ANALYSIS OF COMMUNICATION PROTOCOLS IN MULTI-AGENT SYSTEMS

DRASKO, B. & RAKIC, K.

Abstract: *The use of Multi-Agent Systems (MAS) in different fields of implementation comes with challenges. Some systems are generally more complex, while some do not require robust infrastructure. In both cases, choosing the appropriate MAS communication protocol is essential. This paper aims to list and explain the key protocols such as FIPA-ACL, KQML, MQTT, AMQP, WebRTC, and CoAP. Analysis and comparison between protocols will be presented by showing their strengths and weaknesses. For example, FIPA-ACL and KQML are flexible in agent coordination, while lightweight protocols like MQTT and CoAP are ideal for resource-constrained systems such as IoT. Additionally, the paper discusses the performance and security trade-offs of using peer-to-peer protocols like WebRTC and robust messaging systems like RabbitMQ. The paper will provide guidelines for selecting the appropriate protocol based on system requirements, considering efficiency and reliability in MAS applications.*

Key words: *MAS, FIPA-ACL, KQML, MQTT, AMQP*

Authors´ data: Dipl.-Ing. **Drasko**, B[oris]*; Assist. Prof. Dr. Sc. **Rakic**, K[resimir]*, * University of Mostar, Trg hrvatskih velikana 1, 88000, Mostar, Bosnia and Herzegovina, boris.drasko@sum.ba, kresimir.rakic@fsre.sum.ba

This Publication has to be referred as: Drasko, B[oris] & Rakic, K[resimir] (2024). Analysis of Communication Protocols in Multi-Agent Systems, Chapter 08 in DAAAM International Scientific Book 2024, pp.103-116, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-42-6, ISSN 1726-9687, Vienna, Austria
DOI: 10.2507/daaam.scibook.2024.08

1. Introduction

When designing a MAS, the question arises as to which protocol to use considering various circumstances such as the complexity and robustness of the system and available resources. The purpose of this paper is to provide a detailed analysis of multi-agent protocols used in inter-agent communication so that practitioners can easily evaluate and compare different protocols and accordingly decide which best suits their needs when designing MAS.

Every Agent in MAS has a specific goal and an assigned task. For MAS to work efficiently Agents need to exchange information, coordinate action and ultimately make decisions based on the information collected, to achieve collective goals set. To simplify, Agents need a way to communicate with each other, thus making communication a fundamental component of MAS. Communication protocols for MAS have been developed over the years, each designed to meet specific requirements in various operational environments.

Historically, Agent Communication Language (ACL) such as FIPA-ACL was developed by the Foundation for Intelligent Physical Agents (FIPA), a standards organization established to promote interoperability among agent-based systems. It is one of the most widely used protocols for agent communication. It supports communicative acts such as requesting, informing, and proposing, thus facilitating flexible and complex agent interaction (FIPA, 1997).

Another ACL developed in the early 1990s by researcher Tim Finin and his team is KQML (Knowledge Query and Manipulation Language). It was designed for knowledge sharing and query exchange, but it was replaced by FIPA-ACL due to its wider adoption and clearer semantics (Finin et. al., 1994).

Some of the non-ACL protocols that are widely used for Agent Communication are MQTT (Message Queuing Telemetry Transport), AMQP (Advanced Message Queuing Protocol), WebRTC (Web Real-Time Communication) and CoAP (Constrained Application Protocol).

It’s important to note that the difference between ACLs and non-ACLs is that the ACLs were primarily developed with a focus on inter-agent communication based on speech act theory (Austin, 1962).

In essence, it means that these protocols were based on human interaction and can handle complex interactions such as agreements, negotiations and task delegation between agents (Rygallo et al., 2009).

Non-ACLs, on the other hand, are not capable of complex semantic understanding but focus on data transmission and reliable messaging.

2. Key Communication Protocols in MAS

We can arrange communication protocols in MAS by categories, as follows:

- Agent Communication Languages (ACL): FIPA-ACL, KQML.
- Publish-Subscribe Protocols: MQTT, AMQP.
- Peer-to-Peer (P2P) Protocols: WebRTC.
- Request-Response Protocols: HTTP/REST, CoAP.

- Shared Memory Systems: Tuple Spaces.
- Messaging Systems: JADE, RabbitMQ.
- Security Protocols: TLS, Blockchain-Based.

In the following sections, a detailed analysis of each protocol by category is presented, the mode of functioning, their advantages and disadvantages are explained, and a comparison of existing protocols is given.

3. Agent Communication Languages

3.1 FIPA-ACL

FIPA ACL uses a performative messaging system to relay not only the data but also the intent behind it. A performative message carries intent behind it in the form of a *request*, *inform*, *propose*, *confirm*, or *reject*. An agent who receives the message can thus recognize the intent behind the message and can make informed decisions and respond to that intent based on the context of communication.

Structure of FIPA-ACL message includes: (FIPA, 2002):

- *Performative*. Defines the intended purpose of the message (e.g., request, inform, query, propose).
- *Sender*. Identifies the agent sending the message.
- *Receiver*. Identifies the recipient(s) of the message.
- *Content*. The actual information being exchanged or the subject of the message.
- *Language*. Specifies the language used to express the content (e.g., Prolog, XML, …).
- *Ontology*. Defines the vocabulary and the domain of discourse that both agents must understand for meaningful communication.
- *Protocol*. Specifies the interaction protocol to manage the sequence of message exchanges (e.g., a negotiation or auction protocol).
- *Conversation ID*. Allows the agents to identify and track a specific conversation.
- *Reply With*. Used when the sender expects a response to the message.
- *In Reply To*. Provides a reference to the original message when sending a response.
- *Reply By*. A timestamp indicating when a reply is expected.

FIPA-ACL can be implemented with use of various programming languages including Java, Python, C++, Prolog. It is best used in complex systems distributed multi-agent systems where autonomous agents need to exchange semantically rich, structured information to coordinate, negotiate, or collaborate dynamically, such as in smart cities, supply chain management, or autonomous robotics.

Here are the key disadvantages of using FIPA ACL (Bordini et al.,2005):

- *Complexity*. FIPA ACL's structured message format and performatives can add complexity to agent development, making implementation more challenging, especially for simple systems.
- *Overhead*. The abstraction and richness of communication in FIPA ACL can introduce overhead, which may not be suitable for real-time or resource-constrained environments that require fast, low-latency communication.
- *Limited Adoption*. Despite being a standard, FIPA ACL is not widely adopted across many industries, limiting available support, tools, and community resources for developers.
- *Interoperability Issues*. Although FIPA ACL aims to promote interoperability between agents, differences in ontology, language, or agent platforms can still lead to communication problems if not carefully managed.
- *Performance*. FIPA ACL's higher-level communication protocol may lead to inefficiencies in systems that require high throughput and minimal delay, such as real-time financial trading or industrial automation.

3.2 KQML

A KQML message consists of (Finin et.al., 1994):

- *Performative*. Indicates the communicative intent (e.g., ask, tell, achieve).
- *Sender*. The agent sending the message.
- *Receiver*. The intended recipient(s).
- Content. The information or query being communicated.
- *Language*. Specifies the language used in the content.
- *Ontology*. Defines the vocabulary for shared understanding.

Key features of KQML include:

- *Message Wrapping*. KQML allows flexible message wrapping for a range of communicative acts.
- *Content Agnosticism*. Separates message content from its structure, enabling varied data exchanges.

Some of the advantages of using KQML are:

- *Simplicity*. Easier to implement than more complex communication languages like FIPA ACL.
- *Flexibility*. Supports diverse content languages, making it versatile for different domains.
- *Expressiveness*. Covers a wide range of communicative acts (e.g., querying, informing, commanding).

Using KQML also brings certain disadvantages:

- *Lack of Semantics*. KQML does not enforce strong semantics, which can cause interpretation inconsistencies.
- *No Built-In Interaction Protocols*. Unlike FIPA ACL, it lacks predefined protocols for agent interactions.
- *Limited Adoption*. Its use has diminished in favour of more structured alternatives like FIPA ACL.

3.3 Comparison of FIPA ACL and KQML

While KQML is simpler and more flexible, FIPA ACL offers more structured communication with built-in interaction protocols and stronger semantics. KQML is suitable for lightweight communication needs, while FIPA ACL is better for more complex multi-agent environments.

4. Publish-Subscribe Protocols

Publish-Subscribe (Pub-Sub) protocols are used for efficient and scalable message distribution. Two commonly used Pub-Sub protocols are MQTT and AMQP.

4.1 Message Queuing Telemetry Transport

Message Queuing Telemetry Transport (MQTT) is a lightweight Pub-Sub protocol developed for low-bandwidth, high-latency, and unreliable networks, commonly used in IoT applications (Banks & Gupta, 2014). MQTT uses a central broker that receives messages from publishers and forwards them to subscribers based on topics. The publisher and subscriber are completely decoupled, allowing for flexible and dynamic communication.

MQTT supports three levels of Quality of Service (QoS):

- *QoS 0 (At most once)*. The message is sent once and may be lost if the network fails.
- *QoS 1 (At least once)*. The message is guaranteed to be delivered but may be duplicated.
- *QoS 2 (Exactly once)*. Ensures that the message is received exactly once, making it suitable for critical data transmission.

Advantages of using MQTT include:

- *Low overhead*. Ideal for resource-constrained devices.
- *Simple to implement*. The protocol is easy to integrate into systems with limited capabilities.
- *Scalability*. Supports large-scale deployments with minimal resource consumption.

Disadvantages of using MQTT are:

- *No built-in message persistence*. Messages may be lost if the broker fails.
- *Basic security*. MQTT relies on external protocols (e.g., TLS) for security features.

4.2 Advanced Message Queuing Protocol

Advanced Message Queuing Protocol (AMQP) is a robust messaging protocol designed for enterprise-grade systems requiring high reliability and advanced features. Unlike MQTT, AMQP provides more complex messaging patterns and guarantees message delivery (Ayanoglu et al., 2017).

AMQP uses a broker to route messages between publishers and subscribers, but offers more sophisticated message exchange patterns like queues, routing keys, and exchanges.

AMQP allows message queuing, where messages can be stored by the broker and delivered at a later time, ensuring no loss of data.

There are two delivery guarantees:

- *At least once*. Guarantees that the message will be delivered, though duplicates are possible.
- *Exactly once*. Ensures that each message is delivered exactly once without duplication.

Advantages of using AMQP include:

- *Reliability*. Provides robust mechanisms for message delivery guarantees.
- *Advanced features*. Supports multiple message-routing patterns, including topic-based and direct communication.
- *Security*. AMQP includes built-in mechanisms for secure communication.

Disadvantages of using AMQP are:

- *Heavyweight*. AMQP is resource-intensive, making it unsuitable for lightweight or resource-constrained environments.
- *Complexity*. Implementation can be more challenging due to its extensive feature set.

4.3 Comparison of MQTT and AMQP

The comparison between two previously discussed protocols is shown in Tab. 1.

Feature	MQTT	AMQP
Use case	IoT, low-bandwidth environments	Enterprise-grade applications
Overhead	Low	High
QoS	0, 1, 2 (Basic)	At least once, exactly once
Security	External (TLS)	Built-in mechanisms
Message Persistence	Not guaranteed	Guaranteed through queuing

Tab. 1. Comparison of MQTT and AMQP

5. Request-Response Protocols

These protocols are essential for client-server interactions. Two prominent examples of such protocols are HTTP/REST and CoAP. Their features are explained below.

5.1. Hypertext Transfer Protocol / Representational State Transfer

Hypertext Transfer Protocol (HTTP) is the protocol for web-based communication, and Representational State Transfer (REST) is an architectural style that builds on it. RESTful services use HTTP methods like GET, POST, PUT, and DELETE for communication (Fielding, 2000).

REST is stateless, meaning every client request to the server is independent and contains all the necessary information for processing. REST is widely used for web APIs, transferring data typically in JSON or XML format.

Advantages of using HTTP/REST include:

- *Scalability*. Statelessness allows easy horizontal scaling.
- *Simplicity*. It uses well-established HTTP, making it easy to implement.

Disadvantages of using HTTP/REST are:

- *Overhead*. HTTP's headers can be large, making it inefficient for constrained environments.
- *Latency*. High network overhead and statelessness can introduce delays, particularly for real-time applications.

5.2 Constrained Application Protocol

Constrained Application Protocol (CoAP) is designed for constrained devices and networks, such as those in IoT applications. It operates over UDP, ensuring lower overhead than HTTP (Shelby, 2014). CoAP uses a request-response model similar to HTTP but is optimized for low-power, constrained devices. It supports both unicast and multicast communication.

Advantages of using CoAP include:

- *Efficiency*. It uses less bandwidth and power, with smaller message sizes.
- *Interoperability*. CoAP can be easily mapped to HTTP, facilitating integration with web-based systems.

Disadvantages of using CoAP are:

- *Limited feature set*. CoAP lacks the robustness and features of HTTP, such as built-in security.
- *Reliability*. Being UDP-based, it can suffer from packet loss, although retransmission mechanisms help mitigate this.

5.3 Comparison of MQTT and AMQP

The comparison between two previously discussed protocols is shown in Tab. 2.

Feature	HTTP/REST	CoAP
Protocol	TCP-based	UDP-based
Overhead	High	Low
Message Size	Larger (JSON/XML)	Smaller (binary)
Reliability	TCP ensures reliability	UDP with retransmission

Tab. 2. Comparison of HTTP/REST and CoAP

6. Shared Memory Systems

Shared Memory Systems in MAS allow agents to communicate by writing and reading data from a common memory. Tuple Spaces, originating from Linda, a coordination language developed by David Gelernter, provide a shared memory model that facilitates this interaction by using tuples ordered lists of elements.

In Shared Memory Systems within Multi-Agent Systems (MAS), agents do not directly send messages to one another but instead they interact through a common memory space. Tuple Spaces, which serve as this shared memory, allow agents to write, read, and take tuples, which are ordered collections of data, facilitating indirect communication. This approach enables decoupled interactions, as agents do not need to be aware of each other's existence or state, leading to a more flexible and scalable coordination mechanism.

6.1 Tuple Spaces Concept

In a Tuple space, agents perform three main actions (Gelernter, 1985):

- *Write (out)*. Insert a tuple into the space.
- *Read (rd)*. Retrieve a tuple without removing it.
- *Take (in)*. Retrieve and remove a tuple from the space.

Advantages of using Tuple spaces in distributed systems include:

- *Decoupling*. Tuple spaces decouple agents in both time and space, making interactions flexible and asynchronous.
- *Coordination*. Useful for coordinating distributed agents without direct communication.

Disadvantages of using Tuple spaces in distributed systems are:

- *Scalability*. Managing large numbers of tuples can be inefficient.
- *Security risks*. Shared access can expose the system to security vulnerabilities.

6.2 Use in Distributed Systems

Tuple spaces are highly suited for distributed computing where agents work asynchronously. The data-driven model allows agents to interact via the shared tuples, rather than through direct messages (Carriero & Gelernter, 1989).

Advantages of using Tuple spaces in distributed systems:

- *Asynchronous interaction*. No need for real-time communication between agents.
- *Fault tolerance*. Data remains accessible until consumed.

Disadvantages of using Tuple spaces in distributed systems:

- *Retrieval delays*. Searching for specific tuples in large spaces can slow performance.
- *Consistency*. Ensuring data consistency in dynamic environments is challenging.

7. Messaging Systems

Messaging Systems facilitate communication between agents. Two prominent messaging systems are JADE and RabbitMQ, each serving different needs in agent communication and message distribution.

7.1 Java Agent DEvelopment Framework

Java Agent DEvelopment Framework (JADE) is a software framework that simplifies the development of multi-agent systems by providing a platform for agents to communicate using a message-based architecture. It complies with the specifications, ensuring interoperability between different agents (Bellifemine et al., 2007).

JADE agents communicate using the Agent Communication Language (ACL). Each message consists of a performative (the type of communicative act), sender, receiver, and content.

Advantages of using JADE include:

- *FIPA-compliance*. Ensures standardization and interoperability across different MAS implementations.
- *Extensibility*. The modular design allows developers to easily add new functionalities or agent behaviours.
- *Fault-tolerance*. JADE agents can be distributed across multiple platforms, ensuring system resilience.

Disadvantages of using JADE are:

- *Overhead*. JADE's compliance with FIPA standards and its complex messaging protocol can introduce overhead, especially for lightweight systems.
- *Performance*. JADE may not be as efficient in high-throughput environments where message delivery needs to be fast.

7.2 RabbitMQ

RabbitMQ is an open-source message broker that implements the Advanced Message Queuing Protocol (AMQP). It allows applications to communicate asynchronously via message queues, making it suitable for both agent-based systems and general distributed architectures (Alvaro & Videla, 2012)

RabbitMQ facilitates communication through producers (which send messages) and consumers (which receive messages). Messages are sent to exchanges, which then route them to queues based on routing rules. RabbitMQ ensures reliable delivery of messages using acknowledgment mechanisms and supports message persistence for durability.

Advantages of using RabbitMQ include:

- *High-throughput*. RabbitMQ is designed for high-performance environments and supports a large number of concurrent messages.
- *Reliability*. With features like message acknowledgment, delivery guarantees, and persistence, RabbitMQ ensures that messages are not lost.

- *Flexibility*. It supports multiple messaging patterns such as direct, fanout, and topic-based routing.

Disadvantages of using RabbitMQ are:

- *Complexity*. RabbitMQ can be more complex to configure, especially for systems that do not require its advanced features.
- *Overhead*. The need for message acknowledgment and routing through exchanges can introduce latency in time-sensitive applications.

7.3 Comparison of JADE and RabbitMQ

The comparison between two previously discussed protocols is shown in Tab. 3.

Feature	JADE	RabbitMQ
Use case	MAS development, agent communication	General messaging, distributed systems
Protocol	FIPA-ACL	AMQP
Scalability	Moderate	High
Reliability	Fault-tolerance across agents	Message acknowledgment, persistence
Complexity	High (FIPA compliance)	Moderate (Advanced features)

Tab. 3. Comparison of JADE and RabbitMQ

8. Security Protocols

Security protocols are essential to ensure secure communication, data integrity, and system trustworthiness. Two widely used protocols are Transport Layer Security and Blockchain-based security.

8.1 Transport Layer Security

Transport Layer Security (TLS) is a cryptographic protocol designed to provide secure communication over a network. It evolved from Secure Sockets Layer (SSL) and is widely used to secure web traffic, email, and other Internet communications (Rescorla, 2001).

TLS uses a combination of symmetric and asymmetric encryption to ensure data confidentiality and integrity. A TLS session begins with a handshake process where the server and client exchange keys and authenticate each other using digital certificates. Afterward, all communication is encrypted using symmetric keys.

TLS also provides message integrity through the use of hash functions, ensuring that data has not been altered during transmission. The protocol supports multiple encryption algorithms, allowing clients and servers to negotiate the strongest available cipher suite based on their capabilities. Additionally, TLS incorporates Perfect Forward Secrecy (PFS), ensuring that even if long-term keys are compromised, past communications remain secure by using ephemeral session keys for each connection.

Some key features of TLS are:

- *Confidentiality*. TLS encrypts data to prevent unauthorized access.
- *Integrity*. Message integrity is ensured using cryptographic hashing (e.g., HMAC).
- *Authentication*. Uses X.509 certificates to authenticate the identities of communicating parties.

Advantages of using TLS include:

- *Widely adopted*. TLS is used extensively across the web, making it a standard for securing communications.
- *End-to-end security*. TLS provides encryption from client to server, protecting data in transit.

Disadvantages of using TLS are:

- *Resource-intensive*. The TLS handshake and encryption mechanisms can introduce latency and resource overhead, especially in systems with constrained resources.
- *Certificate management*. TLS requires a public key infrastructure (PKI) for certificate management, which can be complex to implement.

8.2 Blockchain-Based Security

Blockchain-based security leverages the decentralized and immutable nature of blockchain technology to secure communications, data storage, and transactions. A blockchain is a distributed ledger that records data in blocks, which are linked together in a chain using cryptographic hashes (Nakamoto, 2008)

Blockchain relies on consensus mechanisms, such as Proof of Work (PoW) or Proof of Stake (PoS), to validate transactions and ensure the integrity of the ledger. Every participant in the network holds a copy of the blockchain, making it tamper-resistant.

Blockchain-based security uses the decentralized nature of blockchain technology to ensure that no single entity controls the system, enhancing resilience against attacks. Each block in the blockchain contains a cryptographic hash of the previous block, creating a secure chain that is nearly impossible to alter without the consensus of the entire network. Consensus mechanisms like Proof of Work (PoW) or Proof of Stake (PoS) validate transactions, ensuring that only legitimate actions are recorded in the ledger. By distributing the ledger across all participants, blockchain technology provides transparency and tamper-resistance, making it highly secure for various applications.

Some key Features of Blochain-based security are:

- *Decentralization*. Blockchain eliminates the need for a central authority, distributing control across the network.
- *Immutability*. Once data is recorded in a block, it cannot be altered without consensus from the majority of the network.
- *Transparency*. All transactions are visible to participants, ensuring accountability.

Advantages of using Blochain-based security include:

- *Tamper-proof*. The immutability of the blockchain ensures that data or transactions cannot be altered or forged.

- *Decentralized trust*. Blockchain eliminates the need for trust in a single authority, distributing trust across the network.

Disadvantages of using Blochain-based security are:

- *Scalability*. Blockchain can suffer from scalability issues, with transaction speeds often slower than centralized systems.
- *Energy consumption*. Consensus mechanisms like PoW are energy-intensive, making blockchain less sustainable for high-frequency transactions.

8.3 Comparison of TLS and Blockchain-Based Security

The comparison between two previously discussed protocols is shown in Tab. 4.

Feature	TLS	Blockchain-Based Security
Architecture	Centralized, client-server	Decentralized, peer-to-peer
Confidentiality	Encryption with symmetric and asymmetric keys	Cryptographic hashing for data integrity
Scalability	High (with proper infrastructure)	Moderate (depends on consensus)
Key Use Case	Web and Internet communication	Securing distributed transactions and data

Tab. 4. Comparison of TLS and Blockchain-Based Security

9. Conclusion on Communication Protocols in MAS

In MAS, communication protocols play a critical role in enabling effective interaction and coordination between agents. These protocols ensure that agents can exchange information reliably and efficiently, regardless of whether they are operating in centralized, decentralized, or hybrid environments.

Request-response protocols like HTTP/REST and CoAP provide structured frameworks for direct communication between agents, with REST being well-suited for scalable web-based systems and CoAP designed for resource-constrained IoT environments. On the other hand, Publish-Subscribe (Pub-Sub) protocols such as MQTT and AMQP facilitate asynchronous communication, decoupling the sender and receiver, which enhances scalability in dynamic, large-scale systems.

Moreover, security protocols like TLS and Blockchain-based systems ensure that communication between agents is secure and tamper-proof. TLS offers robust point-to-point encryption, making it ideal for web-based MAS applications, while Blockchain provides a decentralized mechanism that ensures data integrity and trust in distributed environments.

The choice of communication protocol in MAS depends heavily on the specific use case, system requirements, and constraints such as scalability, resource availability, and security needs. As MAS evolve, the selection and optimization of these protocols will remain a key factor in achieving efficient and reliable multi-agent interactions.

In addition to the specific characteristics of each communication protocol, the interoperability and compatibility of these protocols within heterogeneous MAS environments are also crucial considerations. Many MAS implementations involve agents with varying capabilities, operating on different platforms and networks. Therefore, selecting protocols that support seamless integration across diverse systems is essential for ensuring smooth agent communication. Furthermore, the ability to manage network latency, handle failures, and maintain consistent performance under varying loads is another important factor when evaluating communication protocols. By carefully balancing these factors, MAS can achieve robust, scalable, and efficient coordination across different use cases and environments.

This paper enlisted most commonly used communication protocols used when designing MAS. It lists the main features of each, and discusses advantages and disadvantages by category. Finally we compared the protocols by category.

9.1 Further research on MAS protocols

Significant progress has been made in developing communication protocols for Multi-Agent Systems (MAS). However, there are several areas that require further exploration. One critical area is scalability. Current MAS communication protocols often face limitations in scalability with the significant increase in number of agents, leading to performance bottlenecks. Future research should focus on developing protocols that maintain efficiency and reliability in large-scale systems, especially in real-time and dynamic environments.

Another important avenue for research is the integration of MAS with emerging technologies such as 5G, edge computing, and quantum communication. These technologies could enhance the speed and reliability of communication between agents, opening new possibilities for MAS applications in high-stakes environments such as autonomous vehicles, smart cities, and industrial automation.

Security and privacy concerns are also prominent. As MAS becomes more prevalent in fields such as finance, healthcare, and national defense, ensuring secure communication among agents is crucial. Protocols must be able to withstand cyber-attacks and ensure data privacy and integrity, especially in decentralized systems where agents may operate under different jurisdictions or regulations.

Lastly, there is a need for further research into the adaptability and flexibility of MAS protocols. Given that MAS often operates in unpredictable and changing environments, protocols must evolve and adapt dynamically to new conditions without human intervention. (Popirlan & Stefanescu, 2011)

Techniques such as machine learning and artificial intelligence could play a crucial role in creating more adaptive communication systems that can learn and optimize communication pathways based on real-time data.

By addressing these challenges, future research can contribute to the development of more robust, efficient, and secure MAS communication protocols, ensuring their continued relevance and effectiveness across diverse domains.

10. References

Alvaro, A., & Videla, G. (2012). RabbitMQ in Action. Manning Publications. https://www.manning.com/books/rabbitmq-in-action

Austin, J. L. (1962). How to Do Things with Words. Clarendon Press.

Ayanoglu, E., Aytaş, Y and Nahum, D (2017). Mastering RabbitMQ by Packt Publishing

Banks, A., & Gupta, R. (2014). MQTT Version 3.1.1. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

Bellifemine, F., Caire, G., & Greenwood, D. (2007). Developing Multi-Agent Systems with JADE .John Wiley & Sons. https://onlinelibrary.wiley.com /doi/book/10.1002/9780470058411

Bordini, R. H., Dastani, M., Dix, J., & Seghrouchni, A. E. F. (2005). "Multi-Agent Programming: Languages, Platforms, and Applications." https://link.springer.com/book/10.1007/978-0-387-89299-3

Carriero, N., & Gelernter, D. (1989). Linda in context. Communications of the ACM, https://dl.acm.org/doi/10.1145/63334.63337

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation, University of California, Irvine). https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

Finin, T., Fritzson, R., McKay, D., & McEntire, R. (1994). KQML as an agent communication language. In Proceedings of the 3rd International Conference on Information and Knowledge Management, Gaithersburg, MD. DOI: 10.1145/191246.191322

FIPA (2002). "FIPA Communicative Act Library Specification." http://www.fipa.org/specs/fipa00037/SC00037J.html

Gelernter, D. (1985). Generative communication in Linda. ACM Transactions on Programming Languages and Systems, https://dl.acm.org/doi/10.1145/2363.2433

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin.org.

Popirlan, C & Stefanescu, C (2011), A Multi-Agent solution for Contact Centre Improvement, DOI:10.2507/22nd.daaam.proceedings.576

Rescorla, E. (2001). SSL and TLS: Designing and Building Secure Systems. Addison-Wesley Professional. ISBN-10: 0201615983 ISBN-13: 978-0201615982

Rygallo, A.; Zloto , T. & Wolny , R.(2009). A Grammatical Model of the Multi-Agent system, DAAAM INTERNATIONAL SCIENTIFIC BOOK 2009 DOI:10.1007/11802372_5

Shelby, Z., Hartke, K., & Bormann, C. (2014). The Constrained Application Protocol (CoAP). RFC 7252, IETF. https://www.rfc-editor.org/info/rfc7252