 https://doi.org/10.3311/PPee.13925

Periodica Polytechnica Electrical Engineering and Computer Science, 63(4), pp. 274–281, 2019

A New Decomposition Method for Designing Microservices

Omar Al-Debagy[1*], Peter Martinek[1]

[1] Department of Electronics Technology, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, H-1521 Budapest, P. O. B. 91, Hungary
[*] Corresponding author, e-mail: omeraldebagy@gmail.com

Received: 22 February 2019, Accepted: 03 May 2019, Published online: 19 June 2019

Abstract

Many companies are migrating from monolithic architectures to microservice architectures, and they need to decompose their applications in order to create a microservices application. Therefore, the need comes for an approach that helps software architects in the decomposition process. This paper presents a new approach for decomposing monolithic application to a microservices application through analyzing the application programming interface. Our proposed decomposition methodology uses word embedding models to obtain word representations using operation names, as well as, using a hierarchical clustering algorithm to group similar operation names together in order to get suitable microservices. Also, using grid search method to find the optimal parameter values for Affinity Propagation algorithm, which was used for clustering, as well as using silhouette coefficient score to compare the performance of the clustering parameters. The decomposition approach that was introduced in this research consists of the OpenAPI specifications as an input, then extracts the operation names from the specifications and converts them into average word embedding using fastText model. Lastly the decomposition approach is grouping these operation names using Affinity Propagation algorithm. The proposed methodology presented promising results with a precision of 0.84, recall of 0.78 and F-Measure of 0.81.

Keywords

microservice architecture, microservices identification, service extraction, monolithic decomposition

1 Introduction

These days, large companies such as Amazon, Netflix, eBay, and others have utilized cloud architecture to operate their services. Cloud model provides the flexibility to these companies with the purpose of scaling the resources based on the actual consumption of the users [1].

One of the architectural patterns used to implement applications on the cloud is microservice architecture. It is a service oriented architectural (SOA) style where the application consists of small services working together as a single application. These services interact with each other using lightweight methods, for instance REST API [2].

On the other hand, there is the monolithic architecture, where all the services are contained in a single code base. These services exchange information with external systems through multiple mechanisms such as web services, HTML pages, or REST API [3]. Since many companies are migrating from monolithic architecture to microservice architecture, the process of converting a monolithic application into a microservices application is an important step. Microservices' decomposition is the process of converting an existing monolithic application to microservices application architecture. This is done through extracting services from the monolithic application into candidate microservices [4].

This paper presents a method for decomposing monolithic applications into microservices by identifying functions that are related to each other based on their semantic similarity. Furthermore, clustering of these similar functions is also performed to create actual microservice candidates. In the proposed decomposition method of this paper, the monolithic application is considered in the first step of the decomposition process, because microservices will use the functions of the monolithic application by grouping similar functions together.

Finding proper microservices can lead to easier maintainability and scalability of a software [5]. Therefore, microservices decomposition is an essential phase in the process of migrating from monolithic architecture to a microservice architecture. The method of services decomposition or extraction is required to identify microservice candidates from the monolithic application in this process. The decomposition itself is an essential process

Cite this article as: Al-Debagy, O., Martinek, P. "A New Decomposition Method for Designing Microservices", Periodica Polytechnica Electrical Engineering and Computer Science, 63(4), pp. 274–281, 2019. https://doi.org/10.3311/PPee.13925

for the whole experience of migrating to a microservice architecture because creating inappropriate microservices can lead to performance issues in the application and problems with governance policies at the organization. Thus, the properties of microservices like granularity, loose coupling, and high cohesion must be maintained during the decomposition [4].

Several methods and techniques were presented in the literature to extract microservices from monolithic applications. Some used semantic similarity between shared reference concepts in APIs to identify microservices [4]. Use cases and domain models were also applied to extract microservices [6]. These researches will be discussed in detail in the literature review section.

The aim of our research was to provide a new methodology, using word embedding and hierarchical clustering method, to identify microservice candidates in a monolithic application through analyzing the application programming interface "API". OpenAPI [7] specification was used, which is a format description for REST APIs that is useable for automatic processing. OpenAPI specification contains endpoints, operation names, operation parameters, inputs, and outputs of operations. Hence, we used the OpenAPI specification to obtain operation names, which is widely used by developers and is fully machine-readable.

2 Literature Review

Several methods were presented for extracting microservices from a monolithic application. For instance, Gysel et al. [6] developed Service Cutter a service decomposition framework in which domain models and use cases were used to extract coupling information from, and this information was represented as weighted graphs to allocate closely related services. Also, they deployed Newman and Girvan [8] and Epidemic Label Propagation [9] clustering algorithms on the extracted coupling information. One issue with these clustering algorithms is that they require the number of clusters to be assigned in order to be functional, which in this case can be a weak point for this framework because it is hard to define the number of services for large applications. For the evaluation of the method, they developed their own classification metrics to calculate the quality of the candidate services. Service Cutter framework was tested with two sample applications, and the suggested services ranged from good to bad according to the classification method. Overall results were good for one test application and acceptable for the other one [6].

Baresi et al. [4] proposed a solution based on the semantic similarity between the operations available in OpenAPI specifications. They used Schema.org specifications as a vocabulary reference to be mapped with the specifications of the available OpenAPI. Also, a fitness function was applied which was based on DISCO [10], which is a tool that calculates the distributional similarity between two words in a large corpus of the text. The idea of Baresi et al. [4] was to couple standardized OpenAPI specifications with similar semantic characterizations. Baresi et al. [4], evaluated their method using two microservices applications: the first one was a Money Transfer application, which consists of four microservices, and the second application was the Kanban Board which consists of three microservices. The resulted microservices candidates of the first application were 80 % accurate which means 8 of the 10 operations were properly decomposed. In the second application, the accuracy rate was 77 %, 10 of the 13 operations were decomposed correctly. Thus, they claimed that their method was 80 % accurate, if the expected decompositions would have been available in advance. In their paper, they showed that their method produced better results than the Service Cutter [6] method.

Another research is done by Mazlami et al. [11] where a formal model for microservices extraction was introduced through employing the source code of the monolithic application as an input for the method and converting it to a graph representation. After that, clustering algorithms were applied to the graph representation to produce microservices candidates. Their method consists of three stages:

1. the first stage is the monolith stage,
2. the second stage is the graph stage,
3. and the third stage is the microservices stage.

Our proposed decomposition methodology uses word embedding models such as fastText [12] to obtain word representations to find similarities between words in operation names, as well as, using a hierarchical clustering algorithm, which uses message passing between data points, to group similar operation names together into actual microservices. So when compared to Service Cutter [6] method, the proposed method of this research has several advantages. For example, the clustering algorithm does not need to specify the number of clusters in advance because it finds it automatically. Also, when compared to paper of Baresi et al. [4], which is more similar to our approach in terms of using semantic similarity to find relations between operation names, but they

used DISCO [10] "Distributionally related words using co-occurrences" to find semantic similarity between words, while word embedding was used in our approach.

3 Methodology

The decomposition approach that was introduced in this research consists of the OpenAPI specifications as an input, then extracts the operation names from the specifications and converts them into average word embedding using a given model. fastText [12] and Word2Vec [13] models were utilized to obtain word vectors from the operation names. To obtain word representation, a vector is created from input tokens by searching a word embedding model [14]. Algorithm 1 presents a general overview of the proposed decomposition method.

In this research, Word2Vec [14] was trained on Google News corpus, and fastText was trained on different datasets. Nevertheless, before converting the operation names into vectors, removing the stop words was an initial step, because without the removal of stop words might lead to inaccurate results. Also, a list of specific words was created to be removed from the operation names, because these words can change the meaning of the sentence or the operation name in this context. For example, the word "post", "get", "update", and others, which can be found in many operation names.

3.1 Word Embedding Models

The Word2Vec embedding model is a combination of two different algorithms, the continuous bag of words (CBOW) and skip-gram. The skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. The goal of Skip-gram model is to find word embedding that is good for predicting the semantically closest words in a large corpus of text [14].

Algorithm 1 Our proposed decomposition algorithm

```
    Data: OpenAPI Specifications
    Results: microservices' candidates
1   sentences ← θ
2   foreach operationName do
3       sentences ← ConvertToLowerCase(operationName)
4       sentences ← RemoveStopWords(sentences)
5       sentences ← ShorttextToVector(operationName)
6   end
7   microservicesCandidates ← AffinityPropagation(sentences)
8   Return microservicesCandidates
```

fastText [12] is a library for word embedding and sentence classification that was created by Facebook Research Group. It is also an evolution of the Word2Vec model.

In this research, two pre-trained models were used for word representations with two different text corpora. First one is the Word2Vec model with Google News corpus, the second model is the fastText with one million-word vectors trained on Wikipedia 2017. The purpose of using fastText and Word2Vec models is to create a sentence level embedding or operation's name level embedding as used in this research.

3.2 Operation Name Vector

Usually, operation names have more than one word in their names. Therefore, in order to convert the operation name to a word representation vector, we utilized a method of getting the sum of each word vector in the operation name and dividing it by the number of words in the same operation name. In other words, getting the average of word vectors in an operation name as it was proposed by Le and Mikolov [15]. Thus, this method returns the average of all word representations for each operation name. The output needs to be fed by a clustering algorithm such as Affinity Propagation after that.

3.3 Clustering Method

After converting the operation names into vectors based on the word embedding of the pretrained models and removing stop words the clustering is applied. The clustering method was utilized to group similar operation names together in order to create a microservice candidate consist of these similar operation names. The Affinity Propagation [16] algorithm was used, because it defines the number of clusters without the need to specify it beforehand. This clustering algorithm will find the number of microservices that is going to be created from the API of a monolithic application.

Affinity Propagation works by passing messages between data points, also, it finds exemplars, which are unique data points that represent the clusters and each cluster has one exemplar [16]. The purpose of these messages is to find the willingness of the data points to be exemplars. These exchanged messages between the data points are divided into two types. The first type is "responsibility" messages, which are messages sent from data points to candidate exemplars, in order to show if the data points are suitable being a member of the candidate exemplar's cluster. The "responsibility" denoted by $r(i,k)$ in Eq. (1) indicates if

point k is suited to be an exemplar for point i. Responsibilities are exchanged from point i to exemplar to be k:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\} \ . \quad (1)$$

The second type is "availability" messages, which are messages sent from candidate exemplars to other data points, demonstrating if the candidate exemplar is suitable to be an exemplar. The "availability" denoted by $a(i,k)$ in Eq. (2) indicates if point i can choose point k as an exemplar. Availabilities are exchanged between exemplar candidate k and data point i starting from k:

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ s.t } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\} \ . \quad (2)$$

Self-availability is updated in a different way, which can be seen in Eq. (3):

$$a(k,k) \leftarrow \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(i',k)\} \ . \quad (3)$$

Then the algorithm finds pairwise similarities between the data points, and it will identify the clusters by maximizing the total similarity between the exemplars and their data points.

Mézard [17] explained the importance and efficiency of message passing algorithms even on complicated problems. Hence, Affinity Propagation was utilized in this research paper for clustering similar operation names to create candidates for microservices.

In Affinity Propagation algorithm there are three parameters which are related to the performance of the algorithm:

1. The first parameter is damping, which damps the exchange of messages between responsibility and availability to prevent numerical oscillations while updating the values of responsibilities and availabilities [18].
2. The second one is the maximum number of iterations.
3. The third one is the number of iterations with no change in the number of estimated clusters that stops the convergence.

Algorithm 2 summarizes all the steps in Affinity Propagation algorithm.

Algorithm 2 Affinity Propagation algorithm

Data: $\{s(i,j)\}_{i,j \in \{1,..,N\}}$ data similarities and preferences

Results: cluster assignments $\hat{c}$

Availability ← 0

repeat a and r updates until convergence

  $r(i,k) \leftarrow s(i,k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i,k') + s(i,k')\}$

  $a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ s.t. } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\}$

  if $k \neq i$

    $a(k,k) \leftarrow \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(i',k)\}$

  end

end

Return $\hat{c} = \arg\max_k [a(i,k) + r(i,k)]$

3.4 Evaluation Metrics

Calculating the Silhouette coefficient [19] is a method for validating data consistency within clusters. It measures the similarity between an object and its own cluster compared to other clusters. Silhouette coefficient score ranges from −1 to 1, this means an object is matched correctly to its cluster when it has a value of 1 for its silhouette coefficient $s(i)$. This method was used to evaluate the performance of the clustering algorithm while using different values for the algorithm's parameters.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \ , \quad (4)$$

where $a(i)$ represents the average dissimilarity of object i, to all other objects of the same cluster. $b(i)$ represents the smallest average distance between object i and any other object in other clusters.

Also, grid search was used to try different combinations of the three parameters of the clustering algorithm in order to find the optimal values for these parameters.

Precision, recall, and F-measure were the metrics used to evaluate the performance of the proposed decomposition method.

The precision for the clustering method that is used in this paper is the averaged precision of each object or in this case of each operation name. The precision $P(O_O)$ of a given object O_O, which will find the precision of the object O_O in a computed cluster compared with the ideal cluster of the same object $C(O_O)$:

$$P(O_O) = \begin{cases} \dfrac{\left|S_{\tau(O_O)} \cap C(O_O)\right| - 1}{\left|C(O_O)\right| - 1} \ , & \left|C(O_O)\right| > 1 \\ 1 \ , & \left|C(O_O)\right| = 1 \end{cases} \ , \quad (5)$$

where $S_{\tau(O_O)}$ denotes the object related to the same ideal cluster that the selected object O_O belongs to. After calculating the precision of every object, the precision of the clustering P is the average of the precisions of every object.

The recall $R(O_O)$ of a given object O_O, which will find the recall of the object O_O in a computed cluster compared with the ideal cluster of the same object $C(O_O)$:

$$R(O_O)=\begin{cases}\dfrac{\left|S_{\tau(O_O)}\cap C(O_O)\right|-1}{\left|C(O_O)\right|-1}, & \left|\tau(O_O)\right|>1\\ 1, & \left|\tau(O_O)\right|=1\end{cases}, \tag{6}$$

where $S_{\tau(O_O)}$ denotes the object related to the same ideal cluster that the selected object O_O belongs to. After calculating the recall of every object, the recall of the clustering R is the average of the recalls of all the objects.

Furthermore, F-Measure $F1$ was used to get the harmonic mean of recall and precision, where 1 represent the best value and 0 represent the worst result. F-Measure can be calculated as follows:

$$F1=2*\frac{P*R}{P+R}. \tag{7}$$

4 Results and Discussion

For the implementation of the algorithm in this research, Python [20] programming language was utilized with specific libraries for text analysis and clustering such as Gensim [21], NLTK [22], and Sklearn [23].

In order to find the optimal values for the parameters of Affinity Propagation algorithm like damping, the maximum number of iterations, and convergence iterations, grid search approach was used. The test cases were the APIs of four different applications such as, Amazon Web Services, PayPal, Kanban Board, and Money Transfer app. Furthermore, Silhouette coefficient "SC" was utilized to evaluate the performance of the clustering algorithm with different parameter values. The parameter values were a range of numbers. First, damping "DA" value started from 0.5 until 0.9 with 0.1 step size. Second, maximum iteration "MI" ranges from 100 to 1000 with 100 step size. Finally, convergence iteration "CI" ranges from 10 to 100 with 10 step size. Eventually, we found, that the optimal values for the parameters were 0.6 for damping, 300 for maximum iteration, and 50 for convergence iteration. Table 1 shows the highest values of average Silhoutte coefficient score and the parameter values were used to achieve these results.

Fig. 1 illustrates the Silhouette coefficient score of each operation name and its cluster for PayPal's API using affinity propagation algorithm with the optimal parameters that were obtained previously. The average Silhouette coefficient was 0.43.

Table 1 Average Silhouette coefficient scores

App	DA	MI	CI	SC
AWS	0.6	300	50	0.50
Money	0.6	300	50	0.40
PayPal	0.6	300	50	0.43
Kanban	0.6	300	50	0.55

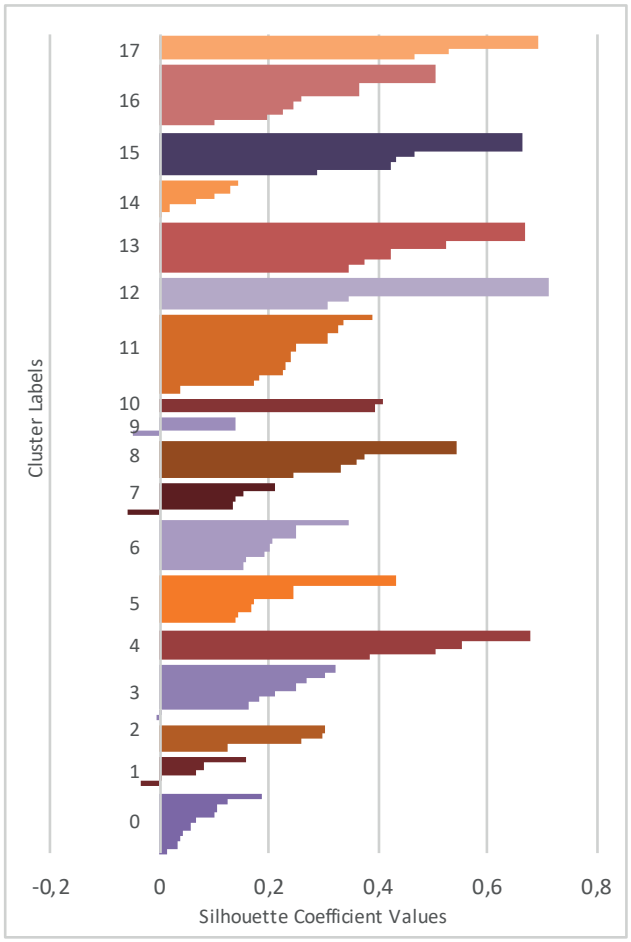

Fig. 1 PayPal's Silhouette coefficient score of each operation name

Once the optimal parameters for Affinity Propagation algorithm were chosen, there was a need to choose which word embedding model should be used for the proposed decomposition method. Several tests with 4 different test cases conducted were chosen. F-Measure was used to compare the performance between Word2Vec and fastText models.

As it is apparent in Fig. 2, the performance of fastText was better than the performance of Word2Vec in terms of F-Measure. Therefore, fastText was selected to be used for conducting the final tests with the 4 different test cases.

After performing several tests, 4 different OpenAPI specifications of different applications were evaluated. These applications were the Money Transfer [24] application

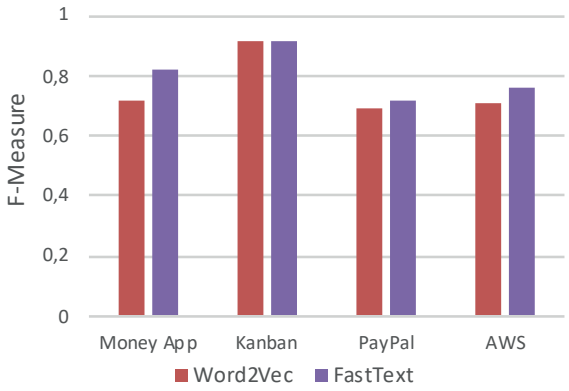

Fig. 2 Performance of Word2Vec vs fastText

with 11 operations and 4 microservices, the Kanban Board [25] application which contained 13 operations and 3 microservices. Both of them were created by Chris Richardson the author of Microservice Patterns [26] book, serving as a good standard for evaluating the performance of the decomposition method presented in this paper.

Subsequently, the application of the proposed decomposition method on these two examples gave an excellent result of 100 % precision and 85 % recall for Kanban Board application, and 82 % precision and recall for Money Transfer application. Table 2 compares the proposed decomposition using our method against the standard design of the two applications. Comparing these results with the results of Baresi et al. [4], our proposed method performed better in the decomposition of the Kanban Board application by decomposing 12 out of 13 operations correctly, in comparison to method of Baresi et al. [4], which decomposed only 10 operations correctly. For the Money Transfer application, 10 of the 11 operations were decomposed correctly using our method, while in research of Baresi et al. [4] only 8 operations were found correctly during the decomposition. Consequently, the proposed method in this paper showed a better performance when compared to other methods in the literature, as research of Baresi et al. [4] showed already that their method performed better than Service Cutter [6].

Furthermore, additional test cases were created with real life examples of applications that are already used in a real life environment. Thus, we searched for companies using microservice architecture in their applications; for example, Netflix, Amazon, PayPal, Twitter, and others [27]. Eventually, Amazon Web Services and PayPal were selected as a case study for the proposed decomposition methodology, because their API was available in OpenAPI specifications definition. So, these were compatible with our method.

Table 2 Comparison of the proposed method's decomposition and the standard decomposition of the two applications

Application	Proposed Decomposition	Optimal Decomposition
Money Transfer	addToAccountUsing createAccountUsing getAccountUsing	addToAccountUsing createAccountUsing getAccountUsing
	createCustomerUsing getAccountsForCustomerUsing getCurrentUserUsing getCustomerUsing getCustomersByEmailUsing getTransactionHistoryUsing	createCustomerUsing getAccountsForCustomerUsing getCurrentUserUsing getCustomerUsing getCustomersByEmailUsing
	doAuthUsing	doAuthUsing
	moneyTransferUsing	moneyTransferUsing getTransactionHistoryUsing
Kanban Board	doAuthUsing	doAuthUsing
	getBoardUsing listAllBoardsUsing saveBoardUsing	getBoardUsing listAllBoardsUsing saveBoardUsing
	backlogTaskUsing completeTaskUsing deleteTaskUsing listAllTasksUsing saveTaskUsing scheduleTaskUsing startTaskUsing updateTaskUsing	completeTaskUsing deleteTaskUsing listAllTasksUsing saveTaskUsing scheduleTaskUsing startTaskUsing updateTaskUsing
	getHistoryUsing	backlogTaskUsing getHistoryUsing

The number of operations in Amazon Web Services API was 318 divided into 47 microservices. On the other hand, PayPal's API has 110 operations scattered on 16 microservices. This shows the size of the applications, and how challenging it is to decompose them manually without any automation.

Performing the decomposition process on these two APIs and comparing the decomposition results with the already available services in the API documentations of each application, our method presented promising results in terms of precision and recall. For example, the precision of the proposed decomposition method was 74 %, and recall was 79 % obtained from decomposing Amazon Web Services API. For PayPal API, the performance was less accurate, precision was 80 %, while recall was 66 %. Table 3 shows the results of all tests using the 4 different applications.

Accordingly, in total there were 4 applications with 452 operations tested using our decomposition method. By getting the precision and recall, F-measure was calculated as was mentioned before. The averaged F-Measure was 81 %, while the averaged precision of all the tests was 84 % and the averaged recall was 78 %. These results showed that the proposed decomposition method is suitable to be a helping tool for software architects by decomposing a monolithic application into a microservices application.

Table 3 The performance of the proposed decomposition methodology

Application	Precision	Recall	F-Measure	# of Operations
AWS	0.74	0.79	0.76	318
Kanban Board	1	0.85	0.92	13
Money Transfer	0.82	0.82	0.82	11
PayPal	0.8	0.66	0.72	110
	Precision Average	Recall Average	F-Measure Average	Total
	0.84	0.78	0.81	452

5 Conclusion and Future Work

This paper proposed a novel approach to identify microservices in the process of migrating from monolithic architecture to a microservice architecture. The proposed method consists of several steps, starts with the extracted operation names from OpenAPI specifications. The second step is the process of converting the operation names into word representation using word embedding models. The third step is the clustering of semantically similar operation names in order to create candidates of microservices. The proposed method showed significantly better results when compared it to other methods from the literature, resulted in F-Measure of 0.81, a precision of 0.84 and a recall of 0.78. Therefore, this can be an aiding addition for software architects in the process of extracting microservices from monolithic applications.

Furthermore, we proposed a new microservices decomposition method using word embedding and hierarchical clustering method to identify potential microservices through analyzing application programming interfaces.

For the future, this work can be improved by finding a different method for calculating the distance between word vectors such as using word mover's distance instead of averaging the word representations in operation names. Another possible area of further research is the developing of new evaluation metrics based on service granularity. The general goodness of the microservices' grouping should be addressed with it directly.

References

[1] Chen, R., Li, S., Li, Z. "From Monolith to Microservices: A Dataflow-Driven Approach", In: 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing, China, 2017, pp. 466–475. http://doi.org/10.1109/apsec.2017.53

[2] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L. "Microservices: Yesterday, Today, and Tomorrow", In: Mazzara, M., Meyer, B. (eds.) Present and Ulterior Software Engineering, Springer, Cham, Switzerland, 2017, pp 195–216. https://doi.org/10.1007/978-3-319-67425-4_12

[3] Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., … Lang, M. "Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures", Service Oriented Computing and Applications, 11(2), pp. 233–247, 2017. http://doi.org/10.1007/s11761-017-0208-y

[4] Baresi, L., Garriga, M., De Renzis, A. "Microservices Identification Through Interface Analysis", In: De Paoli, F., Schulte, S., Broch Johnsen, E. (eds.) Service-Oriented and Cloud Computing, Lecture Notes in Computer Science, vol. 10465, Springer International Publishing, Cham, Switzerland, 2017, pp. 19–33. http://doi.org/10.1007/978-3-319-67262-5_2

[5] Wilde, N., Gonen, B., El-Sheikh, E., Zimmermann, A. "Approaches to the Evolution of SOA Systems", In: El-Sheikh, E., Zimmermann, A., Jain, L. (eds.) Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures, Intelligent Systems Reference Library, Vol. 111, Springer, Cham, Switzerland, 2016, pp. 5–21. http://doi.org/10.1007/978-3-319-40564-3_2

[6] Gysel, M., Kölbener, L., Giersche, W., Zimmermann, O. "Service Cutter: A Systematic Approach to Service Decomposition", In: Aiello, M., Johnsen, E. B., Dustdar, S., Georgievski, I. (eds.) Service-Oriented and Cloud Computing, Lecture Notes in Computer Science, Vol. 9846, Springer International Publishing, Cham, Switzerland, 2016, pp. 185–200. http://doi.org/10.1007/978-3-319-44482-6_12

[7] OpenAPI Initiative "Home - OpenAPI Initiative", [online] Available at: https://www.openapis.org/ [Accessed: 03 May 2019]

[8] Newman, M. E. J., Girvan, M. "Finding and evaluating community structure in networks", Physical Review E, 69(2), article ID: 026113, 2004. http://doi.org/10.1103/physreve.69.026113

[9] Raghavan, U. N., Albert, R., Kumara, S. "Near linear time algorithm to detect community structures in large-scale networks", Physical Review E, 76(3), article ID: 036106, 2007. http://doi.org/10.1103/physreve.76.036106

[10] Kolb, P. "DISCO: A Multilingual Database of Distributionally Similar Words", In: Storrer, A., Geyken, A., Siebert, A., Würzner, K.-M. (eds.) KONVENS 2018 - Ergänzungsband: Textressourcen und lexikalisches Wissen, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Germany, 2008, pp. 5–12. [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.329.9446 [Accessed:13 January 2019]

[11] Mazlami, G., Cito, J., Leitner, P. "Extraction of Microservices from Monolithic Software Architectures", In: International Conference on Web Services (ICWS), Honolulu, HI, USA, 2017, pp. 524–531. http://doi.org/10.1109/icws.2017.61

[12] Joulin, A., Grave, E., Bojanowski, P., Mikolov, T. "Bag of Tricks for Efficient Text Classification", 2016. [online] Available at: http://arxiv.org/abs/1607.01759 [Accessed: 28 January 2019]

[13] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J. "Distributed Representations of Words and Phrases and their Compositionality", 2013. [online] Available at: http://arxiv.org/abs/1310.4546 [Accessed: 23 January 2019]

[14] Ma, C., Xu, W., Li, P., Yan, Y. "Distributional Representations of Words for Short Text Classification", In: Blunsom, P., Cohen, S., Dhillon, P., Liang, P. (eds.) Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing, Association for Computational Linguistics, Red Hook, New York, USA, 2015, pp. 33–38. [online] Available at: http://www.aclweb.org/anthology/W15-1505 [Accessed: 21 January 2019]

[15] Le, Q., Mikolov, T. "Distributed Representations of Sentences and Documents", Proceedings of the 31st International Conference on Machine Learning, 32(2), pp. 1188–1196, 2014. [online] Available at: http://arxiv.org/abs/1405.4053 [Accessed: 23 January 2019]

[16] Frey, B. J., Dueck, D. "Clustering by Passing Messages Between Data Points", Science, 315(5814), pp. 972–976, 2007. http://doi.org/10.1126/science.1136800

[17] Mézard, M. "Where Are the Exemplars?", Science, 315(5814), pp. 949–951, 2007. http://doi.org/10.1126/science.1139678

[18] Refianti, R., Mutiara, A. B., Syamsudduha, A. A. "Performance Evaluation of Affinity Propagation Approaches on Data Clustering", International Journal of Advanced Computer Science and Applications, 7(3), pp. 420–429, 2016. http://doi.org/10.14569/ijacsa.2016.070357

[19] Rousseeuw, P. J. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis", Journal of Computational and Applied Mathematics, 20, pp. 53–65, 1987. https://doi.org/10.1016/0377-0427(87)90125-7

[20] Python Software Foundation "Welcome to Python.org", [online] Available at: https://www.python.org/ [Accessed: 13 February 2019]

[21] Řehůřek, R., Sojka, P. "Software Framework for Topic Modelling with Large Corpora", In: Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks, University of Malta, Valletta, Malta, 2010, pp. 46–50. [online] Available at: https://is.muni.cz/publication/884893/en [Accessed: 23 January 2019]

[22] Loper, E., Bird, S. "NLTK: The Natural Language Toolkit", In: Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics, vol. 1, Association for Computational Linguistics, Stroudsburg, PA, USA, 2002, pp. 63–70. http://doi.org/10.3115/1118108.1118117

[23] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. "Scikit-learn: Machine Learning in Python", Journal of Machine Learning Research, 12(October), pp. 2825−2830, 2011. [online] Available at: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html [Accessed: 20 January 2019]

[24] Richardson, C. "Example code for my building and deploying microservices with event sourcing, CQRS and Docker presentation", [computer program] Available at: https://github.com/cer/event-sourcing-examples [Accessed: 22 January 2019]

[25] Richardson, C. "Multi-user Kanban board built using Eventuate, DDD, microservices, event sourcing, CQRS, and Spring Boot", [computer program] Available at: https://github.com/eventuate-examples/es-kanban-board [Accessed: 22 January 2019]

[26] Richardson, C. "Microservices Patterns: With examples in Java", 1st ed., Manning Publications, New York, United States, 2018.

[27] Novoseltseva E. "Benefits & Examples of Microservices Architecture Implementation", 2018. [online] Available at: https://apiumhub.com/tech-blog-barcelona/microservices-architecture-implementation/ [Accessed: 15 January 2019]