20.21
Travel	65	28	16.09	33.13	121.13
Weather	42	15	8.54	17.16	9.14
Media	19	14	10.9	16.4	28.6
Education	26	15	11.3	15.36	32.46
Messaging	29	20	7.6	11.21	18.25
Location	31	22	5.8	28.32	11.15
All	425	136	17.08	34.2	48.6

Table 3: Web services used in the empirical study.

We considered antipattern types range from eight common antipatterns, namely god object Web service (GOWS), fine- grained Web service (FGWS), chatty Web service (CWS), data Web service (DWS), ambiguous Web service (AWS), redundant port types (RPT), CRUDy interface (CI), and maybe it is not RPC (MNR) (cf. Section II-A). In our study, we employed a 10-fold cross validation procedure. We split our data into training data and evaluation data. For each fold, one category of services is evaluated by using the remaining nine categories as a base of examples (ground-truth). For instance, weather services are analyzed using antipattern instances from travel, shipping, search, science financial, media, education, messaging, and location

services. We use precision and recall [10] to evaluate the accuracy of our approach. Precision denotes the ratio of true antipatterns detected to the total number of detected antipatterns, while recall indicates the ratio of true antipatterns detected to the total number of existing antipatterns.

To answer **RQ1**, we investigate and report on the effectiveness of MOGP, since one of our primary novelties lies in the adoption of the multi-objective formulation. To this end, we implemented random search (RS) with the same fitness functions as MOGP. Indeed, it is important to compare our search technique to random search, since if an intelligent search method fails to outperform random search, then the proposed formulation is not adequate. In addition, we compared our multi-objective algorithm to an existing mono-objective approach where only examples of antipatterns were considered [8] without the use of positive examples of well-designed Web services.

To answer **RQ2**, we use both recall and precision to evaluate the efficiency of our approach in identifying antipatterns.

To answer **RQ3**, we investigated the antipattern types that were detected to find out whether there is a bias towards the detection of specific antipattern types.

To answer **RQ4**, we compared our approach with the SODA-W approach of Palma et al. [5]. SODA-W manually translates antipattern symptoms into detection rules and algorithms based on a literature review of Web service design. All three approaches are tested on the same benchmark described in Table 3.

4.3 RESULT

4.3.1 RESULTS FOR RQ1

The goal of RQ1 is to investigate how well MOGP performs against random search and an existing single-objective approach where only antipattern examples are used [8]. Table 4 and Figure 3 report the comparative results. Over 31 runs, RS did not perform well when compared to MOGP in terms of precision and recall achieving average values of only 29% and 31% respectively on the different Web services. The main reason could be related to the large search-space of possible combinations of metrics and threshold values to explore. The results achieved by MOGP are also better than the mono-objective roach in terms of precision and recall. In fact, the single objective GP technique has an average of 86% and 87% of precision and recall however MOGP has better scores with 94% of precision and 92% of recall on the different Web services. These results confirm that an intelligent search is required to explore the search space and that the use of well-designed Web service examples improved the obtained detection results.

Category	Precision (%)	Recall (%)
Financial	94	92
Science	96	98
Search	94	95
Shipping	96	90
Travel	94	96
Weather	91	96
Media	96	91
Education	97	93
Messaging	92	98
Location	94	91
Average	**89**	**93**

Table 4: MOGP results on the different Web services.

4.3.2 RESULTS FOR RQ2

The results for RQ2 are presented in Table 4. The obtained results show that we were able to detect most of the expected antipatterns in the different categories with a median precision higher than 94%. The higher precision value for travel and Education (97%) can be explained by the fact that these Web services are large than the others and contain a high number of operations and complex types that match the GOWS antipattern. For the Web service weather, the precision is the lowest one (91%), but is still a very acceptable score. This is due to the nature of the antipatterns involved which are typically data or chatty Web services. Indeed, some false positives are recorded for the DWS and CWS antipatterns. These antipatterns are likely to be difficult to detect using metrics alone. Similar interpretations can be made for recall. The obtained results indicate that our approach is able to achieve a recall of 92%. The highest values were recorded for travel services with 96% where most of the detected services are GOWS and AWS. The lowest recall score was recorded for the location service (91%) which is attributable mostly to FGWS. Indeed, location Web services typically provide one or two operations which falsely matches the symptoms of FGWS

4.3.3 RESULTS FOR RQ3

Based on the results of Fig. 3, we observe that MOGP does not have a bias towards the detection of any specific antipattern type. As described the figure, we had an almost equal distribution of each antipattern type. On some Web services such as weather, the distribution is not as balanced. This is principally due to the number of actual antipattern types detected. Overall, all the 8 antipattern types are detected with good precision and recall scores (more than 88%). Most existing guidelines/definitions [1] [5] rely heavily on the notion of size to detect antipatterns.

This is reasonable for antipatterns like GOWS and FGWS that are associated with a notion of size, but for antipatterns like AWS, however, the notion of size is less important and this makes this type of anomaly hard to detect using structural information. This difficulty limits the performance of GP in detecting this type of antipattern. Thus, we can conclude that our MOGP approach detects well all the types of considered antipatterns (RQ3).

4.3.4 RESULTS FOR RQ4

Figure 4 reports the comparison result of MOGP, Ouni et al. [6] [8], and SODA-W. While SODAW shows promising results with an average precision of 71% and recall of 83%, it is still less than MOGP in all the eight considered antipattern types. We conjecture that a key problem with SODA-W is that it simplifies the different notions/symptoms that are useful for the detection of certain antipatterns. Indeed, SODA-W is limited to a set of WSDL interface metrics, but ignores the source code of the Web service artifacts. In fact, service design may look promising at the interface level, but can prove to be an antipattern if the source code is not implemented well. In contrast, our approach is based on both interface and code metrics. Another limitation of SODA-W is that in an exhaustive scenario, the number of possible antipatterns to manually characterize with rules can be very large, and rules that are expressed in terms of metric combinations need substantial calibration efforts to find the suitable threshold value for each metric. By contrast, our approach needs only some examples of antipatterns to generate detection rules. *Figure* 4 also shows that the mono-objective GP [6] provides lower detection results for the eight studied antipatterns with an average of 72% for both precision and recall. The lower performance can be explained by the fact that of the mono-objective formulation is based only on interface metrics that may not be able to capture all possible antipattern symptoms.

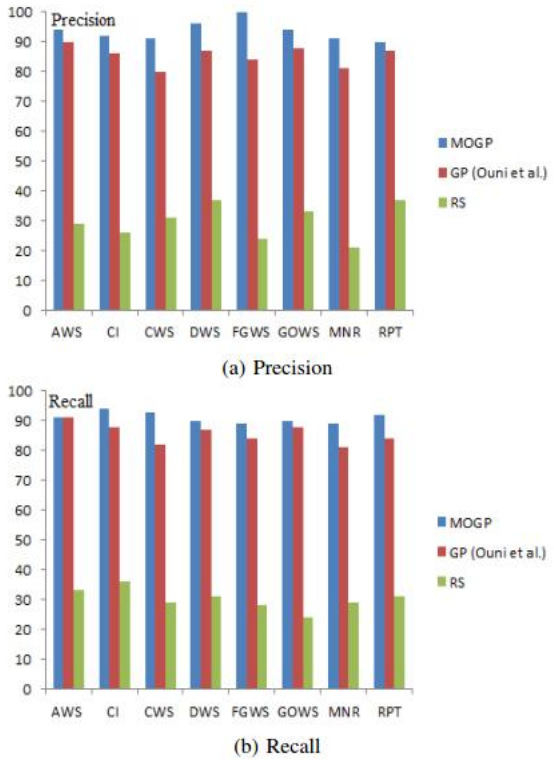

Figure 3: Detection results for each antipatterns type.

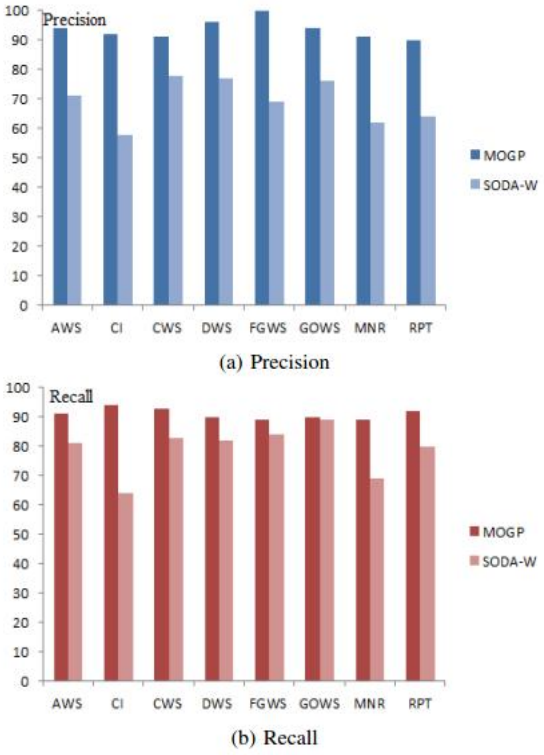

Figure 4: Comparative results of MOGP, Mono-objective GP and SODA-W

CHAPTER 5: RELATED WORK

Detecting and specifying antipatterns in SOA and Web services is a relatively new area. The first book in the literature was written by Dudney et al. [1] and provides informal definitions of a set of Web service antipatterns. More recently, Rotem-Gal-Oz described the symptoms of a range of SOA antipatterns [2]. Furthermore, Kral et al. [3] ´ listed seven "popular" SOA antipatterns that violate accepted SOA principles. In addition, a number of research works have addressed the detection of such antipatterns. Recently, Moha et al. [4] have proposed a rule-based approach called SODA for SCA systems (Service Component Architecture). Later, Palma et al. [5] extended this work for Web service antipatterns in SODA-W. The proposed approach relies on declarative rule specification using a domain-specific language (DSL) to specify/identify the key symptoms that characterize an antipattern using a set of WSDL metrics.

In another study, Rodriguez et al. [11] [12] and Mateos et al. [13] provided a set of guidelines for service providers to avoid bad practices while writing WSDLs. Based on some heuristics, the authors detected eight bad practices in the writing of WSDL for Web services. In other work [14], the authors presented a repository of 45 general antipatterns in SOA. The goal of this work is a comprehensive review of these antipatterns that will help developers to work with clear understanding of patterns in phases of software development and so avoid many potential problems. Mateos et al. [15] have proposed an interesting approach towards generating WSDL documents with less antipatterns using text mining techniques.

Recently, Ouni et al. [6] [8] proposed a search-based approach based on standard GP to find regularities, from examples of Web service antipatterns, to be translated into detection rules. However, the proposed approach can deal only with Web service interface metrics and cannot consider all Web service antipattern symptoms. Similar to [5], the latter consider neither deviation from common design practices nor code metrics, which leads to several false positives. Similarly to SODA-W [5], Moha et al. [16] proposed a description of antipattern symptoms using a domain-specific language (DSL) for their antipatterns detection approach called DECOR. They proposed a consistent vocabulary and DSL to specify antipatterns based on their review of existing work on code smells found in the literature. To describe antipattern symptoms different notions are involved, such as class roles and structures. Symptoms descriptions are later mapped to detection algorithms.

CHAPTER 6: CONCLUSION AND FUTURE WORK

In this thesis, we introduced a new multi-objective approach for the detection of Web Service antipatterns. In our multi-objective adaptation, two fitness functions are used to maximize the coverage of antipattern examples and minimize the coverage of well-designed Web service examples. The proposed approach is evaluated on a benchmark of 415 Web services and eight common Web service antipattern types. Statistical analysis of the obtained results provides compelling evidence that the proposed multi-objective algorithm outperforms mono-objective approaches, random search, and a recent state-of-the art approach with a median precision of more than 94% and a median recall of more than 92%.

As future work, we plan to extend the approach to detect business process antipatterns in SBS in addition to individual Web service antipatterns and automate the correction, through refactoring, of the detected antipatterns.

REFERENCES

[1] B. Dudney, J. Krozak, K. Wittkopf, S. Asbury, and D. Osborne, J2EE Antipatterns. John Wiley; Sons, Inc., 2003.

[2] A. Rotem-Gal-Oz, SOA Patterns. Manning Publications, 2012.

[3] J. Kral and M. Zemlicka, "Popular SOA Antipatterns," in Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. IEEE, 2009, pp. 271–276.

[4] N. Moha, F. Palma, M. Nayrolles, B. J. Conseil, Y.-G. Gueh´ eneuc, B. Baudry, and J.-M. J ´ ez´ equel, "Specification and ´ detection of soa antipatterns," in Service-Oriented Computing. Springer, 2012, pp. 1–16.

[5] F. Palma, N. Moha, G. Tremblay, and Y.-G. Gueh´ eneuc, ´ "Specification and detection of soa antipatterns in web services," in Software Architecture. Springer, 2014, pp. 58–73.

[6]] A. Ouni, R. Gaikovina Kula, M. Kessentini, and K. Inoue, "Web service antipatterns detection using genetic programming," in Proceedings of, ser. GECCO'15. ACM, 2015, pp. 1351–1358.

[7] B. K. Giri, J. Hakanen, K. Miettinen, and N. Chakraborti, "Genetic programming through bi-objective genetic algorithms with a study of a simulated moving bed process involving multiple objectives," Applied Soft Computing, vol. 13, no. 5, pp. 2613–2623, 2013.

[8] A. Ouni, M. Kessentini, and K. Inoue, "Search-based web service antipatterns detection," in IEEE Transactions on Services Computing, to appear. IEEE, 2016.

[9] "Experimental data," https://github.com/ouniali/ WSantipatterns, accessed: 2015-08-14.

[10] W. B. Frakes and R. Baeza-Yates, Eds., Information Retrieval: Data Structures and Algorithms. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992.

[11] J. M. Rodriguez, M. Crasso, C. Mateos, and A. Zunino, "Best practices for describing, consuming, and discovering web services: a comprehensive toolset," Software: Practice and Experience, vol. 43, no. 6, pp. 613–639, 2013.

[12] J. M. Rodriguez, M. Crasso, A. Zunino, and M. Campo, "Automatically detecting opportunities for web service descriptions improvement," in Software Services for e-World. Springer, 2010, pp. 139–150.

[13] C. Mateos, A. Zunino, and J. L. O. Coscia, "Avoiding WSDL Bad Practices in Code-First Web Services," SADIO Electronic Journal of Informatics and Operational Research, vol. 11, no. 1, pp. 31–48, 2012.

[14] M. A. Torkamani and H. Bagheri, "A Systematic Method for Identification of Anti-patterns in Service Oriented System Development," International Journal of Electrical and Computer Engineering, vol. 4, no. 1, pp. 16–23, 2014.

[15] C. Mateos, J. M. Rodriguez, and A. Zunino, "A tool to improve code-first web services discoverability through text mining techniques," Software: Practice and Experience, vol. 45, no. 7, pp. 925–948, 2015.

[16] N. Moha, Y. Gueheneuc, L. Duchien, and A. Le Meur, "Decor: A method for the specification and detection of code and design smells," Software Engineering, IEEE Transactions on, vol. 36, no. 1, pp. 20–36, Jan 2010.Ashok, B., Joy, J., Liang, H., Rajamani, S.K., Srinivasa, G., and Vangala, V.: 'DebugAdvisor: a recommender system for debugging', in Editor (Ed.) (ACM, 2009, edn.), pp. 373-382

[17] Dudney B., Krozak J., K., Asbury S., and Osborne D., J2EE Antipatterns. John Wiley; Sons, Inc., 1st edition (2003).

[18] Rotem-Gal-Oz A., Bruno, E., and Dahan, U., SOA Patterns. Manning Publications, 38-62 (2012).

[19] Kral J. and Zemlicka M., "Popular SOA Antipatterns," in Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. IEEE, pp. 271–276 (2009).

[20] Moha N., Palma F., Nayrolles M., Conseil B. J., Gueheneuc Y.-G., Baudry B., and Jezequel J.-M., "Specification and detection of soa antipatterns," in Service-Oriented Computing. Springer, pp. 1–16 (2012).

[21] Palma F., Moha N., Tremblay G., and Gueheneuc Y.-G., "Specification and detection of soa antipatterns in web services," in Software Architecture. Springer, pp. 58–73 (2014).

[22] Bard J.. Practical Bilevel Optimization: Algorithms and Applications. The Netherlands: Kluwer, Vol 30 (1998).

[23] Giri B. K., Hakanen J., Miettinen K., and Chakraborti N., "Genetic programming through bi-objective genetic algorithms with a study of a simulated moving bed process involving multiple objectives," Applied Soft Computing, vol. 13, no. 5, pp. 2613–2623 (2013).

[24] Ouni A., Kessentini M., and Inoue K., "Search-based web service antipatterns detection," in IEEE Transactions on Services Computing, to appear. IEEE, pp. 1-21 (2016).

[25] Frakes W. B. and Baeza-Yates R., Eds., Information Retrieval: Data Structures and Algorithms. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. (1992).

[26] Rodriguez J. M., Crasso M., Mateos C., and Zunino A., "Best practices for describing, consuming, and discovering web services: a comprehensive toolset," Software: Practice and Experience, vol. 43, no. 6, pp. 613–639 (2013).

[27] Rodriguez J. M., Crasso M., Zunino A., and Campo M., "Automatically detecting opportunities for web service descriptions improvement," in Software Services for e-World. Springer, pp. 139–150 (2010).

[28] Ankur S., Pekka M., Anton F., Kalyanmoy D., Multi-objective Stackelberg game between a regulating authority and a mining company: A case study in environmental economics. IEEE Congress on Evolutionary Computation, Cancun, Mexico, 478–485 (2013).

[29] Mateos C., Zunino A., and Coscia J. L. O., "Avoiding WSDL Bad Practices in Code-First Web Services," SADIO Electronic Journal of Informatics and Operational Research, vol. 11, no. 1, pp. 31–48 (2012).

[30] Torkamani M. A. and Bagheri H., "A Systematic Method for Identification of Anti-patterns in Service Oriented System Development," International Journal of Electrical and Computer Engineering, vol. 4, no. 1, pp. 16–23 (2014).

[31] Mateos C., Rodriguez J. M., and Zunino A., "A tool to improve code-first web services discoverability through text mining techniques," Software: Practice and Experience, vol. 45, no. 7, pp. 925–948 (2015).