Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 19 No. 24 (2025) | OPEN ACCESS

https://doi.org/10.3991/ijim.v19i24.56299

PAPER

Enhancing Distributed System Performance with an Intelligent Multi-Agent Microservices Architecture

Sara Zouad[1,2] (✉), **Mahmoud Boufaida**[1]

[1]LIRE Laboratory, Abdelhamid Mehri – Constantine 2 University, Constantine, Algeria

[2]Computer Sciences Department, Larbi Ben Mhidi University, Oum-El-Bouaghi, Algeria

sara.zouad@univ-constantine2.dz

ABSTRACT

In a rapidly evolving digital landscape, distributed systems are becoming increasingly complex, making it essential to adopt architectures that can adapt in real-time. Business process optimization now depends on intelligent, dynamic, and modular systems. To address this need, we propose a hybrid architecture called multi-agent microservices (MAMS), which merges the principles of microservices with those of multi-agent systems (MAS). Each microservice is represented by an autonomous agent that can discover, compose, and collaborate with other services through REST interfaces while adapting in a decentralized manner to contextual changes. The uniqueness of our approach lies in integrating deep reinforcement learning (DRL) into the decision-making process. This technique enables agents to learn how to flexibly select optimal service combinations based on quality of service metrics, such as availability, response time, and reliability. Unlike traditional methods, our system ensures continuous adaptation to load fluctuations or service failures. The main contributions of this work include intelligent and autonomous distributed orchestration, decision-making driven by adaptive learning, and significant improvements in flexibility, resilience, and efficiency of services in complex environments. A case study on travel planning demonstrates the effectiveness of our solution, which surpasses monolithic architectures in personalization, user satisfaction, and operational efficiency.

KEYWORDS

business processes, microservices, multi-agent systems (MAS), service composition, deep reinforcement learning (DRL), interoperability, quality-of-service, REST

1 INTRODUCTION

In a constantly changing economy, optimizing business processes [1] has become a vital strategic tool for maintaining competitive advantage. Automating these processes is now essential for managing the increasing complexity of inter-system interactions, minimizing human error, and boosting productivity. Major technological

Zouad, S., Boufaida, M. (2025). Enhancing Distributed System Performance with an Intelligent Multi-Agent Microservices Architecture. *International Journal of Interactive Mobile Technologies (iJIM)*, 19(24), pp. 76–93. https://doi.org/10.3991/ijim.v19i24.56299

Article submitted 2025-04-29. Revision uploaded 2025-09-15. Final acceptance 2025-09-17.

breakthroughs, particularly in web services [2], multi-agent systems (MAS) [3], and semantic technologies [4], have enabled this transformation.

However, integrating these technologies simultaneously within a distributed, adaptive, and real-time operational architecture still presents significant challenges. In particular, combining the MAS and microservices paradigms [5] creates complex issues related to the dynamic orchestration of services [6], ensuring semantic consistency among diverse components, and enabling autonomous adaptation to changing contexts. Existing approaches often address these areas separately or within static architectures, which limits their ability to meet the needs of modern environments.

In this context, our research presents an innovative hybrid architecture, called multi-agent microservices (MAMS) [7], which aims to overcome these limitations. This architecture rests on three technological pillars:

1. Intelligent agents for autonomous decision-making,
2. Semantic web services for dynamic discovery and integration,
3. Microservices for greater modularity and scalability.

The originality of our approach stems from integrating deep reinforcement learning (DRL) [8]–[9], which enables agents to learn how to optimally assemble services based on quality factors (response time, availability, reliability) while continuously adapting to user preferences and environmental conditions.

By addressing the challenges of distributed orchestration, semantic interoperability, and contextual adaptation, this contribution presents a comprehensive and innovative solution applicable across various fields, including tourism, healthcare, and education, where responsiveness and personalization are crucial.

This paper is divided into six sections. In Section 2, we review related works. Section 3 describes the methods and materials used in this study. Section 4 outlines the proposed architecture, comprising various components. In Section 5, we evaluate the results and demonstrate the architecture's performance through a series of experiments. Finally, in Section 6, we present our conclusions and outline some directions for future work.

2 SOME RELATED WORKS

The combination of MAS, microservice architectures, and DRL provides a promising approach to developing autonomous and adaptable systems that can continuously improve their performance. This technological synergy enables the creation of intelligent and flexible solutions suited to distributed and evolving environments.

This integrated approach provides companies with a powerful tool for optimizing their business processes amidst rising demands for operational efficiency, flexibility, and responsiveness. It enables automation and dynamic adaptation to change while maintaining greater organizational agility.

The MAMS framework enhances existing approaches by introducing dynamic QoS-aware orchestration, driven by three key innovations: (1) semantic service composition using reinforcement-learning-based agents that interpret formal specifications at runtime, (2) self-adaptive transformation techniques to ensure semantic consistency during topology changes, and (3) combined static semantic verification and runtime validation checks to enforce QoS constraints.

These advancements establish MAMS as a more comprehensive and context-aware solution, as summarized in the comparison below and illustrated in Table 1.

Table 1. Comparison of existing approaches

Reference	Main Objective	Methodology	Results	Limits
Pereira et al [10]	Efficient deployment of microservices in a Kubernetes multi-cluster	RL agents with DeepSets network for multi-cluster placement	Significant reduction in latency and deployment costs (32x improvement)	The complexity of the model is dependent on the quality of the training data
Peddinti et al [11]	Optimizing microservices orchestration for system efficiency	RL with MDP, Kubernetes integration, hierarchical learning	Reduced response times, better resource allocation	High cost of calculation, complexity of agent training
Qi Si et al [12]	Dynamic cloud resource allocation for microservices	Multi-agent DRL with performance prediction (PAUDS)	Reduce resource consumption while maintaining QoS	Collecting and modelling complex data
Cui et al [13]	Autoscaling of Mobile Edge Computing (MEC) based on MARL	Multi-agent RL for autoscaling	Improved scalability and reduced latency	Focus on autoscaling, less on semantic orchestration
Zhang [14]	Automated architectural composition via multi-agent DRL	Multi-agent DRL for architectural spatial composition	Efficient composition automation with dynamic adaptation	Specific application to spatial architecture, necessary extension to microservices
Acar et al [15]	Improving MAS interoperability via standardized microservice interfaces	Encapsulation of agents as Docker containers with RESTful APIs; runtime platforms for monitoring and security	Encapsulation of agents as Docker containers with RESTful APIs; runtime platforms for monitoring and security	Encapsulation of agents as Docker containers with RESTful APIs; runtime platforms for monitoring and security

The MAMS architecture differs from previous approaches by integrating decentralized microservices that act as autonomous agents, while also combining QoS criteria with semantic consistency in service selection and composition. Furthermore, incorporating DRL techniques allows for the refinement of orchestration processes, leading to an overall enhancement in system performance.

3 METHODS AND MATERIALS

This Section offers a brief overview of the current research landscape in MAMS architecture, covering key terms and concepts that will be used throughout the article.

3.1 Agent paradigm

Agents are autonomous computer entities that can interact with their environment and independently pursue goals [16]. In MAS, multiple agents work together to solve complex tasks that are often divided among different entities or areas of expertise. MAS's strength, effectiveness, and ability to promote interoperability in distributed environments with fragmented data, control, and knowledge set it apart.

These features make it an ideal solution for issues with web-based service technologies, such as system opening, relationship complexity, and resource decentralization.

More specifically, MAS are well-suited for modeling the dynamic composition of web services, where components must be actively selected, arranged, and coordinated based on user needs and available resources.

This work aims to leverage MAS capabilities to develop a flexible coordination and composition mechanism that allows agents to collaborate effectively with web services. The goal is to create a system that can respond to user requests in a dynamic and relevant way while also adapting to various distributed environments.

3.2 Microservices techniques

A few years ago, microservices architectures gained significant popularity, and major companies such as Amazon, Uber, and Netflix began implementing them to build their information systems [17]–[18]. The core idea of this approach is to split applications into separate, independent components called microservices, each dedicated to a specific function.

Unlike monolithic systems, where all modules are combined into a single software entity, microservices are designed to work independently while collaborating to reach a common goal. This modularity simplifies development, maintenance, and, most importantly, deployment in distributed and complex environments.

The similarities between the microservices concept and MAS, where autonomous entities collaborate to provide services to the user, are quite apparent. Microservices can be viewed as modern software agents with enhanced interoperability and resilience.

Scalability, fault tolerance, and technological flexibility are essential benefits of microservices. Each microservice can be developed using a different programming language or environment, tailored to the specific needs. Furthermore, this architecture encourages loosely coupled systems, where modifications to one component do not impact others. This enables faster development of new features, independent testing, and continuous deployment.

3.3 Combination of multi-agents and microservices

The MAMS architecture combines microservices architecture with MAS to offer a new approach for designing complex systems in distributed environments.

This method aims to enhance coordination and collaboration among microservices while maintaining increased system flexibility and adaptability by leveraging the intelligence and independence of agents.

A key advantage of MAMS is its capacity to provide enhanced modularity, increased scalability, and resilience to cumulative patches. MAMS improves overall system efficiency by integrating agents into the microservices architecture, enabling decentralized decision-making and more intelligent service orchestration.

Each microservice in this model is represented by an agent microservice (AMS), which functions as an autonomous agent with specific capabilities. These AMSs are packaged as independent microservices and communicate with each other using standardized protocols such as RESTful APIs. They can exchange information, negotiate, and collaborate to achieve shared goals.

Depending on the system's needs, agent communication can be either synchronous or asynchronous. This offers considerable flexibility in managing interactions and adapting to performance limits.

3.4 Web service composition

Web service composition involves integrating multiple independent web services to create a new, more complex service that performs a specific task or meets a particular need. The process includes choosing appropriate services, defining their inputs and outputs, and specifying how they should be connected.

Web service composition is typically performed using service-oriented architectures (SOA), which provide a standardized approach to developing and integrating web services. SOA enables the creation of modular, reusable services that can be easily combined and reconfigured to form new composite services.

The main advantages of web service composition include greater flexibility, scalability, and interoperability, as well as the ability to create tailored solutions by combining existing services instead of building everything from scratch. However, there are also challenges to web service composition, such as the need for effective service discovery, service matching, and service orchestration.

Web service composition is an important area of research and development in web services. Many approaches and tools are available to support the process of creating and managing composite services.

The combination of MAS and microservices, using DRL tools for web service composition, is a promising approach that aims to enhance the efficiency and scalability of web services while enabling them to adapt to changing requirements and environments. However, this approach is still in its early stages of development, and several challenges need to be addressed before it can reach its full potential.

3.5 DRL techniques

Deep reinforcement learning is a specialized branch of machine learning that combines reinforcement learning (RL) algorithms with deep neural networks. DRL has been applied in various fields, including robotics, gaming, and natural language processing.

In web service composition, DRL can enhance the process by using environmental feedback. For instance, an agent might use DRL to choose the optimal sequence of services to meet a user's request, taking into account current service availability.

The use of DRL in web service composition can also address the issue of QoS optimization by learning to select services based on QoS metrics such as response time, availability, and reliability. Moreover, DRL can learn to adapt to changing environments by continuously updating the service composition based on new information.

3.6 Adopted methodology

The development of the system proposed in this work follows a refined and structured methodology, adapted from [19], to meet the needs of intelligent and distributed environments. This improved methodology used in this research is divided into four consecutive phases: analysis, architectural design, implementation, and experimental evaluation. Each phase is designed to address the constraints of intelligent, distributed systems while ensuring modularity, interoperability, and real-time adaptability of the proposed system, as shown in Figure 1.

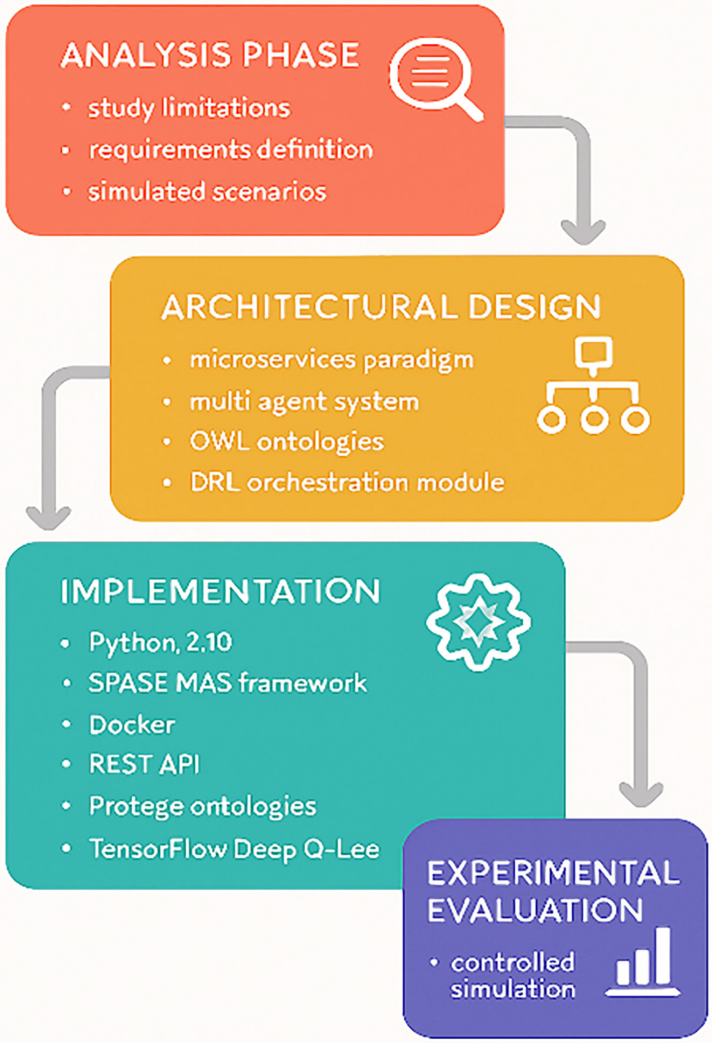

Fig. 1. The adopted methodology

Analysis phase. This phase aims to identify the limitations of traditional architectures (monolithic or centralized), especially concerning scalability, resilience, adaptability, and interoperability. Functional requirements are outlined in terms of dynamic discovery, component autonomy, intelligent service composition, and compliance with QoS standards. Non-functional requirements include modularity, reusability, flexibility, compatibility across different environments, and support for runtime modifications.

Simulated scenarios help define agents, microservices, and their expected interactions.

Architectural design. The overall structure is built on three supporting principles:

- The microservices paradigm guarantees functional decomposition, isolation, scalability, and independent maintenance of services.
- The multi-agent model enables decentralized decision-making, autonomous components, and flexible interaction management.
- Web Ontology Language (OWL) ontologies are used to formalize metadata, ensure semantic interoperability between agents, and enrich contextual understanding.

Additionally, a DRL module has been integrated into the agent orchestrator, responsible for learning optimal service composition strategies based on quality criteria and execution contexts.

Implementation. The system implementation is built on a consistent technology stack.

- Language: Python 3.10. [20]
- Multi-agent framework: SPADE (Smart Python Agent Development Environment). [21]
- Microservices & orchestration: Docker & Docker Compose. [22]
- Inter-agent communication and services: REST API.
- Semantic modeling: Ontologies designed with Protégé. [23]
- Machine learning: Deep Q-Learning algorithms implemented [24] with TensorFlow 2.0. [25]

Agents are contained within autonomous microservices, each able to perform specialized tasks, self-describe, and interact flexibly with other services through REST interfaces.

Experimental evaluation. System evaluation relies on controlled simulation where synthetic user profiles interact with agents. The system is tested in dynamic scenarios with changes in load, service availability, and user preferences.

Performance is assessed based on several quantitative indicators:

- Average response time;
- Service completion rate;
- Composition accuracy;
- Use of system resources (CPU, memory);
- Degree of personalization;
- Simulated user satisfaction.

The results are compared with a similar version in monolithic architecture to highlight the improvements in efficiency, flexibility, and adaptability enabled by multi-agent microservices.

4 AN ARCHITECTURE BASED ON MAS, MICROSERVICES, AND DRL

As shown in the previous section, the MAMS architecture utilizes the principles of microservices and MAS to improve the effectiveness of the web services model in multi-agent systems. It uses technologies such as REST to enable interactions among agents and facilitate communication between components (microservices and agents). The architecture is designed to meet QoS requirements and can enhance its performance over time by integrating DRL algorithms. This enables agents to learn from their interactions with the environment and optimize their behavior based on rewards or penalties, thereby supporting informed decision-making and enhancing QoS metrics such as response time, reliability, and availability.

The architecture components work together to deliver personalized, intelligent, and adaptive services while ensuring high performance and scalability. The architecture, shown in Figure 2, consists of the following key components.

- **UI** (user interface), which acts as the entry point for user interactions
- **TMS** (task manager service) handles task management and assignment

- **SDA** (service discovery agent), which finds and manages available agents
- **AMS** (agent microservices), specialized agents that carry out specific tasks

Furthermore, **SCA** (service composer agent) intelligently designs and optimizes services through reinforcement learning methods.

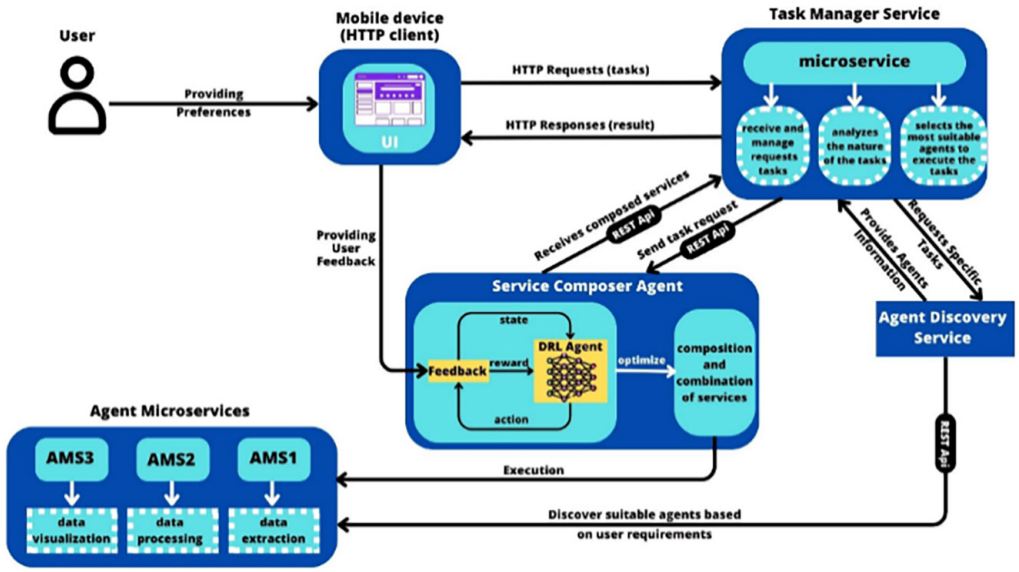

Fig. 2. The overall architecture of MAMS

4.1 Description of the MAMS Architecture

This section describes each component of the multi-agent microservices.

User interface (UI). This is the system's front end, which communicates with the system by sending requests to the task manager service and receiving the results of the composed web services.

Task manager service (TMS). This microservice handles task requests from the user interface and assigns them to the correct agent for execution. It also tracks task progress and sends status updates back to the user interface.

Service discovery agent (SDA). This microservice is responsible for detecting all available agents in the system. It uses an ontology to represent and organize information about these agents. By leveraging the ontology, the SDA can deliver a more efficient and effective service discovery process, enhancing the overall performance and scalability of the system.

Agent microservices (AMS). We define three types of AMS, each responsible for specific tasks, such as:

- AMS1 (agent microservice 1) handles data extraction, involving collecting and retrieving data from different sources.
- AMS2 (agent microservice 2) handles data processing, including transforming and analyzing the collected data.
- AMS3 (agent microservice 3) handles data visualization by presenting processed data in a clear and intuitive way.

The notation AMSi refers to a generic agent that is assigned dynamically to a specific task within a process. Similarly, AMSn denotes the last agent in the chain, tailored to the type of service requested.

This modular approach enables the creation of tailored services by flexibly combining agents based on each user's specific needs.

Service composer agent (SCA). It is one of the agents in the MAMS architecture responsible for combining the services of other agents to deliver a final web service. It interacts with the other agents and determines the best combination of services to meet the needed functionality. This agent uses DRL algorithms to train the agents in selecting and combining services based on the environment, thereby enhancing the agents' performance over time.

5 BEHAVIOR OF THE SYSTEM

The behavior of the MAMS system relies on distributed coordination among agents, managed by the SCA. The SCA oversees the entire dynamic service composition process, considering contextual constraints (such as latency, availability, and overload) and QoS objectives.

5.1 Specification of interaction protocol

The system follows a clearly defined sequence:

- The user submits a request through the interface (UI).
- The TMS receives the request, processes it, and then sends it to the SCA.
- The SCA consults the SDA to identify available agents, their current capabilities, and constraints (through semantic metadata).
- A composition decision is made by the SCA, relying on a DRL algorithm trained to optimize the combination of services based on QoS metrics.
- The selected services (AMS1, AMS2, and AMS3) are called in an optimal order, and the results are reassembled by the SCA and sent back to the user via the TMS.

5.2 Composition process

At this stage in the SCA, functionalities from multiple agents are combined to create a new service that can fulfill a user's request. For example, if a user wanted to extract data from a website and visualize the results, SCA would decide which parts of those features to use for each step of the process (e.g., select AMS1 for data extraction and AMS3 for data visualization).

Then, these services are combined to create a new web service using various technologies, including service matching, selection, and optimization, as shown in Figure 3. This process may involve the most efficient and effective way to transfer data between different agents, optimize the order of service execution, and handle errors or failures that can occur during service composition.

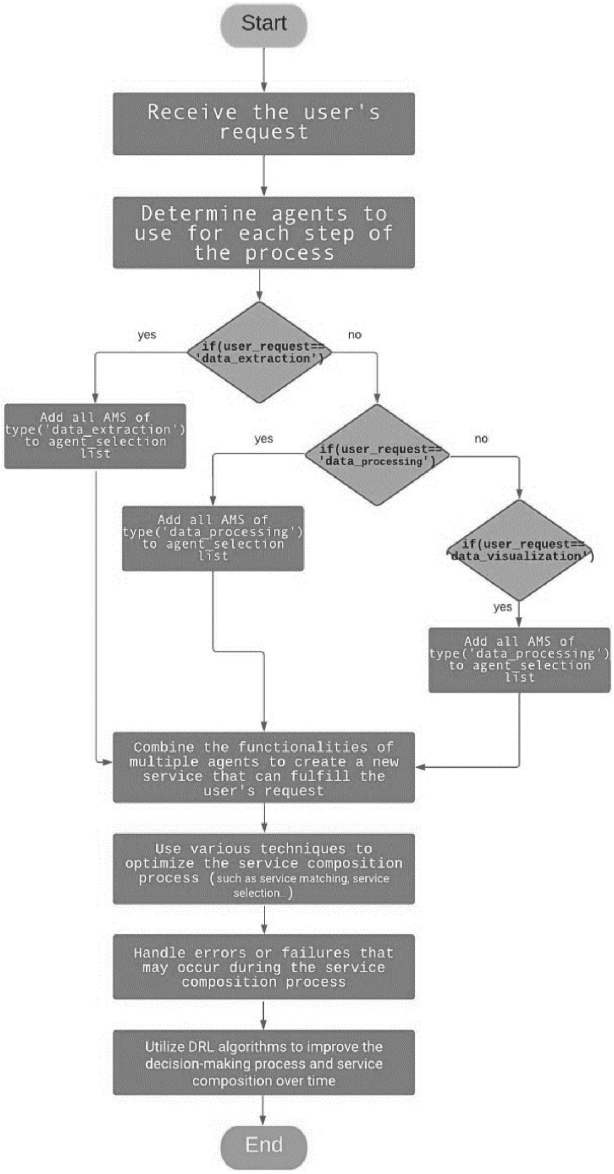

Fig. 3. The flowchart illustrating the service composition process

The practical implementation of the MAMS architecture depends on a dynamic, multi-stage process for creating and managing semantic web services. This process includes four main stages, each playing a crucial role in ensuring the system's efficiency and flexibility.

Discovery stage. Currently, the SDA gathers and organizes information about available AMS. It also identifies each agent's capabilities, costs, and constraints (such as region or availability). In our system, all AMS1-type agents are designed to collect user preference data.

Composition stage. At this stage, the SCA decides how the relevant AMS can be combined to create a travel planning service in response to the user's request. The SCA might use reinforcement learning to optimize the weights assigned to different QoS metrics.

The agent would consider the weight vector W as part of the state it learns.

The action space consists of combinations of services C and possible weight increments ΔW.

The environment evaluates (C, W) samples and provides a reward based on user feedback, calculated as:

$$Reward = Score(C) - l \cdot |ActualQoS(C) - TargetQoS(W)|$$

Here, Score is the user rating, and *TargetQoS* is the ideal *QoS* profile as defined by W.

The agent trains a policy network $\pi(C, W)$ using Q-learning [24]:

$$Q(C, W, C', \Delta W') \leftarrow Q + \alpha[R + \gamma \cdot \max_{A'} Q(C', W', A') - Q(C, W, A)]$$

It explores random major C and incremental major delta major W and chooses higher major Q combinations over time.

It samples more (C, W) pairs and refines the weight values that maximize average user rewards.

For new requests, it exploits the optimal π to select high-scoring combinations.

This end-to-end learning enables the agent to customize QoS weighting based on users' value.

The Q-network can estimate the expected long-term reward.

- The Q-network has an input layer for state features (request, weights) and an output for each action (combination).
- Internal layers learn complex nonlinear representations of states.
- During training, the agent samples batches of (state, action, reward, next_state) tuples from the replay memory.
- For tuples where next_state is non-terminal:
- The target Q-value is calculated using the Bellman equation [26]:

$$Q_{\text{target}} = \text{reward} + \gamma \cdot \max_{a'} Q(\text{next\_state}, a')$$

Here, γ is the discount factor, and Q(next_state, a') comes from the target Q-network [27].

- For terminal next_states, the target is simply the reward.
- It calculates the loss as the mean squared error between the current Q-network output and the target.
- Backpropagation [28] is used to update the Q-network weights to minimize this loss through gradient descent [28]
- Over multiple iterations, the Q-network learns to accurately predict the discounted cumulative reward for state-action pairs based on experiences.
- This enables it to assess options without exhaustively sampling every combination.

So, the Q-network approximates the optimal action-value function through reinforcement learning based on interactions.

Execution stage. The TMS assigns tasks to an appropriate AMS. Each agent performs its role, while the TMS oversees task execution and status, and updates the user interface with progress to keep the user informed in real-time.

Optimization stage. In its final stage, the SCA will use feedback from execution to apply DRL techniques for improving future designs. This includes identifying which agents best contribute to specific tasks and enhancing decision-making based on feedback from users, usage trends, and overall QoS metrics like response time, reliability, and availability.

6 CASE STUDY AND DISCUSSION

In its final phase, the SCA will use feedback from implementation and apply DRL techniques to improve future compositions. This involves identifying which agents most effectively contribute to specific tasks and optimizing decision-making based on user feedback, usage trends, and general QoS metrics such as response time, reliability, and availability.

This section will explain and demonstrate the details of these steps using the travel planning system example to validate the relevance and effectiveness of the MAMS approach in real-world scenarios. As shown in Figure 4, travel planning involves various activities, including searching for flights, booking hotels, renting cars, and suggesting activities or attractions at the destination.

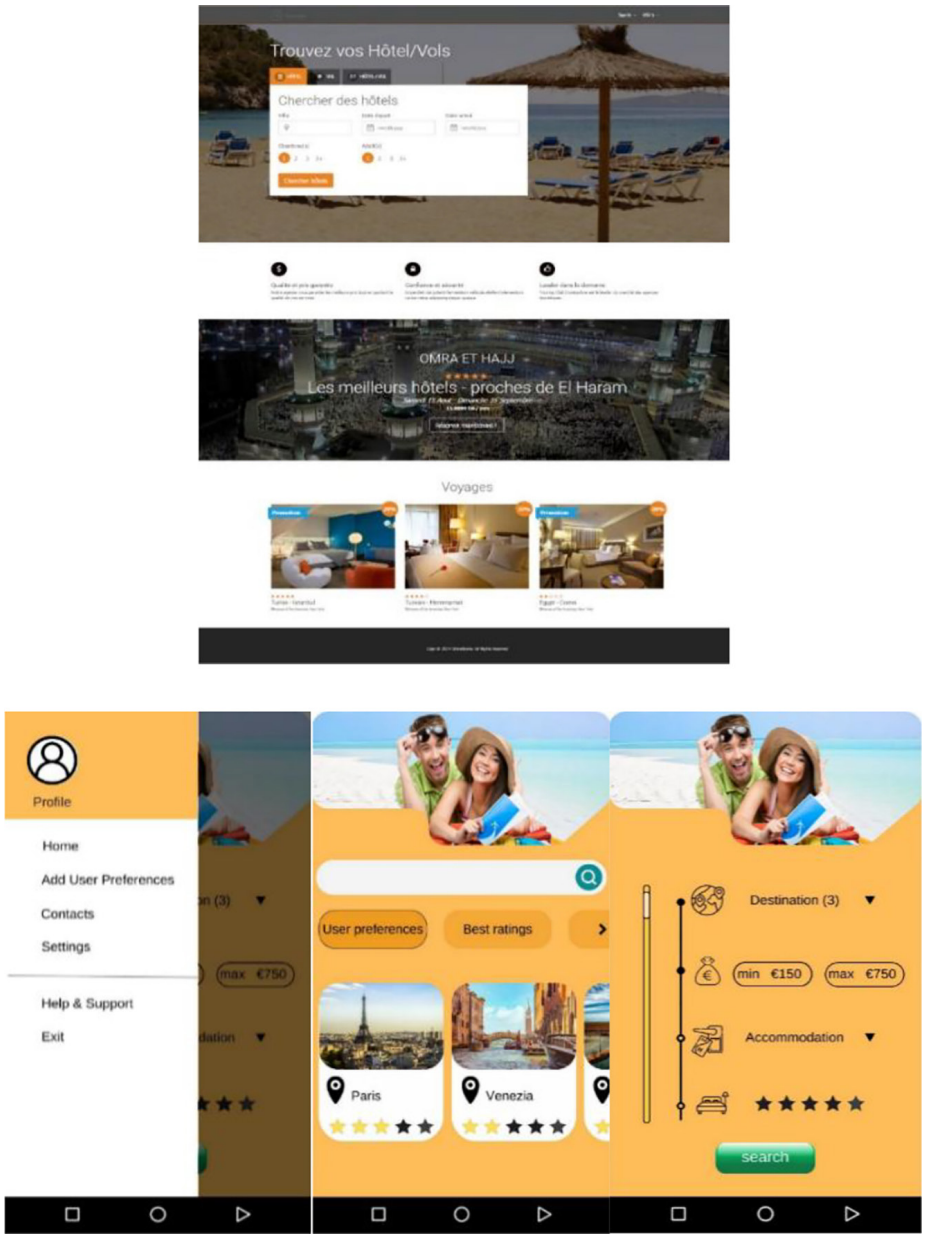

Fig. 4. A user interface of the travel planning system

Through agents and microservices, the MAMS architecture helps automate and personalize travel planning by handling tasks such as searching, combining, and executing travel services based on individual preferences and needs, such as budget and interests. It also manages and organizes different agents handling various cases into a single travel planning system.

7 PERFORMANCE EVALUATION OF THE SYSTEM

To evaluate the system's performance, we established four criteria, randomly simulated preferences for 100 users' travel services, and measured the following:

- Completion rate: indicates the percentage of tasks the system successfully completes for a user request.

$$\text{Completion Rate} = \frac{\text{Completed Tasks}}{\text{Total Requested Tasks}}$$

- Response time: This indicates the total duration the system takes to reply to a user's request. It is measured from the moment the request is entered until the system provides results to the user.
- Accuracy rate: This refers to the percentage of recommended travel plans that align with the user's requirements and preferences.
- Efficiency: It measures resource utilization, such as CPU and memory usage.

Table 2. Key performance metrics of the MAMS framework

Completion Rate	Response Time	Accuracy Rate	Efficiency
100%	2 min	93%	80%

Based on the results shown in Table 2, our system responds to any request, on average, within a maximum of two minutes. The accuracy of a recommendation provided by the system is evaluated based on the user's actual preferences.

As shown in Figure 5, the algorithm's calculated accuracy was 93%. The review also evaluated resource usage to measure the system's efficiency. Based on our observation, the system effectively utilizes approximately 70% of the available resources.

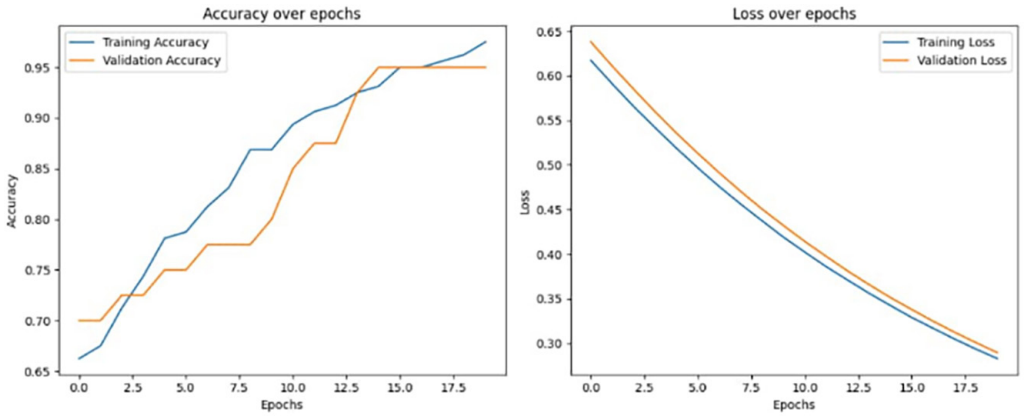

Fig. 5. Accuracy and loss over training epochs

In addition to the above, different QoS measures can also be used to evaluate the performance of a MAMS system.

- Throughput: The requests processed per second by the system.
- Availability: The percentage of time the system is operational and working properly.
- Reliability: The capacity to perform consistently without failure or errors.
- Scalability: The ability to handle more traffic and requests.

The results obtained are shown in Table 3.

Table 3. QoS metrics

Throughput	Availability	Reliability	Scalability
1150 requests/sec	99.9%	80% success rate	Up to 10,000 concurrent users

Using these metrics, we can compare the performance of microservices in MAMS architecture with that of traditional monolithic architectures, as shown in Table 4.

Table 4. MAMS vs. Monolithic architecture performance comparison

Metric	MAMS	Monolithic	Error ± (IC 95%) MAMS
Response time (ms)	1759.84	2671.65	141.48
Precision (%)	92.52	84.80	7.72
Completion rate (%)	100.01	91.67	2.17
CPU usage (%)	64.67	82.71	2.90
Throughput (requests/sec)	1141.93	841.99	40.02
User satisfaction (/5)	4.58	3.82	0.17

Note: Table of values with 95% confidence intervals.

All differences are statistically significant with p-values less than 0.001.

The confidence intervals display clear margins, confirming that MAMS's performance is significantly better than the monolithic architecture on all indicators.

The scatter plot shown in Figure 6 illustrates the strengths and trade-offs of both architectural approaches in terms of accuracy. The MAMS architecture demonstrates higher accuracy (%), which may qualify it for applications requiring highly accurate and reliable results. We also examine the criteria of user satisfaction and personalization.

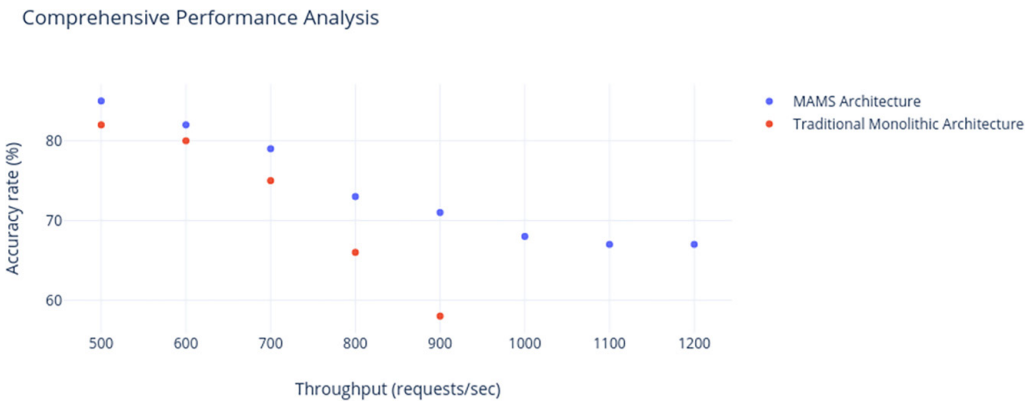

Fig. 6. Accuracy vs. Throughput: MAMS vs. Monolithic

- User satisfaction: It reflects how pleased users are after receiving the recommended travel plan. The system's user satisfaction rate is 90%, based on user feedback where users give 4 out of 5 stars, showing that users generally value the recommendations provided.

$$User\ Satisfaction\ Rate = (\sum User\ Ratings/(Nbr\ or\ Users \times Maximum\ Rating)) \times 100$$

- Personalization: Measures how accurately recommendations align with the user's preferences. It displays a personalization score of 70%, meaning the recommendations are fairly well tailored to the user.

$$Personalization\ Rate = (Total\ Nbr\ of\ User\ Preferences/Nbr\ of\ Matched\ Preferences \times 100)$$

7.1 Dynamic scenario testing and results

To demonstrate the system's robustness and adaptive intelligence, we simulated various scenarios typical of a real distributed environment.

Failure scenarios and limitations.

Scenario 1: Microservice failure (AMS unavailable)

When an AMS assigned to a specific task is temporarily unavailable or produces an error:

- The SCA immediately consults the SDA to find alternative agents with equivalent function.
- The DRL model updates its policy by emphasizing more dependable options moving forward.
- An adaptive retry/backoff mechanism is activated to reduce unnecessary overload.

Scenario 2: High network latency or slow response times

The SCA continuously tracks response times:

- If latency exceeds a critical threshold, the affected service is temporarily de-prioritized in the decision-making process.
- The DRL uses this data as a penalty in its reward function, steering the learning process toward more efficient combinations.

Scenario 3: System overload (spikes in concurrent requests)

During a large influx of users:

- The TMS uses a queuing system with contextual prioritization, such as for authenticated users.
- The DRL adjusts compositions by decreasing the number of agents involved.
- Without compromising perceived QoS ("graceful degradation" approach).
- Load indicators are monitored in real-time within the states analyzed by the Deep reinforcement learning.

Observed success scenarios.

Scenario 1: Adjusting to user preferences

Thanks to DRL, the system has learned to prefer certain suppliers for users who care about cost or comfort, boosting the satisfaction rate to 90%.

Scenario 2: Improved response time

The distributed composition allowed a response time of under 2 seconds to be maintained in 85% of cases, even under load.

Scenario 3: Resilience

In case a service becomes unavailable (e.g., flight API fails), the SDA provides an equivalent alternative to ensure continuity.

8 CONCLUSION

This study introduces an innovative architecture called MAMS for the dynamic composition of semantic web services in distributed environments. By integrating the modularity of microservices, the autonomy of multi-agent systems, and the adaptive learning abilities of DRL, MAMS facilitates intelligent, real-time service orchestration that meets both QoS requirements and user preferences. Key technical contributions include self-learning orchestration guided by a continuously trained DRL model, decentralized decision-making for optimized service selection based on evolving QoS criteria, and semantic interoperability maintained through ontologies to ensure composition consistency. Future plans aim to enhance MAMS as a robust foundation for intelligent service orchestration across various domains. These include adapting it for IoT and edge environments with lightweight agents that incorporate pre-trained models, integrating federated learning to protect privacy while enabling collaborative learning, and customizing for specific domains like healthcare, education, and smart cities through tailored ontologies and reward functions. Additionally, security-focused orchestration policies will strengthen its use in critical applications. Ultimately, MAMS paves the way for autonomous, scalable, and context-aware service ecosystems.

9 REFERENCES

[1] M. Hammer, "What is business process management?" in *Handbook on Business Process Management 1*. in International Handbooks on Information Systems, J. vom Brocke and M. Rosemann, Eds., Springer, Berlin, Heidelberg, 2014, pp. 3–16. https://doi.org/10.1007/978-3-642-45100-3_1

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web services," in *Web Services: Data-Centric Systems and Applications*, Springer, Berlin, Heidelberg, 2004, pp. 123–149. https://doi.org/10.1007/978-3-662-10876-5_5

[3] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018. https://doi.org/10.1109/ACCESS.2018.2831228

[4] M. Hepp, P. De Leenheer, A. De Moor, and Y. Sure, *Ontology Management: Semantic Web, Semantic Web Services, and Business Applications*. New York, NY: Springer Science & Business Media, 2007. https://doi.org/10.1007/978-0-387-69900-4

[5] J. Jordanov, D. Simeonidis, and P. Petrov, "Containerized microservices for mobile applications deployed on cloud systems," *Int. J. Interact. Mob. Technol. (iJIM)*, vol. 18, no. 10, pp. 48–58, 2024. https://doi.org/10.3991/ijim.v18i10.45929

[6] M. Fahad, N. Moalla, and Y. Ourzout, "Dynamic execution of a business process via web service selection and orchestration," *Procedia Comput. Sci.*, vol. 51, pp. 1655–1664, 2015. https://doi.org/10.1016/j.procs.2015.05.299

[7] S. Zouad and M. Boufaida, "Using multi-agent microservices for a better dynamic composition of semantic web services," in *Proceedings of the 4th International Conference on Advances in Artificial Intelligence*, 2020, pp. 47–52. https://doi.org/10.1145/3441417.3441423

[8] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Inf. Fusion*, vol. 85, pp. 1–22, 2022. https://doi.org/10.1016/j.inffus.2022.03.003

[9] M. M. Al-Nawashi, O. M. Al-hazaimeh, N. M. Tahat, N. Gharaibeh, W. A. Abu-Ain, and T. Abu-Ain, "Deep reinforcement learning-based framework for enhancing cybersecurity," *Int. J. Interact. Mob. Technol. (iJIM)*, vol. 19, no. 3, pp. 170–190, 2025. https://doi.org/10.3991/ijim.v19i03.50727

[10] J. Santos *et al.*, "Efficient microservice deployment in Kubernetes multi-clusters through reinforcement learning," in *NOMS 2024–2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–9. https://doi.org/10.1109/NOMS59830.2024.10575912

[11] S. R. Peddinti, B. K. Pandey, A. Tanikonda, and S. R. Katragadda, "Optimizing microservice orchestration using reinforcement learning for enhanced system efficiency," *Distrib. Learn. Broad Appl. Sci. Res. Annu.*, vol. 7, 2021. https://ssrn.com/abstract=5119917

[12] Q. Si, J. Shi, W. Li, X. Lu, and P. Pu, "DeepMRA: An efficient microservices resource allocation framework with deep reinforcement learning in the cloud," in *Advanced Intelligent Computing Technology and Applications, ICIC 2024*, in Lecture Notes in Computer Science, D. S. Huang, X. Zhang, and Y. Pan, Eds., vol. 14863, Springer, Singapore, 2024, pp. 455–466. https://doi.org/10.1007/978-981-97-5581-3_37

[13] L. Cui, T. Shi, R. Lu, and T. Zhang, "Autoscaling in mobile edge computing based on multi-agent reinforcement learning," in *Proceedings of the 2023 9th International Conference on Communication and Information Processing*, 2023, pp. 520–527. https://doi.org/10.1145/3638884.3638966

[14] Z. Zhang, Z. Guo, H. Zheng, Z. Li, and P. F. Yuan, "Automated architectural spatial composition via multi-agent deep reinforcement learning for building renovation," *Autom. Constr.*, vol. 167, p. 105702, 2024. https://doi.org/10.1016/j.autcon.2024.105702

[15] B. Acar *et al.*, "OPACA: Toward an open, language-and platform-independent API for containerized agents," *IEEE Access*, vol. 12, pp. 10012–10022, 2024. https://doi.org/10.1109/ACCESS.2024.3353613

[16] H. S. Nwana and D. T. Ndumu, "An introduction to agent technology," in *Software Agents and Soft Computing Towards Enhancing Machine Intelligence: Concepts and Applications*, 2005, pp. 1–26. https://doi.org/10.1007/3-540-62560-7

[17] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, and Culture*. Sebastopol, CA: O'Reilly Media, Inc., 2016.

[18] J. Jordanov, D. Simeonidis, and P. Petrov, "Containerized microservices for mobile applications deployed on cloud systems," *Int. J. Interact. Mob. Technol. (iJIM)*, vol. 18, no. 10, pp. 48–58, 2024. https://doi.org/10.3991/ijim.v18i10.45929

[19] S. Zouad and M. Boufaida, "An agent-oriented methodology for business process management," in *Business Modeling and Software Design, BMSD 2020*, in Lecture Notes in Business Information Processing, B. Shishkov, Ed., Springer, Cham, 2020, pp. 287–296. https://doi.org/10.1007/978-3-030-52306-0_19

[20] A. Downey, *Think Python*. Sebastopol, CA: O'Reilly Media, Inc., 2012.

[21] J. Palanca, J. A. Rincon, C. Carrascosa, V. Julián, and A. Terrasa, "A flexible agent architecture in SPADE," in *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation, The PAAMS Collection, PAAMS 2022*, in Lecture Notes in Computer Science, F. Dignum, P. Mathieu, J. M. Corchado, and F. De La Prieta, Eds., vol. 13616, 2022, pp. 320–331. https://doi.org/10.1007/978-3-031-18192-4_26

[22] R. Smith, *Docker Orchestration*. Birmingham, UK: Packt Publishing Ltd., 2017.

[23] A. Stadnicki, F. F. Pietroń, and P. Burek, "Towards a modern ontology development environment," *Procedia Comput. Sci.*, vol. 176, pp. 753–762, 2020. https://doi.org/10.1016/j.procs.2020.09.070

[24] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, pp. 279–301, 2020. https://doi.org/10.1146/annurev-statistics-031219-041220

[25] V. Silaparasetty, *Deep Learning Projects Using TensorFlow 2*. Berkeley, CA: Apress, 2020. https://doi.org/10.1007/978-1-4842-5802-6

[26] V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013. https://doi.org/10.48550/arXiv.1312.5602

[27] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the deep Q-network," *arXiv preprint arXiv:1711.07478*, 2017. https://doi.org/10.48550/arXiv.1711.07478

[28] T. Frerix, T. Möllenhoff, M. Moeller, and D. Cremers, "Proximal backpropagation," *arXiv preprint arXiv:1706.04638*, 2017. https://doi.org/10.48550/arXiv.1706.04638

10 AUTHORS

Sara Zouad is a professor in the Department of Computer Science at the University of Oum El Bouaghi, Algeria. She earned her bachelor's degree in 2004 and her master's degree in Computer Science in 2007. Since 2012, she has been teaching a variety of courses, including Advanced Databases, Information Systems, Statistical Data Processing, Artificial Intelligence tools, and Computer Vision. Her research interests include Artificial Intelligence, Multi-Agent Systems, Semantic Web Services, Business Informatics, and intelligent distributed architectures. She has published works in international conferences and journals (E-mail: sara.zouad@univ-constantine2.dz).

Mahmoud Boufaida is an emeritus professor of computer science at the Faculty of New Information and Communication Technologies at Constantine 2 University in Algeria. After leading the Distributed Computing laboratory for several years, he now heads the "Information Systems and Knowledge Bases" research group within the same lab. His research mainly focuses on cooperative information systems, business processes, advanced databases, and service-oriented architectures. He has also worked in computer-assisted learning, software engineering, and human-computer interfaces. He has initiated and coordinated numerous national and international research projects on information system interoperability and application integration in businesses. He is a founding member of the Algerian Academy of Sciences and Technologies (E-mail: mahmoud.boufaida@univ-constantine2.dz).