Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search

Islem Saidani[1], Ali Ouni[1](✉), Mohamed Wiem Mkaouer[2], and Aymen Saied[3]

[1] ETS Montreal, University of Quebec, Montreal, QC, Canada
islem.saidani@ens.etsmtl.ca, ali.ouni@etsmtl.ca
[2] Rochester Institute of Technology (RIT), Rochester, NY, USA
mwm@se.rit.edu
[3] Concordia University, Montreal, QC, Canada
m_saied@encs.concordia.ca

Abstract. We introduce in this paper a novel approach, named *MSExtractor*, that formulate the microservices identification problem as a multi-objective combinatorial optimization problem to decompose a legacy application into a set of cohesive, loosely-coupled and coarse-grained services. We employ the non-dominated sorting genetic algorithm (NSGA-II) to drive a search process towards optimal microservices identification while considering structural dependencies in the source code. We conduct an empirical evaluation on a benchmark of two open-source legacy software systems to assess the efficiency of our approach. Results show that *MSExtractor* is able to find relevant microservice candidates and outperforms recent three state-of-the-art approaches.

Keywords: Microservices · Search-based software engineering · Legacy decomposition · Microservices architecture

1 Introduction

In this paper, we introduce a novel approach namely *MSExtractor*, that formulate the microservices extraction problem as a multi-objective combinatorial optimization problem to decompose an OO legacy application into a set of cohesive, loosely-coupled microservices. We employ the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [1], as search method to drive the decomposition process and find the near-optimal trade-off between two objectives: (1) minimize coupling (inter-service dependencies), and (2) maximize cohesion (intra-services dependencies) while leveraging the structural information embodied in the source code. MSExtractor aims at supporting software developers and architects by providing a decision-making support in their design decisions for their microservices migration.

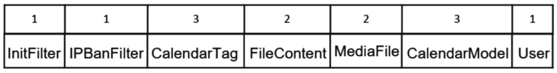
We conduct an empirical study to evaluate our approach on a benchmark of two open source Java legacy applications. Results show that MSExtractor is able to extract cohesive and loosely coupled microservices with higher performance than three recent state-of-the-art approaches.

2 Approach

We formulate the automated extraction of microservices from a legacy application as a combinatorial optimization problem, in which a search algorithm explores alternative combinations of classes from an input legacy system. Given legacy system composed of a set of classes to be decomposed into microservices, there are many ways in which the microservice boundaries can be drawn leading to different possible class combinations. The problem is a graph partitioning problem, which is known to be NP-hard and therefore seems suited to a metaheuristic search-based techniques [2].

To identify such instances of candidate microservices, MSExtractor proceeds to (i) create a set of new empty microservices, and (ii) assign each class to a unique microservice. The process should assign each class to exactly one microservice, and have no empty microservices. Then, MSExtractor uses NSGA-II [1] in order to find the optimal solutions that provide the best trade-off between our two objective functions.

Figure 1 shows a simple microservices decomposition example. A simple solution $X = \{1, 1, 2, 3, 1, 1, 2\}$, for example, denotes a decomposition of seven classes into three microservices. The classes $InitFilter$, $IPBanFilter$ and $User$ are in the microservice m_1, $Product$, $CarItem$ and $Category$ are in m_2, and finally, $Order$ and $Catalog$ are in m_3. Moreover, different class dependencies exist in order to implement the required functionalities by the microservice. An appropriate decomposition should maximize the cohesion within a microservice while minimizing coupling between the extracted microservices.



1	1	3	2	2	3	1
InitFilter	IPBanFilter	CalendarTag	FileContent	MediaFile	CalendarModel	User

Fig. 1. An example of a microservice decomposition solution (snippet) from `JPetstore`.

Source code dependencies are widely used in software engineering to measure how strongly related are the elements of a software system, *i.e.*, methods, classes, packages, etc. [3]. MSExtractor is based on a combination of *structural* measures to detect the dependencies among classes. In a nutshell, structural dependency for two given classes represents the shared method calls between them. We use two popular structural measurements to define our fitness objectives.

Objective Functions. To evaluate the quality (*i.e.*,the fitness) of a candidate microservices decomposition solution, we define a fitness function that evaluates multiple objective and constraint dimensions. Each objective dimension refers to a specific value that should be either minimized or maximized for a solution to be considered “better” than another solution. In our approach, we optimize the three following objectives:

1. **Cohesion:** The cohesion objective function is a measure of the overall cohesion of a candidate microservices decomposition. The cohesion of a candidate microservice m is denoted by $Coh(m)$ and defined as the complement of the average of all pairs of classes belonging to the microservice m. Then, the cohesion objective function corresponds to the average cohesion value of all microservice candidates in a decomposition. This objective function should be maximized to ensure that each candidate microservice contains strongly related classes and does not contain classes that are not part of its functionality.
2. **Coupling:** The coupling objective function measures the overall coupling among the microservice in a decomposition $\mathcal{M}$. We define the coupling between two microservices m_1 and m_2 as the average similarity between all possible pairs of classes from m_1 and m_2. The coupling objective function corresponds to the average coupling measures between all possible pairs of microservices in the decomposition. This objective function is to be minimized. The lower the coupling value between all candidate microservices, the better is the decomposition quality.

3 Empirical Evaluation

In this section, we present the results of our evaluation for the proposed approach, MSExtractor. The goal of this evaluation is to assess the efficiency of our approach in identifying appropriate microservices and compare it with available state-of-the-art approaches. This study aims at answering the following research question:

– **RQ1.** To what extent can MSExtractor identify relevant microservices?

Empirical Setup. To evaluate our approach, we conduct an experimental study on a benchmark of two legacy web applications namely JPetstore[1], and Springblog[2]. To answer RQ1, we employ four evaluation metrics to assess the quality of the identified microservices based on measuring their *functional independence*. This measure assesses the extent to which microservices exhibit a bounded context and present their own functionalities with low coupling to other microservices. In particular, four metrics were commonly used in recent studies [4–6] to assess the quality of Web service interfaces.

– ***CHM*** *(CoHesion at Message level):* CHM is inspired by LoC_{msg}, a widely used metric to measure the cohesion of a service at the message level [4–6].

[1] https://github.com/mybatis/jpetstore-6.
[2] https://github.com/Raysmond/SpringBlog.

- ***CHD*** *(CoHesion at Domain level):* CHD is inspired by LoC_{dom}, a widely used metric to measure the cohesion of a service at the domain level [4,5].
- **OPN** *(OPeration Number):* OPN computes the average number of public operations [4,7] exposed by an extracted microservice to other candidate microservices. The smaller OPN is, the better.
- **IRN** *(InteRaction Number):* IRN represents the number of method calls among all pairs of extracted microservices [4,8]. The smaller is IRN, the better is the quality of candidate microservices as it reflects loose coupling.

State-of-the-Art Comparison. We evaluate the performance of our approach, we compare it against three recent state-of-the-art approaches, namely *FoME* [4], *MEM* [9], and *LIMBO* [10]. We selected these three state-of-the-art methods as they use different decomposition techniques, and have been selected in recent comparative studies [4].

Table 1. The results achieved by MSExtractor, FoME, MEM, and LIMBO.

System	Metric	MSExtractor	FoME	MEM	LIMBO
Jpetstore	CHM	**0.5–0.6**	0.7–0.8	**0.5–0.6**	**0.5–0.6**
	CHD	**0.6–0.7**	**0.6–0.7**	**0.6–0.7**	**0.6–0.7**
	OPN	28	**22**	39	68
	IRN	**33**	35	48	329
SpringBlog	CHM	**0.5–0.6**	0.7–0.8	0.6–0.7	0.6–0.7
	CHD	**0.6–0.7**	0.8–0.9	0.8–0.9	0.7–0.8
	OPN	10	**7**	21	147
	IRN	**21**	26	30	238

3.1 Results

Table 1 presents the achieved results by each of the approaches, *MSExtractor*, and the compared approaches, *FoME* [4], *MEM* [9], and *LIMBO* [10]. The metrics CHM and CHD reflect the cohesion of the identified microservices, while the metrics OPN and IRN reflect the coupling. Higher cohesion metrics values indicate better performance while lower coupling metrics values indicate better performance. The cohesion results are provided in the form of an interval, e.g., [0.5–0.6], instead of specific values since slight differences between CHM or CHD values are not significant. We observe from the table that our approach, MSExtractor, outperforms the three competing approaches in the two studied systems, in the majority of metrics. In particular, for smaller systems such as `JPetStore` (24 classes), the achieved results on the four metrics are comparable. Indeed, this system represents a relatively smaller search space where deterministic approaches may achieve high performance. For larger systems, such as `Roller` and JForum (340 and 534, respectively), there is a clear superiority achieved by MSExtractor compared to the three compared approaches in terms of both CHM and CHD, as well as IRN.

We can also observe from Table 1 that FoME tends to provide better results in terms of OPN in three out of the two systems. This superiority is justified by the fact that FoME excludes a relatively important number of classes that are not covered by the dynamic analysis scenarios. These excluded classes will be, in turn, excluded from the candidate microservices. Obviously, ignoring a number of classes may improve coupling, but would provide functionally incomplete microservice candidates. These classes are generally related to exceptions, e.g., the classes *MailingException*, *FilePathException*, *BootstrapException* from the project Roller, or to other third-party or no-behavior classes, e.g., *YoutubeLinkTransformer*, *MessageHelper*, and *SecurityConfig* from the project Springblog. Such exclusion of classes from microservices would result in an incomplete architecture and would require a manual inspection by developers performing the migration.

4 Conclusions and Future Work

In this paper, we proposed MSExtractor, a novel approach that tackles the microservices extraction problem and formulates it as a multi-objective combinatorial optimization problem. Specifically, MSExtractor employs the non-dominated sorting genetic algorithm (NSGA-II) to drive a search process towards an optimal decomposition of a given legacy application while considering structural dependencies in the source code. Our evaluation demonstrates that MSExtractor is able to extract cohesive and loosely coupled services with higher performance than three recent state-of-the-art approaches.

As we only focused on the identification of microservices boundaries, we plan in our future work to investigate the other steps of the migration process towards containerization and pre-deployment configuration of our microservices candidates. We also plan to evaluate our approach form developers and software architects perspective on more systems. We also plan to consider non-functional criteria that are essential in the context of microservices architecture, including the scalability and availability of the system. We further plan on challenging the effectiveness of NSGA-II, being the main search algorithm used in MSExtractor, by performing a comparative study with other popular search algorithms, namely the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [1], and Strength Pareto Evolutionary Algorithm 2 (SPEA2) [11].

References

1. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
2. Mkaouer, W., et al.: Many-objective software remodularization using NSGA-III. ACM Trans. Softw. Eng. Methodol. (TOSEM) **24**(3), 17 (2015)

3. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)
4. Jin, W., Liu, T., Zheng, Q., Cui, D., Cai, Y.: Functionality-oriented microservice extraction based on execution trace clustering. In: 2018 IEEE International Conference on Web Services (ICWS), pp. 211–218. IEEE (2018)
5. Athanasopoulos, D., Zarras, A.V., Miskos, G., Issarny, V., Vassiliadis, P.: Cohesion-driven decomposition of service interfaces without access to source code. IEEE Trans. Serv. Comput. **8**(4), 550–562 (2015)
6. Ouni, A., Wang, H., Kessentini, M., Bouktif, S., Inoue, K.: A hybrid approach for improving the design quality of web service interfaces. ACM Trans. Internet Technol. (TOIT) **19**(1), 4 (2018)
7. Adjoyan, S., Seriai, A.-D., Shatnawi, A.: Service identification based on quality metrics object-oriented legacy system migration towards SOA. In: SEKE: Software Engineering and Knowledge Engineering (2014)
8. Newman, S.: Building Microservices: Designing Fine-grained Systems. O'Reilly Media, Sebastopol (2015)
9. Mazlami, G., Cito, J., Leitner, P.: Extraction of microservices from monolithic software architectures. In: 2017 IEEE International Conference on Web Services (ICWS) (2017)
10. Andritsos, P., Tzerpos, V.: Information-theoretic software clustering. IEEE Trans. Softw. Eng. **31**(2), 150–165 (2005)
11. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: improving the strength Pareto evolutionary algorithm, TIK-report, vol. 103 (2001)