Article

A Framework for Enhancing Multi-Agent Systems: Leveraging Microservices and DLT to Optimize Message Overhead and Load Balancing

Ching Han Chen [1], Ming Fang Shiu [1,*]

[1] Department of Computer Science and Information Engineering National Central University, Taiwan; pierre@g.ncu.edu.tw (C.H.C); 108582003@cc.ncu.edu.tw (M.F.S.)
* Correspondence: 108582003@cc.ncu.edu.tw; Tel.: +886-3-4207151 ext.35211

Abstract: This article delves into the deployment of Multi-Agent Systems (MAS) within the realm of distributed computing. It capitalizes on the flexibility afforded by the publish-subscribe architecture and employs the holonic agent structure to adeptly model complex systems. The study introduces a framework utilizing this architecture to address the critical challenges of message overhead and load balancing—key to the reliability of expansive systems. Drawing inspiration from the decentralized systems that underpin cryptocurrency technologies, it seeks to promote independent and scalable interactions among agents. The MAS framework is enriched by principles of Microservices and Distributed Ledger Technology, emphasizing durable and efficient operations for intricate tasks. The article lays out the foundational design phase and sets the stage for subsequent experiments aimed at enhancing communication and operational efficacy within the system.

Keywords: Multi-Agent Systems, Distributed Artificial Intelligence, Publish-Subscribe Architecture, Decentralized Computing

1. Introduction

Advancements in AI and computing hardware have spurred the adoption of deep learning networks in cloud and edge computing, enabling the integration of distributed intelligent nodes into unified services. Central to this is Multi-Agent Systems (MAS), a key element of Distributed Artificial Intelligence (DAI) [1], leveraging autonomous agents to achieve common goals [2]. This paper presents a practical framework for distributed computing environments using MAS's publish-subscribe architecture. We demonstrate its effectiveness in fostering collaboration among distributed intelligent agents. MAS technology, with its parallel computation, robustness, scalability, cost-effectiveness, and reusability, is ideal for large-scale systems [3]. However, challenges like message overhead and load balancing require sophisticated strategies such as message compression, QoS adjustments, and consensus algorithms [4].

Message overhead is a common issue in Many-to-One communications, where multiple client agents communicate with a single agent, leading to unnecessary message traffic as clients sift through irrelevant messages [5]. Load balancing also poses challenges in One-to-Many scenarios, such as optimizing response efficiency and fault tolerance within a cluster. We address these challenges through a dynamic decentralized framework [6].

Microservices refine MAS technology by offering a scalable, adaptable architecture compatible with holonic and publish-subscribe models [7]. Breaking down complex systems into smaller services mitigates message overhead and facilitates load balancing

via distributed processing, enhancing robustness and resource management. Microservices support various communication patterns and service discovery mechanisms, effectively addressing MAS's design challenges.

Distributed Ledger Technology (DLT) enhances MAS by promoting autonomy, scalability, and secure transactions without centralized control [8]. Adopting decentralized consensus principles, DLT enables autonomous decision-making and adaptation, boosting system resilience and efficiency. DLT's immutable record-keeping and distributed consensus algorithms reduce communication overhead and facilitate local data processing, marking a strategic advancement for dynamic coordination, secure interactions, and efficient decentralized operations in MAS.

Our approach develops a Multi-Agent Systems model based on the holonic agent concept, integrating Microservices and Distributed Ledger Technology (DLT) to address communication challenges. For Many-to-One communication, Microservices improve scalability and manageability, ensuring efficient message distribution. DLT tackles One-to-Many challenges by ensuring secure, consensus-driven interactions among agents. Microservices for flexibility and DLT for secure aggregation, validating the framework's efficiency in managing many-to-many communications within a decentralized MAS model.

The thesis explores integrating Multi-Agent Systems in distributed computing. The Introduction outlines the research problem and objectives. The Design section details the MAS model's development, emphasizing publish-subscribe architecture and holonic agents. The Experiment section rigorously tests system performance, focusing on message overhead and load balancing. The Discussion analyzes outcomes and challenges, and the Conclusion summarizes findings and future research directions.

2. Design

The design chapter explores the integration of Multi-Agent Systems in distributed computing, focusing on message overhead and load balancing. We present a strategy leveraging Microservices for adaptability and Distributed Ledger Technology for secure, consensus-based interactions. Using class diagrams and algorithms, we detail the structure and dynamics of our framework, illustrating component interactions and operationalizing our strategies. Focused experiments assess the design's effectiveness in reducing message overhead and enhancing load balancing, connecting theoretical groundwork with practical implementation.

2.1. Holonic Structures in Multi-Agents System

Multi-Agent Systems (MAS) effectively address issues involving multiple computational units, with each unit designated as an agent [9]. MASs form networks of interacting agents that pool resources to tackle complex challenges beyond the capacity of individual agents. Among various organizational frameworks, holonic MAS are advantageous for constructing intricate AI systems [10]. A holon is a stable entity integrating sub-holons within a broader system, originally describing social dynamics in biological species [11]. This concept, with its hierarchical organization and interactive capabilities, has been applied to simulate behaviors in manufacturing and business industries [12]. Within a MAS, what appears as a single agent may consist of multiple subagents.

Each holon is managed by a head agent capable of environmental interaction and communication. In homogeneous MASs, head agent selection can be arbitrary, as in rotating leadership in wireless sensor networks. In heterogeneous setups, selection depends on agents' distinct capabilities. Holons may merge to form superholons, as

shown in Fig. 1, where agents H-21 and H-31 coordinate communication with H-1, while H-4 is an atomic agent without subagents.

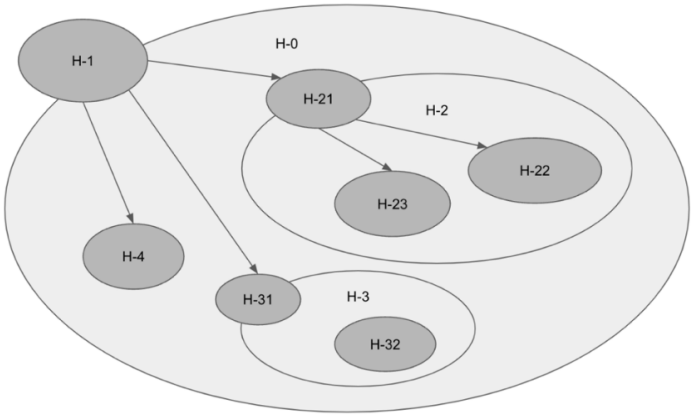


Fig. 1 Example of a holonic MAS

2.2. Foundational design

The holonic MAS architecture, mirroring physiological systems, effectively simulates biological tissues. This study uses holonic MAS to design complex, intelligent agents that integrate various perceptions and actions. The resulting HolonicAgent highlights this sophisticated integration.

The research encapsulates the Publish-Subscribe paradigm within the abstract `MessageBroker` class, ensuring flexibility with essential operations like start, stop, publish, and subscribe. Specific implementations, including `MQTT_Broker`, `DDS_Broker`, and `ROS_Broker`, cater to different communication frameworks and scenarios. This modular approach allows interchangeability of messaging systems.

To simplify HolonicAgent complexity, the BrokerNotifier interface serves as an intermediary for MessageBroker interactions, promoting loose coupling and modularity. Fig. 2 illustrates these setups, enhancing agent agility and manageability. The next phase will tackle advanced communication patterns.

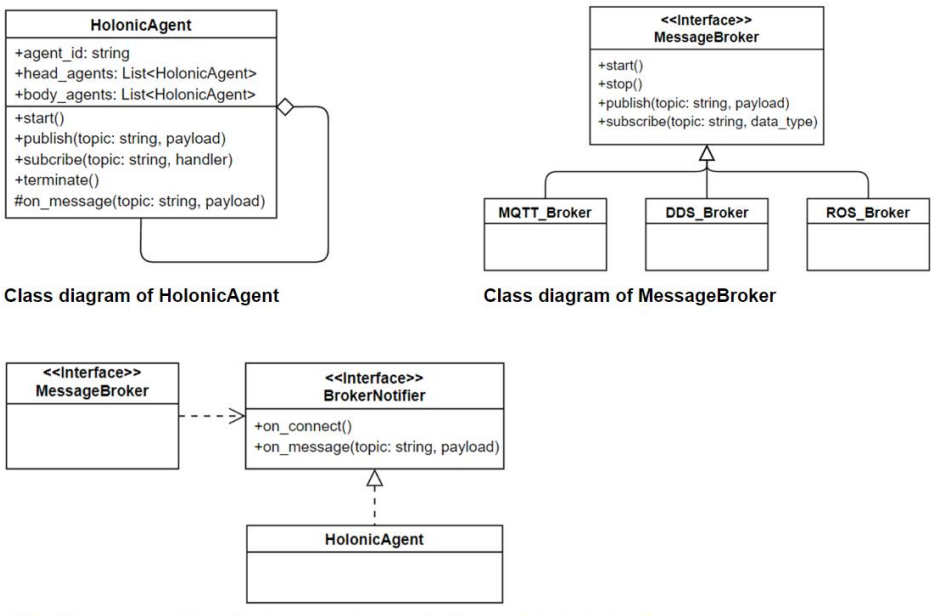


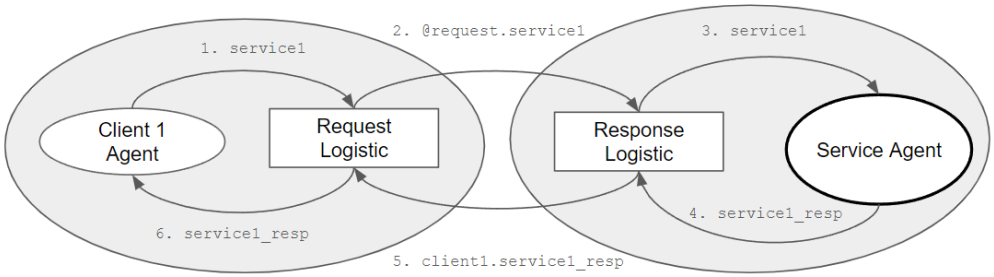
Fig. 2 Class diagrams of the holonic MAS architecture.

Building upon these foundational designs, the subsequent phase will address the challenges associated with advanced communication patterns.

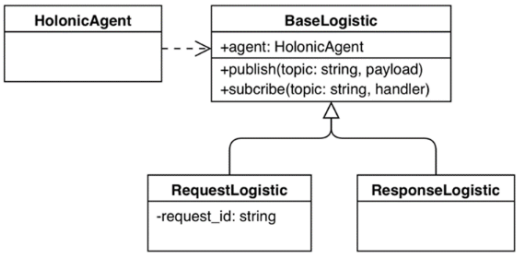
2.3. Tackling Many-to-One Communication Efficiency

In Multi-Agent Systems, the Many-to-One communication model addresses a request-response problem within a publish-subscribe communication architecture. In this scenario, all clients receive every message and then filter out the unnecessary ones, causing excessive data traffic.

To solve the challenge in Multi-Agent Systems without disrupting the agents' original logic, we introduced Logistic objects as selective couriers. These ensure messages reach only the intended clients, maintaining agents' process integrity. Centrally managing request-response interactions, these objects enhance communication efficiency and integrate seamlessly with existing infrastructure, avoiding extensive redesigns. Serving as intermediaries, the Request and Response Logistics manage message topics and facilitate effective bidirectional communication, shown in Fig. 3. This method ensures messages reach their recipients without broadcasting to all clients, maintaining system effectiveness with minimal modifications to agent workflows.



Logistic Request-Response in Many-to-One communication model.



Class diagram of logistic.

Fig. 3 Many-to-One communication model with logistic objects.

The Many-to-One Communication model enhances network efficiency by using Logistics objects to ensure messages reach only intended clients, preventing congestion and allowing seamless integration with current systems.

2.4. Strategies for One-to-Many Communication Challenges

In Multi-Agent Systems, One-to-Many communications can cause conflicts when two service agents respond simultaneously, complicating coordination. Therefore, efficient load balancing and dynamic agent participation are crucial. We use decentralized coordination to optimize system responsiveness and scalability. By leveraging distributed architectures, agents can dynamically engage and disengage based on real-time demands, maintaining network equilibrium and enhancing performance.

Distributed Ledger Technology (DLT) enhances decentralized coordination by maintaining a consistent record of transactions through replication and computational trust, ensuring transparency and security without a central authority. Using consensus algorithms, DLT autonomously determines the executing agent within a dynamic group, ensuring equitable load distribution and operational efficiency. A specially designed logistic object acts as an intermediary, managing task allocation to optimize load balancing across the cluster (Fig. 4). This approach allows agents to focus on primary tasks without managing communication and load distribution, improving system efficiency and responsiveness.

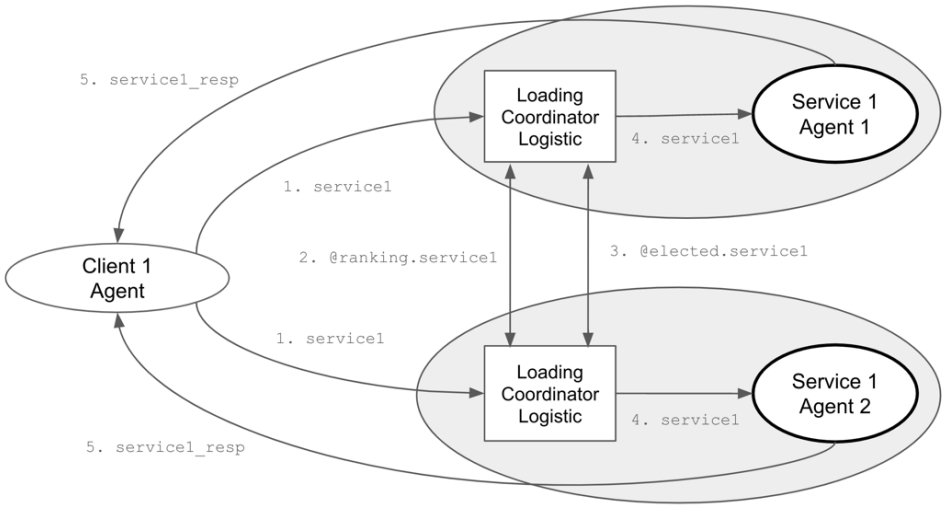


Fig. 4 Loading Coordinator Logistic in One-to-Many communication model.

Every LoadingCoordinator employs the following consensus mechanism to identify the optimal executing agent, initiated by a coordinator who subscribes to specific topics to monitor new tasks and gather load-based rankings from various agents. Upon the arrival of a new task, the coordinator conducts an election by distributing its own ranking to assert its candidacy for handling the task. It then collects and aggregates rankings from all participating agents, which reflect each agent's current load and availability. Using a consensus algorithm, the coordinator determines the most suitable leader for the task, based on the lowest rank. If the coordinator's agent is elected, it directly undertakes the task; otherwise, it remains on standby, ready for future task assignments and elections. This process enhances scalability and efficiency in distributed computing environments. Here is the consensus algorithm:

1. **Subscription Setup**: Upon initialization, the LoadingCoordinator subscribes to specific topics to listen for new tasks and ranking information.
2. **Task Arrival**: When a new task arrives (start), the coordinator begins an election to determine which agent should handle the task, publishing its own ranking based on its current load and a random factor.
3. **Rank Collection**: As rankings from other agents arrive (rank), they are collected and stored.
4. **Leader Determination**: After a brief period to allow all ranks to be submitted, the coordinator determines the leader (determine) based on the lowest rank.
5. **Task Assignment**: If the responsible agent is elected, it will throw out the next request (if any), repeat this consensus algorithm, and begin the task.

This decentralized approach, which is a consensus algorithm, ensures that tasks are allocated fairly and efficiently, leveraging the system's distributed nature to optimize performance and responsiveness. The ranking algorithm is based on the contributions of the service agents. The coordination process duration varies depending on the number of service agents involved. Utilizing Distributed Ledger Technology, if other tasks arise before coordination is completed, they will be recorded in a first-in, first-out manner for subsequent processing. The elected leader of the current round will initiate the retrieved tasks for a new round of work coordination.

2.5. Integration Strategies for Many-to-Many Communications

The Many-to-Many communication model in Multi-Agent Systems combines selective messaging from Many-to-One with dynamic coordination from One-to-Many. Using Request-Response and LoadingCoordinator logistics (Fig. 5), this architecture guides messages and manages load balancing among multiple agents. The Request-Response Logistics handle point-to-point communication, ensuring accurate and efficient request-response matching. Simultaneously, the LoadingCoordinator Logistic orchestrates task distribution among service agents based on current load and capacity, optimizing network operations.

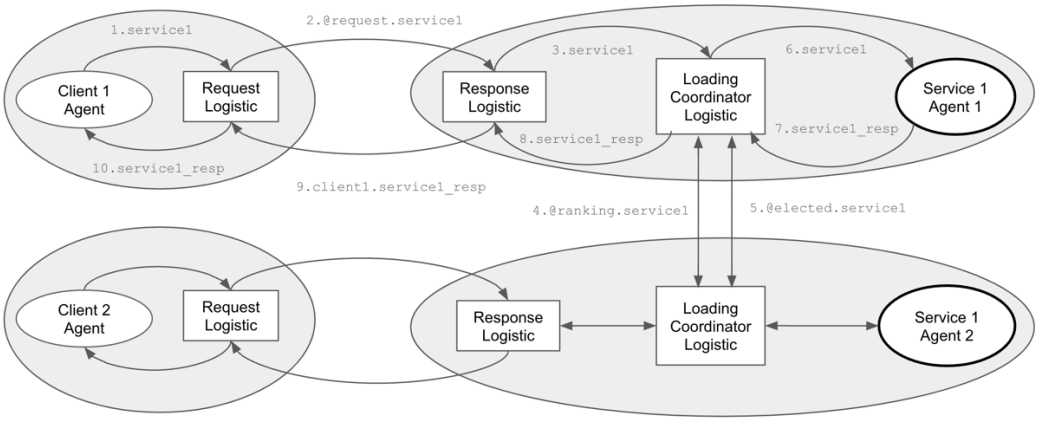


Fig. 5 Sequence Diagram of Integration

This model is particularly beneficial in sectors like smart grids, healthcare, and supply chain management. It optimizes interactions among numerous entities, improving responsiveness and operational efficiency. By facilitating precise communication and equitable task distribution, it enhances the scalability and flexibility of distributed systems.

3. Experiment

In this experiment, we evaluate the efficacy of our Multi-Agent Systems architecture, integrating Microservices and Distributed Ledger Technology for optimized message distribution and load balancing. Through methodical tests, we measure key performance indicators like message overhead and load distribution efficiency, providing empirical data to validate our design and highlight areas for future enhancement.

3.1. Experiment 1: Evaluating Request-Response Efficiency in Reducing Message Overhead

To investigate the performance differences between a standard publish-subscribe communication method and a many-to-one technique in a system where multiple client

agents request services from a single service agent, we design an experiment with the following setup:

3.1.1 Experiment Design

Objective:

Compare the response time and message throughput between normal communication and many-to-one techniques under varying loads from multiple client nodes.

Parameters:

1. Number of Client Nodes: 10, 50, 100 (to test under different scales).
2. Messages per Client: 100, 500, 1000 (to vary the load).
3. Message Size: Fixed at 1 KB to maintain consistency in data size across tests.
4. Message Frequency: 1 message per second per Client.
5. Service Efficiency: 100 messages per second.

Metrics to Measure:

1. Response Time: Time taken from sending a request to receiving a response.
2. Resource Usage: CPU and memory utilization of the service node.

3.1.2 Result and Observations

Table 1. Comparison for both the normal and many-to-one communication

Clients/Messages	Communication Method	Response Time (ms/message)	Total Response Messages	Increased CPU Usage (%)	Increased Memory Usage (GB)
10 Clients/ 100 Messages	Normal	20.81	1000	1.42	0.54
	Many-to-One	21.57	100	1.95	0.36
50 Clients/ 500 Messages	Normal	53.23	25000	12.27	0.78
	Many-to-One	22.78	500	6.85	0.88
100 Clients/ 1000 Messages	Normal	2768.60	100000	14.02	1.67
	Many-to-One	25.24	1000	9.39	1.84

Key Observations:

1. The "Normal" communication method experiences substantial increases in Response Time due to processing bottlenecks caused by a high volume of Total Response Messages. For example, 100 clients generate 100,000 messages, significantly slowing response times. In contrast, the "Many-to-One" method maintains stable, low response times by handling only the necessary messages, reducing overhead and improving efficiency.
2. The "Many-to-One" method shows a moderate increase in CPU usage due to fewer messages but a slight increase in memory usage from the logistic mechanism. This balance efficiently manages resources while effectively handling communications.

Results show that many-to-one communication maintains lower response times and CPU usage despite increased memory usage due to logistic mechanisms, demonstrating its efficiency under various client loads.

3.2. Experiment 2: Evaluating the Coordination Strategies on Response Efficiency

This study assesses the LoadingCoordinator logistic model's effectiveness in managing high-volume requests from a single client to multiple service agents. By testing different numbers of agents and request complexities, it aims to optimize response times and resource utilization.

3.2.1 Experiment Design

Objective:

To evaluate the efficiency and effectiveness of the LoadingCoordinator logistic model in managing a high volume of requests from a single client to multiple service agents in a One-to-Many communication model, focusing on optimizing response times and resource utilization.

Parameters:

1. Number of Service Agents: 1, 10, 50, 100 - To test scalability and the ability to handle high loads across an increasing number of service agents. The single agent count serves as a control group.
2. Number of Requests: Fixed at 1000 - Consistent high load to assess how the system handles continuous heavy demand.
3. Request Complexity: Processing time per request is fixed at 1 second, focusing on assessing the impact of the number of service agents on response efficiency.
4. Inter-Arrival Time of Requests: Fixed at 0.01 seconds - To ensure a continuous and consistent stream of incoming requests.

Metrics to Measure:

1. Response Time: Time taken from sending a request to receiving a response.
2. Total Processing Time: Total time to complete all requests.
3. Resource Usage: CPU and memory utilization of the service node.

3.2.2 Result and Observations

Table 2. Comparison for both the normal and one-to-many communication

Clients / Services 1000 messages	Total Processing Time (ms)	**Efficiency per Unit Service (%) *Note 1**	**Increased CPU Usage (%)**	**Increased Memory Usage (GB)**
1 Clients / 1 Service	1,005,106	99.12	0.83	0.209
1 Clients / 10 Services	118,229	84.58	2.08	0.479
1 Clients / 50 Services	34,554	58.70	7.21	1.040
1 Clients / 100 Services	56,335	17.75	14.20	2.313

Note 1:

- m = Number of messages = 1,000
- p = Service processing time per message = 1,000 milliseconds
- s = Number of services
- t = Total processing time
- Efficiency per Unit Service = $\frac{m \times p}{s \times t} \times 100\%$

Key Observations:

1. **Total Processing Time**: Increasing services from 1 to 50 significantly reduces total processing time, dropping from over 1 million milliseconds to about 34,554 milliseconds, highlighting improved parallel processing. However, processing time increases with 100 services, indicating potential overhead or resource contention issues.

2. **Efficiency per unit service:** Efficiency inversely correlates with the number of services. It is highest at 99.12% with one service but drops to 18.47% with 100 services, suggesting diminishing returns due to increased complexity and coordination overhead.

3. **Increased CPU and Memory Usage**: As services increase from 1 to 100, CPU usage rises from 0.83% to 14.20%, and memory usage grows from 0.209 GB to 2.313 GB, indicating higher resource demands.

In conclusion, increasing services improves processing time up to a point but also increases CPU and memory usage and decreases efficiency. Finding an optimal balance between resource usage and performance is crucial for one-to-many communication models.

5. Conclusions

This research integrates Multi-Agent Systems with Microservices and Distributed Ledger Technology to enhance distributed computing. Our framework improves scalability, robustness, and efficiency by strategically deploying these technologies. It features a holonic agent organization with a publish-subscribe model, enabling flexible system design for complex agent interactions.

Microservices minimize message overhead and enhance load balancing by breaking systems into smaller, autonomous services. This modular approach improves resource efficiency and supports dynamic scaling. Load balancing enables decentralized consensus and interactions, boosting system resilience and privacy while reducing central control risks. Experimental results show our framework effectively addresses MAS challenges in communication and responsiveness. Future research will refine these integrations, focusing on AI-driven decision-making, fault tolerance, and new consensus algorithms.

References

1. Chaib-Draa, B., Moulin, B., Mandiau, R., & Millot, P. (1992) Trends in distributed artificial intelligence. *Artificial Intelligence Review*, *6*(1), 35-66. https://doi.org/10.1007/BF00155579
2. Balaji, P. G., & Srinivasan, D. (2010) An introduction to multi-agent systems. In *Innovations in multi-agent systems and applications-1*, pp 1-27. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-14435-6_1
3. Gaud, N., Galland, S., Gechter, F., Hilaire, V., & Koukam, A. (2008) Holonic multilevel simulation of complex systems: Application to real-time pedestrians simulation in virtual urban environment. *Simulation Modelling Practice and Theory*, *16*(10), 1659-1676. https://doi.org/10.1016/j.simpat.2008.08.015
4. Van Glabbeek, R., Deac, D., Perale, T., Steenhaut, K., & Braeken, A. (2022). Flexible and Efficient Security Framework for Many-to-Many Communication in a Publish/Subscribe Architecture. In *Sensors, 22*(19), 7391. MDPI. https://doi.org/10.3390/s22197391
5. Ferraz Junior, N., Silva, A. A. A., Guelfi, A. E., & Kofuji, S. T. (2022). Performance evaluation of publish-subscribe systems in IoT using energy-efficient and context-aware secure messages. In *Journal of Cloud Computing, 11*(6). SpringerOpen. https://doi.org/10.1186/s13677-022-00278-6
6. Nandagopal, M., Uthariaraj, V.R. (2011). Decentralized Dynamic Load Balancing for Multi Cluster Grid Environment. In *Meghanathan, N., Kaushik, B.K., Nagamalai, D. (eds) Advanced Computing. Communications in Computer and Information Science*, vol 133. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17881-8_15

7. Hannousse, A., Yahiouche, S. (2021). Securing microservices and microservice architectures: A systematic mapping study. In *Computer Science Review, 41*. Elsevier. https://doi.org/10.1016/j.cosrev.2021.100415
8. Perdana, A., Robb, A., Balachandran, V., Rohde, F. (2021). Distributed Ledger Technology: Its Evolutionary Path and the Road Ahead. *In Information & Management, 58*(3). Elsevier. https://doi.org/10.1016/j.im.2020.103316
9. Khosla, R., & Ichalkaranje, N. (2004) *Design of intelligent multi-agent systems: human-centredness, architectures, learning and adaptation* (Vol. 162). Springer Science & Business Media.
10. Rodriguez, S., Hilaire, V., Gaud, N., Galland, S., & Koukam, A. (2011) Holonic multi-agent systems. In *Self-organising Software,* pp 251-279. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-17348-6_11
11. Koestler, A. (1967) The ghost in the machine, hutchinson & co. *Publishers Ltd., London*.
12. Giret, A., & Botti, V. (2005) Analysis and design of holonic manufacturing systems. In *18th International Conference on Production Research (ICPR2005)*.