Agents in Software Development Architectures[*]

Marco Loaiza[1,*,†], Claudio Savaglio[2,†], Niaz Hussain Arijo[3,†], Gianluca Aloi[4,†], Giancarlo Fortino[5,†] and Raffaele Gravina[6,†]

[1]*Department of Informatics, Modeling, Electronics and Systems engineering (DIMES), University of Calabria, Italy*

Abstract

Over the last few decades, information technologies in the field of computing systems and software architectures have evolved to provide companies with robust solutions to run their business logic. The transition from monolithic applications to Service-Oriented Architecture (SOA) and micro-services represents a fundamental shift in the way companies design, build, and deploy their software. This evolution has been driven by the need for greater scalability, flexibility, and agility in response to the dynamic and demanding landscape of modern application requirements. As software development continues to evolve, the incorporation of intelligent agents has become widespread to provide smartness within distributed system as well as to design and manage complex scenarios, such as the device-edge-cloud continuum and IoT ecosystems. This paper explores the role of intelligent agents within SOA and micro-services architectures, highlighting the spectrum of benefits and limitations they bring to the development life cycle. We also discuss challenges common to both architectures and identify solutions that each architecture provides with respect to the other.

Finally, to overcome some of these challenges, a new approach, stemmed in the context of the EU funded MLSysOps Project, is presented; the proposed approach utilizes agents and Machine Learning (ML) as-a-service to provide inbound and outbound intelligence in the system.

Keywords

device-edge-cloud continuum, multi agent systems, microservices, MLSysOps

1. Introduction

In recent decades, the advancement of technology and software architectures has increased the complexity of computing environments to a degree that exceeds the manageable limits for human system administrators. Initially, with monolithic applications, companies consolidated data, databases, and business logic on the same host by running their applications as a single, unified entity. In this approach, all components, including the user interface, business logic, and database access, are tightly integrated into a unified code-base that encompassed various services [1]. The described architecture faces challenges in scalability, flexibility, and adaptability. Scaling monolithic applications is a proven challenge due to the aggregation of popular and less-

CEUR Workshop Proceedings (CEUR-WS.org)

used services, leading to inefficient resource usage. Collaboration among multiple developers on a shared code-base complicates the development, and deploying new versions requires restarting the system as a whole, potentially impacting user experience. Additionally, the architecture lacks flexibility in modifying features and reusing components across applications, and its limited error tolerance means a single component error can cause system-wide failures.

Service Oriented Architecture (SOA) was proposed to solve this challenges and also to provide integration between solutions for distributed services across different organizations and operational systems aiming for the reusability of a service by multiple end user applications. It relies on simple Remote Procedure Calls (RPC), such as SOAP (Simple Object Access Protocol), commonly used for communication. Since the advent of this software architecture, computing environments have evolved into open and distributed systems. The typical enterprise environment is now a complex, heterogeneous, multi-vendor landscape, posing challenges in configuration, maintenance, and troubleshooting [2].

Micro-services architecture emerged as the next evolutionary step beyond SOA, with lightweight communication channels, prioritizing simplicity in inter-service interactions. This architectural approach directs intelligence and control to individual service endpoints, thus enhancing the decoupling between services, addressing issues of scalability and adaptability in the face of rapidly changing technological landscapes. In this intricate scenario of technological progress, the trajectory from monolithic applications to SOA and, subsequently, to micro-services architecture reflects an ongoing quest for solutions that aligns with the dynamic nature of contemporary computing environments [3].

Until this end, agent-based computing (ABC) is one of the well-known paradigms that proposes an improved structuring mechanism for distributed applications. ABC is characterized by autonomous, intelligent entities capable of migrating across distributed systems; it offers a promising avenue for addressing the escalating complexities in modern computing. In conjunction with SOA and micro-services, ABC contribute to a holistic approach that transcends traditional boundaries. The seamless integration of software agents with SOA enables the creation of agile, distributed services that can be leveraged across diverse organizations and operational systems. Moreover, when juxtaposed with micro-services architecture, ABC contributes to the ongoing quest for enhanced scalability, adaptability, and intelligence at the service endpoints. This triad of agents, SOA, and micro-services represents a forward-looking strategy, offering a robust response to the intricate challenges posed by today's multifaceted computing landscape.

Various studies in the field have undertaken the task of comparing mobile agents with SOA and micro-service architectures, attempting to discern their individual strengths and weaknesses [4, 5, 6]. However, out of these comparative analyses, a growing body of research emphasizes the potential synergies that emerge when these paradigms are combined. In this paper, we delve into the interaction between software agents, SOA, and micro-services, with the aim of analyzing and highlighting the benefits and limitations associated with choosing one over the other, ultimately selecting the most fitting option for IoT applications in the Device-Edge-Cloud (DEC) continuum.

The remainder of the paper is structured as follows: **Section 2** describes the background, providing an introduction to the key features of the technologies and paradigm involved. **Section 3** offers an overview of the state of the art, including an analysis of the benefits and limitations

of utilizing agents within both SOA and micro services architectures. The MLSysOps approach and its possible contributions to the open limitations is presented in **Section 4**. Final remarks conclude the paper.

2. Background

This section introduces key concepts related to SOA and micro-services architectures.

2.1. Service Oriented Architecture

SOA emerged as a response to challenges posed by large monolithic applications, aiming to enhance service re-usability across multiple end-user applications and representing an evolution from the conventional Client-Server model in distributed systems.
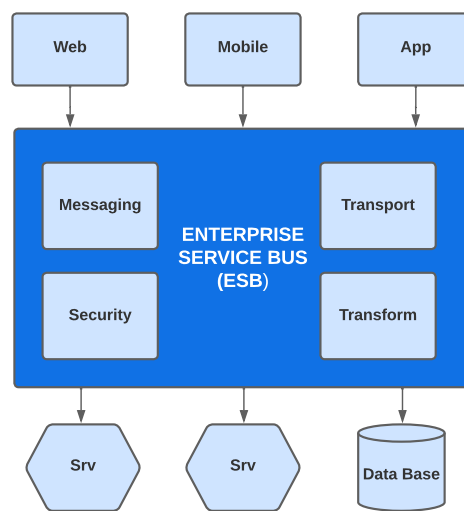
Figure 1: Service Oriented Architecture.

SOA promotes loosely coupled, reusable, and dynamically assembled services, providing explicit boundaries between autonomous services on different servers to meet application requirements and address the limitations of monolithic applications. A common scenario involves clients requesting services through various user interfaces, with the Enterprise Service Bus (ESB) handling the requests. The ESB, designed to seamlessly connect services directly with end-users and render the physical network transparent to applications, incorporates some service functions to link widely located services with the end user, as illustrated in Figure 1. It translates end-user requests into suitable message types for each service, fostering their re-usability across different organizations for numerous customers [2, 5].

In the software industry, SOA has gained substantial popularity for developing distributed enterprise-wide applications, emphasizing software re-usability and integrity in diverse environments through open standards. Companies often leverage SOA by discovering and combining

internet-available services. Alternatively, some organizations prefer internal control, creating new services tailored to their specific environments with quality-attribute properties.

SOA facilitates integration solutions for distributed services across various organizations and operational systems, relying on straightforward RPC like SOAP for communication. It emphasizes the use of services across the web, aligning with the trend of distributed and interconnected systems.

2.2. Microservices

In today's digital landscape, users expect a dynamic user experience across diverse devices, requiring frequent service updates. To meet this demand, the architectural trend emphasizes breaking applications into smaller, independent services. This approach is often compared to SOA, critiquing its centralized integration approach. Micro-services are viewed as an alternative, diverging from SOA by eliminating a service bus, focusing on simple communication channels, and enhancing service decoupling [2]. Microservices, operating as independent or subdivided components, break down substantial services into manageable elements as shown in Figure 2.
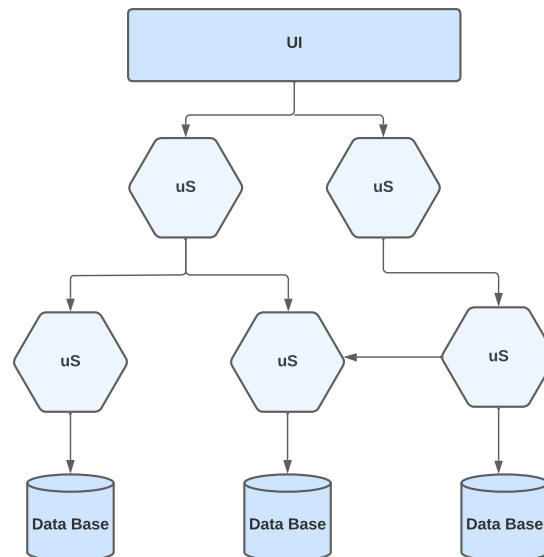
Figure 2: Microservice Architecture.

Each microservice, capable of at least one functional operation, promotes independence and reusability. Breaking databases into smaller ones benefits specific customer types, distributing the load and allowing shared databases to operate as services. Despite numerous benefits like technology diversity and scalability, microservices pose challenges in distributed settings, complicating tasks like service discovery, security and resource management, as well as communication optimization. Microservices architecture, characterized by small, independent services running in separate processes, originated with the goal of deploying software parts independently. These autonomous components isolate fine-grained business capabilities,

communicating via standardized interfaces and lightweight protocols. While offering benefits in scalability and productivity, microservices require careful consideration due to distributed complexities[3].

Understanding microservices in practice is crucial, revealing motivating advantages and critical challenges. This insight support decision-making for system migration or developing applications under this architecture. Overall, the approach enhances software maintainability but demands thoughtful consideration of its challenges.

3. Agents and (Micro)Services: benefits and limitations

Agent-based Computing (ABC) has been recognized as a comprehensive and effective enabler for cooperating within decentralized, dynamic, and open IoT ecosystems [7]. In particular, software agents represent an elegant programming paradigm that emerged in parallel the client-server model, offering an enhanced structuring mechanism for distributed applications. Essentially, an agent can augment the functionalities of a service or microservice, thereby maximizing robustness and performance within the system. In contrast, the introduction of the agent and its abstractions intrinsically brings other issues, complexity and overhead.

To delve further into this effectiveness, it is essential to examine the combination of agents with both SOA and microservices. Such explorations contribute to a deeper understanding of how these paradigms can be combined to optimize various aspects of system architecture and performance.

The following are some of the benefits found in the literature review of using agents with both architectures:

- Agents in both SOA and microservices exhibit **increased flexibility, autonomy, and collaboration capabilities**. In SOA, agents excel in making dynamic run time changes based on internal rules, enhancing robustness. In microservices, agents enable adaptive responses to change environments, contributing to improved scalability and flexibility in system design. They efficiently distribute workloads and handle complex tasks, enhancing overall performance and resource utilization [2, 4, 5, 6, 8, 9, 10].
- The integration of SOA and MAS create the synergy to enhance self-properties in distributed assembly systems, allowing agents to transcend research and function as application components. This fusion facilitates seamless integration, automates processes, can optimizes power consumption through negotiation, and promotes semantic interoperability. In microservices, agents enhance **collaboration, modularity, and scalability**, improving overall system performance and reliability [4, 5, 6, 11, 12, 8, 13, 9, 14, 15].
- Agents also support the management and optimization of distributed energy resources, load balancing, and markets. Their ability to **distribute processing tasks enhances scalability and resource utilization** in diverse contexts. Furthermore, agents, by perceiving and responding to changes promptly, improve system responsiveness. In cloud applications, agents provide an intelligent and autonomous approach to managing and scaling the available resources, e.g. using Reinforcement Learning [2, 5, 12, 16, 10].
- Agents serve a versatile role in negotiating service-level agreements (SLA's) across enterprise boundaries, encompassing quality specifications for both infrastructure and

application. The robust support of SOA frameworks enhances the **quality of agent systems**. Agents contribute to **performance optimization** by executing tasks locally, minimizing the reliance on network bandwidth and resulting in faster response times. In the realm of microservices, agents play a key role in learning resource usage behavior and determining optimal auto scaling metrics for **Quality of Service (QoS)**, thereby reducing the need for specialized knowledge and streamlining maintenance efforts in application resource management [2, 5, 11, 10].

- Agents, with capabilities spanning **knowledge, reasoning, planning, scheduling, and inter-agent communication**, are highly relevant for autonomic computing within SOA. In SOA, agents bring intelligence to web services, fostering interoperability among systems and components. Their ability to execute tasks locally reduces the demand for network bandwidth, leading to improved response times and overall performance. In a microservice architecture, agents play a crucial role in managing and interacting with distributed systems in a loosely coupled manner, fostering independence and replicability among system components [2, 6, 11, 12, 16, 15].

- Agents play a pivotal role in various domains, enabling the autonomous operation of systems. In conjunction with microservices, agents contribute with **intelligent and autonomous behavior**, enhancing overall system functionality and decision-making processes. Their autonomy in SOA allows independent action and decision-making beyond predefined behavior or contracts. Agents improve fault tolerance and resilience in microservices by autonomously detecting and recovering from failures, thereby minimizing system disruptions. Additionally, agents facilitate the integration of heterogeneous systems and microservices, acting as intermediaries to translate and mediate between different protocols, data formats, and interfaces. They support the implementation of intelligent and personalized services by learning from user interactions and preferences, providing tailored recommendations and proactive assistance [2, 4, 6, 10, 11, 12, 8, 13, 14].

On the other hand some limitations found of combining agents and SOA are grouped as follows:

- Resource-intensive agent operations in diverse architectures, including SOA, present challenges to overall **system performance and scalability**. The computational demands and memory requirements of agents, along with potential coordination overhead as the number of agents increases, can impact scalability. Coordinating agent behavior within any architecture, particularly in expansive systems, poses inherent challenges. Additionally, the training and deployment of agents in various contexts demand substantial computational resources and processing power, potentially leading to scalability issues [4, 11].

- Challenges in integrating Multi-Agent Systems (MAS) and SOA arise from perceived competition, **lack of standardization, and interoperability** issues among different agent platforms and frameworks. The autonomous nature of agents poses difficulties in ensuring alignment with organizational goals and compliance with policies. Incorporating agents with web services in SOA may require additional effort for embedding intelligence and ensuring interoperability. Issues such as limited support for hosting agent classes,

challenges with mobile agents, and difficulties in integrating agent-based systems with existing technologies further impede widespread adoption. Additionally, the lack of standardization in Agent Communication Languages (ACLs) and potential limitations in interpretation of Reinforcement Learning (RL) agents contribute to interoperability concerns in SOA [4, 12, 11].

- In SOA, the dynamic and autonomous behavior of agents can lead to challenges in predictability, making it difficult to forecast their outputs. Additionally, microservices, while sharing agent-like qualities such as autonomy and collaboration, may **lack inherent learning and adaptation capabilities**, which could be a consideration for future system upgrades [4, 8].
- Differences in control systems, with the Foundation of Intelligent Physical Agents (FIPA) specification are not suitable for experimental setups and SOA control structures for production systems, highlight considerations in system design. The coordination and communication between agents and services in distributed environments introduce potential overhead and latency. For microservices, well-defined communication protocols, like HTTP or messaging systems, are crucial, necessitating careful consideration to ensure proper communication and system functionality, especially during microservices migration or updates. Challenges in **coordinating and communicating between agents in a distributed environment** may pose difficulties in maintaining consistent behavior and synchronization [5, 12, 13].

The following limitations are also presented in the context of using agent and micro service architectures:

- Implementing self-organization and emergence in several systems through MAS and SOA introduces concerns due to potential **risks of unpredictability and disruptions.** Security is a critical consideration, particularly when agents have access to sensitive data and interact with external services, as their autonomous behavior can become vulnerable to malicious attacks or unauthorized access. Addressing security in agent-based systems within SOA necessitates robust measures, including trusted hosts, secure communication protocols, and secure agent profiles to safeguard sensitive information. The autonomous nature of agents raises challenges in ensuring security and privacy, making the design and implementation of protective measures complex, especially in microservices environments with distributed data exchange. [2, 6, 11, 5, 12, 13, 9, 16, 14]
- MAS face scalability and performance limitations in large-scale systems, particularly in the context of SOA. Concerns about the **scalability of agent-based systems in SOA arise from the potential impact on system performance**, as the number of agents and their interactions increases. Agents may introduce **computational overhead and memory usage challenges**, affecting the performance of microservices. Additionally, coordination, negotiation, and communication between agents can contribute to additional overhead and latency in the system, emphasizing the need for innovative performance evaluation techniques and optimization methods for effective resource allocation and arrangement [12, 9, 16, 14, 15].
- Integration of agents in systems, particularly within SOA, introduces heightened complexity. **This complexity spans development, maintenance, coordination, and**
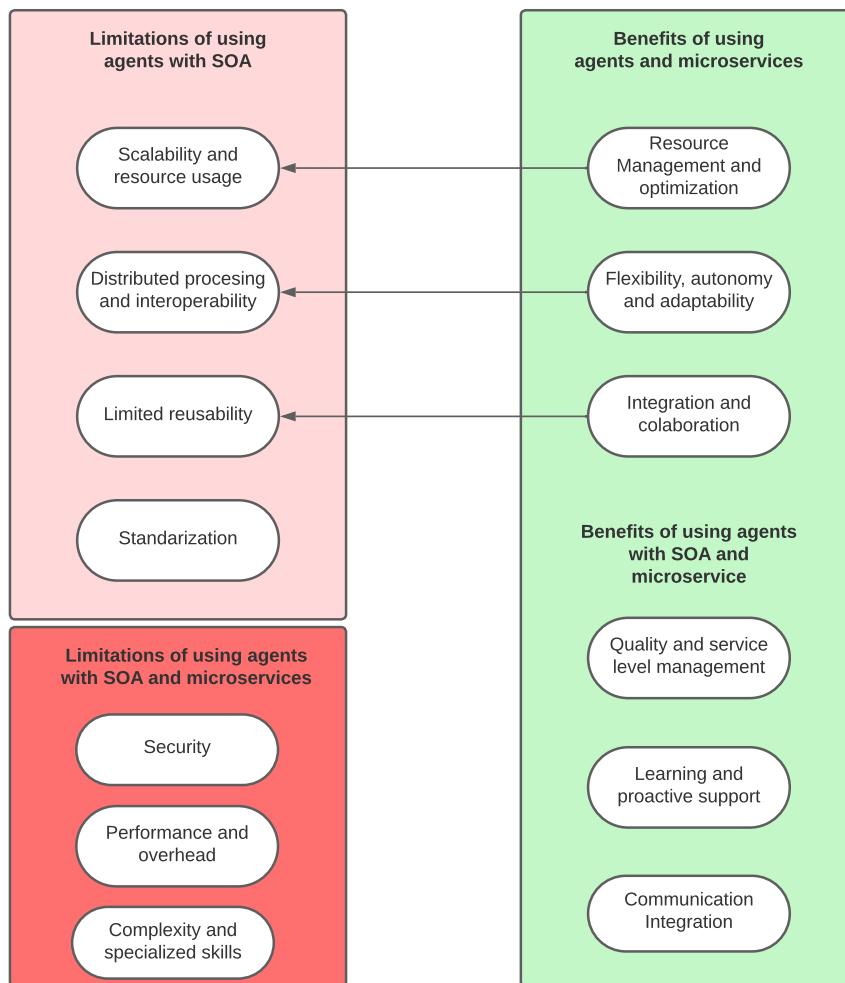
Figure 3: Benefits and limitations of agents in software architectures.

management aspects, necessitating **specialized skills and expertise**. Factors such as persistence management, coordination mechanisms, and the potential for performance overhead contribute to the intricate nature of agent-based systems. The challenges extend to aspects like debugging, troubleshooting, and adapting to dynamic environments, making the implementation and management of agent-central architectures resource-intensive and demanding careful design considerations [2, 4, 5, 6, 11, 12, 8, 13, 16, 14, 15, 10].

A summary of the preceding points is depicted in Figure 3, where both the limitations and benefits of employing agents in software architectures can be observed. It is clear that the combination of agents and microservices offers certain advantages that address the limitations found in the combination of agents and SOA.

To illustrate, if we encounter issues like scalability and resource usage in the context of agents combined with SOA, we can address them by taking advantage of the effective resource

management and optimization features found in agents combined with microservices. Likewise, challenges related to distributed processing and interoperability, as discussed in the context of agents and SOA, can be resolved by utilizing the flexibility, autonomy, and adaptability offered by agents combined with microservices. Moreover, the problem of limited reusability often associated with SOA combined with agents can be overcome by integrating and collaborating with agents combined with microservices, which enhances the agent's capacity for improvement. Additional benefits are intrinsic to both types of architectures, as is evident from the earlier discussion, where microservices can be viewed as an evolution of SOA.

Finally, common limitations emerge when deploying agents in both architectures, implying persistent challenges. In the following section, we introduce MLSysOps, a project designed for automatic and adaptive resource management and application deployment. This initiative is poised to tackle some of the challenges associated with these limitations.

4. The MLSysOPS use case

The MLSysOps project represents a crucial initiative aimed at navigating the complexities surrounding resource management and application deployment within DEC continuum systems, leveraging the power of ML techniques. It is conceived as a Platform-as-a-Service (PaaS) solution,dedicated to streamlining the management, and deployment of distributed IoT applications. These applications, comprising interconnected components with diverse resource demands and considerations for quality of service, find a unified platform in MLSysOps. The advantages explained in the previous section seamlessly extend to the MLSysOps project. In particular, the utilization of this agent framework to oversee microservice-oriented IoT applications within the DEC Continuum offers a range of functionalities, encompassing autonomy, intelligence, flexibility, and integration.
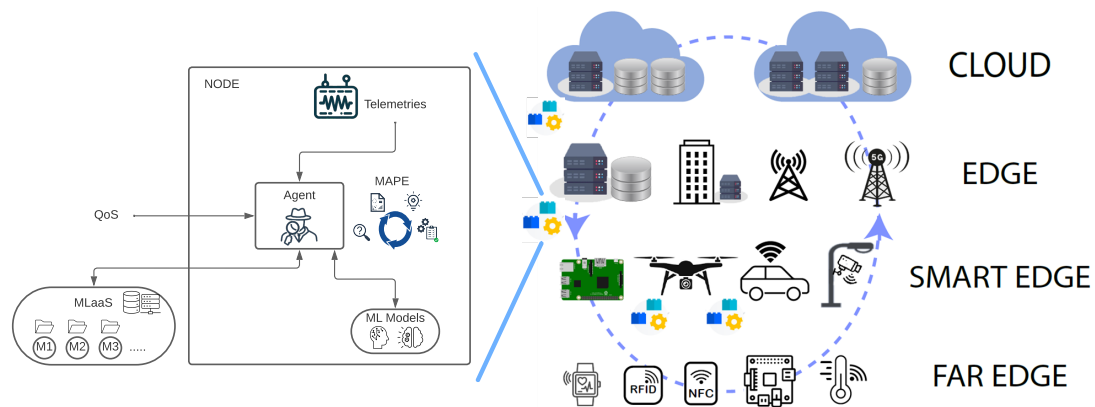
Figure 4: MLSysOps agent operation.

As illustrated in Figure 4, agents, equipped with their full range of functionalities, will be capable of internally and externally employing ML models on each node to oversee the execution

of the application. Additionally, a ML-as-a-service repository will be available, allowing agents to choose from a variety of models to govern the entire system based on its current state. This approach effectively addresses and overcomes the dynamic challenges inherent in the ecosystem.

Furthermore, addressing the limitations within the MLSysOps project involves harnessing the power of ML techniques. The application of ML is instrumental in enhancing security by incorporating well-established anomaly detection models. This robust security measure complements the amalgamation of agents and microservices. Similarly, performance optimization and reduction of overhead are achieved through continuous monitoring system state and efficient manage of the resources facilitated by ML. This dynamic orchestration of services serves to preemptively mitigate potential performance constraints, ensuring a seamless operational experience.

While it is acknowledged that the implementation of such agents may demand a certain level of complexity and specialized skills, it is imperative to recognize that this limitation extends beyond system behavior. Emphasizing the evolutionary leap represented by this promissory advancement can effectively dispel concerns about complexity, encouraging broader adoption of this architecture.

5. Conclusion

In the era of expanding IoT ecosystems, where the inclusion of devices and services is a daily occurrence, effectively dealing with this vast, dynamic, and diverse environment poses a significant challenge. In this direction, the exploitation of software agents holds great promise to facilitate the automation of this complex management landscape. Our analysis has delved into the utilization of agents within SOA and microservice architectures, shedding light on their advantages and limitations.

Upon comparison, it becomes evident that incorporating agents into a microservices scenario aligns seamlessly with the distributed nature of the IoT. This integration brings forth essential functionalities such as autonomy in resource management, flexibility, adaptability, integration, and collaboration, which are instrumental in realizing the vision of autonomous computing. Nevertheless, certain limitations persist, presenting opportunities for refinement.

The MLSysOps framework, as presented, signifies a significant stride in addressing these challenges. The amalgamation of agents and microservices, coupled with the incorporation of ML techniques, promises to embed intelligence into the automation of the dynamic and heterogeneous DEC continuum. This integrated approach not only harnesses the inherent capabilities of agents and microservices but also addresses and surpasses the previously identified limitations.

The future work involves the practical implementation of the MLSysOps framework and rigorous validation through numerous tests using real-world distributed IoT applications. This real-world validation will not only attest to the feasibility of the proposed framework but also contribute valuable insights for further enhancement. As we navigate the future of IoT ecosystems, the synergistic integration of agents, micro-services, and ML techniques stands as a promising avenue for overcoming the intricacies of managing this ever-expanding landscape.

Acknowledgments

References

[1] 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: 2015 10th Computing Colombian Conference (10CCC), IEEE, 2015, pp. 583–590.

[2] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, Y. Al-Hammadi, The evolution of distributed systems towards microservices architecture, in: 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), IEEE, 2016, pp. 318–325.

[3] T. Cerny, M. J. Donahoo, M. Trnka, Contextual understanding of microservice architecture: current and future directions, ACM SIGAPP Applied Computing Review 17 (2018) 29–45.

[4] P. Lotfallahtabrizi, Y. Morgan, Comparing autonomy and collaboration between agent-oriented architecture and service-oriented architecture, in: 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), IEEE, 2017, pp. 25–31.

[5] R. L. Hartung, Service oriented architecture and agents: Parallels and opportunities, Agent and Multi-agent Technology for Internet and Enterprise Systems (2010) 25–48.

[6] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, Y. Al-Hammadi, Iot applications: From mobile agents to microservices architecture, in: 2018 International conference on innovations in information technology (IIT), IEEE, 2018, pp. 117–122.

[7] C. Savaglio, M. Ganzha, M. Paprzycki, C. Bădică, M. Ivanović, G. Fortino, Agent-based internet of things: State-of-the-art and research challenges, Future Generation Computer Systems 102 (2020) 1038–1053.

[8] P. Krivic, P. Skocir, M. Kusek, G. Jezic, Microservices as agents in iot systems, in: Agent and Multi-Agent Systems: Technology and Applications: 11th KES International Conference, KES-AMSTA 2017 Vilamoura, Algarve, Portugal, June 2017 Proceedings 11, Springer, 2017, pp. 22–31.

[9] L. Ribeiro, J. Barata, A. Colombo, Mas and soa: A case study exploring principles and technologies to support self-properties in assembly systems, in: 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, IEEE, 2008, pp. 192–197.

[10] A. A. Khaleq, I. Ra, Development of qos-aware agents with reinforcement learning for autoscaling of microservices on the cloud, in: 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), IEEE, 2021, pp. 13–19.

[11] F. M. Brazier, J. O. Kephart, H. V. D. Parunak, M. N. Huhns, Agents and service-oriented

computing for autonomic computing: A research agenda, IEEE Internet Computing 13 (2009) 82–87.

[12] P. Vrba, V. Mařík, P. Siano, P. Leitão, G. Zhabelova, V. Vyatkin, T. Strasser, A review of agent and service-oriented concepts applied to intelligent energy systems, IEEE transactions on industrial informatics 10 (2014) 1890–1903.

[13] F. Siqueira, J. G. Davis, Service computing for industry 4.0: State of the art, challenges, and research opportunities, ACM Computing Surveys (CSUR) 54 (2021) 1–38.

[14] Y. Kalyani, R. Collier, Hypermedia multi-agents, semantic web, and microservices to enhance smart agriculture digital twin, in: 2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), IEEE, 2023, pp. 170–171.

[15] T. Daradkeh, A. Agarwal, Modeling and optimizing micro-service based cloud elastic management system, Simulation Modelling Practice and Theory 123 (2023) 102713.

[16] R. W. Collier, E. O'Neill, D. Lillis, G. O'Hare, Mams: Multi-agent microservices, in: Companion proceedings of the 2019 world wide web conference, 2019, pp. 655–662.