Patterns for Migration of SOA Based Applications to Microservices Architecture

Vinay Raj* and Ravichandra Sadam

National Institute of Technology Warangal, Telangana, India
E-mail: rvinay1@student.nitw.ac.in
**Corresponding Author*

Received 10 August 2020; Accepted 08 January 2021;
Publication 06 July 2021

Abstract

Service oriented architecture (SOA) has been widely used in the design of enterprise applications over the last two decades. Though SOA has become popular in the integration of multiple applications using the enterprise service bus, there are few challenges related to delivery, deployment, governance, and interoperability of services. To overcome the design and maintenance challenges in SOA, a new architecture of microservices has emerged with loose coupling, independent deployment, and scalability as its key features. With the advent of microservices, software architects have started to migrate legacy systems to microservice architecture. However, many challenges arise during the migration of SOA to microservices, including the decomposition of SOA to microservice, the testing of microservices designed using different programming languages, and the monitoring the microservices. In this paper, we aim to provide patterns for the most recurring problems highlighted in the literature i.e, the decomposition of SOA services, the size of each microservice, and the detection of anomalies in microservices. The suggested patterns are combined with our experience in the migration of SOA-based applications to the microservices architecture, and we have also used these patterns in the migration of other SOA applications. We evaluated these patterns with the help of a standard web-based application.

Journal of Web Engineering, Vol. 20_5, 1229–1246.
doi: 10.13052/jwe1540-9589.2051

Keywords: Distributed systems, service oriented architecture, microservices, migration, migration patterns.

1 Introduction

Distributed systems have become more popular over the last two decades as the demand for independent design and deployment of web applications has increased [1]. Today's world demands the timely delivery of business needs and all that needs to be online. Distributed systems play a key role in the accelerated delivery of services with continuous integration and continuous development. Service oriented architecture (SOA) is the architectural style of distributed applications with service as the main design component. SOA is primarily used for the integration of various components with the middleware feature using Enterprise Service Bus (ESB) [2]. It follows several design principles such as loose coupling, interoperability, statelessness, etc. SOA gained more popularity with its implementation mechanism called web services. SOA is the concept and web services are the implementation of the concept [3].

A web service is a software program that can be accessed and discovered based on web-based protocols such as HTTP and REST. Web services use XML (WSDL) to access information and share messages between different services. The typical web service architecture consists of three components, namely a service provider, a service requester, and a service registry, which maintains all the web services. Service provider and requester are web services where the latter requests for information and the former responds with the information. Nevertheless, the use of XML based WSDL documents makes the application complex, and security problems occur with these APIs. RESTful web services do have a range of problems, including reusability and composition issues. In addition, SOA applications are less scalable, as the components are highly dependent on each other. The use of ESB makes the services tightly coupled with middleware and ESB becomes a single point of failure. Also, SOA services are deployed as a single archive file and are often hosted on a single server. With the increase in user requirements, the same archive files are deployed several times in a limited period of time. Many SOA services are becoming increasingly monolithic in size with ever-changing business requirements [4]. Monolithic services make the application complex and difficult to manage. Hardware infrastructure costs also increase if the application needs to be scaled whenever the input load increases. In the recent past, a new design style for distributed applications

has emerged that overcomes all the challenges of service oriented architecture based applications.

Microservices is a new style of designing enterprise applications which is based on SOA principles with additional features. It is a way of designing applications where each component is designed using a lightweight protocol and deployed independently [5]. Microservices uses the REST communication protocol and the JSON data exchange format for the exchange of messages between services. It follows the concept of the Single Responsibility Principle (SRP) where only one business function should be performed by each service. Continuous Integration (CI) and Continuous Delivery (CD) are the two core principles of microservice architecture. Applications designed with microservices are loosely coupled, scalable, and designed independently. Microservices are well suited to the cloud environment as containers are used for deployment of the applications. The main advantage of using microservices over other architectural styles is that only the required service is deployed independently without having an impact on the other services of the application [6]. The use of containers renders the services auto-scalable. Each service has its own database and configuration environment for the processing of the business requests. Moreover, applications are migrating towards cloud [24] and because of the diverse benefits, companies have started migrating their existing legacy applications to microservices architecture. Netflix, Amazon, and Google have started developing their applications with this new style [7].

As SOA services are becoming complex and difficult to manage, it is important to migrate these applications to the microservices architecture. The migration of the legacy application to new architecture poses many design challenges as the entire systems need to be updated [8] [23]. Similarly, migration of SOA-based application to microservices also exhibits challenges, including identification of candidate microservices from legacy source code, setting up a configuration environment for newly developed microservices, testing of services built with different programming languages, integration of polyglot services, debugging, and monitoring of migrated services [9].

In software engineering, patterns are used to solve the commonly occurring problems that occur during the SDLC phases of the application [10]. Design patterns provide the best solutions for the challenges with practical applicability for all business requirement scenarios. Patterns provide a common vocabulary which helps in reducing the complexity of the application. They act as a framework for presenting tested solutions to issues that are suitable in any given context. Every pattern has few sections with which the

solution provided is understood. This includes the criteria, context, problem, solution, challenges, and the illustration or a diagram of the solution. In addition, migration patterns can also be used to support issues that occur during the migration from one architecture to another [11]. There are many gains from the exploitation of design patterns during the design phase of microservices. There are broader advantages of using migration patterns during migration, as it is the new architectural style [12].

There are very few or no design patterns defined for the problems in the design of microservices in the literature [13]. We consider the stated problems and present the patterns that help with the migration process to microservices. The detailed information on the problems, solutions, and the challenges for the problems are presented in Section 4. We use graph theory concepts to represent the applications as service graph along with an internal task graph. Using graph theory offers a simple and better approach to the migration challenges.

The remaining paper is structured as follows. Related works done to provide solutions to the defined problems are addressed in Section 2. The introduction of the service graph and task graph is discussed in Section 3. The proposed patterns are presented in Section 4 and the evaluation of the patterns is provided in Section 5. Section 6 concludes the paper.

2 Related Work

The use of design patterns in the migration of architectures is less explored. Few researchers have contributed the patterns related to the migration of monolithic applications in the literature which are discussed here. Lessons learned while migrating legacy monolithic applications using the patterns are reported in [14]. Several patterns for migration of monolithic applications to microservices and how each pattern benefits the migration process are addressed in [15]. The migration patterns needed for complete migration and its related challenges are presented in [16]. The patterns for rearchitecting the existing legacy applications are also discussed. The pattern for the decomposition of the monolithic application is provided, but we are considering patterns for migration and decomposition of monolithic SOA services. A new pattern called strangler pattern is introduced that adds new microservices to the monolithic code [17]. In the long run, the monolithic services will gradually be replaced with microservices.

Patterns for migration of SOA applications to microservices are proposed [18], but these patterns do not support the independent development

and deployment of services. Under these scenarios, ESB is still used for communication between services. Legacy software modernization using microservices is proposed with the help of patterns [19] however, the approach is not appropriate for migrating service oriented applications.

To the best of our knowledge, no research or very little work has been performed in the literature on the migration of SOA-based applications to microservices architecture. The definition of service graph along with the task graph needed for the extraction of microservices is discussed in the following section.

3 Preliminaries

We define a formal model called service graph which resembles any service based application. By considering the inputs and outputs from the APIs of the application, we create a service graph. We use this service graph to demonstrate the implementation of the patterns.

3.1 Service Graph

Service graph (SG) is a standard graph created for the visual analysis of communication and dependence between the services of an SOA application. The generalized form of any SOA-based application as a service graph is shown in Figure 1.

3.1.1 Service definition

Let a graph G(V,E) be a service graph 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 service graph 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).

$$S_i =< {P_1}^i, {P_2}^i, {P_3}^i, \cdots\cdots, {P_n}^i, \Lambda > \tag{1}$$

where S_i is the logical service instance, ${P_k}^i$ indicates k^{th} process implementing logical service functionality f_i through the programmatic interface I_i and Λ represents network communication function between individual processes [20]. As each service is a set of processes, the dependency among

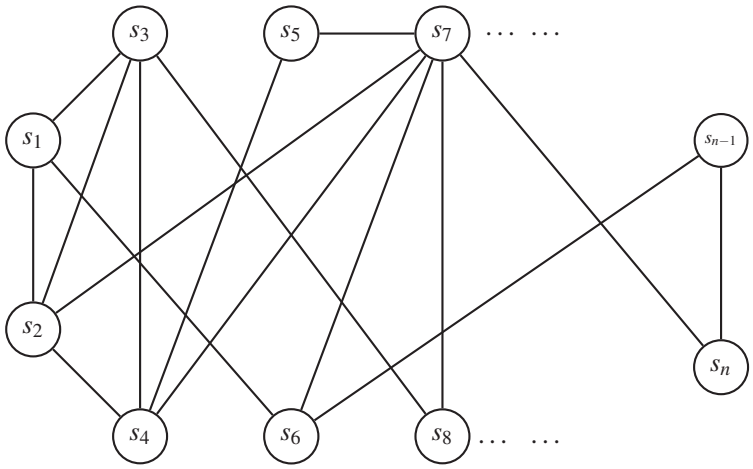

Figure 1 Formal Representation of SOA application.

the processes inside each service is represented as task graph discussed in next section.

3.2 Task Graph

Task graph is a directed acyclic graph where each node represents the process and edge between the node represents the dependency of the node on the other node. Each service in SOA may contain one or more processes performing various tasks and we therefore generate a task graph for the processes in each service. The task graph represents an application with a directed acyclic graph G(V, E) where V is a set of nodes, where each node represents a process and where E is a collection of arcs between the communicating processes. The service graph with the task graphs within each SOA service is represented as shown in Figure 2 where S_1,S_2,S_3,... are services and p_1,p_2,p_3,... are processes inside each service.

4 Migration Patterns

Migration to the microservice architecture has broader benefits when compared with SOA, including scalability and independent deployment. Since few SOA services appear to be monolithic in nature, it is difficult to manage and the complexity of the application increases. The need to move

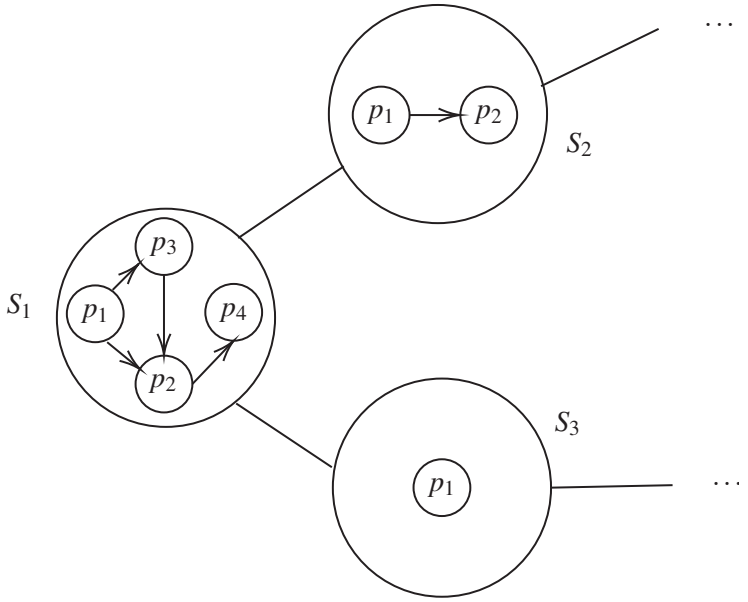

Figure 2 Service graphs containing task graphs.

to microservices has therefore gained attention in the recent past. Several problems emerge during the migration of SOA-based applications to microservices as defined in section 1. We, therefore, present patterns to help with easy migration to microservices.

4.1 Pattern 1: Decomposition of an SOA Service to Microservices

This pattern is used to address the recurring challenge faced by software architects while migrating legacy SOA based applications to microservices.

Requirement: There is a complex SOA based application and the software architects plan to migrate the application to the microservices architecture which will reduce the scalability issues and increase application performance. Also, the effort required for migration should be reduced. There are few monolithic services in the application and few basic services that can be called microservices. The presence of monolithic services makes the application complex and the best approach is to migrate the application to microservices.

Problem: How to split the monolithic services in the SOA based application to microservices? How to extract the microservices from the SOA services?

Solution: Graph theory has been widely used to solve many complex problems since it is easy to represent the components and their dependencies as a graph. Service graph, along with its internal task graphs, is a formal representation of every SOA application and it helps in identifying the monolithic services.

The architecture of microservices follows the single responsibility principle and states that each service should perform only one business function. SOA based applications can contain few services that perform only one task and these services may be regarded as microservices directly. This reduces the migration effort as we do not need to consider all the services for migration to microservices.

Further, identification of monolithic services or the services which need to be migrated to microservices should be considered for the extraction of microservices. Using graph properties, from the service graph, find out the order of each task graph and the service with the order as one can be directly considered as microservices and services with order more than one should be treated as monolithic services. Only such monolithic applications should be considered for the extraction of microservices.

Using the graph partition method [22], we can break the monolithic services into microservices and the independent task graph nodes represent the microservices. We use the service graph to identify complex services and generate microservices.

Challenges: Representing the SOA framework as a service graph is difficult for large enterprise applications, as it includes a large number of services and tools required for generating such large graphs is challenging.

4.2 Pattern 2: Size of Each Microservice

This pattern is used when designing the microservices or extracting from service oriented based applications.

Requirement: There is a complex SOA based application and the architects want to migrate the application to microservices architecture. During migration, the SOA services are partitioned to generate the microservices. Yet one of the main issues faced by software architects, microservices developers, and practitioners is what the size of each microservice will be. By definition, microservices are small, independent, and scalable services that

are deployed using cloud-based technologies. So, the question is how small each microservice should be.

Problem: What should be the size of each microservice? On what basis can we measure the size of the microservices?

Solution: SOA based applications have feature level services and microservices have task level services. Every service in the SOA application consists of many tasks that perform the business requirements. Microservices follow single responsibility principle that requires each service to perform only one business function. We, therefore, use the service graph representation of the SOA based application and consider each task in the task graph to be microservices.

In this case, the size of the microservice is not a measurable metric. When a specific service performs only one operation, it may be considered as a microservice and the size of the service is not considered.

Challenges: While microservices perform a single task, there can be few services that perform multiple computations to meet the business requirements. These microservices can increase the complexity of the application.

4.3 Pattern 3: Bug Detection in Complex Microservices Application

This pattern is used to monitor and find bugs after migrating SOA-based applications to the microservice architecture.

Requirement: An SOA based application is migrated to microservices architecture with the help of Pattern 1 and Pattern 2. When the number of services increases in microservices based application, it is difficult to detect the bug if it happens. Among those multiple microservices, it is difficult to trace the bug and recognize the service that causes the bug in the entire application.

Problem: How to trace the service which is responsible for the bug? How to identify the location of the bug among all available microservices in the application?

Solution: Business requirements are specified prior to the design phase of the software development life cycle. The workflow of the business requirements is defined through the use of UML diagrams in the software requirement specifications document. Nonetheless, despite the nature of the specification, it is difficult to establish a connection between the business requirements

and the services that execute the requirements. As a result, the service graph model allows us to map the criteria and the services that fulfill them.

Using the service graph representation, each workflow of the business requirements is mapped with the nodes in the graph where each node has a service number. We need to define the workflow of each requirement and store the sequence of services it passes to fulfill the business request. Each sequence of service numbers is stored in the database and anytime a particular service fails, we need to check for all the sequences in which the service is involved. If a particular business request has failed, we get the corresponding sequence of the business requirement and trace only in the services involved in the sequence.

Challenges: If the application is complex, it is difficult to create a service graph and store all the sequences in the database. Also, often a specific service can be involved in several sequences, so it is difficult to determine the appropriate sequence for a given bug.

5 Evaluation

We use a standard web-based application, Vehicle Management System (VMS) [21] built based on SOA principles to test the proposed patterns. This application is used to select, customize, and purchase vehicles and its parts using a web interface. This VMS web application is implemented and we created a service graph (SG_SOA) using the API documents as shown in Figure 3. There are eight services in the application and each service performs specific business tasks. Using the concept of task graph, the processes within each service are depicted as a task graph along with the edges between the processes. Using our earlier work of extracting microservices from SOA based applications [22], we have created the service graph for microservices as shown in Figure 4. The details of the services of both SOA and microservices based application along with the representations in service graphs are presented in Table 1. SG_SOA indicates the service graph of SOA based application and SG_MSA indicates the service graph of microservices based application. Both SG_SOA and SG_MSA are used in the validation of the proposed patterns discussed in the below sections.

5.1 Pattern 1

From the given SOA based application, we need to extract the microservices by decomposing the monolithic services of SOA. There are eight services in

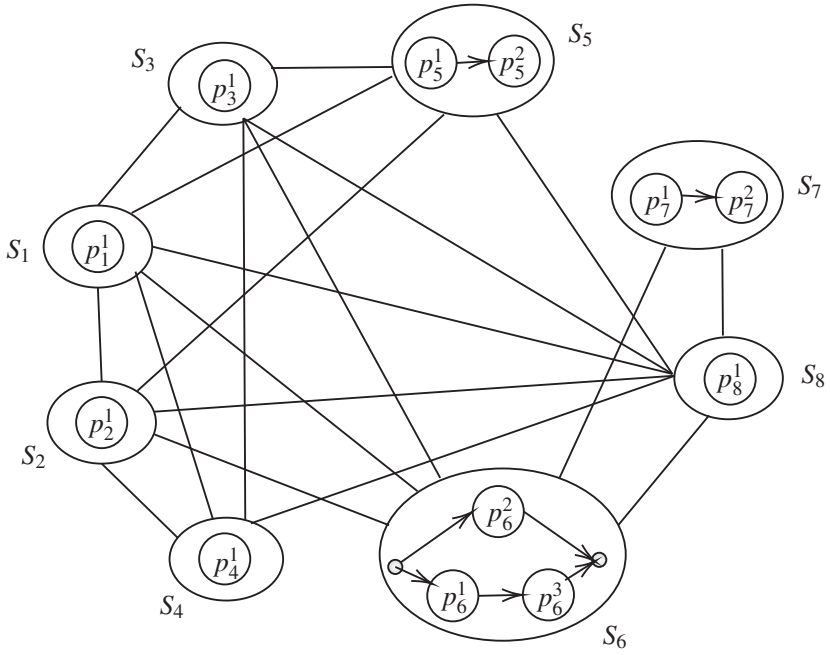

Figure 3 SG_SOA: Service graph representation of SOA based vehicle management system.

Table 1 Details of services of both SOA and Microservices based applications

Notation in SG_SOA	SOA Services	Microservices	Notation in SG_MSA
S_1	Config Service	Config Service	ms_1
S_2	Part Service	Part Service	ms_2
S_3	Product Service	Product Service	ms_3
S_4	Compare Service	Compare Service	ms_4
S_5	Incentives & Pricing Service	Incentives Service	ms_5
		Pricing Service	ms_6
S_6	Dealer & Inventory Service	Dealer Service	ms_7
		Dealer Locator Service	ms_8
		Inventory Service	ms_9
S_7	Lead service	Get-A-Quote Service	ms_{10}
		Lead Processor Service	ms_{11}
S_8	User Interface Client	User Interface Client	ms_{12}

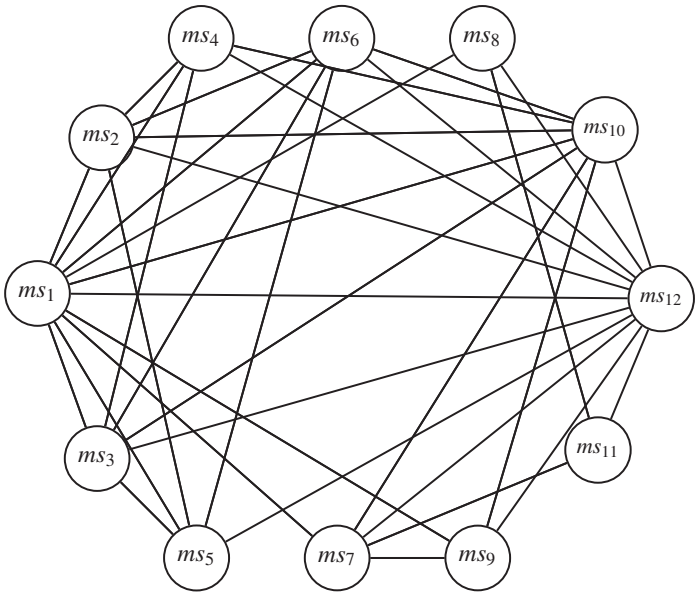

Figure 4 SG_MSA: Service graph representation of microservices based web application.

the application and they are named as S1, S2, S3, ... S8 in the service graph and each service has an internal task graph. Applying the pattern, the degree of each graph indicates the number of nodes in the graph. So, we need to calculate the order for each subgraph. Using the service graph information, order of S_1, $|S_1| = 1$, as we have only one node in the service S_1. Similarly, for all other services, $|S_2| = 1$, $|S_3| = 1$, $|S_4| = 1$, $|S_5| = 2$, $|S_6| = 3$, $|S_7| = 2$, and $|S_8| = 1$.

Now, we need to consider only the services with order greater than one, so we have three services S_5, S_6, and S_7 which should be migrated to microservices. Using the graph partition approach, we can break the monolithic services to generate the microservices. The service graph provides an overview of the services and their interaction with other services and helps in easy migration for software architects.

5.2 Pattern 2

The size of the microservices is immeasurable. Designing the service graph along with task graphs helps us in identifying the size of the microservice. By the definition of microservices, each service should perform only one business task and there is no specific metric to determine the size of the

Table 2 Mapping of business requests with workflows

Business Requests	Sequence of Services
BR1	$ms_{12} - ms_1 - ms_3 - ms_2 - ms_6 - ms_5 - ms_9$
BR2	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_4$
BR3	$ms_{12} - ms_9 - ms_3 - ms_2 - ms_6 - ms_5 - ms_4$
BR4	$ms_{12} - ms_8 - ms_9$
BR5	$ms_{12} - ms_9 - ms_{10} - ms_{11} - ms_8 - ms_7$
BR6	$ms_{12} - ms_1 - ms_7 - ms_8$
BR7	$ms_{12} - ms_1 - ms_5 - ms_6$

microservice. As a consequence, nodes in the task graph itself represent the microservices and each service performs only one task.

5.3 Pattern 3

In the complex application of microservices, if a service fails or a bug occurs, we need to trace the route cause of the failure. The applications designed using microservices generally have a complex network and it is difficult to trace and monitor the applications.

For the chosen application, the business requests should be stored in database. From the service graph given in Figure 4, we extract the sequence of the services in the order of execution and map with the business request number. We have studied the chosen application and few of the requests are considered. The possible business requests of the given application are stored as shown in Table 2.

From the logs and monitoring data, if any service fails, the root cause of the failure for the particular business request can be traced. For example, if the service ms_7 fails, it is involved in business requests 5 and 6. We, therefore, need to analyze the error with the business requests. Business request 5 is to get the quote, generate the lead and select the proper dealer for the vehicle and business request 6 is to configure the dealer along with the dealer's location. Based on the business request and the error, the bug can be easily traced and the problem can be quickly resolved.

6 Conclusion

Patterns help in solving the issues faced during the software development life cycle with better solutions. The migration of an application from one

architecture to another produces many challenges as the entire system gets updated. Similarly, migration of the existing SOA application to microservices also presents many challenges at different phases of migration and design. To address the most common recurring issues, we propose patterns that helps with easy migration. The problems addressed are identified in a literature study on the migration of legacy applications to microservices. The solutions provided are presented with our experience in migrating SOA based applications to microservices. The proposed patterns are best illustrated and evaluated using a standard case study application. However, the proposed patterns needs to be validated by applying on large enterprise applications. In the future, we are planning to develop new migration patterns and to validate them with case studies.

References

[1] T. Salah, M.J. Zemerly, C.Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi. The evolution of distributed systems towards microservices architecture. In *International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 318–325, IEEE 2016.

[2] L. Garces-Erice. Building an enterprise service bus for real-time SOA: A messaging middleware stack. In *33rd Annual IEEE International Computer Software and Applications Conference*. Vol. 2, pp. 79–84, 2009.

[3] V. Raj, and S. Ravichandra. Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services. In *3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pp. 1531–1536, 2018.

[4] N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering.* Springer, Cham. pp. 195–216, 2017.

[5] J. Thönes. Microservices. In *IEEE software*, 32(1), pp. 116–116, 2015.

[6] K. Brown and B. Woolf. Implementation patterns for microservices architectures. In *Proceedings of the 23rd Conference on Pattern Languages of Programs*, pp. 1–35, 2016.

[7] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, S. Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *10th Computing Colombian Conference (10CCC)*, pp. 583–590, IEEE 2015.

[8] B. Boehm and R. Turner. Management challenges to implementing agile processes in traditional development organizations. *IEEE software*, 22(5), pp. 30–39. 2005.

[9] S. Tyszberowicz, R. Heinrich, B.Liu, and Z. Liu. Identifying microservices using functional decomposition. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, (pp. 50–65). Springer, Cham. 2018.

[10] J. Zalewski. Real-time software architectures and design patterns: Fundamental concepts and their consequences. *Annual Reviews in Control*, 25, pp. 133–146. 2001.

[11] F. Lakhani. and M.J. Pont. Using design patterns to support migration between different system architectures. In *5th International Conference on System of Systems Engineering*, (pp. 1–6). IEEE. 2010.

[12] J. Soldani, D.A. Tamburri, and W.J. Van Den Heuvel. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146, pp. 215–232, 2018.

[13] S. Hassan and R. Bahsoon. Microservices and their design trade-offs: A self-adaptive roadmap. In *IEEE International Conference on Services Computing (SCC)*, pp. 813–818, 2016.

[14] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing*, pp. 201–215. Springer, Cham. 2015.

[15] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3), pp. 42–52. 2016.

[16] A. Balalaie, A. Heydarnoori, P. Jamshidi, D.A. Tamburri, and T. Lynn. Microservices migration patterns. *Software: Practice and Experience*, 48(11), pp. 2019–2042. 2018.

[17] D. Taibi, V. Lenarduzzi, and C. Pahl. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5), pp. 22–32. 2017.

[18] Leymann, F., Fehling, C., Wagner, S. and Wettinger, J. Native cloud applications: Why virtual machines images and containers miss. In *Proceedings of the 6th International Conference on Cloud Computing*, pp. 7–15. SciTePress. 2016.

[19] H. Knoche, and W. Hasselbring. Using microservices for legacy software modernization. *IEEE Software*, 35(3), pp. 44–49. 2018.

[20] A. Yanchuk, A. Ivanyukovich, and M. Marchese. Towards a mathematical foundation for service-oriented applications design. *Journal of Software* 1(1):32–9. 2006.
[21] P. Bhallamudi, S. Tilley, and A. Sinha. Migrating a Web-based application to a service-based system-an experience report. In *11th IEEE International Symposium on Web Systems Evolution* pp. 71–74. IEEE. 2009.
[22] V. Raj, and S. Ravichandra. A service graph based extraction of microservices from monolithic services of SOA. *Software: Practice and Experience*. 2020;.
[23] Tajamolian M, Ghasemzadeh M. A Versioning Approach to VM Live Migration. *International Journal of Engineering*. 2018 Nov 1;31(11):1838-45.
[24] Jeyanthi N, Shabeeb H, Durai MS, Thandeeswaran R. Reputation based Service for Cloud User Environment. *International Journal of Engineering (IJE) Transactions B: Applications*. 2014 Aug 1;27(8):1179–84.

Biographies

Vinay Raj has received his Masters degree from BITS-Pilani, India in the year 2016. He is currently Ph.D. student in the Department of Computer Science and Engineering of National Institute of Technology Warangal (NITW), India. He has previously worked in TATA Consultancy Services (TCS), Hyderabad as Developer and Analyst in SOA projects. He has published one conference paper in the area of microservices. His research interests include Service Oriented Architecture, Microservices and Software Engineering.

Ravichandra Sadam has received his Ph.D. in computer science from National Institute of Technology (NIT) Warangal, India in the year 2015. He is currently an associate professor in the department of computer science and engineering , National Institute of Technology (NIT) Warangal, Telangana, India. His research interests include Software Engineering, Software Architecture, Design Patterns and Service Oriented Architecture. He is a member of IEEE and ISTE.