Micro-services Transactions Resilience Across Bounded Domains: An Architecture Perspective

Sreenivasa Rao Vangala
Advisory Solutions Architect
DXC Technologies Ltd
Hyderabad, India

Ravi Kiran Mallidi
School of Computer Applications
Lovely Professional University
Punjab, India

V.L. Prasuna Appili
Department of Information Technology, MGIT, Gandipet
Hyderabad, India

ABSTRACT
In the digital transformation world, applications across industries are getting migrated to Cloud platforms. Off latest buzzword in the digital transformation world is zero touch solution. Also, several industry-specific SaaS-based frameworks are getting released in the market, which is in cloud native. Organizations are adopting buy and use the above-mentioned SaaS products to reduce development time and cost and overcome resource skill crunch. Thereby beat the competition. However, few enterprises and regulatory-specific applications can't be replaced overnight. Hence, enterprises are migrating applications to Cloud native solutions to achieve complete or near-zero digital transformation. The complex nature of business rules requires communication between diversified cloud products, typically called bounded context or domain-driven architecture. The concept of data mesh has come into place to achieve scalable and secured enterprise systems. However, there are no data patterns in the area to take care of CRUD (Create, Read, Update, and Delete) operations, i.e., address the CRUD operations failure instances between boundaries. The current work aims to develop architectural concepts for managing CRUD operations failure and achieving transactional resilience in domain-driven architecture.

Keywords
Microservices Best Practices, API Gateway, Domain Driven Architecture, Bounded Context, Digital Transformation, Enterprise Architecture, Transactional Process Resilience, Tran Process (TP) Monitor, TP Audit, Event Handler.

1. INTRODUCTION
Software systems are developed initially to turn around the work at a faster phase with zero or no errors as against doing the same work manually. Also, improvements like enhanced user experience, Enterprise Application Integration (EAI), data sharing, cloud computing, Digital Transformation, etc., in Information Technology (IT) have further necessitated to architect the state of enterprise systems which are compliant with industry-specific regulatory processes, secured, scalable, reliable and resilient.

The IT systems built during the initial days are procedure-oriented and easy to build. However, they are very expensive and time-consuming regarding maintenance or enhancements. With the invention of new technologies and IT infrastructure, the industry has moved to Object Oriented Methodology (OOM). While OOM systems are better than procedure-oriented systems, maintainability and future enhancements are time-consuming and expensive in terms of both development and infrastructure. Technological advancements like microservices [1,2], agile/scrum and Dev Ops framework software engineering and deployment methodologies [3] respectively have given pathways to developing systems in a shorter time. As a further development, an API-based approach has been introduced in [4], adding flexibility to enterprise systems in disparate system integration.

As mentioned above, several systems across the industry have been architected, designed, and developed using microservices. The supporting level infrastructure has also been architected by infrastructure engineers and architects for non-cloud environments, and the same has been taken care of by Cloud native infrastructure. As a next step, the IT industry has moved towards SaaS products. These SaaS products are nothing but developing industry-specific frameworks in a cloud-native environment. With the development of SaaS-based and migrating existing on-prem applications to the cloud as part of application modernization, the data mesh has come into place to establish effective communication across cloud boundaries. In an enterprise, if one looks at end-to-end transaction perspective, the flow of any enterprise systems is as follows:

Step I: The data is entered in the front-end channel by an end user and submitted for action

Step II: The user request is re-directed to the API gateway for invoking the respective domain microservice

Step III: CRUD operation triggered as part of the microservice invocation. The said CURD operations are of two types: i) executed in the native cloud boundary, ii) outside the cloud boundary

Step IV: Receive the result and route it to the respective subsystem or front-end channel

In the execution scenarios mentioned above steps I-IV are ideal. However, in practice there could be several systems-related issues, like DBMS systems may be down, and interfaces or gateways may not respond whenever a transaction moves and hits destination systems; as a result of this, CRUD operations might fail. There are well-architected and designed data patterns to handle CRUD operations failure within the bounded context or cloud-native boundary. The crux of the matter is what if a CRUD operation fails outside of a designated cloud boundary? The current work concentrates on architecting a framework for CRUD operations failure outside the cloud boundary. An effort has been taken to architect Transaction Process (TP) Monitor, TP Audit, and event handler concepts to ensure that CRUD operations failures are handled. This will ensure that the enterprise systems behavior is more consistent, achieving resilient system behavior.

This paper is organized as follows: Section II describes Literature Review, Section III describes TP and TA architecture concepts, Section IV describes a few case studies

which authors have come across in their day-to-day work, and Section V Conclusions and further work.

2. LITERATURE REVIEW

The IT systems were initially developed as a standalone system, and data is transferred to dependent systems through a manual batch process. The recent trends in the technology space of IT have moved to digital transformation and ZERO touch implementation. This has necessitated the application modernization from monolithic to microservices-based systems. The legacy modernization from the monolithic has been described in [1]. The microservices architecture with a function-first approach has been described in [2]. Microservices architecture with agile/scrum and dev ops has been described in [3]. The enhanced framework for enhancing any microservices system is routing microservices through API Gateway, which has been implemented in [4]. Multi-tenant-based architecture has been described in [5]. Complexity-based microservices implementation and domain-based microservices implementations are described in [6] and [7] respectively. An event-driven-based implementation further boosts the microservices-based implementation, and the same has been described in [8,9 and 10]. Adding streaming to any microservices-based implementation on top of event-driven architecture has further improved the performance of the system and handles real-time systems, and the same has been discussed in [11] and [12]. Application modernization with microservices in the Cloud infrastructure has been described in [13]. Cloud and Dev Ops for microservices has been described in [14]. The digital transformation maturity model has been described in [15], and the same digital transformation specific to the finance industry has been described in [16]. The reference model for digital transformation has been described in [17]. The serverless cloud computing concepts have been covered in [18]. The data mesh concepts, domain driven architecture has been described in [19].

3. MICROSERVICES AND THE DOMAIN DRIVEN ARCHITECTURE

The entire system is divided into subsystems based on business functionality in a domain-driven architecture. Business processing related to that subsystem is built in the respective subsystems or cloud infrastructure as described in [19]. The typical view of sample bounded context has been depicted in Fig 1.

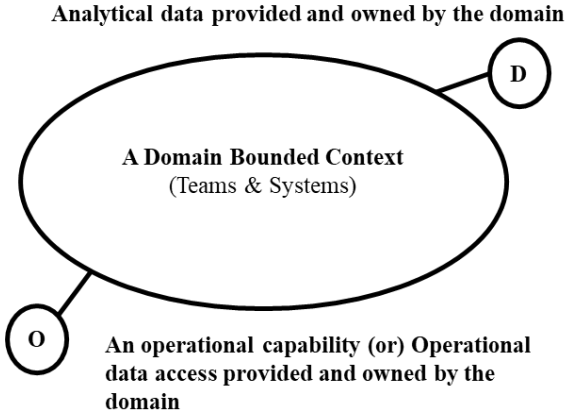

Fig 1: A sample view of a domain-driven context

In Fig 1, each domain can expose one or more operational Application Programming Interface (APIs) and one or many analytics endpoints. The real-time example of domain-driven architecture is depicted in Fig 2.

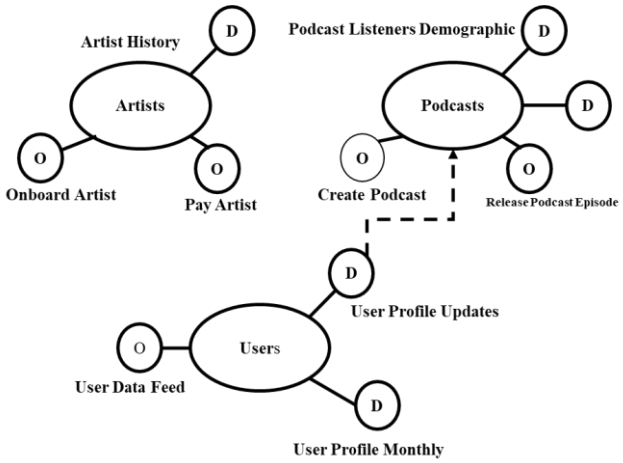

Fig 2: Representation of a domain-driven architecture

Each of the domains represented in Fig. 2 is called a bounded context, and communication among bounded contexts happens through translation maps. The said bounded contexts communication is of two types: i) process and ii) Data. The bounded contexts of communication has been depicted in Fig 3

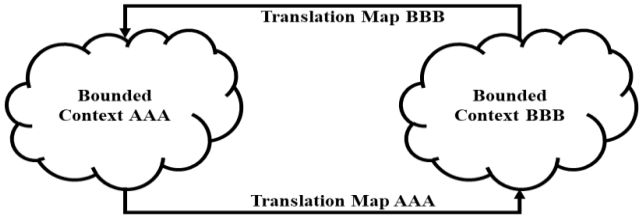

Fig 3: Communication across Bounded contexts

The basic principles of bounded context are

One should treat each SaaS/SAP/Standalone application as a bounded context to solve a disparate application issue

As mentioned above, each bounded context should contain process and data

Bounded context should communicate effectively at the boundary of the domain

Communication in the transactional context mentioned in Fig. 3 happens through microservices. There are no design patterns on how best to manage DBMS CRUD operations on a set of data access patterns. Some patterns for two-phase commits in technologies such as Spring. But these does not provide robust transaction management capabilities such as those required in domains.

In addition, event-driven data architecture requires event delivery to be robust and reliable. This kind of complex operation can't be met by a single microservice context. Simple SQL transaction capabilities don't provide the necessary functionality for these operations.

The ultimate solution for the above-said transactional issues is the TP Monitor, which has three components. They are as follows:

TP Monitor – Creates transaction ID and monitors the flow within a given microservice

TP Audit – Records the current state of any given transaction

and can be used to record a failure point in a given transaction

Event Manager – stores the status of any given event and notifies any subscriber to that event

The above scenarios are depicted in the following Fig 4.

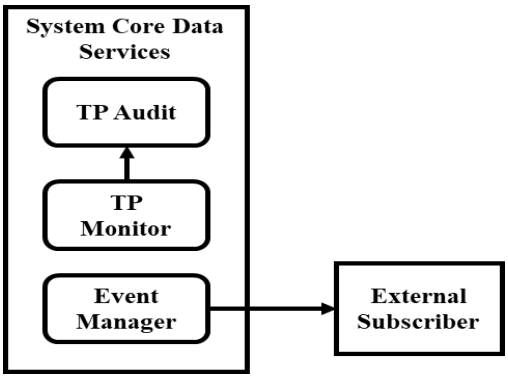

Fig 4: Representation of Transaction Process Components

The role of any TP monitor is to monitor the list of activities in memory transactions by inspecting transactions that have passed a designated timeout period. The timeout transactions are logged, and an exception has been raised for operations resolutions. The active transaction monitor process has been depicted in the following Fig. 5.

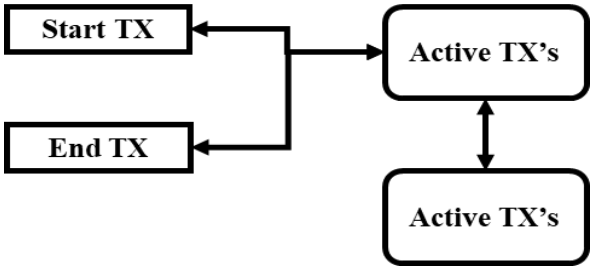

Fig. 5: Active transaction monitoring process

The transaction monitoring processes in the microservices context has been depicted in Fig 6.

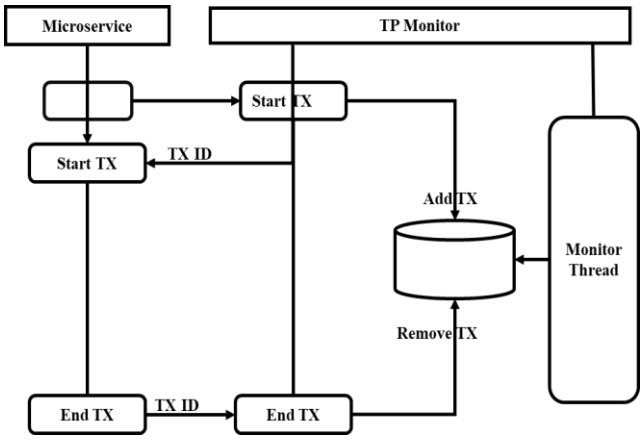

Fig 6: Active Transaction Process Scenario in the Microservices context

In Fig 6, the following sequence of events occurs during any DBMS transaction execution.

TP monitor looks for unmatched transactions IDs and compares them with a transaction timeout value

Raises an exception if any unmatched exception is found

As soon as any exception is thrown by the TP monitor, the TP audit process will come into action to log the intermediate states of the monitored transaction. All states will be logged and persisted for future reference. This should link to our standard audit framework and the same has been depicted in Fig 7.

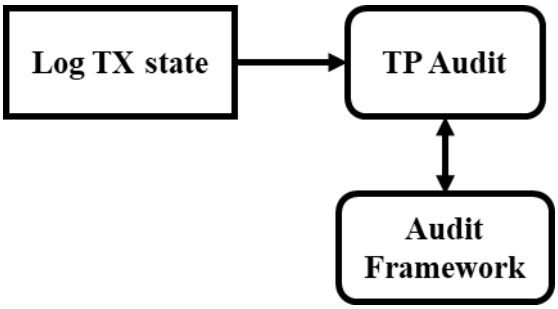

Fig 7: TP Audit Flow

The event handler is a mechanism that comes into the picture whenever a given transaction that requires an event to be published will post an intent at the beginning of the transaction. This handler will monitor for expired event intents and log timeout events. On successful transaction completion, the transaction will post the required update event. Events will be sent via a messaging component which will notify (ensure delivery) of updated events to the registered subscribers. Failure of a subscriber to acknowledge event receipt will cause the raising log and sync error exceptions to the IT operations. The event handler mechanism is depicted in Fig 8.

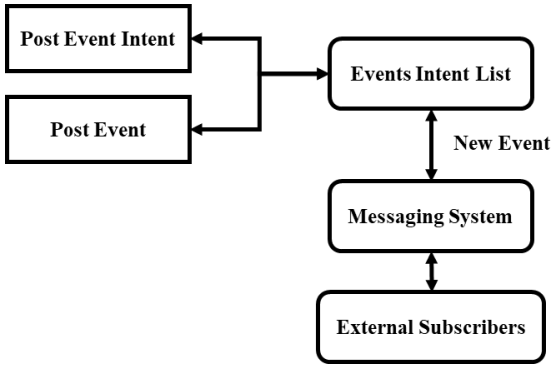

Fig 8: TP Event Handler

The messaging capabilities should be proxied out through the API gateway as many connected systems in the cloud and outside of the Cloud boundary.

4. SYSTEM ARCHITECTURE METHODOLOGY

The current TP monitoring system has been architected in such a way that it overlays an existing data services architecture. The existing data services architecture has been depicted in Fig 9, which is outside the scope of the current work.

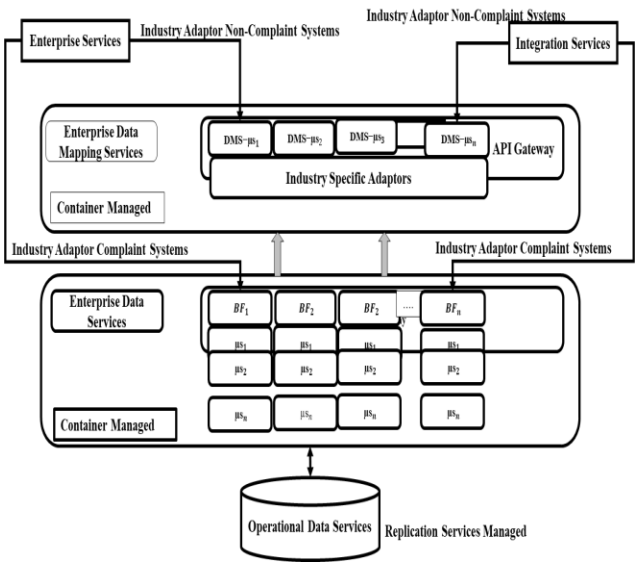

Fig 9: High-Level Architecture of Translation System with Adaptors

Data mapping and services view with the TP monitor are shown in Fig 10 below.

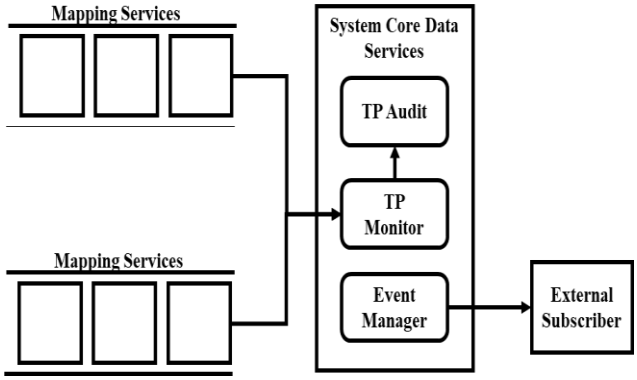

Fig 10: Data Mapping services and Data Services Architectural view

The end-to-end transactional view in a practical implementation has been depicted in the following Fig 11.

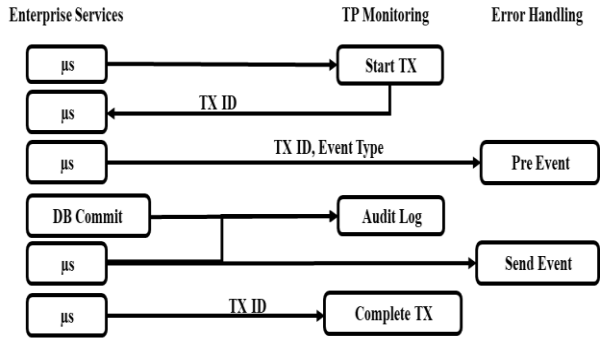

Fig. 11: End-to-End Transactional view

The sample end-to-end architecture view of simple transaction events and transaction plus events has been depicted in Fig. 12 and Fig. 13, respectively.

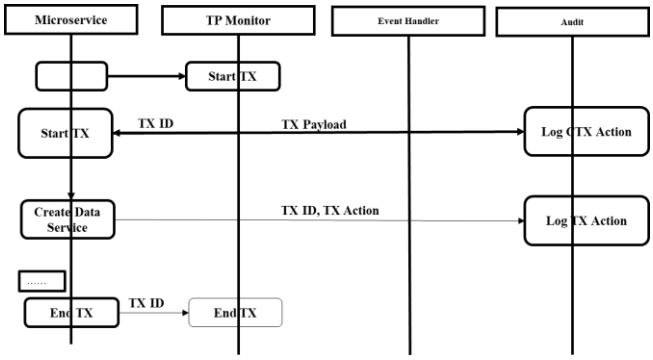

Fig. 12: An architectural view of a simple transaction event

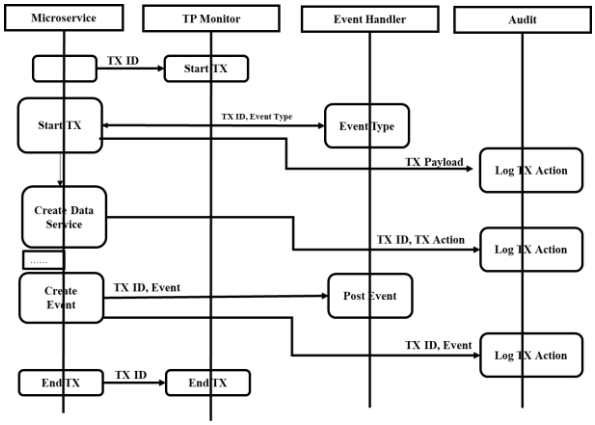

Fig 13: An architectural view of transaction plus event

The deployment architecture view for the current system of consideration has been depicted in Fig 14.

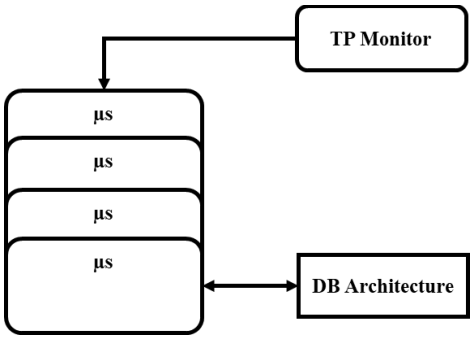

Fig 14: The Deployment Architecture view of microservices TP monitor and DB Architecture

There are multiple ways in which transactions could fail. Some of the examples have been captured in the following Table 1

Table 1: Few examples of transactions failure scenarios

Scenario	Sub-Scenario	Reason
µs (microservice)	µs native failure	Event thrown by µs
	COTS product failure	Occurs during event consumption phase
DB Write Fail	Partial	Out of N transactions there is a possibility that 20-30% (approximately) transactions might fail

	Full	Out of N transactions, all of them could fail due to DB connectivity issue
Messaging system fail		The messaging system is unable to receive messages due to connectivity failure to messaging system
Event consumer fail		The failure might occur at event consumer level i.e., failure of microservices or respective Cloud native functions or any other
Monitoring functions failure		The Cloud DB monitoring functionality might fail

5. CONCLUSION

In the current work, authors have purely focused to bring various architectural scenarios with respect to Database transactions failure and resilience in a microservices context at an enterprise level. However, there could be few situational specific issues arise from the business architecture perspective. These issues will be different for different industries i.e., issues in Banking, Finance Service and Insurance (BFSI) will not be same as that of Retail, Telecom, Media and Entertainment, Oil and Gas, etc. Also, the applicability of the current work with respect to enterprise architecture level framework like i) Application Architecture, ii) Data Architecture and iii) Technology Architecture could be another dimension one could consider. As an extension of this work one could pick-up a domain specific on prem to Cloud migration problem in a digital transformation world and apply the above detailed concepts and compare the results. After successful execution of the above work the concepts could be extended to cross domains.

6. REFERENCES

[1] Bakar, H. K. A., Razali, R., & Jambari, D. I. A Guidance to Legacy Systems Modernization.

[2] J. -P. GOUIGOUX and D. TAMZALIT, ""Functional-First" Recommendations for Beneficial Microservices Migration and Integration Lessons Learned from an Industrial Experience," *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2019, pp. 182-186, doi: 10.1109/ICSA-C.2019.00040.

[3] Muhammad Waseem and Peng Liang, Microservices Architecture in DevOps, 2017 24th Asia-Pacific Software Engineering Conference Workshops, DOI: 10.1109/APSECW.2017.18

[4] Chris Richardson, Building Microservices: Using an API Gateway, Retrieved 10.06.2019 from https://www.nginx.com/blog/building-microservices-using-an-api-gateway/

[5] C. Batista, B. Proença, E. Cavalcante, T. Batista, F. Morais and H. Medeiros, "Towards a Multi-Tenant Microservice Architecture: An Industrial Experience," 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), 2022, pp. 553-562, doi: 10.1109/COMPSAC54236.2022.00100.

[6] N. Santos and A. Rito Silva, "A Complexity Metric for Microservices Architecture Migration," *2020 IEEE International Conference on Software Architecture (ICSA)*, 2020, pp. 169-178, doi: 10.1109/ICSA47634.2020.00024

[7] Michel Cojocaru, Alexandru Uta and Ana Oprescu, MicroValid: A Validation Framework for Automatically Decomposed Microservices, 2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), DOI: 10.1109/CloudCom.2019.00023.

[8] Ervin Đogić, Samir Ribić, Dženana Đonko, Monolithic to Microservices redesign of event driven integration platform, 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), DOI: 10.23919/MIPRO.2018.8400254

[9] Tony Clark, Balbir S. Barn, Event Driven Architecture Modelling and Simulation, Conference Paper, December 2011, DOI: 10.1109/SOSE.2011.6139091.

[10] Svetoslav Zhelev and Anna Rozeva, Using microservices and event driven architecture for big data stream processing, Proceedings of the 45th International Conference on Application of Mathematics in Engineering and Economics (AMEE'19), AIP Conf. Proc. 2172, 090010-1–090010-8; https://doi.org/10.1063/1.5133587

[11] S. R. Vangala, B. Kasimani and R. K. Mallidi, "Microservices Event Driven and Streaming Architectural Approach for Payments and Trade Settlement Services," *2022 2nd International Conference on Intelligent Technologies (CONIT)*, 2022, pp. 1-6, doi: 10.1109/CONIT55038.2022.9848178.

[12] R. K. Mallidi, M. Sharma and S. R. Vangala, "Streaming Platform Implementation in Banking and Financial Systems," *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*, 2022, pp. 1-6, doi: 10.1109/ASIANCON55314.2022.9909500.

[13] B. Althani, S. Khaddaj and B. Makoond, "A Quality Assured Framework for Cloud Adaptation and Modernization of Enterprise Applications," *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, 2016, pp. 634-637, doi: 10.1109/CSE-EUC-DCABES.2016.251.

[14] A. Balalaie, A. Heydarnoori and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," in *IEEE Software*, vol. 33, no. 3, pp. 42-52, May-June 2016, doi: 10.1109/MS.2016.64.

[15] T. Aguiar, S. Bogea Gomes, P. Rupino da Cunha and M. Mira da Silva, "Digital Transformation Capability Maturity Model Framework," *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, 2019, pp. 51-57, doi: 10.1109/EDOC.2019.00016.

[16] T. Butler, "What's Next in the Digital Transformation of Financial Industry?," in *IT Professional*, vol. 22, no. 1, pp. 29-33, 1 Jan.-Feb. 2020, doi: 10.1109/MITP.2019.2963490.

[17] S. Bogea Gomes, F. M. Santoro, M. Mira da Silva and M. -E. Iacob, "A Reference Model for Digital Transformation and Innovation," *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, 2019, pp. 21-30, doi: 10.1109/EDOC.2019.00013.

[18] W. Lloyd, M. Vu, B. Zhang, O. David and G. Leavesley, "Improving Application Migration to Serverless Computing Platforms: Latency Mitigation with Keep-Alive Workloads," *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 195-200, doi: 10.1109/UCC-Companion.2018.00056

[19] Max Schultze, Arif Wider, Data Mesh in Practice, O'Reilly Media, Inc, ISBN: 9781098108496.