Towards Engineering LLM-Enhanced Multi-Agent Systems: A Critical Examination of Roles

Tansu Zafer Asici[1][0000−0003−0495−8198], Önder Gürcan*[,2,3][0000−0001−6982−5658], and Geylani Kardas[1][0000−0001−6975−305X]

[1] Ege University,
International Computer Institute,
35100, Izmir, Türkiye
[2] NORCE Norwegian Research Center AS
Center for Modeling Social Systems
Kristiansand, Norway
[3] Coral Protocol
Canary Wharf, London, UK

Abstract. This paper proposes a structured approach to integrating Large Language Models (LLMs) into Multi-Agent Systems (MAS) by revisiting and extending the fundamental Agent-Oriented Software Engineering (AOSE) concept of "roles." Traditional AOSE methodologies provide well-defined processes for modeling agents, roles, goals, and interactions, yet contemporary LLM-based MAS frameworks typically lack such systematic engineering foundations. We highlight how ad hoc development practices in LLM-enhanced MAS—often driven by prompt engineering or role-playing strategies—can lead to inconsistencies and reduced maintainability. Through a critical examination of role definition, specification, and implementation, we identify several gaps in terms of software engineering. To bridge these gaps, we propose a hybrid role-based architecture where we treat *roles as first-class entities at run-time* encapsulating both traditional AOSE design principles and LLM-driven functionalities. By laying this groundwork, we aim to foster more robust, scalable, and transparent engineering of LLM-enhanced MAS.

Keywords: Multi-Agent Systems · Large Language Models · Agent-Oriented Software Engineering · Role Abstraction · Action Execution.

1 Introduction

Large Language Models (LLMs), with their advanced natural language processing capabilities, have opened new avenues for enhancing the capabilities of Multi-Agent Systems (MAS). Examples of these include improved resource coordination and decentralized collaboration [80, 44], enhanced memory and context handling with hierarchical memory models [80], advanced problem-solving [30] and simulation capabilities [25], scalability and flexibility through frameworks for rapid prototyping [15] and scalable deployment [17]. The rapid advancement

of LLMs has also sparked growing interest from research and industry leveraging agent-based solutions, accelerating the development of practical MAS applications and dedicated innovative tools. LLM-enabled MAS applications span multiple domains, including education [43], 6G communications [42], healthcare [4], financial analysis [74], agriculture [48], crime detection [66] and social simulations [31, 32]. Dedicated tools and frameworks aiming to leverage the benefits of LLMs to create more efficient, autonomous, and scalable MAS, facilitating their application across various domains, are being proposed at an increasing pace [16, 51, 79, 44].

However, the existing tools and frameworks often lack a dedicated engineering methodology, resulting in LLM-enabled MAS applications being frequently developed in an ad hoc manner, which can lead to inconsistencies and inefficiencies in their implementation and deployment. This challenge is further compounded by a significant gap in integrating LLMs within MAS, particularly in leveraging the extensive body of knowledge from Agent-Oriented Software Engineering (AOSE). AOSE provides a rich set of principles, methodologies, and tools specifically designed to address the complexities of developing agent-based systems [8, 72, 36]. The under-utilization of established AOSE methodologies [58, 11, 24, 55, 75] and frameworks for integrating LLMs into MAS hinders the optimization of design and implementation, ultimately affecting system quality, reliability, and maintainability. Because, by providing structured processes and clear guidelines, these methodologies help development teams systematically plan, implement, and test multi-agent software, reducing the likelihood of errors and costly rework.

Software engineering methodologies are built upon conceptual elements that serve as high-level representations (or "blueprints") of concepts within a system. Since they do not strictly dictate the underlying details, these elements can be implemented in many ways depending on the chosen technology, programming language, system constraints, or performance needs. The AOSE literature outlines key conceptual elements such as agents, roles, goals, actions, interactions, organizational structures, communication protocols, and mental states (e.g., beliefs, desires, intentions). However, adapting them to real-world LLM-enabled MAS applications remains *underexplored*.

Based on this observation, in this paper, as a first step towards engineering LLM-enhanced MAS, we study the role concept and its realization, as it is the standard building block for specifying agents in AOSE methodologies [1]. The contributions of this study are as follows:

- We survey and categorize current LLM-powered MAS tools and frameworks (e.g., AutoGPT, CAMEL, MetaGPT, LangChain), analyzing their development workflows and methodological limitations.
- We identify gaps between traditional AOSE methodologies and contemporary LLM-enhanced MAS implementations.
- We provide a comprehensive, structured analysis of the role concept in MAS, covering its definition, specification, and implementation from both AOSE and LLM-enhanced perspectives.

– We propose a hybrid architecture that treats roles as first-class run-time entities, encapsulating both AOSE principles and LLM functionalities.

The organization of this paper is as follows: Section 2 provides a brief background about engineering multi-agent systems. Section 3 describes how LLM-enabled MAS are engineered in practice and provides an evaluation in terms of agent-oriented software engineering. Section **??** discusses the gaps, implications and challenges. In Section 5, we propose a hybrid role-based architecture that encapsulates both traditional AOSE design principles and LLM-driven functionalities. Section 6 concludes the paper by summarizing the findings, and outlines potential future work.

2 Engineering Multi-Agent Agents

MAS methodologies provide structured frameworks for the analysis and design of agent-based systems, enabling agents to perform specific tasks effectively. For example, GAIA [75] offers systematic modeling for complex systems by explicitly defining agent roles, responsibilities, and interaction protocols. Roles represent specific functionalities and responsibilities assigned to agents, whereas protocols regulate interactions among agents. The MaSE [53] methodology supports the design process by analyzing systems as sets of roles and tasks. These tasks and roles are linked to agent goals and subsequently combined to define agent classes in the design phase. O-MaSE [18, 20] extends MaSE by introducing meta-models and modular components, emphasizing agent-environment interactions. This facilitates modeling of systems involving environmental processes and rules.

Tropos [11] views agents as social actors structured around concepts such as goals, plans, capabilities, and beliefs. In Tropos, roles represent abstract behaviors of actors within specific contexts, while positions indicate the set of roles an actor may assume. Prometheus [55] simplifies agent design by adopting a goal- and plan-oriented approach, linking goals explicitly to functionalities. Functionalities are defined in terms of actions, messages, and data handled by agents, thereby streamlining system development. ASEME [68] takes a platform-independent, abstract, and modular approach, modeling actor behaviors and control flows by integrating actors, roles, goals, and use scenarios. In the final step, code compatible with agent development platforms such as JADE is automatically generated. The AGRE [24] model defines agents in terms of roles, groups and environments within an organizational context. This allows agents to adopt flexible and adaptive behaviors across multiple roles, facilitating the design of dynamic and modular systems (see [64] for a detailed example).

Although existing AOSE methodologies provide various tools for the structured definition of roles, responsibilities, tasks, and interactions, they lack the flexibility and creativity offered by LLMs. This constitutes a significant challenge for traditional methodologies in adapting to today's rapidly evolving application domains.

3 Practical Engineering of LLM-enabled MAS

This section describes the design principles and configuration approaches used by current LLM-powered MAS development tools to support practical engineering, highlighting features that enhance their overall usability.

3.1 Existing Tools and Frameworks

Current tools for developing LLM-enabled Multi-Agent Systems (MAS) can be grouped into three categories based on their primary functionality:

- Autonomous Task-Oriented Frameworks: Tools like AutoGPT[4], BabyAGI[5], SuperAGI[6], and XAgent[7] use iterative task loops and autonomous planning-execution mechanisms to achieve goals with minimal human intervention.
- Role-based Collaborative Frameworks: CAMEL [52], ChatDev [60], and MetaGPT [39] configure multiple agents with specialized roles, using structured interactions to collaboratively perform complex tasks, particularly in software development contexts.
- General Multi-Agent Orchestration Frameworks: AgentVerse [17], AutoGen [76], CrewAI[8], AWS MAO[9], OpenAI Swarm[10], and LangChain[11] facilitate flexible configuration and management of agent interactions, emphasizing modularity, scalability, and integration.

3.2 Common Development Workflows and Practices

Developers in practice leverage these tools to rapidly prototype and implement LLM-enabled MAS for a variety of tasks. A common workflow begins with defining the roles or objectives of each agent, often via careful prompt engineering or scripting using the framework's API. For example, using a library like AutoGen or LangChain, a developer can instantiate agents with distinct personas (e.g., a "Planner" agent and an "Executor" agent) and then script an interaction loop between them. Typically one agent may be tasked to break down a problem and another to solve sub-problems, or one may critique and improve the other's outputs. Similarly, with role-based systems such as ChatDev or MetaGPT, the developer provides an initial high-level task (for instance, a software feature request), and the system spawns multiple agent instances (developer, tester, manager, etc.) that message each other to gradually refine and implement the

[4] AutoGPT, https://agpt.co/, accessed February 4, 2025.
[5] BabyAGI, https://babyagi.org/, accessed February 4, 2025.
[6] SuperAGI, https://superagi.com/, accessed February 4, 2025.
[7] XAgent, https://github.com/OpenBMB/XAgent/, accessed February 4, 2025.
[8] CrewAI, https://www.crewai.com/, accessed February 4, 2025.
[9] AWS Multi-Agent Orchestrator, https://awslabs.github.io/multi-agent-orchestrator/, accessed February 4, 2025.
[10] OpenAI Swarm, https://github.com/openai/swarm, accessed February 4, 2025.
[11] LangChain, https://python.langchain.com/, accessed February 4, 2025.

solution. In these multi-agent conversations, each agent's prompt template encodes its persona and scope of work, which guides the agent's behavior throughout the dialogue. Developers often iterate on these prompts to steer the agents toward productive interactions (e.g. ensuring the "tester" agent knows how to systematically find faults in the "developer" agent's code output).

Another prevalent development pattern is the use of autonomous task loops, epitomized by AutoGPT and BabyAGI. In this pattern, once a user supplies an initial goal, the agent system itself iteratively generates sub-tasks, executes them, evaluates results, and adjusts the plan or creates new tasks as needed. Developers employ such patterns to offload not just single-step queries to LLMs, but entire project workflows. For instance, a user might ask AutoGPT to "research and write a report on market trends", upon which the system will autonomously break the job into smaller tasks (e.g. data gathering, analysis, drafting) and cycle through them until completion. In practice, frameworks like SuperAGI or AWS's MAO provide higher-level interfaces for running these autonomous agents, allowing developers to configure resources (for example, a vector database for long-term memory) and to monitor the agent's progress through a dashboard.

Building LLM-enabled MAS today is an iterative and experimental process. Developers rely heavily on observing agent behaviors through logs or real-time monitors and then refining the system. If the multi-agent system fails to converge to a good solution, one might adjust the prompts, add a new agent (for example, an agent whose sole role is to critique solutions or to summarize the discussion so far), or tweak the interaction protocol (for instance, enforcing that agents communicate in a structured format or brief bullet points to reduce ambiguity). In summary, current best practices involve defining clear agent roles, endowing agents with the necessary tools or external knowledge sources, simulating their interactions in a controlled environment, and iteratively improving their prompts and logic. Over time, repeated patterns (such as a planner-executor pair, or a questioner-responder-reviewer trio) are becoming templates that developers can reuse across projects, gradually forming an evolving playbook for how to effectively combine multiple LLM agents for different classes of problems.

3.3 Methodologies and Guidelines in Current Tools

Given the relative infancy of LLM-enabled multi-agent systems, most of the existing tools do not prescribe a rigorous software engineering methodology for design or implementation, nor do they explicitly reference classical Agent-Oriented Software Engineering (AOSE) frameworks (e.g., Gaia, Tropos) from the multi-agent systems literature. Development is often guided by example use cases and the built-in templates provided by each tool, rather than by a formal process model. For instance, AutoGPT and BabyAGI emerged as experimental prototypes shared via open-source repositories; users typically follow the provided README or community-written guides to adapt them to new tasks, engaging in a lot of trial-and-error. The emphasis is on achieving functional goals (e.g., "*let the agent autonomously handle my email workflow*") and tweaking the agent's prompts or code as needed when issues arise, rather than on adherence

to a standardized development lifecycle. In essence, the methodology is *ad hoc* – developers iteratively refine the system until it performs the desired behavior, which is feasible given the high-level nature of LLM prompts but lacks the guarantees of traditional engineering approaches.

Some frameworks do offer informal guidelines aligned with their design philosophies. For example, MetaGPT incorporates the concept of Standardized Operating Procedures (SOPs) from human teamwork as part of its prompting strategy. This provides a semi-structured template for how agents should collaborate on a complex task (e.g., a defined sequence of phases such as requirement analysis → design → implementation → testing in a software project). A developer using MetaGPT is encouraged to follow this template when setting up agent roles and conversation order, which is a form of methodology albeit specific to the software-development domain. Similarly, ChatDev's enforced structure of a virtual software team implicitly serves as a development guideline: it suggests that to solve a problem (say, build a new software feature), one should instantiate a team of agents with complementary roles and have them communicate in a logical order reflecting a typical software engineering process. These tool-specific conventions (often inspired by real-world workflows) provide starting points for developers, but they are not generalizable frameworks one could apply to any MAS project in a systematic way. Notably, we find little to no evidence that these new platforms build on established AOSE methodologies – for instance, their documentation and papers do not mention using Gaia's role models or Tropos's goal diagrams to design agent societies. The emphasis is more on empirical effectiveness (does a given configuration of agents solve the task?) rather than a priori design correctness. In practice, the lack of formal methodology means that much of the development knowledge is tacit, residing in the experience of the developers or shared through blog posts and forums rather than encoded in the frameworks themselves.

3.4 Incorporation of Software Engineering Principles

Current LLM-enabled MAS frameworks implicitly support software engineering principles like modularity and reusability by dividing complex problems into specialized agents with clearly defined roles. Tools such as LangChain emphasize role specialization, which simplifies the development, testing, and potential reuse of agent components across projects. Role specialization aligns well with the principle of single responsibility, enhancing overall system maintainability.

However, significant challenges persist. The behavior of agents, defined largely through prompts, lacks transparency and formal analyzability, complicating maintainability—minor prompt changes can unpredictably influence entire systems. Frameworks like AutoGen offer visual debugging to mitigate this, but robust, systematic methods remain scarce. Additionally, runtime-based traceability provided by logging agent interactions doesn't easily map back to original requirements, limiting systematic verifiability and role clarity.

Scalability also poses issues. While selective message routing and orchestrators, such as those employed by OpenAI Swarm and AWS MAO, improve

manageability of agent interactions by efficiently handling multiple agent roles, empirical evidence of their effectiveness in large-scale scenarios remains limited.

Overall, while current frameworks demonstrate foundational support for modular, reusable agent roles, the absence of formal methodologies, traceability mechanisms, and comprehensive scalability strategies highlights critical areas needing development to fully integrate classical software engineering principles into practical MAS engineering.

4 A Critical Examination of Roles

AOSE has long provided solid analytical and design foundations for MAS. AOSE methodologies encompass fundamental concepts such as agents, roles, goals, actions, interactions, organizational structures, and communication protocols. Among these concepts, the *role* is a key concept as it is used to define responsibilities and capabilities of agents. In many AOSE methodologies, a common method is handling agent roles and then aggregate them to form complete agents [1]. Consequently, as a first step towards bringing AOSE methodologies and the practical LLM-enabled MAS development together, we start by critically examining the role concept which is a fundemantal conceptual element in AOSE.

In software engineering, conceptual elements are typically characterized by three facets: their definition, specification, and implementation. Accordingly, we will examine, in order, the role concept's definition (Section 4.1), specification (Section 4.2), and implementation (Section 4.3).

4.1 Role Definition

In AOSE, a role is an abstract definition that encapsulates a set of *responsibilities*, *behaviors*, and *interaction protocols* expected of an agent within a multi-agent system. It specifies what actions an agent should perform, how it should interact with other agents, and what obligations or constraints it must observe in a given organizational or social context. By decoupling the expected behaviors from the agent's concrete implementation, the role concept promotes modularity, reusability, and flexibility, allowing agents to adopt or change roles dynamically as system requirements evolve.

Different methodologies approach the concept of role in various ways. GAIA [75] emphasizes that roles encompass responsibilities, protocols, and permissions to fulfill specific functions. MaSE [53] structures roles through use case scenarios and sequence diagrams during system specification. O-MaSE [18, 20] extends MaSE to include environmental interactions, deepening role modeling. Tropos [11] defines a role as an abstract representation of a social actor's behavior in a specific context, while Cabri et al. [13] highlight the interactional aspects of this approach. Prometheus considers a role as an element that reflects system functionality and defines the actions performed by agents. ASEME [70] presents dynamic role models derived from system use case scenarios, modeling the influence of roles on agent-internal behavior through control flow. Finally, AGRE [24]

defines a role as a functional position within a group, emphasizing that multiple agents can share the same role as an abstraction.

In AOSE, roles are typically an *analysis* and/or *design-time* concept rather than a *run-time* concept. In some cases (e.g. GAIA), the role concept is only used during the analysis phase and does not exist in the design. Over the years, one of the significant problems faced by the MAS community was transitioning the concept of a role from an abstract design into a concrete implementation within an agent framework, primarily because roles often did not exist at run-time, making effective mapping challenging [9].

In the context of an LLM, a role can be defined as a predetermined behavioral profile or identity assigned to the model (such as "an experienced medical advisor" or "a technical support expert"), guiding its responses, language style, information focus, and context-awareness to accomplish specific tasks or interactions effectively. However, it **does not** encapsulate interaction protocols; instead, it encapsulates the responsibilities and behaviors to ensure appropriate and task-specific responses. Here, the role concept is a *design-time* concept, interpreted and executed by the LLM at *run-time*.

At *design-time*, the developer defines what role the model will assume and how it should behave, while at *run-time*, the LLM actively applies and adapts these guidelines dynamically during user interactions. However, at *run-time*, the LLM processes these guidelines as instructions or contextual cues rather than maintaining explicit "roles" and "actions" in the agent-oriented sense (with goals, behaviors, or organizational structures). Essentially, the concept of roles and actions is supplied and interpreted by users and developers through prompts and conversation structure, rather than by the LLM's own inherent representation. While LLMs can appear to perform roles and actions (e.g., adding two numbers), this "execution" is still just a text-based transformation (generating a numeric answer) rather than an explicit representation of action, roles, or goals as understood in AOSE. It is effectively a pattern matching or textual reasoning process, not an agent-based execution driven by internally modeled objectives.

4.2 Role Specification

Specification of a role provides a detailed, often formal, description of what actions it can perform and how it can interact with other roles [34, 46]. The behavior is often described in terms of permissions, responsibilities, activities, and interactions [5]. This detailed breakdown is used to guide the implementation of the role. In the following, we give an ordered list of methods for specifying roles, arranged by increasing levels of formality and precision—from the most informal, conceptual descriptions to the most rigorous, mathematically precise and verifiable specifications:

1. Natural Language Description: The use of natural language in role definition offers both accessibility and flexibility. In this approach, users define a role by explicitly and comprehensively describing its tasks, responsibilities, interaction patterns, and objectives using natural language expressions.

2. Graphical Modeling Languages (e.g., UML[12], SysML[13], AUML[14]): These provide visual diagrams to depict roles, their relationships, and interactions. They strike a balance between clarity and formality, making them well suited for conceptual modeling and communication among stakeholders [6, 29, 67].
3. Business Process Modeling Languages (e.g., BPMN[15]): Designed for illustrating business workflows, BPMN specifies roles within processes clearly. It is more structured than free-form graphics yet still primarily serves as a communication tool rather than a formal specification [54].
4. Architectural Description Languages (ADLs) (e.g., ACME [26], AADL [3]): ADLs focus on system architecture, detailing how roles (often as components or connectors) integrate into and interact within an overall system. They add structure by specifying interfaces and inter-component relationships [23, 57].
5. Programming Code: Implementing role specifications in programming languages makes them executable. While this approach is concrete and precise in terms of behavior, it often intertwines specification with implementation details rather than remaining purely abstract [63].
6. Domain-Specific Languages (DSLs): DSLs are custom-tailored to particular domains. They capture role details and constraints that are specific to the domain context, offering both precision and relevance while potentially varying in formal rigor [21].
7. Behavioral Modeling Languages (e.g., Statecharts [35], Petri nets [59]): These languages emphasize the dynamic aspects of roles—modeling state transitions, interactions over time, and concurrency. They provide a formal way to describe how roles behave under various conditions [14, 69].
8. Ontology and Semantic Web Languages (e.g., OWL[16], RDF[17]): By defining roles within a network of semantically related concepts, these languages offer a formal framework that supports logical inference and interoperability. They are especially useful when roles need to be integrated into larger knowledge representation systems [33, 40].
9. Logic Programming Languages (e.g., Prolog [28]): Using a declarative, rule-based approach, logic programming languages allow you to specify roles in terms of logical constraints and relationships. This method supports formal reasoning and can automatically infer properties of roles [38].
10. Formal Specification Languages (e.g., Z [71], Alloy [41], TLA$^+$ [49], VDM [45], B [2]): These are the most mathematically rigorous methods available. They provide precise, unambiguous specifications that can be used for formal proofs and verification, ensuring that role properties and interactions meet strict correctness criteria [27, 56, 61].

These various methods for role specification comprehensively outline the theoretical framework and expectations of a role within a system. These methods

[12] OMG UML, https://www.omg.org/spec/UML, accessed on February 4, 2025.
[13] OMG SysML, https://www.omgsysml.org/, accessed on February 4, 2025.
[14] AUML, https://auml.org/auml/, accessed on February 4, 2025.
[15] BPMN, https://www.omg.org/bpmn/, accessed on February 4, 2025.
[16] OWL, https://www.w3.org/TR/owl-ref/, accessed on February 4, 2025.
[17] RDF,https://www.w3.org/TR/rdf11-concepts/, accessed on February 4, 2025.

not only determine which actions a role will perform and how it will interact with other roles but also provide crucial insights into how these abstract concepts can be transformed into concrete actions during the implementation process.

In the context of LLMs and LLM-enabled agents, role specification relies solely on natural language descriptions. These textual inputs are interpreted and refined through a detailed prompt engineering process, enabling LLMs to capture the user's intent. This process ensures that the generated responses align with the desired expertise or narrative style, thereby improving overall performance. In essence, natural language serves as a vital bridge between human intent and model behavior—the clarity and scope of the language used in defining a role directly shape the quality and relevance of the model's outputs. However, to achieve less error-prone role specifications, it is preferable to use the more precise methods described above. These specifications can then be *translated* into natural language descriptions when interacting with LLMs.

Besides, role specialization and reusability are partially feasible within LLM-based systems, particularly when supported by structured prompt engineering and modular design frameworks. By defining roles through detailed persona descriptions—encompassing aspects such as background, expertise, tone, and communication style—LLMs can consistently emulate specific behaviors across various contexts. This approach not only enhances the model's ability to maintain a coherent persona but also facilitates the reuse of these role definitions across different tasks and applications. However, challenges remain in ensuring that specialized roles do not lead to overfitting or reduced generalization capabilities. Techniques such as Role Prompting Guided Domain Adaptation (REGA) [73] address this by preserving the general capabilities of LLMs while allowing for effective domain-specific adaptations. REGA employs strategies like self-distillation and role integration to mitigate issues like catastrophic forgetting and inter-domain confusion. Having said, while frameworks and methodologies are emerging to bring more structure to LLM role definitions, LLMs still lack the formal structure of object-oriented programming (OOP), making it challenging to ensure consistency and reusability across different contexts, and they do not yet match the rigor and predictability offered by OOP paradigms.

4.3 Role Implementation

A role defines a set of actions that an agent can perform to achieve its objectives within a multi-agent system (MAS). These actions are guided by the structural arrangements associated with the role, which may include interacting with the environment, gathering information, making decisions, and communicating with other agents. By adopting roles, agents can dynamically respond to changing conditions and collaborate effectively on complex tasks. The specific actions associated with a role can be implemented in the following ways.

Actions Directly Called within Code Roles are implemented using programming languages and are invoked directly by agents within their code. As

Fig. 1. Actions directly called within code

shown in Figure 1, an action is first added to the relevant role. Then, Agent requests to perform the role. For an Agent to perform its Role, it has to execute various actions. After execution, actions produce outcomes, which are returned to the agent by the role. This method is more common in AOSE methodologies such as GAIA [78], Prometheus [55], MaSE [53] and is based on the implementation of role definitions directly as methods or plans in the agent code.

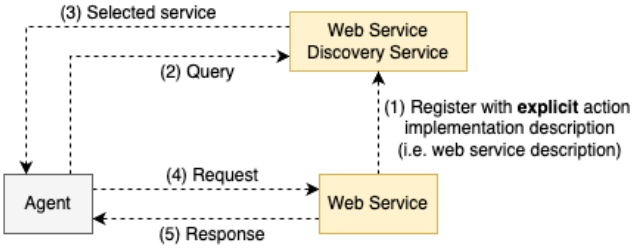

Fig. 2. Actions called as Semantic Web Services

Actions called as Semantic Web Services Agents dynamically discover and invoke actions via services, aligning with modern microservices architectures. In Figure 2, firstly the action as *Web Service* registers its capabilities to Directory Service. Then, at any time, Agent sends a request to find the appropriate service. Once selected, *Web Service* is invoked, performs the required operations, and returns the output information in (5) [12].

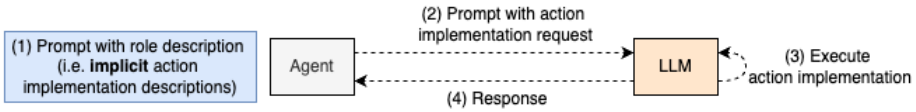

Fig. 3. Actions Triggered for LLM to Performs Its Role

Actions Triggered for LLM to Perform Its Role An LLM autonomously performs tasks by assuming a specific role, enabling artificial intelligence to ex-

hibit autonomous behavior. In the first step of Figure 3, Agent initiates a two-stage prompt process before forwarding it to LLM. In (2), Role supplies its description, and in (3), it sends a prompt containing the action request. In (4), LLM executes the required action-whether generating a response, handling a query, or performing another designated task and returns the output or action outcome back to Agent [17, 52, 60].

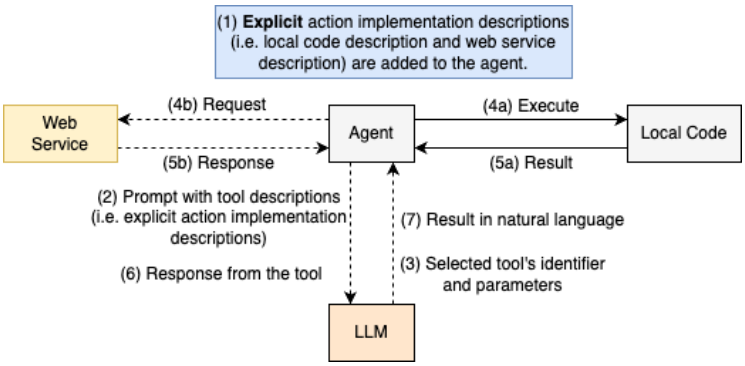

Fig. 4. Actions Triggered by LLM-Controlled Tools

Actions Triggered by LLM-Controlled Tools Agents use natural language queries or commands to trigger actions through LLMs, leveraging advancements in AI and natural language processing. As shown in Figure 4, in (1), a specific action definition is explicitly added to the Agent. Then, in (2), the Agent sends a prompt to the LLM, including descriptions of the defined tools or actions. In (3), the LLM selects the most suitable tool from the available options. The chosen action is then either executed by the Agent within *Local Code* in (4a) or performed by sending a request to the *Web Service* in (4b). In (5a), the result from the *Local Code* is returned to the Agent. In (5b), the response from the *Web Service* is returned to the Agent. In (6), the Agent forwards these results to the LLM. Finally, in (7), the LLM translates these results into natural language and returns them to the Agent.

Actions Triggered by LLM-Controlled MCP Model Context Protocol (MCP)[18] is a robust and open protocol specifically designed to facilitate seamless integration of data sources and external tools into LLM-enhanced MAS. MCP operates on a client-server model, systematically managing resources (e.g., documents, databases), tools (e.g., API calls, file operations), and standardized prompt templates required by LLMs. As shown in Figure 5, MCP Server provides

[18] MCP, https://modelcontextprotocol.io/, accessed on March 31, 2025.

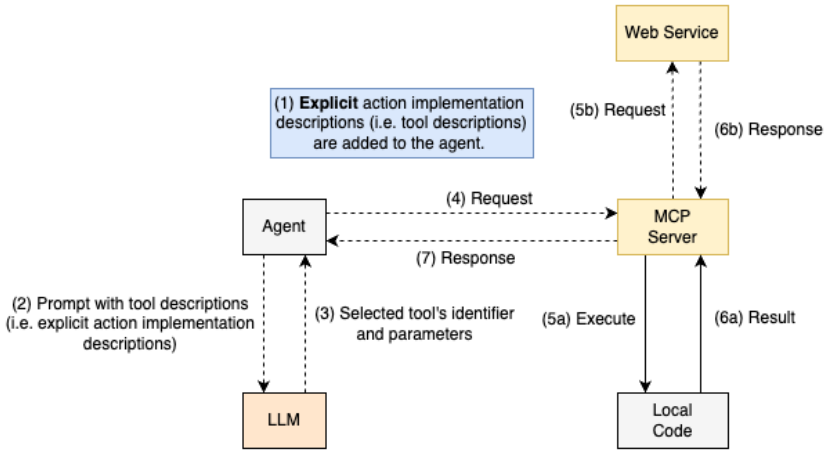

Fig. 5. Actions Triggered by LLM-Controlled MCP

these resources and tools, while LLM acting as MCP Client establishes connections, enabling Agent to efficiently discover and utilize available resources.

In LLM-supported MAS, the selection of role implementation methods must be carefully evaluated based on various criteria, including system performance, security, flexibility, and maintenance ease. Directly invoked actions within code provide developers with full control, enabling efficient debugging, performance optimization, and security. While this method ensures minimal latency and high computational efficiency, its static nature limits flexibility, leading to issues such as code redundancy, maintenance complexity, and scalability challenges. On the other hand, actions invoked as semantic web services enhance dynamic adaptability and reusability through runtime updates and service extensions. While centralized maintenance facilitates cross-platform integration, dependency on external service providers introduces risks such as API changes, service outages, security vulnerabilities, and network latency [47]. Actions triggered by LLM-controlled tools simplify complex operations via natural language interactions, providing a user-friendly environment and supporting rapid prototyping [65]. However, this approach comes with the risks of unpredictable LLM behavior [22], potential errors [77], and high resource consumption [19]. Additionally, actions triggered by an LLM assuming a role offer high adaptability, creativity, and continuous learning capabilities. The ability to assign roles using natural language further enhances accessibility. However, this method also presents challenges such as a lack of transparency [37], security and privacy risks [62], and significant computational demands.

In this section, we have shown that roles are not necessarily explicitly represented an implementation of a concept. In AOSE, roles are often used as abstract concepts during *design-time* and typically lack explicit representation at *run-time*. However, this absence hinders the clear mapping between agents' be-

haviors and their corresponding roles, thereby limiting the system's dynamic adaptability. As highlighted in frameworks such as ROPE [7] and JaCaMo [10], this limitation underscores the need for explicit run-time role modeling. Lhaksmana et al. [50] further stress the critical advantages of defining roles explicitly at run-time, particularly in self-organizing multi-agent systems. We thus argue that, in LLM-supported MAS, roles should be treated as *first-class entities at run-time* to effectively manage diverse application domains and support adaptability. Such an approach enhances agents' ability to adapt dynamically, increases run-time flexibility, and promotes overall system sustainability. This shift toward explicit run-time role representation is embodied in the hybrid role-based architecture presented in Section 5.

5 Proposed Role-based Architecture

Based on the aforementioned observations, this section lays the foundation for an initial architecture for LLM-enhanced MAS engineering. Our goal is to establish a comprehensive framework that integrates traditional AOSE concepts (e.g., role, role implementation) with LLM-based dynamic capabilities. Due to the requirements of hybrid usage and encapsulation, we propose maintaining role definitions explicitly within agents. By encapsulation, we mean bundling data and the actions that operate on that data in a role into a single unit. This way, we can protect the role's internal state from unintended interference or misuse. Moreover, encapsulation promotes modularity and maintainability, making it easier to modify and debug the role while ensuring data integrity.

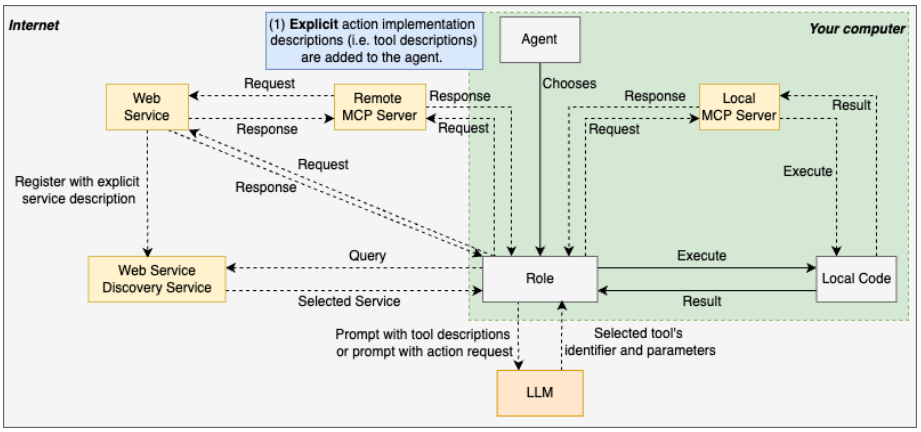

Fig. 6. Proposed Role Based Architecture

Role definitions can encompass one or multiple role implementation approaches outlined in Section 4.3 (Figure 6). This means that a role may involve

actions that are executed locally, actions accessed via web services, tasks performed by locally implemented large language model (LLM) tools, or operations carried out directly by LLMs.

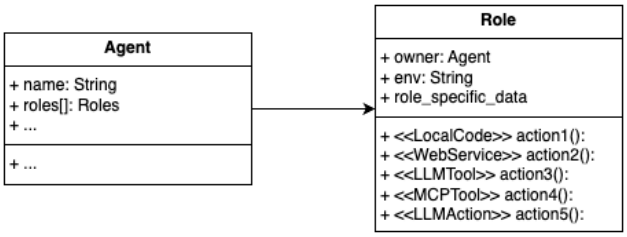

Fig. 7. Proposed Role Concept as UML.

Consequently, we design the LLM-enabled role concept using the UML graphical modeling language since it is more precise compare to natural language. Figure 7 shows a simple UML class model in which an agent with roles. Each Role belongs to one Agent (its owner), has a string indicating the environment where the role is played, and may include various role specific data. A Role can have five types of "actions" with different stereotypes—indicating, for example, LocalCode, WebService, LLMTool, MCPTool, or LLMAction—that can be invoked when an agent is acting in that particular role.

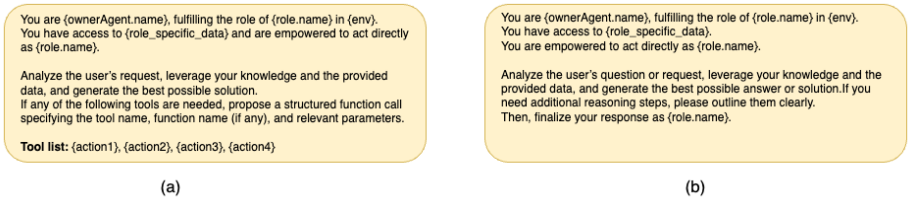

Fig. 8. Figure (a) is a prompt template for a role configured to use tools when necessary. Figure (b) is a prompt template for a role that generates a response directly through the LLM.

When an agent intends to determine or guide its actions within a specific role using outputs generated by an LLM, it employs a dynamically generated prompt. This prompt is constructed by selecting one of the two prompt templates depicted in Figure 8. Figure (a) illustrates a scenario in which the agent performs an action by utilizing external tools based on recommendations provided by the LLM, whereas Figure (b) represents a scenario in which the agent directly utilizes the textual output from the LLM as its response. Placeholders in these prompts-

such as `ownerAgent.name`, `role.name`, `env`, and `role_specific_data`—are dynamically replaced at runtime with concrete values derived from the respective role instance and its owning agent. Consequently, the LLM obtains explicit contextual information about the agent's identity, the role being enacted, the operational environment, and the available capabilities. Thus, the agent interprets the output generated by the LLM and either executes appropriate actions through designated tools or directly incorporates the LLM's response into its communication processes.

6 Conclusion

In this study, we shed light on the fact that existing LLM-based tools and frameworks do not sufficiently leverage AOSE's extensive body of knowledge. While LLM-based frameworks and tools have made impressive strides in enabling powerful and flexible agent behavior, they often do so through ad hoc design approaches that compromise maintainability, reusability, and scalability.

To address this issue, we conducted a focused study on the concept of roles, a foundational element in AOSE, and examined how it is defined, specified, and implemented in both traditional AOSE and LLM-based systems. Our analysis revealed key differences and limitations in current LLM-enabled role modeling, particularly the absence of formal structure and runtime support.

However, adapting the concept of "role" in LLM-assisted MAS development and transforming it into concrete actions brings both new opportunities and challenges. Hybrid solutions that combine the solid principles of traditional AOSE methodologies with the innovative dynamism of LLMs can be effective in overcoming these challenges. Based on this observation, we proposed a hybrid role-based architecture that encapsulates both traditional AOSE design principles and LLM-driven functionalities. This initial architecture is able to express roles that have various actions types while encapsulating them.

By bridging the conceptual foundations of AOSE with the dynamic potential of LLMs, our work lays the groundwork for a more principled engineering methodology for LLM-enhanced MAS. Future work will focus on refining this architecture and validating its effectiveness through practical case studies involving real-world MAS applications.

References

1. Abdalla, R., Mishra, A.: Agent-oriented software engineering methodologies: Analysis and future directions. Complexity **2021**(1), 1629419 (2021)
2. Abrial, J.R., Hoare, A., Chapron, P.: The b-book. (No Title) (1996)
3. Aerospace, S.: Sae architecture analysis and design language (aadl) annex volume 1: Annex a: Graphical aadl notation. Annex C: AADL Meta-Model and Interchange Formats, Annex D: Language Compliance and Application Program Interface Annex E: Error Model Annex, AS5506/1 (2011)

4. ALMutairi, M., AlKulaib, L., Aktas, M., Alsalamah, S., Lu, C.T.: Synthetic arabic medical dialogues using advanced multi-agent llm techniques. In: Proceedings of The Second Arabic Natural Language Processing Conference. pp. 11–26 (2024)
5. Bakar, M., Ghoul, S.: A methodology for auml role modeling. p. 74 – 81 (2011). https://doi.org/10.1109/ISIICT.2011.6149600
6. Bauer, B., Odell, J.: Uml 2.0 and agents: how to build agent-based systems with the new uml standard. Engineering applications of artificial intelligence **18**(2), 141–157 (2005)
7. Becht, M., Gurzki, T., Klarmann, J., Muscholl, M.: Rope: Role oriented programming environment for multiagent systems. In: Proceedings Fourth IFCIS International Conference on Cooperative Information Systems. CoopIS 99 (Cat. No. PR00384). pp. 325–333. IEEE (1999)
8. Bergenti, F., Gleizes, M.P., Zambonelli, F.: Methodologies and software engineering for agent systems: the agent-oriented software engineering handbook, vol. 11. Springer Science & Business Media (2006)
9. Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J.J., Pavon, J., Gonzalez-Perez, C.: Faml: a generic metamodel for mas development. IEEE Transactions on Software Engineering **35**(6), 841–863 (2009)
10. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming **78**(6), 747–761 (2013)
11. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**, 203–236 (2004)
12. Cabral, L., Domingue, J., Motta, E., Payne, T., Hakimpour, F.: Approaches to semantic web services: an overview and comparisons. In: The Semantic Web: Research and Applications: First European Semantic Web Symposium, ESWS 2004 Heraklion, Crete, Greece, May 10-12, 2004. Proceedings 1. pp. 225–239. Springer (2004)
13. Cabri, G., Leonardi, L., Puviani, M.: Service-oriented agent methodologies. In: 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007). pp. 24–29. IEEE (2007)
14. Celaya, J.R., Desrochers, A.A., Graves, R.J.: Modeling and analysis of multi-agent systems using petri nets. In: 2007 IEEE International Conference on Systems, Man and Cybernetics. pp. 1439–1444. IEEE (2007)
15. Chandrasekaran, M.: Enhancing efficiency and flexibility of rapid prototyping for scalable multimodal intelligent agents. In: 2024 Artificial Intelligence for Business (AIxB). pp. 66–71. IEEE (2024)
16. Chen, G., Dong, S., Shu, Y., Zhang, G., Sesay, J., Karlsson, B., Fu, J., Shi, Y.: Autoagents: A framework for automatic agent generation. p. 22 – 30 (2024)
17. Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.M., Yu, H., Lu, Y., Hung, Y.H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., Zhou, J.: Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors (2024)
18. Dam, K.H., Winikoff, M.: Comparing agent-oriented methodologies. In: International Bi-conference Workshop on Agent-Oriented Information Systems. pp. 78–93. Springer (2003)
19. Dang, Y., He, Y., Xu, M., Ye, K.: Resource management for gpt-based model deployed on clouds: Challenges, solutions, and future directions. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **15252 LNCS**, 95 – 105 (2025). https://doi.org/10.1007/978-981-96-1528-5_7

20. DeLoach, S.A., Valenzuela, J.L.: An agent-environment interaction model. In: International Workshop on Agent-Oriented Software Engineering. pp. 1–18. Springer (2006)
21. Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G., Mernik, M.: Sea_l: a domain-specific language for semantic web enabled multi-agent systems. In: 2012 Federated Conference on Computer Science and Information Systems (FedCSIS). pp. 1373–1380. IEEE (2012)
22. Deng, Y., Zhang, W., Pan, S.J., Bing, L.: Multilingual jailbreak challenges in large language models (2024)
23. Dissaux, P., Marc, O.M., Rubini, S., Fotsing, C., Gaudel, V., Singhoff, F., Plantec, A., Nguyen-Hong, V., Tran, H.N.: The smart project: Multi-agent scheduling simulation of real-time architectures. In: Embedded Real Time Software and Systems (2014)
24. Ferber, J., Michel, F., Báez, J.: Agre: Integrating environments with organizations. In: International Workshop on Environments for Multi-Agent Systems. pp. 48–56. Springer (2004)
25. Gao, C., Lan, X., Li, N., Yuan, Y., Ding, J., Zhou, Z., Xu, F., Li, Y.: Large language models empowered agent-based modeling and simulation: a survey and perspectives. Humanities and Social Sciences Communications **11**(1) (2024). https://doi.org/10.1057/s41599-024-03611-3
26. Garlan, D., Monroe, R., Wile, D.: Acme: An architecture description interchange language, white paper. Computer Science Department, Carnegie Mellon University, Pittsburgh PA, submitted for publication (1997)
27. Ghafarollahi, A., Buehler, M.J.: Atomagents: Alloy design and discovery through physics-aware multi-modal multi-agent artificial intelligence. arXiv preprint arXiv:2407.10022 (2024)
28. Giannesini, F., Kanoui, H., Pasero, R., Van Caneghem, M.: Prolog. Addison-Wesley Longman Publishing Co., Inc. (1986)
29. Guedes, G.T.A., Vicari, R.M.: Applying auml and uml 2 in the multi-agent systems project. In: Advances in Conceptual Modeling-Challenging Perspectives: ER 2009 Workshops CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS, Gramado, Brazil, November 9-12, 2009. Proceedings 28. pp. 106–115. Springer (2009)
30. Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N.V., Wiest, O., Zhang, X.: Large language model based multi-agents: A survey of progress and challenges. p. 8048 – 8057 (2024)
31. Önder Gürcan, Falck, V., Rousseau, M.G., Lima, L.L.: Towards an llm-powered social digital twinning platform. In: Proceedings of The 23rd International Conference on Practical applications of Agents and Multi-Agent Systems (PAAMS 2025) (2025)
32. Önder Gürcan: LLM-Augmented Agent-Based Modelling for Social Simulations: Challenges and Opportunities. In: HHAI 2024: Hybrid Human AI Systems for the Social Good, Frontiers in Artificial Intelligence and Applications, vol. 386, pp. 134–144. IOS Press (2024). https://doi.org/10.3233/FAIA240190
33. Haarslev, V., Möller, R.: Racer: An owl reasoning agent for the semantic web. In: Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with the. pp. 91–95 (2003)
34. Hameurlain, N., Sibertin-Blanc, C.: Specification of role-based interactions components in multi-agent systems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **3390**, 180 – 197 (2005). https://doi.org/10.1007/978-3-540-31846-0_11

35. Harel, D.: Statecharts: A visual formalism for complex systems. Science of computer programming **8**(3), 231–274 (1987)
36. Henderson-Sellers, B., Giorgini, P.: Agent-oriented methodologies. Igi Global (2005)
37. Hepenstal, S., Zhang, L., Wong, B.L.W.: The impact of system transparency on analytical reasoning (2023). https://doi.org/10.1145/3544549.3585786
38. Hjelmblom, M.: Deontic action-logic multi-agent systems in Prolog. Högskolan i Gävle (2008)
39. Hong, S., Zheng, X., Chen, J., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S.K.S., Lin, Z., Zhou, L., et al.: Metagpt: Meta programming for multi-agent collaborative framework. arXiv preprint arXiv:2308.00352 (2023)
40. Hui, K.Y., Chalmers, S., Gray, P.M., Preece, A.D.: Experience in using rdf in agent-mediated knowledge architectures. In: International Symposium on Agent-Mediated Knowledge Management. pp. 177–192. Springer (2003)
41. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Transactions on software engineering and methodology (TOSEM) **11**(2), 256–290 (2002)
42. Jiang, F., Peng, Y., Dong, L., Wang, K., Yang, K., Pan, C., Niyato, D., Dobre, O.A.: Large language model enhanced multi-agent systems for 6g communications. IEEE Wireless Communications **31**(6), 48 – 55 (2024). https://doi.org/10.1109/MWC.016.2300600
43. Jiang, Y.H., Liu, T.Y., Zhuang, X., Hu, H., Li, R., Jia, R.: Enhancing educational practices with multi-agent systems: A review (2024)
44. Jin, A., Ye, Y., Lee, B., Qiao, Y.: Decoagent: Large language model empowered decentralized autonomous collaboration agents based on smart contracts. IEEE Access **12**, 155234 – 155245 (2024). https://doi.org/10.1109/ACCESS.2024.3481641
45. Jones, C.B.: Systematic software development using VDM, vol. 2. Prentice Hall Englewood Cliffs (1990)
46. Koning, J.L., Hernández, I.R.: Limitations in auml's roles specification. IFIP Advances in Information and Communication Technology **163**, 79 – 82 (2005). https://doi.org/10.1007/0-387-23152-8_10
47. Kumar, D., Ma, Z., Durumeric, Z., Mirian, A., Mason, J., Halderman, J.A., Bailey, M.: Security challenges in an increasingly tangled web. p. 677 – 684 (2017). https://doi.org/10.1145/3038912.3052686
48. Kuska, M.T., Wahabzada, M., Paulus, S.: Ai for crop production – where can large language models (llms) provide substantial value? Computers and Electronics in Agriculture **221**, 108924 (2024). https://doi.org/10.1016/j.compag.2024.108924
49. Lamport, L.: Specifying systems: the tla+ language and tools for hardware and software engineers (2002)
50. Lhaksmana, K.M., Murakami, Y., Ishida, T.: Role-based modeling for designing agent behavior in self-organizing multi-agent systems. International Journal of Software Engineering and Knowledge Engineering **28**(01), 79–96 (2018)
51. Li, C., Chen, H., Yan, M., Shen, W., Xu, H., Wu, Z., Zhang, Z., Zhou, W., Chen, Y., Cheng, C., Shi, H., Zhang, J., Huang, F., Zhou, J.: Modelscope-agent: Building your customizable agent system with open-source large language models. p. 566 – 578 (2023)
52. Li, G., Hammoud, H., Itani, H., Khizbullin, D., Ghanem, B.: Camel: Communicative agents for" mind" exploration of large language model society. Advances in Neural Information Processing Systems **36**, 51991–52008 (2023)
53. Loach, S.A.D., Wood, M.: Developing multiagent systems with agenttool. In: Intelligent Agents VII Agent Theories Architectures and Languages: 7th International

Workshop, ATAL 2000 Boston, MA, USA, July 7–9, 2000 Proceedings 7. pp. 46–60. Springer (2001)
54. Onggo, B.S., Karpat, O.: Agent-based conceptual model representation using bpmn. In: Proceedings of the 2011 Winter Simulation Conference (WSC). pp. 671–682. IEEE (2011)
55. Padgham, L., Winikoff, M.: Prometheus: A practical agent-oriented methodology. In: Agent-oriented methodologies, pp. 107–135. IGI Global (2005)
56. Paiva, P.Y.A., Saotome, O., Brandauer, C.: Specification and verification of a multi-agent coordination protocol with tla+. In: 2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC). pp. 207–212. IEEE (2018)
57. Park, S., Sugumaran, V.: Designing multi-agent systems: a framework and application. Expert Systems with Applications **28**(2), 259–271 (2005)
58. Pavón, J., Gómez-Sanz, J.J., Fuentes, R.: The ingenias methodology and tools. In: Agent-oriented methodologies, pp. 236–276. IGI Global (2005)
59. Petri, C.: Kommunikation mit automaten: Phd thesis/institut f¨ ur instrumentelle mathematik.-bonn, 1962 (1962)
60. Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., et al.: Chatdev: Communicative agents for software development. In: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 15174–15186 (2024)
61. Ramzan, M., Ali, A., Akram, S., Qayyum, Z.U.: Formal specification of multi-agent environment using vdm-sl. In: 2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT). pp. 150–154. IEEE (2011)
62. Rathod, V., Nabavirazavi, S., Zad, S., Iyengar, S.S.: Privacy and security challenges in large language models. p. 746 – 752 (2025). https://doi.org/10.1109/CCWC62904.2025.10903912
63. Ross, R., Collier, R., O'Hare, G.M.: Af-apl–bridging principles and practice in agent oriented languages. In: Programming Multi-Agent Systems: Second International Workshop ProMAS 2004, New York, NY, USA, July 20, 2004, Selected Revised and Invited Papers 2. pp. 66–88. Springer (2005)
64. Roussille, H., Gürcan, Ö., Michel, F.: Agr4bs: A generic multi-agent organizational model for blockchain systems. Big Data and Cognitive Computing **6**(1) (2022). https://doi.org/10.3390/bdcc6010001, https://www.mdpi.com/2504-2289/6/1/1
65. Sathe, G., Choudhary, V., Bhagat, D.: Comprehensive review on large language models (llms). vol. 2, p. 3097 – 3102 (2024)
66. Sathya, J., Fernandez, F.M.H.: An optimizing crime detection in social media platforms using multiagent ontology-based approach. p. 956 – 966 (2023). https://doi.org/10.1109/ICOSEC58147.2023.10276325
67. Sha, Z., Le, Q., Panchal, J.H.: Using sysml for conceptual representation of agent-based models. In: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. vol. 54792, pp. 39–50 (2011)
68. Spanoudakis, N., Moraitis, P.: Using aseme methodology for model-driven agent systems development. In: International workshop on agent-oriented software engineering. pp. 106–127. Springer (2010)
69. Spanoudakis, N.I.: Engineering multi-agent systems with statecharts: Theory and practice. SN Computer Science **2**(4), 317 (2021)
70. Spanoudakis, N.I., Moraitis, P.: The aseme methodology. International Journal of Agent-Oriented Software Engineering **7**(2), 79–107 (2022)

71. Spivey, J.: The Z Notation: A Reference Manual. Prentice-Hall international series in computer science, Prentice Hall (1992)
72. Sturm, A., Shehory, O.: Agent-oriented software engineering: revisiting the state of the art. Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks pp. 13–26 (2014)
73. Wang, R., Mi, F., Chen, Y., Xue, B., Wang, H., Zhu, Q., Wong, K.F., Xu, R.: Role prompting guided domain adaptation with general capability preserve for large language models. In: Duh, K., Gomez, H., Bethard, S. (eds.) Findings of the Association for Computational Linguistics: NAACL 2024. pp. 2243–2255. Association for Computational Linguistics, Mexico City, Mexico (Jun 2024). https://doi.org/10.18653/v1/2024.findings-naacl.145
74. Wawer, M., Chudziak, J.A., Niewiadomska-Szynkiewicz, E.: Large language models and the elliott wave principle: A multi-agent deep learning approach to big data analysis in financial markets. Applied Sciences (Switzerland) **14**(24) (2024). https://doi.org/10.3390/app142411897
75. Wooldridge, M., Jennings, N.R., Kinny, D.: The gaia methodology for agent-oriented analysis and design. Autonomous Agents and multi-agent systems **3**, 285–312 (2000)
76. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., Wang, C.: Autogen: Enabling next-gen llm applications via multi-agent conversation framework. arXiv preprint arXiv:2308.08155 (2023)
77. Ye, J., Li, S., Li, G., Huang, C., Gao, S., Wu, Y., Zhang, Q., Gui, T., Huang, X.: Toolsword: Unveiling safety issues of large language models in tool learning across three stages. vol. 1, p. 2181 – 2211 (2024). https://doi.org/10.18653/v1/2024.acl-long.119
78. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) **12**(3), 317–370 (2003)
79. Zhang, C., Yang, K., Hu, S., Wang, Z., Li, G., Sun, Y., Zhang, C., Zhang, Z., Liu, A., Zhu, S.C., Chang, X., Zhang, J., Yin, F., Liang, Y., Yang, Y.: Proagent: Building proactive cooperative agents with large language models. vol. 38, p. 17591 – 17599 (2024). https://doi.org/10.1609/aaai.v38i16.29710
80. Zou, H., Li, R., Sun, T., Wang, F., Li, T., Liu, K.: Cooperative scheduling and hierarchical memory model for multi-agent systems. In: 2024 IEEE International Symposium on Product Compliance Engineering-Asia (ISPCE-ASIA). pp. 1–6. IEEE (2024)