Received: 14 June 2021 | Revised: 28 February 2022 | Accepted: 1 March 2022

DOI: 10.1002/spe.3081

RESEARCH ARTICLE

WILEY

A service graph based extraction of microservices from monolith services of service-oriented architecture

Vinay Raj | **Sadam Ravichandra**

Department of Computer Science and Engineering, National Institute of Technology Warangal, Warangal, Telangana, India

Correspondence
Vinay Raj, Department of Computer Science and Engineering, National Institute of Technology Warangal, Warangal, Telangana, India.
Email: rvinay1@student.nitw.ac.in

Abstract

Service-oriented architecture (SOA) has been widely used to design enterprise applications in the past two decades. The services in SOA are becoming complex with the increase in changing user requirements and SOA is still seen as monolithic from a deployment perspective. Monolithic services make the application complex, and it becomes difficult to maintain. With the evolution of microservices architecture, software architects started migrating legacy applications to microservices. However, existing migration approaches in the literature mostly focus on migrating monolithic applications to microservices. To the best of our knowledge, very few works have been done in migrating SOA applications to microservices. One of the major challenges in the migration process is the extraction of microservices from the existing legacy applications. To address this, we propose an approach to extract the candidate microservices using graph based algorithms. In particular, four algorithms are defined: (i) construction of *service graph* (SG), (ii) construction of *task graph* (TG) for each service of the a SOA application, (iii) extraction of candidate microservices using the SG of SOA application, and (iv) construction of a SG for a microservices application to retain the dependencies between the generated microservices. We chose a SOA-based web application to demonstrate the proposed microservices extraction approach and extracted the microservices. Additionally, we have evaluated the extracted microservices and compared them with SOA based services.

KEYWORDS

microservices, migration, monoliths, service graph, service oriented architecture

1 | INTRODUCTION

Distributed systems have evolved rapidly in the past two decades as the demand for the fast delivery of services has increased.[1] Distributed system has emerged from the monolithic style of client-server architecture to service-oriented architecture (SOA), and then to the trending microservices. Monolithic applications are built as a large block of code and deployed as a single archive file.[2] Since the applications are built as a single piece of code, the entire application must be

Abbreviations: CS, coupling of services; DAG, directed acyclic graph; NoS, number of services; RCS, relative coupling of services; SCF, service coupling factor; SG, service graph; SOA, service-oriented architecture; TG, task graph; VMS, vehicle management system; WSDL, Web Services Description Language.

deployed for every update. It becomes difficult to make changes as the entire application needs to be stopped.[3] Hence, monolithic applications are hard to maintain, and the complexity of the application increases. To avoid the challenges in monolithic applications, the SOA style was adopted.[4]

SOA is a style for designing applications where each component is a service. It has been widely used in integrating large enterprise applications using enterprise service buses as the communication medium.[5] A service is the primary component in SOA based systems, and each service implements the business requirements without being aware of other service states.[6] SOA provides loosely coupled reusable and scalable services. A typical implementation of SOA is based on web services with interfaces defined in Web Services Description Language (WSDL).[7] A web service is a loosely coupled software application that can be discovered, described, and accessed based on XML and standard web protocols over intranets, extranets, and the Internet. The WSDL document used in web services can exhibit design problems with the increase in size, which could impact quality attributes like performance and maintainability.[8] The use of WSDL based solutions declined after REST and HTTP protocols were used for the exchange of messages with JSON for data exchange format.[9] However, the use of RESTful web services also exhibited service reuse and composition issues. Additionally, with the increase in software enhancements and changes in business requirements, services in the SOA system are tending toward monolithic making the applications domain-specific and less reusable. For example, if ESB fails due to any reason, the entire application ends up failing as all the services are dependent on the ESB. ESB becomes a single point of failure. Nevertheless, architects could not get the fundamental use of SOA and left behind the key design principles.[10] Therefore, with the evolution of microservices architecture, the need to understand the architectural concepts and design principles of microservices is on demand.

Microservices are a new architectural style of designing applications that follows the single responsibility principle in which each service performs only one task.[11] Microservices use lightweight protocols like REST/JSON for communication between the services and cloud containers for deployment. Containers are scalable, secure, and reliable.[12] It follows continuous integration and continuous delivery strategy while designing applications. Services with high load can be dynamically detected and scaled independently.[13]

Though monolith applications are very easy to design, test, and deploy, the addition of new features and bug fixing consume more time when compared to microservices.[14] It is difficult for developers to work independently in monolithic applications. In contrast, in microservices, developers can design, test, and deploy the applications independently, and they also have the technological freedom to choose the programming language of their choice.[15] Moreover, monolithic applications can scale only in one direction, and we cannot scale each component independently. We can scale the application in multiple dimensions in microservices, and each component can be scaled independently.

The primary difference of microservices style with SOA is the emphasis on scalability, independence, and semantic cohesiveness of each component in the system.[16] In contrast with SOA, microservices are required to be self-contained with data, user interface, and databases. Software architects may believe that microservice is SOA done right, but microservice architecture is about designing isolated services with a strong focus on data isolation.[17]

Because of the diverse benefits, IT companies have started designing their applications using microservices architecture, and few of them have started migrating their applications to microservices.[18] There are numerous reasons which trigger migration toward microservices. As architects are unaware of the effort and cost estimation required for designing the application from scratch, migrating is the best approach.[19] A systematic mapping study conducted by Di Francesco et al. states that research for the migration to microservices is at an early stage.[20] Migrating applications to the cloud have also aroused for migrating to microservices as it suits better in the cloud environment.[21] Technical debt has reduced by migrating the existing legacy application to microservices, and maintenance has improved.[22]

Some recent research works focus on migration of existing applications to microservices architecture. Balalaie et al.[23] have proposed migration patterns which are useful in migrating non cloud native applications to cloud native microservices based applications. The proposed patterns are related to migration of monolithic applications to microservices. De Iasio and Zimeo[24] proposed a framework to support synchronization among microservices to avoid unavailability of dependent microservices to achieve the goal of the application. Freire et al.[25] proposed an approach for migration of monolithic applications to microservices using aspect oriented programming. The approach can be done with zero downtime and with minimal changes. Laso et al.[26] proposed a framework based on microservices which is helpful in integration of humans in the deployment of IoT devices. Filisbino Passini et al.[27] comprehensive analysis on the design of self-adaptive service oriented applications (Self-Apps). The authors also highlight that there is a need to study the differences between both SOA and microservices architecture.

In this article, for migrating SOA based services into microservices, a three-step approach is proposed. In the first step, the service graph (SG) construction algorithm is presented for the given SOA application, and for each service, task graphs (TGs) are constructed using another algorithm. In the second step, we present a microservices extraction algorithm using the SG, and generate the candidate microservices. Finally, the SG construction algorithm for the microservices application is proposed, which helps in retaining the dependencies of each microservices. Each node in the graph will represent a service, and the edge between the node represents the dependency between the services. This three-step approach for extraction of microservices is fully automated.

The remaining part of the article is organized as follows. The background information for this work is presented in Section 2. SG and TG, along with the proposed algorithms, are discussed in Section 3. Case study and extraction of microservices along with the construction of SGs are discussed in Section 4. The evaluation of extracted microservices is discussed in Section 5, and Section 6 concludes the article.

2 | BACKGROUND

In this section, the challenges for migrating the applications, issues identified during migration process, existing migration approaches, and the problem description are discussed in this section.

2.1 | Challenges in migration

A major challenge in migration is identifying the appropriate partition of the system into microservices.[28] An overview of the lessons learned and challenges while migration to microservices is discussed by da Silva Filho et al.[29] Few challenges include decoupling of services, effort estimation for migration, identification of service boundaries, and the effort to analyze every part of the system and decide what should be converted to microservice. Multi-tenancy, statefulness, and data consistency are a few other challenges of microservices migration.[30] However, a feedback study conducted by Henry A et al. identified that more than 50% of the responses state that finding the right way to break the legacy applications is the major difficulty, and 49% of the responses state that the complex task during migration is to overcome tight coupling.[17] Therefore, in this article, we strive to propose a solution for extracting the candidate microservices from existing legacy service-oriented applications.

2.2 | Existing microservice extraction approaches

Considering the major challenge of extracting microservices, some researchers have contributed to the solution in the past few years. Tyszberowicz et al.[28] have proposed an approach to identify microservices using functional decomposition. This method is not for extracting microservices from existing applications; rather, it is for designing microservices from requirement specifications. Similar work for the extraction of candidate microservices from application code using a clustering algorithm is proposed by Kamimura et al.[31] The relation between extracted candidates and the whole structure of the software is also visualized. However, the proposed approach can be used just to analyze the system before actual migration. An exploratory study conducted by Carvalho et al.[32] finds that customization and variability are needed after the extraction of microservices from legacy systems. As our approach extracts microservices from SOA based systems, they can be used directly without any customization of the services. Gysel et al.[33] have proposed an approach called service cutter for service decomposition. In this approach, a tool that supports structured service decomposition through graph cutting is designed where internal structure is decomposed based on coupling criteria. Also, the user has to provide the software artifacts as input to extract services. Mazlami et al.[34] have proposed a clustering algorithm to extract the microservices from monolithic applications. This approach considers classes as an atomic unit of computation, and not all monolithic will be based on classes. Baresi et al.[35] have proposed a technique for microservices identification through interface analysis. In this approach, decomposition is based on reference vocabulary and open API. However, this approach cannot suit service-based architectures as the decomposition of artifacts is based on vocabulary, which is not considered a best practice.[36] A functionality oriented microservice extraction method is proposed by Jin et al.,[37] which identifies the dependencies using the execution traces. There are many limitations in their method as it is not fully automated, and the coverage of test cases may not be accurate.

A similar set of works related to extraction of microservices have been carried out by few researchers. A graph based clustering approach to refactor the monolith application is proposed by Desai et al.[38] The pure monolith application is formed as a cluster of services based on the classes defined. Their approach is purely applicable to object oriented system and it is not automated. In References 39 and 40, Kalia et al. have proposed an AI based toolchain called Mono2Micro which extracts the microservices from monolith application based on the business use case similarity by generating group of classes. In their approach, the user has to manually define the use cases and the proposed approach is limited only to web based applications. Another similar work carried out by Jin et al.[41] which is also applicable only for pure object oriented systems. It is based on the execution traces and it is difficult to judge for microservices based applications. However, all the above approaches have proposed for extraction of microservices from object oriented based monolithic applications. The primary focus of this work is to extract microservices from SOA based applications.

A recent study conducted by Ponce et al.[42] states that 90% of the proposed techniques use the design element as input and applicable only for object-oriented software. All the aforementioned approaches directly or indirectly depend on user inputs and are either manual or semi-automatic approaches. Also, the above approaches discussed focus mostly on the migration of monolithic applications to microservices architecture.

Few efforts have been contributed to overcome the challenges in SOA based applications. Martha and Lenglart[43] have proposed an approach for web services engineering named Web Service Development Life Cycle where each web service can be developed independently from other services in the enterprise. It is merely a streamlining approach for the existing application but not migration to microservices. This approach reduces the problems related to the performance and availability of web services and has few limitations. *Verb-based* and *noun-based* decomposition techniques are used to partition the web services from a large service in the enterprise. It is not always possible to divide the services uniformly, as mentioned in the approach. As microservices are said to be SOA done well, migrating existing applications to microservices architecture is the best solution.

2.3 | Problem description

With the dependency on ESB and the use of heavyweight protocols for the exchange of messages, SOA applications become less scalable, and the complexity increases with the increase in change requirements. SOA is still seen as monolithic from a deployment perspective.[44] The ability of independent service deployment and elastic scalability in microservices makes SOA applications as legacy.[45] Also, as mentioned in Section 1, many design challenges exist with the implementation of SOA using web services. To the best of our knowledge, very few works have been proposed in the literature to migrate SOA based web services to microservices. One such approach is proposed by Tusjunt et al.[46] to migrate web services based on business capabilities, and scenario base analysis. It identifies vocabularies and their relationships and generates the microservices. The process of identifying microservices through vocabularies is not considered as best practice. However, the proposed approach is specific to online shopping domain and cannot be applied to other domains, and identifying a complete set of vocabulary is difficult. Therefore, we consider the migration of SOA based applications to microservices architecture in this work.

3 | GRAPH-BASED APPROACH

Graph theory has been widely used in solving many complex problems in software engineering as the flow of messages and dependency between the software components can be graphically represented. As services are the software components in SOA based applications, we develop a new graph called SG to extract the candidate microservices. We start by introducing the concept of SG, which plays a fundamental role in our proposed approach.

3.1 | Service graph

SG is a regular graph generated for the visual analysis of communication and dependency between the services of an SOA application. The generalized form of any given SOA based application as a SG is shown in Figure 1.

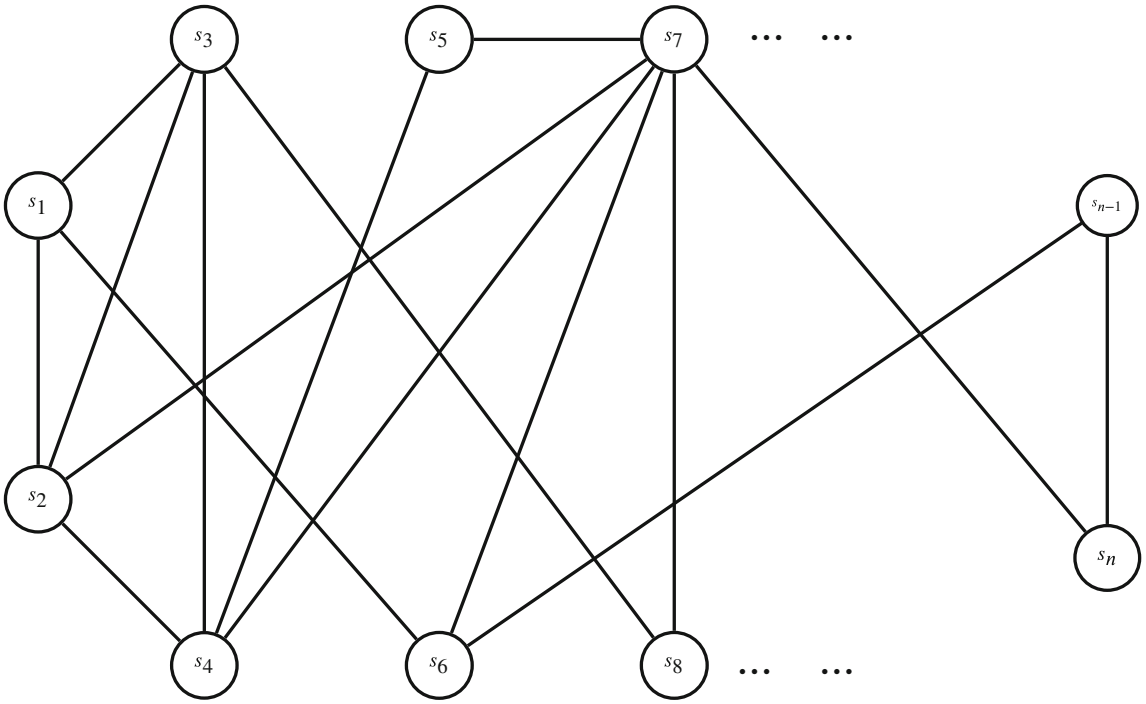

FIGURE 1 Formal representation of SOA application

3.1.1 | Service definition

Let a graph $G(V, E)$ be a SG with n nodes, where the nodes of the graph represent a set of services in the application, and edges between the nodes represent the interactions or dependency each service has with other services in the application. Let $V = \{s_1, s_2, s_3, \dots\}$ be the nodes of the SG where $s_1, s_2, s_3, \dots$ are services and $E = \{(s_1, s_2), (s_1, s_3), (s_2, s_4), \dots\}$ be the edges between the nodes which represent the dependency between the services. A service can be represented as a set of coordinating and interacting processes as defined in Equation (1).

$$s_i = \langle P_1^i, P_2^i, P_3^i, \dots, P_n^i, \Lambda \rangle, \tag{1}$$

where s_i is the logical service instance, P_k^i indicates kth process implementing logical service functionality f_i through the programmatic interface I_i, and Λ represents network communication function between individual processes.[47] SGs can be used in many software engineering activities like effort estimation, fault detection, and monitoring of the services.

3.1.2 | SG construction

Given an SOA application, we need to construct the SG representation which helps in the extraction of the microservices. Every SOA based application has an API document that contains the complete information about the services, operations in each service, and input/output parameters of each operation. If the application is implemented using web services, then we will have a WSDL file as the API document and if we implement the application as normal services, then we have XML format of the complete application. The inputs for each operation can be either initial inputs for the business request or output of other services. Hence, the input and output sets extracted for each operation are equivalent to each other. The input and output parameters can be of any datatype. Operations in each service are termed as a process. Using this API input, we present Algorithms 1 and 2 to construct the SG for the SOA application including the TGs.

Algorithm 1. Service_Graph_Construction_for_SOA

Require: API (WSDL) file of SOA based application
Ensure: Service Graph $G = (V, E)$

Begin
Read API file
Convert API to XML format
Parse the generated XML
Extract serviceNames from parsed file
▷ Each serviceName represents a service
V=$\{s_1, s_2, \cdots s_n\}$
where $s_i \leftarrow$ service and n $\leftarrow$ number of services
for each service s_i, i $\leftarrow 1, n$ **do**
    Extract inputs and outputs of each service s_i
end for
s_k.input $\leftarrow$ input set of service s_k
s_k.output $\leftarrow$ output set of service s_k
Let s_i.input $\sim$ s_j.output is considered as equivalence between two services s_i and s_j
for $i \leftarrow 1, n$ **do**
    for $j \leftarrow 1, n$ **do**
        if i $\neq$ j and s_i.output $\cap$ s_j.input $\neq \emptyset$ **then**
            E = E $\cup$ $\{(s_i, s_j)\}$
            ▷ Add edge from s_i to s_j in G
        end if
    end for
end for
for each service $s_i \in V$ **do**
    Call Task_Graph_Construction(s_i)
end for
return G
End

We need to use parsing tools to parse the API files. In Algorithm 1, we can skip the first two steps if the application is not built using web services. For the normal implementation of SOA as services, XML file can be directly generated and we need not convert it again to XML format. The extracted inputs and outputs from the API are equivalent to each other as the elements which act as input for one service are the outputs of its previous services and vice versa. Hence, the input and output sets are equivalent to each other. From the SG, we can observe that services are connected with edges, which represent the communication/dependency between the services. The output parameters of a particular service are given as input to the connected service. In the proposed algorithm, if s_i and s_j represent two input and output sets of two services, they both are said to be connected or communication exists between both s_i and s_j if at least one output element of s_i exists in input set of s_j. In the SG, we are adding undirected path between the services as the entry point can be from any services in the application based on the business requirements. As each service is a set of processes, the dependency among the processes inside each service is represented as TG discussed in next section.

3.2 | Task graph

TG is a directed acyclic graph (DAG) where each node represents the process and an edge between nodes represents the dependency of one node on another. Each service in SOA may contain one or more processes performing different tasks (based on the definition of service from Equation 1) and therefore we generate TG for processes in each service. The TG

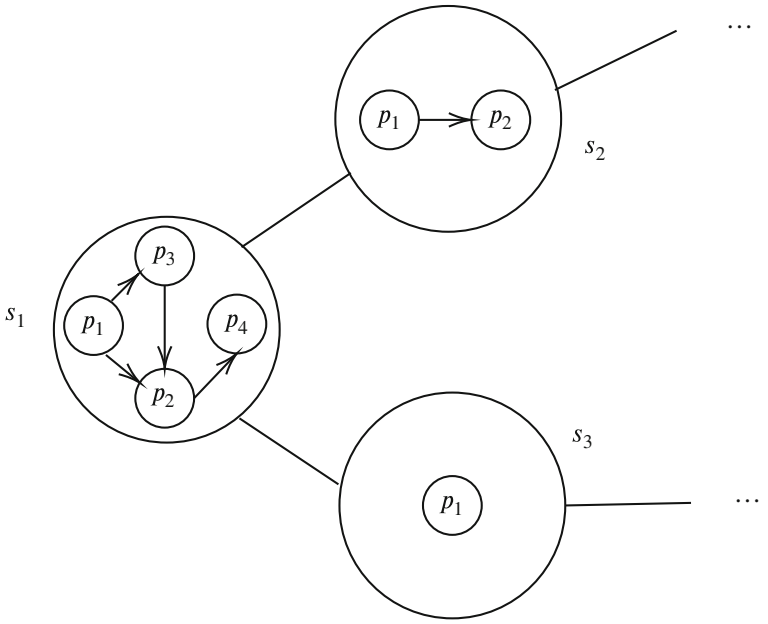

FIGURE 2 Service graphs containing task graphs

represents the application with a DAG $G(V, E)$ where V is the set of nodes, each node representing a process and E is the set of arcs between communicating processes. SG with the TGs inside each SOA service is represented as shown in Figure 2 where $s_1, s_2, s_3, \ldots$ are services and $p_1, p_2, p_3, \ldots$ are processes inside each service.

As per the design concepts of SOA and the proposed SG, whenever there is a dependency between processes across the services, the communication or dependency is expressed as the dependency between the services which hold such processes. The dependency is generated with the help of inputs and outputs and such parameters are passed between the services. Hence, the dependency still exists between the services only.

3.2.1 | TG construction

The processes in each service as represented as a TG as shown in Figure 2. Each service which is built using the concepts of SOA will have API file representing the operations performed in the service. We may have WSDL file or XML file based on the design approach of the application. For generating the SG, complete API of the application is used whereas to generate the TG for a single service, we consider the API of the particular service only. The API document comprises of the set of operations along with the input and output parameters involved in the operation. Services in SOA do not have constraint like microservices to perform only one business task. Therefore, they can be multiple operations in each service and we represent the processes as TG using Algorithm 2. Similar to the approach in Algorithm 1, the input and output elements of each service are equivalent. In TG, the processes execute the business requests in some particular order. Hence, we have represented the edges as directed paths.

3.3 | Microservices extraction algorithm

Now, we extract the microservices from the SG using Algorithm 3 and it generates a set of microservices as the output. The Set_m in the algorithm indicates the set of candidate microservices.

As per the definition of microservices architecture every service should perform only one operation/task. In our proposed approach, if the number of processes/tasks in SOA services is equal to one, it can be considered as a microservice since it is performing only one task. So in Set_m, at Step 7, we are adding such services which can be directly used as microservices (represented as services s_j). In Step 11, if the SOA services have more than one processes/task, we consider each process as a microservice.

Algorithm 2. Task_Graph_Construction(s_i)

Require: API (WSDL) file of a SOA service
Ensure: Task graph G_t=(V_t, E_t)
1: **Begin**
2: Read API file
3: Convert API to XML format
4: Parse the generated XML
5: Extract operations from parsed file
6: ▷ Operation is designed as a process p_i
7: $V_t = \{p_1, p_2, \cdots, p_m\}$
8: where $p_i \leftarrow$ process and m $\leftarrow$ number of processes
9: **for each** process p_i, i $\leftarrow 1, m$ **do**
10: Extract inputs and outputs of each process p_i
11: **end for**
12: p_k.input $\leftarrow$ input set of service p_k
13: p_k.output $\leftarrow$ the output set of service p_k
14: Let p_i.input $\sim p_j$.output is considered as equivalence between two processes p_i and p_j
15: **for** $i \leftarrow 1, m$ **do**
16: **for** $j \leftarrow 1, m$ **do**
17: **if** i $\neq$ j and p_i.output $\cap$ p_j.input $\neq \emptyset$ **then**
18: $E_t = E_t \cup \{(p_i, p_j)\}$ ▷ Add edge from p_i to p_j in G_t
19: **end if**
20: **end for**
21: **end for**
22: **return** G_t
23: **End**

3.4 | SG generation for microservices

The nodes in the SG represents the services and edges between the services represent the interactions each services has with other services to complete the business task. After extracting the services, in our approach, we retain the interactions of each services by constructing the SG for microservices based application also. It helps the developers to easily migrate the application and reduces the effort required in migration process (Algorithm 4).

Thus, the generated graph $G' = (V', E')$ represents the SG for the microservices application where V' represent the services in the microservices application and E' represents the dependency among different microservices. The generation of SG and extraction of microservices using the proposed approach is demonstrated through a case study application.

The proposed algorithms can be useful in the easy migration of an SOA based application to microservices architecture. The list of possible candidate microservices are generated from the SG of microservices. The communication between the microservices is also represented in the SG. Since both SOA and microservices architecture are service based, the same service artifact can be used in microservices as well. Once the microservices are designed, the only change required is the process of deployment. Cloud containers are used for deployment of the microservices.

4 | CASE STUDY APPLICATION

We applied our approach to a web-based application to demonstrate the extraction of microservices. We chose the application "vehicle management system (VMS)" as it is designed based on SOA.[48] The VMS application is used by the customers to select, customize, and purchase vehicles along with its parts through a web interface. The application includes eight services which include *config service, part service, product service, compare service, incentives and pricing service, dealer*

Algorithm 3. Microservices_Extraction

Require: Service graph $G = (V, E)$
Ensure: Set of candidate microservices Set_m
1: **Begin**
2: $V = \{s_1, s_2, \cdots, s_n\}$ where s_i is a task graph.
3: **for** $i \leftarrow 1, n$ **do**
4: visit the node $s_i \in$ V, if not visited before
5: calculate the order of node s_i
6: **if** $\mid s_i \mid = 1$ **then**
7: $Set_m = Set_m \cup s_i$
8: **else**
9: **for** $j \leftarrow 1, m$ **do**
10: i. visit node p_j where $p_j \in s_i$
11: ii. $Set_m = Set_m \cup p_j$
12: **end for**
13: **end if**
14: **end for**
15: **return** Set_m
16: **End**

Algorithm 4. Service_Graph_for_Microservices

Require: Service graph of the SOA application, $G = (V, E)$ with $V = \{s_1, s_2, \ldots, s_n\}$ where $s_1, s_2, \ldots, s_n$ are task graphs.
Ensure: Service graph of microservices application, $G' = (V', E')$
1: **Begin**
2: Let the given service graph vertex $s_i = (V_i, E_i)$ is a directed acyclic graph (DAG) of order k_i where, $V_i = \{p_i^1, p_i^2, p_i^3, \ldots p_i^{k_i}\}$.
3: Compute $V' = \cup_{i=1}^{n} V_i$
4: Compute $E' = \{(p_i^s, p_j^t) : p_i^s \in V_i, p_j^t \in V_j, (s_i, s_j) \in E, 1 \leq s \leq k_i, 1 \leq t \leq k_j\} \cup E_1 \cup E_2 \ldots \cup E_n$.
5: **return** G'
6: **End**

and inventory service, lead service, and user interface client. We implemented the application using TIBCO Business Works software and designed them as web services. All these eight services in the application are RESTful and stateless. We designed the SG of the application by using Algorithms 1 and 2. The SG along with its internal TGs of the VMS application is illustrated in Figure 3.

4.1 | Extraction of microservices

Applying Algorithm 3, we now extract the set of candidate microservices from the constructed SG as shown in Figure 3. Let the $Set_m = \emptyset$ initially. We need to visit each node of the graph and calculate the order of TG in that particular node. Therefore, the order of each service in graph G is as given below.

$$|s_1| = 1, |s_2| = 1, |s_3| = 1, |s_4| = 1, |s_5| = 2, |s_6| = 3, |s_7| = 2, |s_8| = 1.$$

For services s_5, s_6, s_7, we need to visit each node of the TG and add it to the Set_m. Since the remaining services are having only one task, the services are represented as processes for better understanding. Therefore, the final set of candidate microservices is given as $Set_m = \{p_1^1, p_2^1, p_3^1, p_4^1, p_5^1, p_5^2, p_6^1, p_6^2, p_6^3, p_7^1, p_7^2, p_8^1\}$.

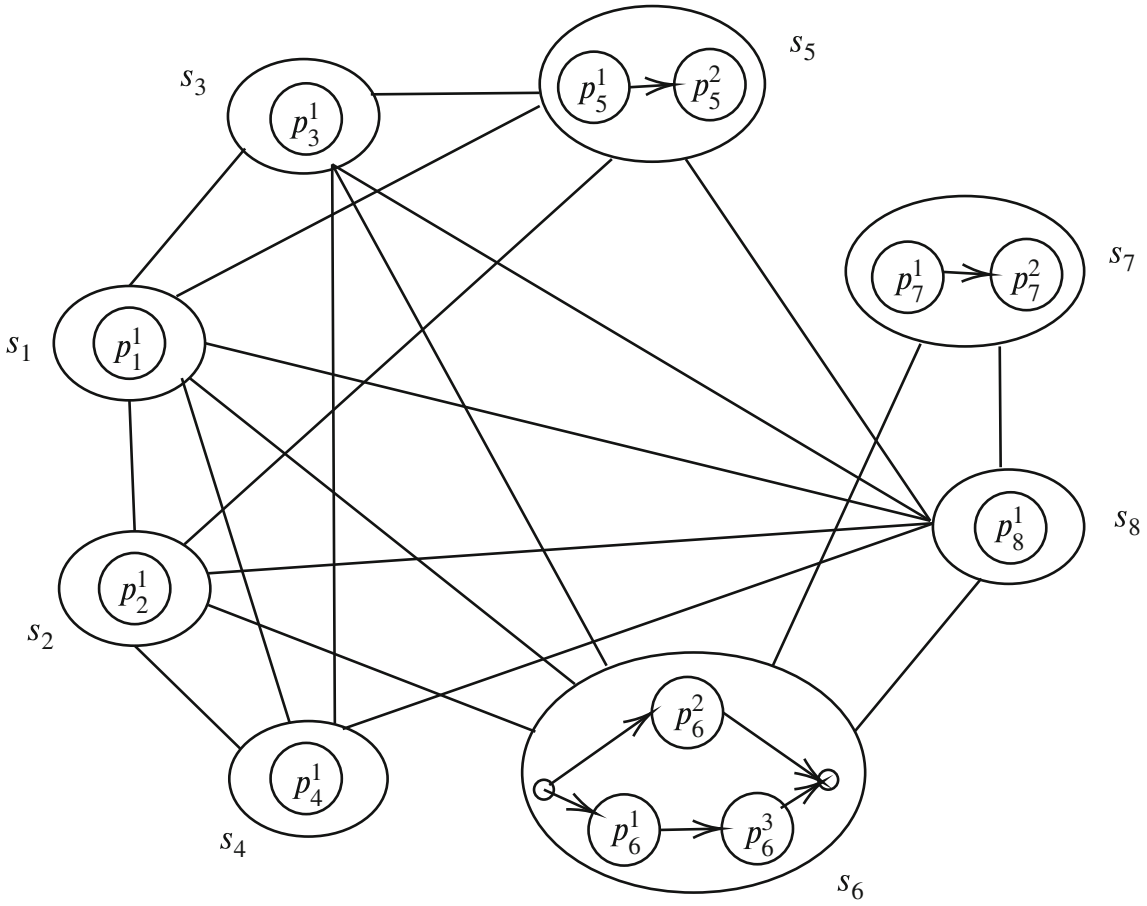

FIGURE 3 Service graph representation of SOA based vehicle management system

4.2 | Construction of SG

The proposed approach to extract microservices described in this section is based on the eight services in VMS application. We construct the SG of the application using the approach discussed in Section 3.1. Let us consider the SG of SOA application as shown in Figure 3 as input graph $G = (V, E)$. As there are eight services in the given SG, it represents the eight vertices of the graph. From the SG, $V = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$.

From the API of the application, the inputs and outputs of each service are extracted to form the edges between the services. We consider the inputs of one service is equivalent to the output of another service if the input and output data format (such as XML or JSON) is identical. The data format of each service may or may not be identical and as per the business requirements and processing of the business request, it must be formatted to a single form. Hence, the input and output of each service are considered equivalent. For example, the output data format of a particular service in MuleSoft is transformed implicitly to match with the input data format of another service using DataWeave. Similarly, in TIBCO business works, the data format needs to be explicitly transformed. Therefore, the inputs and outputs of any service are considered equivalent in Algorithms 1 and 2. The edge set is constructed using the proposed algorithm.

$$E = \{(s_1, s_2), (s_1, s_3), (s_1, s_4), (s_1, s_5), (s_1, s_6), (s_1, s_8), (s_2, s_4), (s_2, s_5), (s_2, s_6), (s_2, s_8), (s_3, s_4), (s_3, s_5), (s_3, s_6), (s_3, s_8), (s_4, s_8), (s_5, s_8), (s_6, s_7), (s_6, s_8), (s_7, s_8)\},$$

where $s_1, s_2, \ldots, s_8$ are the services of the application which internally consists of DAGs. Now we represent vertices and edges of DAG in each service. The first service G_1 is represented as $s_1 = (V_1, E_1)$ where $V_1 = \{p_1^1\}$ as it consists of only one

process in the service s_1 and $E_1 = \emptyset$ as it has no dependency with other processes. Similarly, other services are represented as below.

$$
\begin{aligned}
s_2 = (V_2, E_2) &: \text{ where } V_2 = \{p_2^1\} \text{ and } E_2 = \emptyset,\\
s_3 = (V_3, E_3) &: \text{ where } V_3 = \{p_3^1\} \text{ and } E_3 = \emptyset,\\
s_4 = (V_4, E_4) &: \text{ where } V_4 = \{p_4^1\} \text{ and } E_4 = \emptyset,\\
s_5 = (V_5, E_5) &: \text{ where } V_5 = \{p_5^1, p_5^2\} \text{ and } E_5 = \{(p_5^1, p_5^2)\}\\
s_6 = (V_6, E_6) &: \text{ where } V_6 = \{p_6^1, p_6^2, p_6^3\} \text{ and } E_6 = \{(p_6^1, p_6^3)\}\\
s_7 = (V_7, E_7) &: \text{ where } V_7 = \{p_7^1, p_7^2\} \text{ and } E_7 = \{(p_7^1, p_7^2)\}\\
s_8 = (V_8, E_8) &: \text{ where } V_8 = \{p_8^1\} \text{ and } E_8 = \emptyset.
\end{aligned}
$$

We need to determine the target output graph $G' = (V', E')$ which represents the SG of microservices application.

As $V' = \cup_{i=1}^{8} V_i$,

$$V' = \{p_1^1, p_2^1, p_3^1, p_4^1, p_5^1, p_5^2, p_6^1, p_6^2, p_6^3, p_7^1, p_7^2, p_8^1\}.$$

We compute edge set E' by considering the edges in the input graph.

$$
\begin{aligned}
E' = \{&(p_1^1, p_2^1), (p_1^1, p_3^1), (p_1^1, p_4^1), (p_1^1, p_5^1), (p_1^1, p_5^2), (p_1^1, p_6^1), (p_1^1, p_6^2), (p_1^1, p_6^3), (p_1^1, p_7^1), (p_1^1, p_8^1), (p_2^1, p_4^1), (p_2^1, p_5^1), (p_2^1, p_5^2), (p_2^1, p_7^1),\\
&(p_2^1, p_8^1), (p_3^1, p_4^1), (p_3^1, p_5^1), (p_3^1, p_5^2), (p_3^1, p_7^1), (p_3^1, p_8^1), (p_4^1, p_7^1), (p_4^1, p_8^1), (p_5^1, p_5^2), (p_5^1, p_8^1), (p_5^2, p_7^1), (p_5^2, p_8^1), (p_6^1, p_6^3), (p_6^1, p_7^1),\\
&(p_6^1, p_7^2), (p_6^1, p_8^1), (p_6^2, p_7^2), (p_6^2, p_8^1), (p_6^3, p_7^1), (p_6^3, p_8^1), (p_7^1, p_8^1), (p_7^2, p_8^1)\}.
\end{aligned}
$$

Thus the SG for microservices is represented using the graph $G' = (V', E')$. For better understanding of the application in generated graph G', nodes are renamed as services $ms_1, ms_2, ms_3, \ldots, ms_{12}$. We have generated the SG for microservices application using the V' and E' as shown in Figure 4.

4.3 | Discussion on application of proposed approach on case study application

The SG constructed with the proposed approach represents the microservices and their dependencies with other services in the application. Microservices architecture follows the principle of single responsibility, where each service should accomplish only one business task. The VMS application has few services which perform only a single task, and few services are loaded with multiple business requirements. We applied the proposed approach, partitioned the services, and generated individual microservices that perform only a single task. The benefit of our approach is communication between the services in the chosen SOA application hold intact in the SG of microservices application. We also get to know the services which have to be redesigned as microservices as few existing services are performing only one task. Therefore, it helps the developer in easy migration and saves time for migration. The number of services (NoS) can be identified from the SG of microservices. It helps in doing other software engineering activities like effort and cost estimation for migration of the application.

Apart from the benefits of our approach, it has a few limitations in the extraction of microservices. We assume that each process of the TG has one only operation. There can be processes that perform multiple operations. We have taken a simple application to demonstrate our approach and representation of large enterprise applications as SGs may be complex. To overcome this, architects should carefully analyze the system and generate the SG automatically using graph generation tools. The scalability of the proposed approach depends on the implementation of the approach. The proposed approach can handle large applications as well but the generation of such large graphs is a major challenge. In the future, we plan to consider the database also for partition as it is suggested to have an individual database for each microservice.

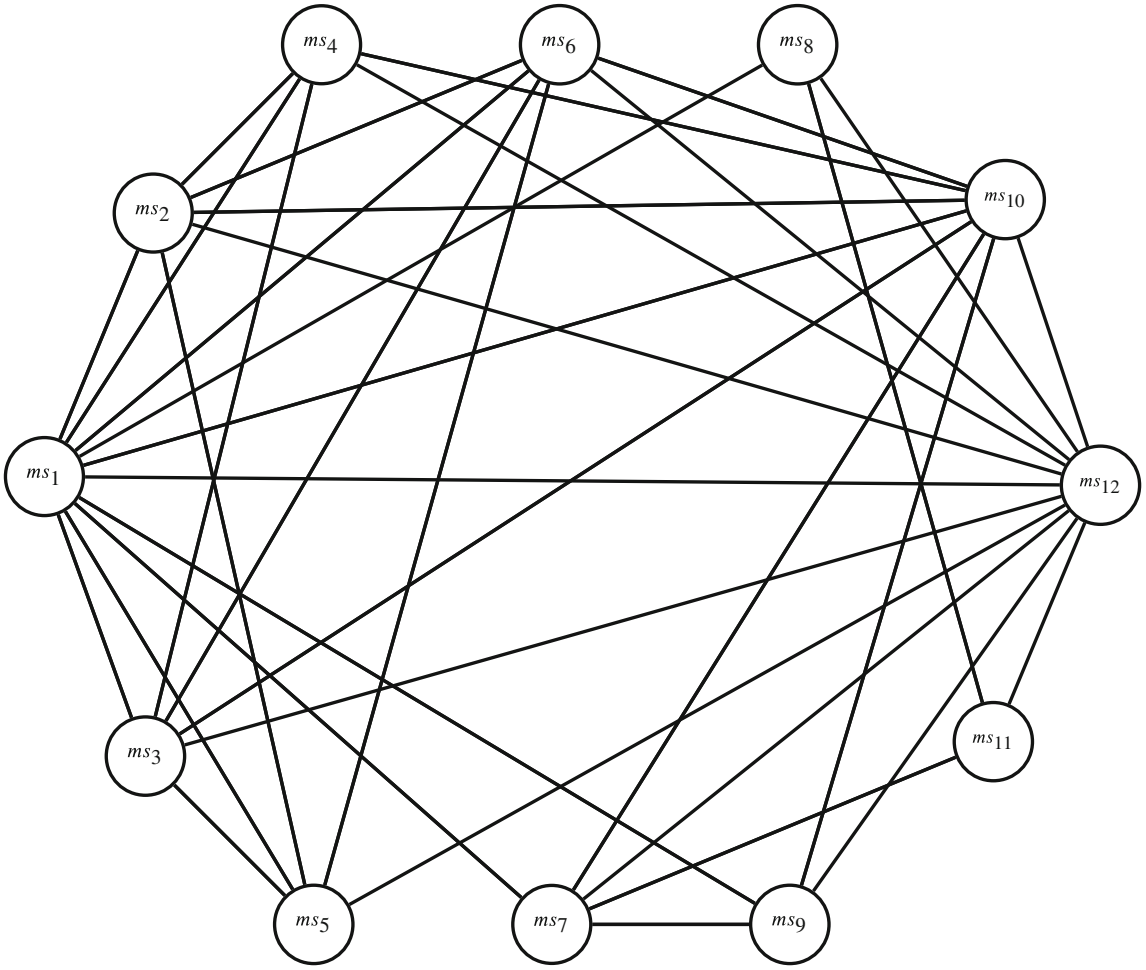

FIGURE 4 Service graph representation of microservices based web application

5 | EVALUATION OF THE EXTRACTED MICROSERVICES

One of the main reasons for migrating toward microservices is that it exhibits better QoS compared to SOA based services. However, to the best of our knowledge, very few works have been done in comparing both SOA based services and microservices. In one of our earlier works, we have compared the services of both the architectures with respect to performance, complexity and scalability.[49] The results conclude that though the complexity of microservices is higher, it has better response time and throughput compared to SOA based services.

5.1 | Evaluation criteria

In addition to the performance and complexity parameters, in this article, we consider loose coupling as evaluation criteria for comparing both microservices and SOA based services. Loose coupling is one of the essential characteristics of the service-oriented design. Coupling is measured by the level of dependency each service has on other services in the system. Coupling between the services should be minimal such that it holds the standards of service design principles. If the coupling between the services increases, the complexity of the architecture also increases. Hence coupling is a crucial factor among all the principles of SOA. Other SOA principles including statelessness, scalability, and so forth, are directly or indirectly related to loose coupling which are discussed below. The relation between coupling and other principles is represented in Figure 5.

- Nature of loose coupling minimizes cross-service dependencies, by this service autonomy is achieved.[50]

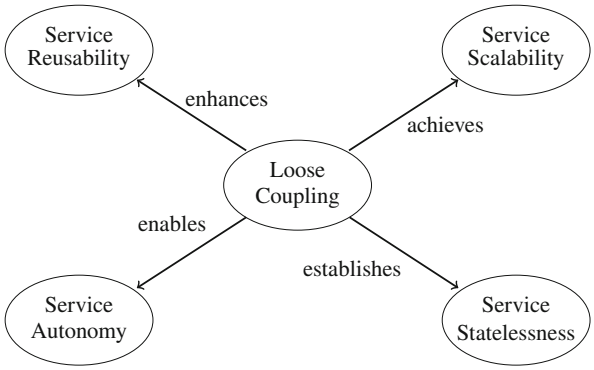

FIGURE 5 Relation between coupling and other SOA principles

- Loose coupling frees the tight dependencies on other components. This increases their availability for reuse opportunities.[51,52]
- Service statelessness is achieved by designing loosely coupled services.[53]
- Scalability is achieved if the services are loosely coupled.[54]

5.2 | Extraction of metric values from SG

SG provides the details of basic metrics which are used to determine other metric values. NoS value is given by the count of nodes in the SG, NoS = n. Coupling of services (CS) value is given by the degree of each node as given in Equation (2).

$$\mathrm{CS}_i = \deg(s_i). \tag{2}$$

Relative coupling of services (RCS) denotes the degree of coupling in a particular service.[20] RCS of service is calculated using the formula in Equation (3).

$$\mathrm{RCS}[s] = \frac{\mathrm{CS}[s]}{\mathrm{NoS}}. \tag{3}$$

The complexity of the application can be measured with this metric. Coupling intensity of a service is directly proportional to the value of the RCS. The complexity of the service-oriented system is also indicated with another metric, service coupling factor (SCF). It is calculated as given in Equation (4).

$$\mathrm{SCF} = \frac{\sum_{s \in S[*]} \mathrm{CS}[s]}{\mathrm{NoS}^2 - \mathrm{NoS}}. \tag{4}$$

SCF metric is used to indicate the overall coupling of the application. The value of SCF ranges between 0 and 1. Lower the SCF value, better is the system. Moreover, any service in a service-oriented system cannot have the values of SCF as 0 as each service should couple at least with one service in the application.

5.3 | Evaluation of SOA based application

Using the SG for SOA based application as given in Figure 3, metric values are calculated and presented in Table 1. The NoS in SOA based application is eight as we have eight nodes in the SG. Coupling value (CS) of services is the number of interactions each service has with other services, and RCS value depends on CS value.

TABLE 1 List of services with CS and RCS values of web services based application

Service #	Service name	Interacting services	CS value	RCS value
1.	Config service	2, 3, 4, 5, 6, 8	6	0.75
2.	Part service	1, 4, 5, 6, 8	5	0.62
3.	Product service	1, 4, 5, 6, 8	5	0.62
4.	Compare service	1, 2, 3, 8	4	0.5
5.	Incentives and pricing service	1, 2, 3, 8	4	0.5
6.	Dealer and inventory service	1, 2, 3, 7, 8	5	0.62
7.	Lead service	6, 8	2	0.25
8.	User interface client	1, 2, 3, 4, 5, 6, 7	7	0.87

TABLE 2 List of services with CS and RCS values of microservices based application

Service #	Service name	Interacting services	CS value	RCS value
1.	Config service	2, 3, 4, 5, 6, 7, 9, 10, 12	9	0.75
2.	Part service	1, 4, 5, 6, 10, 12	6	0.5
3.	Product service	1, 4, 5, 6, 10, 12	6	0.5
4.	Compare service	1, 2, 3, 10, 12	5	0.41
5.	Incentives service	1, 2, 3, 6, 12	5	0.41
6.	Pricing service	1, 2, 3, 5, 10, 12	6	0.5
7.	Dealer service	1, 9, 10, 11, 12	5	0.41
8.	Get-a-quote service	11, 12	2	0.16
9.	Dealer locator service	1, 7, 10, 12	4	0.33
10.	Inventory service	1, 2, 3, 4, 6, 7, 9, 12	8	0.67
11.	Lead processor service	7, 8, 12	3	0.25
12.	User interface client	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	11	0.91

For example, the CS value of *incentives and pricing service* is four as it has communication with *config service, part service, product service* and *user interface client* services. The corresponding RCS is calculated by $\frac{4}{8} = 0.5$, where 8 is the NoS value. Similarly, CS and RCS values are calculated for all services. Services are assigned a number from 1 to 8, coupled services corresponding to each service are also given with the assigned service numbers in Table 1. The interacting services field in the table indicates the services with which the given service has communication in the application.

5.4 | Evaluation of microservices based application

We calculate the CS and RCS values for the services of the application built using the microservices style. The NoS in this style is 12. Each service, its CS and RCS value are presented in Table 2.

5.5 | Results

We evaluated both the systems using the metrics related to coupling derived from SG. The results are presented in terms of coupling as it is an important principle of concentration in this work. The impact of coupling values on other principles are also discussed.

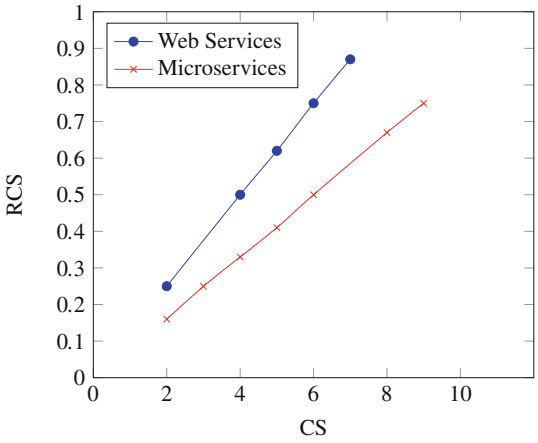

FIGURE 6 Comparison of coupling intensity

5.5.1 | Comparison based on RCS values

The RCS values of both the applications are calculated as shown in Tables 1 and 2. A graph is plotted with the values of CS and RCS values as shown in Figure 6 in which Coupling of Services (CS) values are represented in the X-axis and RCS values are represented in the Y-axis.

It is observed from the graph that microservices architecture has low RCS values. If the coupling values are less, then the architecture is good for designing enterprise applications.

5.5.2 | Comparison based on SCF

- The SCF value can be calculated using the NoS value and the total sum of CS values. From Table 1 of web services based application, the total of CS values is 38, and the NoS value is 8. Using the metric to calculate SCF value,

$$\text{SCF for web services} = \frac{38}{8^2 - 8} = 0.67.$$

- Similarly for microservices application, from Table 2, the sum of CS values is 70, and NoS value is 12.

$$\text{SCF for microservices style} = \frac{70}{12^2 - 12} = 0.53.$$

- Any system with less SCF score has low coupling between the services. The SCF score for web services based application is 0.67, and for microservices-based application, it is 0.53. Therefore, the overall SCF is better for microservices when comparing with web services. Microservices architecture is 20% less complex when comparing the SCF scores with web services architecture for the chosen web-based application.

5.6 | Discussion on comparison

From the above evaluation and comparison of extracted microservices with SOA services, it is clear that microservices has low coupling. Though the NoS is more in microservices application, the overall coupling intensity is less compared to SOA based application. It makes the developers to easily handle additional change requirements and the time taken to deploy the services is reduced. Moreover, as the microservices are extracted from SOA services, it becomes very easy to configure and deploy the extracted services. As already discussed, many features and QoS parameters are dependent on loose coupling, and from the results, it can be concluded that microservices has better QoS values compared to SOA services.

Apart from the chosen coupling metrics, few other metrics are also considered for the detailed evaluation of extracted microservices.

- Calls occurring between recommended partitions: Since the NoS has increased in the microservices based application, the service calls also increase. To satisfy a particular business request, it has to call multiple services to achieve the end response. It also adds to increase in complexity of the application.
- Entropy of business use cases: Microservices are very small unit of code which performs only one task. Since the microservices are extracted from existing SOA system without removing any existing business functionality, the entropy of the business use cases is less and it does not affects the business use cases.
- Size of microservices: There is no standard for measuring the size of each microservice. The size of the microservice can be measured only by verifying whether it performs only one business operation or not.

6 | CONCLUSION

To identify the candidate microservices from SOA applications, we proposed a new approach using the SG. In this approach, we use the SG representation of SOA based application along with the TG in each node of the SG. We presented algorithms for the construction of SG of a given SOA application, microservices extraction, and for constructing the SG of the microservices application, which is to be designed. We selected a VMS application and applied our proposed approach to extract the candidate microservices and generate the SG for microservices. The generated SG acts as a blueprint for the new application to be designed, and it helps in easy and fast migration to microservices. Additionally, the dependencies among the services are also represented in the SG of the microservices application. We have evaluated the extracted microservices w.r.t loose coupling and compared with existing SOA services. It is clear from the results that microservices has low coupling compared to SOA services. The proposed formal model of SG holds good for small applications, and we are yet to test with a large enterprise application. Comparison with other approaches is restricted as this is one of the first attempts to extract the microservices from monolith services of SOA. In the future, we will enhance our approach by considering the limitations, like considering the database for migration.

AUTHOR CONTRIBUTIONS

Vinay Raj has studied the literature and identified the idea. He has proposed the solution and constructed the algorithms. He has drafted the paper. Ravichandra Sadam has verified and evaluated the proposed solutions. The final draft of the paper is edited and corrected by Ravichandra Sadam.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no datasets were generated or analyzed during the current study.

ORCID


Vinay Raj https://orcid.org/0000-0002-4739-5893

REFERENCES

1. Salah T, Zemerly MJ, Yeun CY, Al-Qutayri M, Al-Hammadi Y. The evolution of distributed systems towards microservices architecture. Proceedings of the 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST); 2016:318-325.
2. Megargel A, Shankararaman V, Walker DK. Migrating from monoliths to cloud-based microservices: a banking industry example. In: Ramachandran M, Mahmood Z, eds. *Software Engineering in the Era of Cloud Computing*; Springer; 2020:85-108.
3. Fan CY, Ma SP. Migrating monolithic mobile application to microservice architecture: an experiment report. Proceedings of the 2017 IEEE International Conference on AI & Mobile Services (AIMS); 2017:109-112.
4. Raj V, Sadam R. Evaluation of SOA-based web services and microservices architecture using complexity metrics. *SN Comput Sci*. 2021;2(5):1-10.
5. Raj V, Sadam R. Patterns for migration of SOA based applications to microservices architecture. *J Web Eng*. 2021;20(5):1229-1246.
6. Raj V, Ravichandra S. Microservices: a perfect SOA based solution for enterprise applications compared to web services. Proceedings of the 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT): 2018:1531-1536; IEEE.
7. Christensen E, Curbera F, Meredith G, Weerawarana S. Web services description language (WSDL) 1.1; 2001.

8. Rodriguez G, Esteberena LF, Mateos C, Misra S. Reducing efforts in web services refactoring. Proceedings of the International Conference on Computational Science and Its Applications; 2019:544-559.
9. Baresi L, Garriga M. Microservices: the evolution and extinction of web services? In: Bucchiarone A, ed. *Microservices*. Springer; 2020:3-28.
10. Indrasiri K, Siriwardena P. Designing microservices. In: Indrasiri K, Siriwardena P, eds. *Microservices for the Enterprise*; Springer; 2018:19-61.
11. Xiao Z, Wijegunaratne I, Qiang X. Reflections on SOA and microservices. Proceedings of the 2016 4th International Conference on Enterprise Systems (ES); 2016:60-67.
12. Lyu Z, Wei H, Bai X, Lian C. Microservice-based architecture for an energy management system. *IEEE Syst J*. 2020.
13. Abdullah M, Iqbal W, Mahmood A, Bukhari F, Erradi A. Predictive autoscaling of microservices hosted in fog microdata center. *IEEE Syst J*. 2020.
14. Li S, Zhang H, Jia Z, et al. A dataflow-driven approach to identifying microservices from monolithic applications. *J Syst Softw*. 2019;157:110380.
15. Dmitry NMSS. On micro-services architecture. *Int J Open Inf Technol*. 2014;2(9):24–27.
16. Mazzara M, Bucchiarone A, Dragoni N, Rivera V. Size matters: microservices research and applications. In: Bucchiarone A, ed. *Microservices*; Springer; 2020:29-42.
17. Henry A, Ridene Y. Migrating to microservices. In: Bucchiarone A, ed. *Microservices: Science and Engineering*; Springer; 2019:45.
18. Pahl C, Jamshidi P. Microservices: a systematic mapping study. CLOSER; Vol. 1; 2016:137-146.
19. Taibi D, Lenarduzzi V, Pahl C. Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation. *IEEE Cloud Comput*. 2017;4(5):22-32.
20. Di Francesco P, Lago P, Malavolta I. Architecting with microservices: a systematic mapping study. *J Syst Softw*. 2019;150:77-97.
21. Di Francesco P, Malavolta I, Lago P. Research on architecting microservices: trends, focus, and potential for industrial adoption. Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA); 2017:21-30.
22. Lenarduzzi V, Lomio F, Saarimäki N, Taibi D. Does migrating a monolithic system to microservices decrease the technical debt? *J Syst Softw*. 2020;110710.
23. Balalaie A, Heydarnoori A, Jamshidi P, Tamburri DA, Lynn T. Microservices migration patterns. *Softw Pract Exper*. 2018;48(11):2019-2042.
24. De Iasio A, Zimeo E. A framework for microservices synchronization. *Softw Pract Exper*. 2021;51(1):25-45.
25. Freire AFA, Sampaio AF, Carvalho LHL, Medeiros O, Mendonça NC. Migrating production monolithic systems to microservices using aspect oriented programming. *Softw Pract Exper*. 2021;51(6):1280-1307.
26. Laso S, Berrocal J, García-Alonso J, Canal C, Manuel MJ. Human microservices: a framework for turning humans into service providers. *Softw Pract Exper*. 2021.
27. Filisbino Passini W, Aparecida Lana C, Pfeifer V, Affonso FJ. Design of frameworks for self-adaptive service-oriented applications: a systematic analysis. *Softw Pract Exper*. 2021.
28. Tyszberowicz S, Heinrich R, Liu B, Liu Z. Identifying microservices using functional decomposition. Proceedings of the International Symposium on Dependable Software Engineering: Theories, Tools, and Applications; 2018:50-65.
29. Silva Filho HC, Figueiredo Carneiro GD. Strategies reported in the literature to migrate to microservices based architecture. Proceedings of the 16th International Conference on Information Technology-New Generations (ITNG 2019); 2019:575-580.
30. Furda A, Fidge C, Zimmermann O, Kelly W, Barros A. Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency. *IEEE Softw*. 2017;35(3):63-72.
31. Kamimura M, Yano K, Hatano T, Matsuo A. Extracting candidates of microservices from monolithic application code. Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC); 2018:571-580.
32. Carvalho L, Garcia A, Assunção WK, Bonifácio R, Tizzei LP, Colanzi TE. Extraction of configurable and reusable microservices from legacy systems: an exploratory study. Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A; 2019:26-31.
33. Gysel M, Kölbener L, Giersche W, Zimmermann O. Service cutter: a systematic approach to service decomposition. Proceedings of the European Conference on Service-Oriented and Cloud Computing; 2016:185-200.
34. Mazlami G, Cito J, Leitner P. Extraction of microservices from monolithic software architectures. Proceedings of the 2017 IEEE International Conference on Web Services (ICWS); 2017:524-531.
35. Baresi L, Garriga M, De Renzis A. Microservices identification through interface analysis. Proceedings of the European Conference on Service-Oriented and Cloud Computing; 2017:19-33.
36. Bartsch C, Shwartz L, Ward C, Grabarnik G, Buco MJ. Decomposition of IT service processes and alternative service identification using ontologies. Proceedings of the NOMS 2008-2008 IEEE Network Operations and Management Symposium; 2008:714-717.
37. Jin W, Liu T, Zheng Q, Cui D, Cai Y. Functionality-oriented microservice extraction based on execution trace clustering. Proceedings of the 2018 IEEE International Conference on Web Services (ICWS); 2018:211-218.
38. Desai U, Bandyopadhyay S, Tamilselvam S. Graph neural network to dilute outliers for refactoring monolith application. Proceedings of the AAAI Conference on Artificial Intelligence; Vol. 35, 2021:72-80.
39. Kalia AK, Xiao J, Krishna R, Sinha S, Vukovic M, Banerjee D. Mono2Micro: a practical and effective tool for decomposing monolithic Java applications to microservices. Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering; 2021:1214-1224.

40. Kalia AK, Xiao J, Lin C, et al. Mono2Micro: an AI-based toolchain for evolving monolithic enterprise applications to a microservice architecture. Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering; 2020:1606-1610.
41. Jin W, Liu T, Cai Y, Kazman R, Mo R, Zheng Q. Service candidate identification from monolithic systems based on execution traces. *IEEE Trans Softw Eng*. 2019;47(5):987–1007.
42. Ponce F, Márquez G, Astudillo H. Migrating from monolithic architecture to microservices: a rapid review. Proceedings of the 2019 38th International Conference of the Chilean Computer Science Society (SCCC); 2019:1-7.
43. Martha VS, Lenglart M. Webservices engineering. In: Mohanty H, Pattnaik P, eds. *Webservices*; Springer; 2019:173-196.
44. Microservices EW. *Flexible Software Architectures*. CreateSpace Independent Publishing Platform; 2016.
45. Cerny T, Donahoo MJ, Trnka M. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Appl Comput Rev*. 2018;17(4):29-45.
46. Tusjunt M, Vatanawood W. Refactoring orchestrated web services into microservices using decomposition pattern. Proceedings of the 2018 IEEE 4th International Conference on Computer and Communications (ICCC); 2018:609-613.
47. Yanchuk A, Ivanyukovich A, Marchese M. Towards a mathematical foundation for service-oriented applications design. *J Softw*. 2006;1(1):32-39.
48. Bhallamudi P, Tilley S, Sinha A. Migrating a web-based application to a service-based system-an experience report. Proceedings of the 2009 11th IEEE International Symposium on Web Systems Evolution; 2009:71-74.
49. Raj V, Sadam R. Performance and complexity comparison of service oriented architecture and microservices architecture. *Int J Commun Netw Distrib Syst*. 2021;27(1):100-117.
50. Jammes F, Smit H. Service-oriented paradigms in industrial automation. *IEEE Trans Ind Inform*. 2005;1(1):62-70.
51. Washizaki H, Yamamoto H, Fukazawa Y. A metrics suite for measuring reusability of software components. Proceedings of the 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717); 2004:211-223.
52. Sindhgatta R, Sengupta B, Ponnalagu K. Measuring the quality of service oriented design. *Serv Orient Comput*. 2009;5900:485-499.
53. Pautasso C, Wilde E. Why is the web loosely coupled? A multi-faceted metric for service design. Proceedings of the 18th International Conference on World Wide Web; 2009:911-920.
54. Hasselbring W. Microservices for scalability: keynote talk abstract. Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering; 2016:133-134.

How to cite this article: Raj V, Ravichandra S. A service graph based extraction of microservices from monolith services of service-oriented architecture. *Softw Pract Exper*. 2022;52(7):1661-1678. doi: 10.1002/spe.3081