

## RESEARCH ARTICLE

# Leveraging the Learning Curve: Reusing Existing Architectural Patterns to Design and Implement MAS

ARTHUR CASALS<sup>ID</sup>, (Member, IEEE), AND ANAROSA A. F. BRANDÃO<sup>ID</sup>

Escola Politécnica, USP, São Paulo 05508-220, Brazil

Corresponding author: Anarosa A. F. Brandão (anarosa.brandao@usp.br)

This work was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior–Brazil (CAPES) under Grant 001.

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the Comitê de Ética em Pesquisa (CEP) EACH-USP under Application No. CAAE 40985420.0.0000.5390, and performed in line with the Resolutions CNS n° 466/12 and CNS n° 510/2016 provided by Comissão Nacional de Ética em Pesquisa–Conep.

**ABSTRACT** Recent advancements in AI have led to the development of specialized systems related to multi-agent systems (MAS). However, the inherently collaborative nature of agents is often overlooked, and many of these specialized systems are used as components by other AI systems. From a software engineering perspective, this context can benefit from aligning the architectural characteristics of distributed systems with the inherently distributed nature of MAS. We propose that introducing a minimal set of agent-related concepts into the Distributed Systems (DS) domain can improve the engineering of modern MAS by leveraging techniques from DS engineering with established agent theory. In this study, we recapitulated the common origins of MAS and DS by drawing architectural parallels to establish a unified engineering approach. We then defined a minimal set of agent concepts to perform two practical studies on leveraging MAS development. First, we incorporated these concepts into a DS architectural pattern to design a distributed MAS. We then used these concepts in a graduate course to teach MAS engineering to students with no prior knowledge of agent theory. The learning outcomes from both courses included successful MAS implementation using DS tools and techniques. Although more than two-thirds of these students had no practical experience in developing distributed systems, the average final grade in both courses was above 80%, thus validating our approach. Finally, we discuss how this study supports the development of advanced systems using modern AI techniques consistently with established agent-related research while leveraging established DS techniques and concepts.

**INDEX TERMS** Architectural patterns, distributed systems, multi-agent systems, entity-component-systems.

## I. INTRODUCTION

In 1995, Russell and Norvig [1] stated that “AI is the study of agents”. The agent concept is also used in Distributed Artificial Intelligence (DAI) [2]. Fikes [3] first described the basis for autonomous and cooperative problem-solving in 1982, laying the foundation for agents as autonomous entities used for problem solving.

The associate editor coordinating the review of this manuscript and approving it for publication was Jolanta Mizera-Pietraszko<sup>ID</sup>.

The main concern of DAI is to find a “collaborative solution of global problems by distributed entities” [2]. In this case, entities can possess different natures and levels of complexity, including *reasoning* - thus characterizing the “intelligent” aspect. The problem-solving process is based on the concept of *information sharing*. *Collaboration* arises from shared information being necessary for entities to solve the problems at hand. Simultaneously, the problem is *global* and common to all the entities involved in the problem-solving process. Finally, the process is considered *distributed*

because the entities involved in the problem-solving process can exist in different locations, as determined by logical or geographical separation.

These concepts were also summarized by Demazeau and Müller [4], who introduced the concept of *Decentralized Artificial Intelligence* (DzAI). In their article, they state that DzAI “is concerned with the activity of an autonomous agent in a multi-agent world.” The focus of DzAI is to model autonomous agents that can solve *their problems* or achieve their goals. By being able to possess multiple goals, it is also taken into account that (i) the agent might have to deal with multiple and sometimes contradictory sources of information, and (ii) all of their goals have to be mapped according to the agent’s own restrictions in terms of perceiving and acting over the environment.

Agents can be organized in communities known as *multi-agent systems* (MAS), which are systems composed of multiple agents that interact among themselves in a single environment [1] to solve complex problems. Regardless of their nature, problems in MAS may possess different degrees of complexity, and they can be divided into subproblems (or goals) individually assigned to each agent in the system [5]. MAS are often related to or merged with modern DAI concepts due to their origin and characteristics. In 1996, Parunak [6] stated that structuring a DAI system would require designing both the agent and system architecture, explicitly describing the added value of an MAS in an industrial scenario. The author also stated the benefits of replacing centralized control systems and databases with an MAS whenever a control distribution would be desirable. Modern DAI is categorized as MAS and distributed problem-solving [2]. MAS focuses on how agents coordinate their knowledge and activities, whereas distributed problem-solving focuses on decomposing problems and synthesizing solutions.

Owing to their inherently distributed mechanisms, it is easy to trace a parallel between MAS and *distributed systems*. A distributed system (DS) is “a collection of autonomous computing elements that appear to its users as a single coherent system” [7]. Similar to MAS, their components concurrently interact with each other to achieve a common goal. Additionally, their computing elements (*nodes*) can behave independently.

Among the existing distributed systems currently used on a large scale, the World Wide Web (WWW) is one of the most widely used and developed ones. It was created in 1994 by Berners-Lee et al. as a loosely coupled distributed system for sharing documents [8], focusing on user-generated content and ease of use. A few years later, the author introduced the concept of the Semantic Web [9], extending his original work by adding semantics to the existing data format representation. This extension allows information on the World Wide Web to be understandable to humans and software entities. Designing systems for the Semantic Web involves not only creating systems with distributed capabilities but also systems capable of sharing and reusing

knowledge, which is also part of the focus of MAS. From this point on, we will refer to the Semantic Web simply as “the Web” and systems designed for the Web as “web-based systems.”

Although there are many different architectures for distributed systems [7], web-based systems host some of the most complex use cases for the development of distributed systems because they can be designed for a massive number of simultaneous users that are geographically distributed and make use of both local and distributed resources. In addition, DS can be built with web interfaces, so they can either access existing systems or data on the Web or be accessed by other systems using Web protocols. In our context, this is the case for systems such as ChatGPT<sup>1</sup> and Midjourney,<sup>2</sup> which can be used by other systems via API access and Discord<sup>3</sup> integration. In either case, each system instance is treated as a software agent with different capabilities and responsibilities, similar to a heterogeneous MAS. While there are many plugins and third-party tools that address the use of multiple instances of such systems as a “multi-agent” tool, most of them do not relate to the existing multi-agent research, neither in general nor in its subfields (e.g., coordination, communication, and collaboration).

Designing an MAS from a DS perspective means taking a minimum set of MAS elements and bringing them to the context of a DS. Interestingly, the majority of existing work related to the Web in the MAS domain adopts the opposite perspective: Web technologies (and their related elements) are brought to the context of MAS. We do not intend to argue about the absolute number of web-based distributed system implementation rates when compared with MAS, nor to speculate why or if one is easier to implement than the other. It is a fact, however, that a myriad of tools, libraries, and frameworks are available for the implementation of DS over the Web [10] that encapsulate the specialized knowledge necessary to design distributed systems.

If we emulate the same scenario in the current MAS context, one could argue that the MAS community lacks the abundance of tools or frameworks available for distributed systems. The existing tools and MASs are almost exclusive to the agent community; all extensions are exclusive to other MAS and Agent frameworks, and many of the frameworks have not yet been used to deploy an MAS in production [11]. In addition, none of the most popular MAS frameworks eliminate the need for specialized knowledge, such as goals and plans, or how the system can be implemented considering different organizations, artifacts, and collaborating agents.

The objective of the present work is to propose an approach that uses both existing MAS research and modern DS engineering techniques to leverage the learning curve currently necessary for developing MAS. To achieve this objective, we need to analyze the common evolution of MAS

<sup>1</sup><https://chat.openai.com/>

<sup>2</sup><https://www.midjourney.com/>

<sup>3</sup><https://discord.com/>

and DS to establish a minimum set of MAS concepts that can be introduced into the DS domain and effectively used to build new AI systems.

Once this minimal conceptual set is established, its viability must be tested. This analysis was conducted in two practical studies. First, we used this minimal conceptual set in conjunction with a specific DS architectural pattern to design a distributed MAS. In the second study, we used both the conceptual set and the previous use case to teach graduate students without any previous knowledge about MAS and Agents to design and implement MAS using DS techniques. These courses took place on two different occasions.

This paper is organized as follows: Section II presents our study on the parallels between DS and MAS, with an emphasis on the architectural and communication aspects. In Section III, we explore existing work related to our research, focusing on the use of web technologies in conjunction with MAS. Section IV describes our approach in designing MAS in the context of DS, including our approach for determining a minimal conceptual set of topics in agent theory, the incorporation of such concepts into the DS domain, and practical studies performed to explore our approach. This section also includes the learning outcomes of both the courses. We conclude this paper in Section V with our final considerations and future research directions.

## II. DISTRIBUTED SYSTEMS AND MAS

According to Steen and Tanenbaum [7], distributed systems can be divided into *high-performance distributed computing*, *distributed information*, and *pervasive systems*, depending on their purpose. According to the authors, the reasons for designing a DS should meet four specific goals: (i) making resources easily accessible; (ii) making distribution transparent by hiding the fact that resources are distributed across a network; (iii) offering components that can be easily used by, or integrated into other systems; and (iv) being scalable in size, geographic distribution, or number of administrative organizations (following the scalability dimensions proposed by Neuman [12]).

Similarly, Russell and Norvig [1] stated that MAS is designed to solve problems that are beyond the capabilities of individual agents or monolithic systems. As mentioned before, multi-agent systems are inherently distributed, solving problems through the cooperation and interoperation of multiple agents. Individual agents can be used not only for distributed task allocation but also for wrapping and providing access to legacy systems. Agents can also use distributed resources to retrieve and coordinate information from multiple sources. In addition, agents are highly reusable and composable [13]. Similar to distributed systems, MAS can be built for high-performance computing tasks [14], [15], disseminating information across multiple nodes [16], and for ubiquitous blending with the environment [17].

We trace a parallel between DS and MAS in this paper because while they share multiple elements in their history, both are seen and used differently by the software engineering

community. In particular, Web-based distributed systems are widely used and adopted by many software engineers and organizations across the globe, whereas MAS is not yet mainstream. While we do not have the tools (or data) to affirm that this is because of one reason or another, it is also a fact that the existing resources related to Web-based DS are *distributed systems implemented from a Web perspective*. This means that elements related to the DS research field were introduced and implemented in the context of the Web. Because the Web itself evolved as a distributed system, it is difficult to define the line that separates bringing one context into another. However, it is relatively easy to perceive that most distributed systems theory is shielded from the average web systems engineer. From this perspective, taking MAS elements and bringing them to the context of DS engineering could potentially leverage the development of MAS.

In the following paragraphs, we analyze the main aspects of designing a DS and establish a parallel with the MAS design aspects. These aspects revolve around two specific topics: (i) architecture and (ii) communication. One characteristic common to both DS and MAS is that all the other engineering aspects - from coordination to handling failures - depend on the system's architecture and communication. It can be argued that "available resources" (hardware, auxiliary systems, communication bandwidth, etc.) are also a primary topic when designing MAS and DS. However, because resource analysis is a fundamental part of designing the system architecture in the first place, we do not need to address it. This analysis aims to create a baseline for existing work in both domains, which will later be used to establish the minimal MAS conceptual set mentioned previously. The DS aspects analyzed were based on the organization proposed by Steen and Tanenbaum [7].

### A. ARCHITECTURE

Software architectures represent how software components are logically organized and how they should interact with each other [18]. They use architectural styles to organize system components and make their inherent complexities manageable and understandable. An architectural style "is formulated in terms of components, the way in which components are connected to each other, the data exchanged between components, and finally how these elements are jointly configured into a system" [7]. The components and their relationships follow the definition provided by Selic [19], [20], being modular and providing replaceable interfaces within the environment. In general, connectors mediate everything related to communication between components, including information and control flow.

#### 1) ARCHITECTURE IN DS

According to Steen and Tanenbaum [7], the most important architectural styles used when designing distributed systems are (i) layered architectures, (ii) object-based architectures, (iii) resource-centered architectures, and (iv) event-based

architectures. These styles can be combined within a system, with each part of the system having a different architectural style. Regardless of the architectural style used, DS can be organized in different ways according to the coordination of its components. *Centralized* DSs rely on a central *coordinator*, which is usually responsible for implementing all the data processing. This coordinator is accessed by a *client* who requests and uses the information provided by the coordinator to perform its functions.

In contrast, a *decentralized* DS does not possess a unique component that is solely responsible for data processing. Instead, each component of the system possesses its own data processing responsibilities, and they can communicate with each other without the need for third-party intervention. Centralized and decentralized architectures can be combined into hybrid architectures.

We will not discuss each style in detail (although their names provide a general idea of what is involved in each one). However, we decided to focus our analysis on decentralized DS because of their nature. From an MAS perspective, building a distributed heterogeneous MAS can involve almost all sub-research fields within agent theory, which is ideal when proposing a minimal conceptual set related to a research area. In addition, as mentioned before, there are many different architectures for distributed systems, and web-based systems host some of the most complex use cases for DS development. If we want to consider systems that are partially exposed to the Web (either by deployment or by accessing another web-based system), we need to understand the architectural styles used when building web-based systems.

## 2) WEB ARCHITECTURAL STYLES

In DS, layered architectures are based on organizing software components in layers so that each layer can access the adjacent layers. Top-down layered architectures (i.e., vertically organized) are commonly used in network communication. In this organization, the upper layers can access those immediately below using *calls*. Layered architectures can be organized in different ways [7].

Object-based architectures follow the same principle as layered architectures by dividing software components into objects that can access each other. Each object encapsulates the data and the operations that can be performed on that data. Similarly, services can also be encapsulated and communicate with each other, which is the basis for service-oriented architectures (SOA) [21]. A system designed following SOA principles is composed of multiple services that do not necessarily belong to the same administrative organization or reside on the same server (being, therefore, scalable). Owing to organizational requirements, services can be grouped and divided into layers. These architectural styles are used on the web as *web services* [22].

Resource-based architectures are based on the idea that a DS is a collection of resources individually managed by components. In the context of the Web, such architectures are

seen as an evolution from pure service-based ones. Resources are considered to be “the fundamental building blocks of Web-based systems” [10]. Resource-based architectures still depend on services to access and manipulate resources within a system.

Following the previous architectural style, event-based architectures [23] are based on the concept of decoupling processes. Steen and Tanenbaum [7] use the term “publish-subscribe” to refer to the same idea, since the publisher-subscriber messaging pattern [24] is commonly used for communication (although event streaming [25] is also another communication pattern used in event-based applications). Coordination and integration between services are achieved through the use of events representing changes in the state of data (such as resources), instead of requiring that all services involved in a process know the full specification of the data involved. Publish-subscribe architectures [26] are examples of this architectural style: a given service (publisher) sends a message notifying other services (subscribers) that something has changed in relation to the state of the data. Other services, on the other hand, react and take action upon this warning, without the need to transfer data between services.

Any of these styles can be used to build and organize distributed systems on the Web. However, the final system architecture also depends on the components it possesses and how they interact with each other, as it reflects how the system is organized [18]. Web distributed systems (Web DS), or any other DS, can assume a *centralized* organization (i.e., client-server systems), a *decentralized* organization (i.e., peer-to-peer systems), or a hybrid between the two (i.e., edge-server systems) [7].

Detailing each type of system architecture used by a DS is not relevant to the present study. However, it is important to highlight that different architectural styles can be used as the basis for systems with different organizations (architectures). Different components (using different technologies) can be organized differently, resulting in different system architectures. We will explore this idea later in the text.

## 3) MIDDLEWARE

Another important topic related to DS is *middleware*. In [27], Krakowiak defines middleware as the software layer that lies between the operating system and the applications on each site of the system. The main objective of middleware is to mask the heterogeneity and distribution of the underlying hardware and operating systems by providing common programming abstractions, thus hiding low-level programming details. He also presented four distinct design patterns for distributed object middleware (i.e., middleware used in distributed architectural styles other than layered ones):

- Proxy: intermediates requests between distributed objects (clients/services) such that the objects do not need to know their mutual locations.

- Factory: used to dynamically create families of related objects.
- Adapter: used to provide a different interface for the functions of a specific object (service) in order to comply with the interface expected by another object.
- Interceptor: used to enhance an existing service with new capabilities or to provide it with different means.

Steen and Tanenbaum [7] used the adapter and interceptor patterns when describing middleware organization in distributed systems, focusing on how they modify an existing service. While an adapter (or *wrapper*) transforms the service's interface, the interceptor transforms its functionality. These patterns can also be used to provide access to existing web systems or to establish an interface for web access.

Middleware support for MAS is usually limited, providing support for communication or a broker infrastructure to be used by agents [28]. According to Weyns, common middleware services such as security, persistency, and transactions are often minimally considered in MAS development. This is because domain-specific middleware for MAS usually considers agent communication as the basis for agent coordination. Other concerns are delegated to domain-independent middleware services. This tendency has been verified in popular frameworks such as JADE [29], Jack [30], and JaCaMo [31].

#### 4) MAS ARCHITECTURES

According to [28], multi-agent systems have the following characteristics: (i) each agent has incomplete information or capabilities to solve the problem at hand (implying a limited viewpoint), (ii) there is no system global control, (iii) data is distributed, and (iv) computation is asynchronous.

Regarding implementation, MAS architectures can follow the same styles as those used in DS. However, at the same time, they also satisfy properties related to agents. Agent theory divides MAS architectures into *deliberative architectures* and *reactive architectures* [32] (which can also be combined into hybrid architectures). Deliberative MAS focuses on (i) translating the environment into a useful description and (ii) *reasoning* upon this information and acting appropriately. Reactive MAS, on the other hand, does not use complex symbolic reasoning, and the agents simply react to the environmental information according to a predetermined program.

The parallel between the DS and MAS architectures involves both implementation and agent theory. From the implementation perspective, MAS can use the same architectural styles as DS, as long as agent theory is satisfied. From the agent theory perspective, an MAS possesses distinct elements (agents, environment, and environment resources) and related processes (planning, perceptions, and actions). These elements can also vary according to the agent paradigm adopted; agents based on the belief-desire-intention (BDI) architecture [33] require belief revision processes. Without diving into all existing agent paradigms, it is sufficient to say

that as long as the elements required by the agent theory are present, an MAS can use the same architectural styles used by a DS.

## B. COMMUNICATION

In distributed systems, the concept of processes is derived from the field of operating systems, where it generally represents “a program in execution”. Considering that distributed systems are composed of processes, “communication” usually refers to interprocess communication. In MAS, communication occurs between agents either via an agent communication language (ACL) or through the environment.

### 1) COMMUNICATION IN DS

All communication models in DS are based on message passing: the differences between each model are based on the message type (what is passed between processes) and how it is done. DS communication models are divided into (i) remote procedure calls (RPC), (ii) message-oriented communication, and (iii) multicast communication.

Remote procedure calls allow processes to invoke procedures from other processes. Despite its name, a message is still passed between processes, containing the name of the procedure invoked and its eventual parameters. This communication is usually performed synchronously, in which case the calling process is halted until a response is received. Message-oriented communication, on the other hand, relies solely on messages passed between processes. Each message can contain a full description of what is required from the other process, and it can also be persisted or treated as an independent entity. This can be illustrated by message-queuing systems that support asynchronous communication between processes by handling and persisting messages. Because message-oriented can be either synchronous or asynchronous, message-queuing systems allow processes to communicate with each other even if they are not available simultaneously. Finally, multicast communication is based on the idea of diffusing information from one process to many other processes. Multicasting is usually asynchronous and can also involve message queuing.

### 2) COMMUNICATION IN MAS

In MAS, communication between agents is also achieved through the use of messages. More specifically, an agent communication language is used to establish a structure for the messages exchanged between agents, both in type and meaning. Agents are also capable of engaging in conversations [34]. A conversation can be seen as a pre-arranged protocol or message exchange pattern oriented towards a specific task or objective.

There are different ACLs in place, but the two most consistently used [35], [36] are the FIPA-ACL [37] and KQML [38]. Both languages are based on the *speech acts theory* [39] and are composed of different *performatives*.<sup>4</sup>

<sup>4</sup><https://stanford.library.sydney.edu.au/entries/speech-acts/>

They can be understood as sentences that describe an environment and influence it simultaneously. As such, messages exchanged between agents can represent actions or communicative actions. Despite being relatively old, these standards are still used [40], [41].

### III. RELATED WORK

Agents have been used in conjunction with DS and Web technologies for a few decades [42], [43], [44], [45], [46], [47], [48]. In particular, from the Web perspective (both accessing and providing access using web-based technologies), agents have been commonly used as assistants for web-based services [43], [45]. In parallel, considering the use of Web technologies in the MAS domain, agents can also be used *as* web services [49], [50]. Dickinson and Wooldridge [47], [51], [52] also proposed the use of cognitive agents as web services. There is also existing work on MAS [53], [54], [55], [56] and agent architectures [57], [58] using web services.

The possible beneficial implications of establishing a parallel between MAS and DS have already been discussed previously [59], [60], [61], [62] from different perspectives. In one of these discussions [59], the author discusses the problem of the lack of direct impact of MAS on system development practice, pointing towards a possible study of the parallels between DS and MAS. This topic was also a point of discussion between many researchers from the community before [63], which resulted in a study of the current status of existing tools for aiding the development of MAS [11].

From a Software Engineering perspective, existing model-driven development (MDD) techniques for developing agent models [64], [65], [66] can also be used to model web-based agents [67], [68], [69]. In 2007, a survey on existing agent methodologies based on the agent-oriented paradigm was published [70] with the aim of evaluating the extent to which existing agent-oriented methodologies were also oriented toward the development of services. From an Agent-oriented Software Engineering (AOSE) perspective, the study demonstrated that all evaluated AOSE methodologies could be used to model service-oriented agents. At this point, the focus was on modeling agents as services or generally using concepts around services to facilitate the interaction between agents.

In 2010, however, Ricci et al. introduced a platform called CARtAgO-WS [48], intended to allow the development of service-oriented applications (SOA) populated by agents. This was a more embracing approach from the MAS perspective, since it took into account both the concepts of artifacts - "objects" or services that the agents could use - and the use of heterogeneous agent architectures (including BDI). This idea was further explored and used with other agent frameworks, such as JaCaMo.<sup>5</sup> More recently, Ciortea et al. [71] proposed another step in the same direction, suggesting that the WWW,

in its current state, could be suitable as middleware for Internet-scale multi-agent systems. This is achieved through the use of a resource-oriented layer that allows the application environment to be decoupled from its deployment context. To demonstrate this proposal, an agent environment was developed using JaCaMo. This environment was specifically aimed at the Internet of Things (IoT), emphasizing the separation between the application environment and the deployment context.

If we focus on work related to exposing agents as web services, most existing frameworks and implementations aim to expose agents' capabilities as web services. This is done either by exposing each of its capabilities as a service [46] or by exposing the agent as a service that can be accessed through the web and perform its functions [43]. This is what happens when agents are exposed as services using the JADE framework,<sup>6</sup> for instance [56]: while the agents can be accessed through web-compliant addresses, they are still implemented in the context of a MAS, using an MAS framework (JADE), being compliant with an ACL (FIPA-ACL), and so on.

In this case, web services are used to make the MAS more flexible in terms of accessibility and usage; however, system organization, modeling, and implementation still occur in the MAS context. This is, of course, by design: Dickinson and Wooldridge argued that agents and web services are distinct, since 'agents provide a distinctive additional capability in mediating user goals to determine service invocations' [47]. However, maintaining the MAS context while adopting web technologies tends to be the norm in other areas related to MAS. In a recent study, for example, Ricci et al. [72] explored the problem of engineering distributed environments and organizations for MAS in a scalable manner. The problem in question is explicitly mapped from the agent perspective onto the web: the MAS is represented as a web resource, and each of its related structures (artifacts, agents, etc.) becomes a sub-resource. In this scenario, agents work with artifacts by accessing their related endpoints through web requests (using REST).

It is interesting to note, however, that while different research has been performed on web-based MAS, most of this work perceives the Internet environment as a means of communication. Even larger proposals such as Agentcities [73] and distributed platforms for data processing [74] rely on web protocols and agent communication standards to establish and coordinate communication processes. In contrast, agents, although exposed as services and sharing resources, are reasoning blocks that take advantage of such a structure. In addition, lower-level communication mechanisms are still based on procedure calls, not taking advantage of newer mechanisms (such as asynchronous message passing). Very recent work in this field aims to address such issues [71], [72], [75], [76].

<sup>5</sup><http://jacamo.sourceforge.net/>

<sup>6</sup><https://jade.tilab.com>

In the current state of the Web, using it as middleware for MAS is appealing. At the same time, modeling MAS while taking full advantage of the web environment (not only as a communication-enabled environment) is partially subject to the constant evolution of the Internet itself. From a higher-level perspective, it is possible to model a purely web-based MAS using existing AOSE methodologies (such as GAIA [77] and O-MASE [78], [79]) while abstracting the Internet environment. From a lower-level perspective, however, different communication protocols, Web-specific components, frameworks, and technologies can influence the way a distributed MAS can be designed using Web-related technologies.

#### IV. DISTRIBUTED SYSTEMS AND MAS DESIGN

To illustrate how MAS can be implemented from a DS perspective, we adopt a specific architectural pattern called Entity Component System (ECS). One of the main reasons for choosing this pattern is the use of a *data-oriented design* (DOD) [80]. The principles behind DOD have been around for a while, but they received this name in 2009.<sup>7</sup> DOD is a program optimization approach that focuses on the efficient usage of the CPU cache for manipulating data. As such, it emphasizes the data layout, separating it from the problem domain (data is separated from logic). Object-oriented design (OOD) [81], in contrast, focuses on encapsulating data and attributes within objects. Consequently, the data are inherently contextualized (making it part of the problem domain).

Different studies have been conducted on the advantages of adopting DOD instead of OOD in high-performance systems [82], [83]. One of the common conclusions of these studies is that using DOD makes it easier to scale systems that process large quantities of data. This is particularly interesting for MAS development because, in a multi-agent system, there can be, hypothetically, a myriad of agents simultaneously performing actions based on the same data (environment). DOD is frequently used in game development, and it is also interesting from the MAS perspective. In both MASs and games, the system must be designed to handle multiple autonomous entities (agents and game characters). In game development, non-playing characters (NPCs) frequently interact with each other and with the environment. Their actions can depend on the environment's state and affect it as a result (most of the time in real-time). Many other aspects of an agent's field can also be present in game development (e.g., behavioral models, coordination, cooperation, and communication). As a result, when looking for a DS architectural pattern to test our hypothesis, it makes sense to examine the game development field.

##### A. THE ENTITY-COMPONENT-SYSTEM PATTERN

Within this context, one of the most used DS architectural patterns in complex, massive multiplayer online games

<sup>7</sup><https://gamesfromwithin.com/data-oriented-design>

(MMOGs) is the Entity Component System (ECS) [84], [85]. This pattern, based on the concept of Component-Based Systems [86], uses DOD. It appeared and evolved within the MMOG community<sup>8,9</sup> and was quickly adopted by modern real-time interactive systems (RIS) frameworks [87] and commercial game engines [88]. As a DOD-focused pattern, it emphasizes composition over inheritance by decoupling data from the logic. It is particularly designed for scenarios in which many different, independent entities interact dynamically with the environment and each other.

The ECS pattern is based on three core abstractions:

- **Entities:** A general-purpose object or actor within the system. They are usually unique identifiers used by the system and thus contain no logic or data. Their purpose is to serve as a reference point for the *components* (see below). In a game (or MAS), an entity can represent an agent or specific element in the environment.
- **Components:** Core data structures used by ECSs. Each component stores a single type of data and, as its name implies, they are used solely as data containers, holding no behavior. A single Entity may have multiple attached Components to define its characteristics. In an ECS implementation, Components are stored in arrays or contiguous blocks of memory to promote cache usage efficiency. Because Components are responsible only for storing data, any modifications to the data structures or their components do not affect the overall logic of the program.
- **Systems:** Logic or behavior used within the program. Systems are designed to operate on Entities with specific Component compositions. Within a game, for example, a System for processing “movement” can be used to update all Entities that possess the combination of components related to “position” and “velocity.”

This structure is heavily aligned with the DOD principles, emphasizing data organization for optimal processing efficiency. ECS implementations also use the concepts of (i) “World” (the environment in which the game happens) and (ii) Events to aid the logic behind systems: all entities are susceptible to different events triggered within the world. A Component is extensible and modifies each Entity by providing specific data in the form of properties. Considering a game environment, a given Entity may possess a Component called “Position,” with properties (X, Y, and Z coordinates) that determine where in the world the Entity is located. The organization of a typical ECS is illustrated in Fig. 1.

##### B. AGENTS: A MINIMAL CONCEPTUAL SET

We also need to define a minimal set of agent-related concepts to incorporate into the adopted architectural pattern. For this task, we decided to use an existing university course as a

<sup>8</sup><http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/>

<sup>9</sup><https://unity.com/dots>

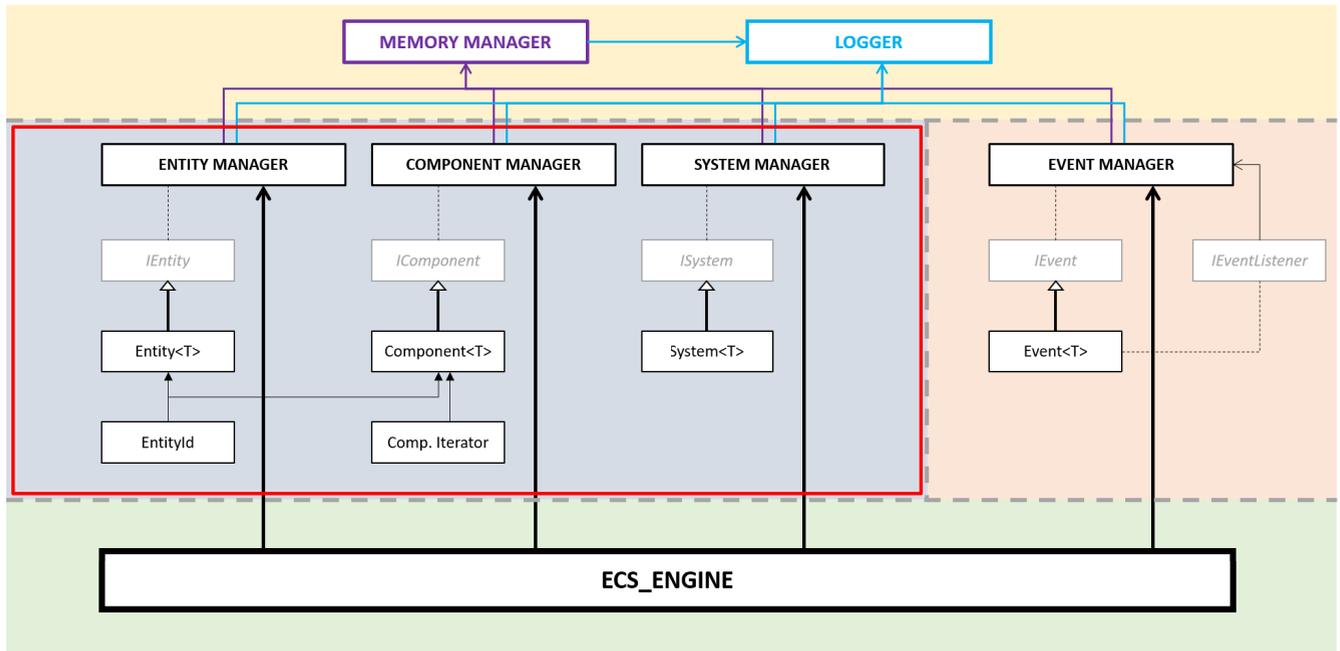


FIGURE 1. A typical ECS architecture.

reference: the introductory MAS course offered by University of São Paulo (USP), Brazil.<sup>10</sup>

Our reasoning was based on two main points. The first refers to the course's public. It was designed for graduate students with no prior knowledge of agent theory. Since our goal involves transferring MAS concepts into a different domain, we benefited from the previous years of curriculum fine-tuning and used the course's structure as a basis. The second point involves the course structure; although mostly conceptual, the course involves a reasonable amount of practical activity, including working with existing MAS frameworks. Minimal knowledge of computer programming is required to do so, which also benefits us, considering that the first ones to benefit from our initiative are DS system engineers and architects.

Considering the points above, we also needed to determine whether the course chosen as a reference was aligned with other university courses on the same topic. This analysis is necessary because our main objective involves leveraging the learning curve necessary to develop MAS. Assuming that agent theory is primarily taught in graduation courses within universities, comparing the structure of the course chosen as a reference against a larger pool of agent-related courses is an adequate verification of its suitability for our intended purposes.

We divide our analysis into two parts. In the first part, we analyzed the syllabi of twenty-six graduation courses related to MAS in two different geographical contexts: local (Brazil) and global. This division was intended to show any

potentially relevant differences in terms of the curriculum structure.

By analyzing these courses, we observed that there were four prevalent books in all syllabi: Wooldridge's [32], Shoham and Leyton-Brown's [89], Russell and Norvig's [1], and Weiss's [90]. From a global perspective, we observed that approximately 81% of them referenced Wooldridge's book, whereas others were much less referenced. Russell and Norvig's book, a common reference for the basic definitions of agents, was referenced in only 54% of all courses. The numbers maintained a similar proportion from a local perspective, except for an increase in the use of Weiss's book by universities in Brazil. This comparison also included the course chosen as a reference. More details of this analysis can be found in Appendix A. Table 1 shows the comparative usage of MAS courses offered by universities worldwide, and Figures 5 and 6 show the distribution of the four main books cited above among both Brazilian and worldwide universities.

The second part of our analysis relates to the structures of the courses. Assuming that each course is structured based on its syllabus, we analyzed the contents of the three books mentioned above to extract a common set of agent-related concepts. We also cross-referenced these concepts with our previous DS and MAS architecture analyses and the latest Computer Science Curricula prepared by the ACM/IEEE-CS/AAAI Joint Task Force [91]. This analysis resulted in the following list of concepts:

- Basic Definitions: agents, different types of agents (reactive, cognitive, hybrid), MAS.
- Agent architectures: cognitive agents, reactive agents, BDI.

<sup>10</sup><https://edisciplinas.usp.br/course/view.php?id=61282>

- Agent-based communication: message passing and ACLs.
- Distributed problem-solving: problem decomposition, solution synthesis.
- Cooperation between agents: cooperative strategies, coordination mechanisms.
- Task planning: planning algorithms, dynamic task allocation.
- Coordination between agents: coordination protocols, synchronization techniques.
- Negotiation between agents: negotiation strategies, conflict resolution.
- Organizations, rules, and norms: organizational structures, normative systems.
- Fault Tolerance: fault detection and recovery, redundancy strategies.
- Learning and Adaptation: machine learning in MAS, reinforcement learning for agent adaptation.
- Scalability and Performance: scalability techniques, performance optimization.
- Security and Privacy: secure agent communication, privacy-preserving mechanisms.
- Human-agent Interaction: interfaces for human-agent interaction, user experience, and usability.
- Agent-Based Simulation: simulation environments, applications of agent-based modeling.
- Interoperability: standards and protocols for interoperability, integration with other systems.
- Autonomous Decision-Making: decision-making algorithms, autonomy levels.
- Ethics and Trust: ethical considerations in MAS, building trust in agent systems.
- Resource Management: resource allocation strategies, efficient resource utilization.
- Advanced AI Techniques in MAS: integrating generative AI and applying deep learning in MAS.

In the list above, “BDI” refers to Belief-Desire-Intention agent architectures [33], and “ACLs” refers to Agent Communication Languages [32]. This preliminary list covers our observations so far, with a list of MAS-related topics that we judged essential to anyone designing a MAS.

It is also interesting to observe that the list above shows high synergy with distributed systems. In DS, architectural patterns provide a structured way to organize components, manage communication, and ensure aspects such as scalability and fault tolerance. The MAS architecture is built on the same principles, extending them with intelligent agents capable of making autonomous decisions, learning, and coordinating among themselves.

From the same perspective, not all of these topics are necessary to design every MAS. Depending on the system’s requirements, some might not be used at all. Advanced AI techniques will not necessarily be present in every MAS, and purely architectural topics (such as fault tolerance) will not necessarily be addressed, considering the number

of frameworks and service providers that encapsulate such concerns. We also need to remember that we want to use this conceptual set in a practical study that involves creating a course on engineering MAS. Therefore, the information taught in such a course must be adequate. Ultimately, we aim for a *minimal conceptual set*.

With the considerations above, the resulting minimal conceptual set is shown below:

- Basic definitions: agents, different types of agents, MAS.
- Agent architectures: cognitive agents, reactive agents, BDI.
- Agent-based communication: message passing and ACLs.
- Cooperation between agents.
- Task planning.
- Coordination between agents.

We excluded some engineering aspects from this list, such as “Security” and “Fault Tolerance.” Although there are specific sub-fields for this kind of research within the agent research field, we believe that these topics should be considered as part of the architectural requirements (similar to resources). Therefore, they do not belong to this list.

### C. BRINGING MAS INTO ECS

To illustrate how MAS concepts can be used with the ECS pattern, we extended an existing MMOG developed with the ECS architectural pattern. This was our first practical study on leveraging MAS development. The objective of this study was to implement a typical MAS scenario within the system, following the premise that MAS elements can be brought to the chosen architectural pattern. We chose an open-source MMOG called Terasology<sup>11</sup> for our base system. This game is implemented using the ECS architectural pattern, and it is open source, giving us access to the core system implementation.

Because we used a game platform, we chose a scenario that would use the main concepts involved in an MAS in the context of the game. This meant using the existing structure of entities and components to (i) create our agents, (ii) create a structure that could be used to group these agents into organizations, (iii) create a structure that could allow collective problem-solving, and (iv) making it all without bringing specialized knowledge from the MAS domain.

From the game perspective, the main problem was to establish collective problem-solving. Using an ECS implementation means isolating the logic from both data and entity identification, which means that all group-related functionalities must be handled by system-level logic. In addition, because we wanted to explore concepts related to deliberative MAS, we decided to implement agents using the notion of behaviors. Popular games already use *behavior trees* [92] to coordinate characters in-game, allowing this structure

<sup>11</sup><https://terasology.org/>

to be used without additional MAS-specific concepts and implementation.

With these considerations in mind, we implemented the aforementioned structures. The agents were described in JSON files. Each file described the agent functionalities, which were implemented as Components. These Components included a base specification for all agents, such as rendering attributes (color, skeletal structure, etc.), the organizations (groups) to which the agent belongs, and a reference to a specific behavior tree. An example of an agent descriptor file is presented below:

```
{
  ``parent``: ``baseAgent``,
  ``Behavior``: {
    ``tree``: ``stray``
  },
  ``Groups``: {
    ``groups``: [ ``Rebels``, ``Green`` ]
  }
}
```

In this agent file, the `parent` description specifies a base for all agents, including their in-game rendering attributes (color, skeletal structure, etc.). The `Behavior` description specifies a specific behavior tree in the form of a JSON-like file (there is no need to implement a specific behavior tree within the source code). All behaviors were described modularly in pre-defined blocks (similar to a workflow). Finally, the `Groups` description specifies the organizations (groups) to which the agent belongs (`Rebels` and `Green`). Similarly to agents and behavior trees, groups are also described by JSON-like files:

```
{
  ``groupLabel``: ``Rebels``,
  ``syncBehavior``: ``false``,
  ``behaviorTree``: ``flockOrAttack``
}
```

Besides the identification (`groupLabel`), the other two parameters refer to the need for synchronized behavior (where all agents act in unison) and the behavior tree that must be imposed on all agents within the group.

Because an ECS Entity does not hold any data other than its own unique identification, its descriptions are in the form of a set of Components. Each Component within the system is implemented using the programming language used by the system (in this case, Java). While logic implementation must be performed within the source code, it is unnecessary to incorporate an agent programming language or declarative expressions of any sort (such as intentions). The most complex notions related to behavior trees (such as how they are interpreted) are encapsulated within a structure that uses a descriptive text file as a reference. Agents can be found solely by their component properties, and communication between them is handled by the system architecture (without needing

an ACL). All functionalities can be described as components and the logic behind them.

This implementation is publicly available as part of Terasology's source code, and was funded as an individual project in the 2019 edition of Google Summer of Code.<sup>12</sup> We adopted the ECS architectural pattern for two reasons. The first one was related to its use and application. Despite their size, MMOGS using this pattern have been deployed in several configurations, from centralized to Web-based DS. Different aspects of more complex DS (e.g., scalability and resource management) can be used without requiring extensions or additions to the pattern. Because it is a data-oriented pattern, every logical requirement (directly related to the agent-related knowledge necessary for each MAS) can be designed and implemented within the pattern. However, existing MASs that follow typical object-oriented patterns have already shown that it is possible to implement any necessary agent capability [11], although they do not necessarily adhere to the original architectural pattern. Although using other general-purpose DS patterns could still showcase the same agent capabilities, we decided to choose a fresh approach in this area.

The second reason relates to the "data-oriented" aspect. This characteristic is especially favorable if we want to bring MAS concepts into the Web DS context because most agent-specific techniques and capabilities are based on data structure and processing. Again, while it is not impossible to do the same with other architectural patterns, using ECS makes it easier to encapsulate the agent capabilities within a system.

If we consider all the possible MAS elements that can be introduced into the system design using this or any DS architectural pattern, they should follow the same rules: they should not break the pattern and should be encapsulated or isolated as much as possible. Otherwise, we would end up reverting to an MAS architecture that uses DS elements, not the opposite.

As the next step in our study, we decided to use the ECS architectural pattern to design an MAS framework. We showed that using an existing ECS implementation makes it possible to build a community of agents. However, adapting an already existing ECS-based system has several disadvantages. For example, it would be difficult to use the same game system used in our case study to control traffic lights in the real world. In addition, because we would be using the system for a different purpose from gaming, it would contain many undesirable software components (e.g., anything related to 3D rendering).

At this point, we will not detail the design of the framework *per se*, as this is not necessary within the scope of the present paper. However, we used it in the next part of our study, as described in the following paragraphs.

<sup>12</sup><https://summerofcode.withgoogle.com/archive/2019/projects/>

#### D. LEVERAGING THE CURVE

Bringing Web technologies into the MAS context is an intuitive approach. However, this creates a disadvantage from the overall engineering perspective. While Web technologies are widely available and can be easily learned by developers worldwide, the same does not apply to MAS theory. Any project implementing an MAS requires considerable specialized knowledge from the developers involved, which poses a challenge to the large-scale adoption of MAS [63].

With this in mind, our second practical study in this work was to use the concepts from the first study to teach students without any previous knowledge of agent theory on how to implement a MAS. The idea behind this study was simple: we concluded that it was *possible* to create a framework using an existing DS architectural pattern, but we wanted to determine if it was *necessary*. For this purpose, we divided our efforts into two parts. The first was to use the previously designed framework to allow the students to encapsulate as many concepts as needed for their projects within the same common platform. The second part was to create a different version of the same course without using an ECS-based framework.

For the first part, we used the existing MAS introductory course offered by USP, Brazil (PCS-5703: multi-agent Systems).<sup>13</sup> We also used this course as the basis for defining our minimal conceptual set of agent-related concepts, implemented within the framework used by students. At the end of the course, we conducted a guided usability experiment aimed at our implemented framework. This project was submitted and approved by the Ethics Committee<sup>14</sup> under the reference CAAE 40985420.0.0000.5390.

The guided experiment included questionnaires to evaluate the initial proficiency of the students regarding MAS and programming. One of the prerequisites for this course was previous exposure to the Java programming language since the students would have to use a Java-based implementation of the framework we previously designed. Using these questionnaires, we were able to profile the students taking the course as follows:

- Software development proficiency: low (13%), medium (56%), and high (31%).
- Professional software development experience: none (13%), 6 months or less (6%), between 6 months and 2 years (31%), and 2 years or more (50%).
- JAVA development proficiency: none (19%), low (44%), medium (31%), and high (6%).
- Development experience using APIs: Yes (37%), No (63%).
- Development experience using Distributed Systems: Yes (19%), No (81%).

During the course, we required students to implement an MAS for an auction mechanism. Our DS-based implementation allowed them to work with all topics covered by our proposed minimal conceptual set. The required auction

mechanism was based on a tutorial scenario for an existing MAS framework, JaCaMo.<sup>15</sup> The scenario in question is called “house building”.<sup>16</sup> We chose it for the required implementation because it is already a proven introductory scenario for building MAS, and it encompasses all theoretical topics within our minimal conceptual set.

In the second part of our study, we decided to create a different course for the students using an engineering-based approach. The new course was offered in 2023 at USP, Brazil, under the name “PCS-5045: MAS Engineering I”.<sup>17</sup> We have amended the previous Ethics Committee submission (under the same reference), which was approved before we started our lectures.

For this course, the requisites were pretty much the same, except for one: we did not ask for any Java proficiency since the students would not use any specific framework. We also asked them to design an MAS to solve a particular problem, chosen from a list of possible cases based on real-world scenarios. While the scenario used in the previous course was ideal for an MAS implementation, it was also *leading* in terms of system requirements - in the sense that it was conceived as part of an implementation tutorial. For this reason, we decided to present the students with a set of scenarios that *could* be solved using MAS.

The students taking the new course were profiled as follows:

- Software development proficiency: low (23%), medium (33%), and high (44%).
- Professional software development experience: none (11%), 6 months or less (11%), between 6 months and 2 years (33%), 2 years or more (45%).
- Development experience using APIs: Yes (56%), No (44%).
- Development experience using Distributed Systems: Yes (33%), No (67%).

We also used the same minimum conceptual set in this course’s lectures. However, since we were not providing a framework for them to use, we presented the basics of the DS and MAS architectural patterns. We also allowed them to use SPADE<sup>18</sup> for their final projects. SPADE is a minimal MAS framework written on top of Python that provides the basic structures for a MAS. However, because we intended to detach their development experience from specific MAS-related tools as much as possible, we only allowed them to use the basic agent structure provided by the framework. This decision was taken considering (i) the duration of the course, (ii) the complexity of the proposed projects, and (iii) the fact that the basic concept of an “agent” - as implemented within the SPADE framework - is

<sup>15</sup><https://jacamo-lang.github.io/>

<sup>16</sup><https://sourceforge.net/p/jacamo/code/HEAD/tree/tut/house-building/>

<sup>17</sup><https://edisciplinas.usp.br/course/view.php?id=110364>

<sup>18</sup><https://spade-mas.readthedocs.io/en/latest/readme.html>

<sup>13</sup><https://edisciplinas.usp.br/enrol/index.php?id=83096>

<sup>14</sup><https://plataformabrasil.saude.gov.br>

already used in the DS area, and it is implemented by widely used DS-specific frameworks such as Akka<sup>19</sup> and Orleans.<sup>20</sup>

**E. LEARNING OUTCOMES**

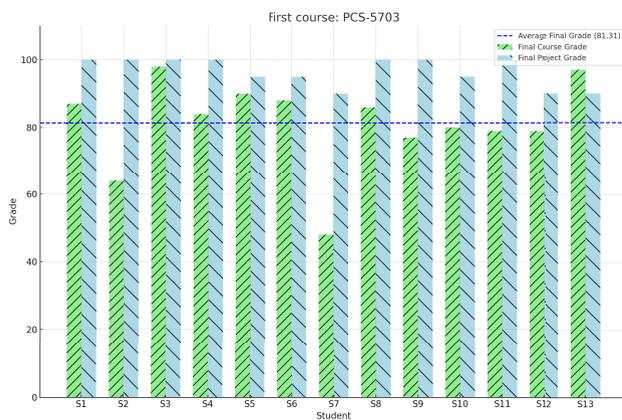
The expected learning outcomes from both courses were:

- Knowledge: Understanding the key characteristics of agents and multi-agent systems.
- Skills: The ability to engineer basic multi-agent systems to solve cooperative problem-solving scenarios.

To measure these outcomes, students were evaluated according to the following method:

- Reading: Each week, students received an article related to the topic of the upcoming class. They were required to read the article and submit a summary of its content the following week.
- Practical work: During the course, the students were required to complete three to four hands-on projects by applying the concepts studied in class. These projects were cumulative, with each new project building on the previous project.
- Seminar: At the end of the course, the students were required to present a seminar on the project they developed, explaining how the concepts studied in class were applied.

Figures 2 and 3 show the final grades obtained by students in both courses. In each graph, two different grades (Y-axis) are shown for each student (X-axis). The left column shows the final grade obtained by the students in the course, and the right column shows the grade obtained by the students in the final project.

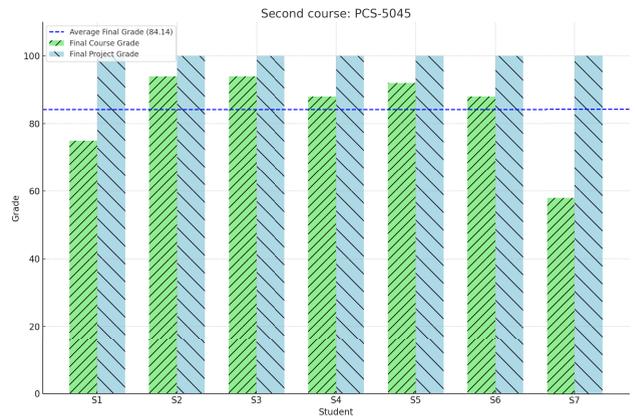


**FIGURE 2. Learning outcome achieved: Grades for the course PCS-5703 (Source: author).**

The average final grade for the first course was 81.31; for the second course, it was 84.10 (out of 100). It is interesting to note that the grade obtained by each student for the final project was always higher than the average final grade. For the first course, none of the students scored lower than 90 in

<sup>19</sup><https://akka.io/>

<sup>20</sup><https://learn.microsoft.com/en-us/dotnet/orleans/>

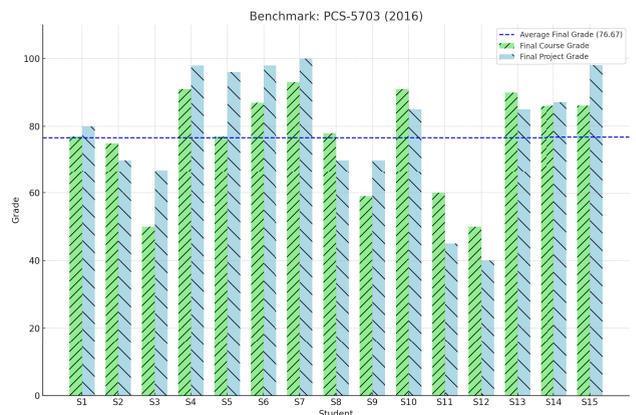


**FIGURE 3. Learning outcome achieved: Grades for the course PCS-5045 (Source: author).**

the final project, whereas in the second course, all students scored 100 out of 100 in the final project.

These grades can be analyzed from the perspective of each course. In the first course, the students were taught using our designed framework. They also used this same framework in their final project. In the second course, the students were oriented towards DS tools and techniques. If we consider that in both courses, (i) the agent-related topics evaluated in each project were the same and (ii) the programming proficiency of the students was, on average, the same, we can infer that the newly adjusted course had a positive impact on their learning outcomes.

In terms of the benchmark, we used the original course (PCS-5703), as it was taught in 2016. In this instance of the course, while the syllabus and topics were the same as we used, the students did not have access to the framework we developed. Consequently, the practical project was based on the MAS-specific framework JaCaMo:



**FIGURE 4. Benchmark: Past grades for the course PCS-5703 (Source: author).**

When we used the grades obtained by the students in a past instance of the course PCS-5703 (Figure 4) as a benchmark, the immediate observation was that the framework we

developed and used in the latest instance of the same course also had a positive impact on the learning outcome.

However, there are a few limitations to the analysis of these courses. The first refers to the benchmark grades: although the course topics were the same, the scope of the practical project conducted by the students was different. This is partly due to the steep learning curve of using MAS-specific frameworks in general. While we understand that part of the merit of using a custom-made framework is to avoid this learning curve, a proper benchmark should involve the same scope for practical projects in both instances of the course.

The second limitation concerns the previous exposition of students on the course topic. In both courses, one or two students had contact with agent-specific frameworks after the course as part of their master's or doctorate topics. Individual feedback from these students regarding using different tools (custom framework or DS tooling) was positive. However, we would need more instances of the same courses and control groups to properly measure the impact of the changes we introduced.

Finally, we collected feedback from students in both courses at the end of the term (during their seminar presentation). This feedback was unanimously positive: the students felt it was much easier to understand agent theory by applying the concepts learned to a practical project during the course.

## V. DISCUSSION AND FUTURE WORK

In the present work, we trace a historical parallel between the evolution of Distributed Systems and Multi-Agent Systems research fields, pointing out the common origins and similarities between them. We compared architectural styles and patterns used in the implementation of MAS and DSs, with an emphasis on the communication mechanisms typically used.

This parallel was then used to point out that while web developers can seamlessly implement distributed systems, the same is not valid for multi-agent systems. Implementing an MAS requires specific knowledge of agent-related theory, even when using available agent-oriented frameworks and libraries. This was pointed out in the past as one of the possible reasons impeding MAS from being widely adopted by the industry [63]. Therefore, we decided to follow the example of the Web and introduce concepts from MAS theory into the DS context.

After establishing an architectural parallel between the MASs and DSs, we created a minimal set of MAS-related concepts that could be introduced into the DS domain. Once this set was created, we tested these concepts within the context of our proposal. For this, we took several cumulative steps. First, we chose a DS architectural pattern representative enough for our study, considering the architectural comparison previously made and the relevant complexity considerations. The pattern chosen was then used in two experiments. The first involved implementing agent-related concepts from our minimal set into an existing system. This

experiment was performed and validated as an accepted result in the 2019 edition of the Google Summer of Code.

The second step was to use the chosen pattern to design a framework that accommodated all concepts contained in our minimal set. Designing such a framework was important for studying the architectural and technical limitations of our approach. This design (and its subsequent implementation) will be discussed in detail in a separate publication.

The next step was to validate the conceptual set more broadly. For this purpose, we have taught two graduate courses at the university. With a more conceptual approach, the first used the previously designed framework and required students with no previous MAS-related knowledge to implement a specific MAS. The second had a practical approach, and the students (with no prior knowledge of MAS-related theory) were required to design and implement an MAS to solve a specific problem. In both cases, the students were only provided with agent theory contained within our proposed minimal conceptual set.

We profiled the students taking both courses and found that their experience and proficiency in software development were similar in both cases. However, students taking the second course had more previous contact with APIs and Distributed Systems.

In both courses, all students were able to implement the required MASs within the DS domain. In the first course, we used a scenario adapted from an implementation tutorial for JaCaMo. Because we were using our framework implementation, we chose this scenario as part of a proven implementation tutorial within the MAS domain. In the second course, however, we asked the students to choose between multiple real-world scenarios with problems that could be solved in several ways, including implementing MAS. We wanted to understand how much of the recently learned agent theory would be used by the students in their implementation.

The learning outcomes of both the courses were positive. In addition to the high-grade average observed in both cases, the final feedback from the students from both courses at the end of the term was unanimously positive: they felt it was much easier to understand agent theory by applying the concepts learned in a practical project along the course. We intend to offer these courses again in the next few years.

In light of this study and the nature of all implementations, we concluded that our approach was successful, although limited by the number of experiments. Considering the nature of our approach, we can use it not only to develop new MASs but also to bring MAS-specific capabilities into existing systems. Establishing a minimum MAS conceptual set and using it in conjunction with existing distributed technologies allows for the development of specialized MAS without depending on specific MAS frameworks. At the same time, it will enable software engineers to incorporate the latest AI advancements and techniques into their work without losing touch with existing agent research and without abstaining from using DS-specific frameworks or techniques.

In terms of broader implications, we were also able to observe a few interesting findings in this study. The first relates to the existing synergy between MAS and DS concepts. By analyzing the core elements taught in agent-related courses, it was clear that most of these concepts were common to DS areas. Although this can be easily explained by the common origin of both research fields, it does not alter the fact that there is a fertile space for cross-pollination between DS and MAS. By using DS-specific tools and techniques, MAS engineers can benefit from topics already well-discussed and treated in the DS field, such as fault tolerance, load balancing, and scaling. Simultaneously, problems usually belonging to the MAS domain can be solved by applying DS techniques. The concept of coordination in MAS is well-aligned with the peer-to-peer (P2P) architecture [93] in DS, and agent modeling and communication can benefit from the use of microservices [94]. The scope of the present work does not include a full study of which of these techniques has already been incorporated across both research fields. However, considering the present state of both fields, we believe that introducing MAS problems into the DS domain would yield better practical results than doing the opposite. We intend to explore this perspective in future studies.

It is also important to note that this approach can be generalized to other AI and engineering systems beyond the MAS and DS domains. Some of these are already being explored, such as integrating computer vision and autonomous systems seen in self-driving cars or combining deep learning and edge computing in smart cities. In a broader scenario, MAS and DS can be used in conjunction with other AI and engineering techniques to produce more sophisticated systems. The key and arguably most important benefit of cross-pollination between AI and engineering fields is leveraging the problems already solved in the engineering field with the latest AI advancements. This perspective is part of what fuels our research, and we believe many opportunities have yet to be fully explored.

## APPENDIX A COMPARATIVE ANALYSIS OF SYLLABI IN MULTI-AGENT SYSTEMS COURSES

This appendix contains a detailed analysis of the usage of four major textbooks in multi-agent systems (MAS) courses, as referenced in their syllabi. We analyzed 26 graduate courses taught in universities worldwide, comparing the usage of each book in Brazilian universities, international universities, and globally (altogether).

### A. TEXTBOOKS FOR MAS GRAD COURSES

By analyzing the course syllabi, we found that the prevalence of the following textbooks (editions and publication years may vary):

- 1) **Multi-agent Systems: Algorithmic, Game-theoretic, and Logical Foundations**, by Yoav Shoham and Kevin Leyton-Brown.
- 2) “**An Introduction to multi-agent Systems**,” by Michael Wooldridge.
- 3) “**Artificial Intelligence: A Modern Approach**,” by Stuart Russell and Peter Norvig.
- 4) “**Multi-agent systems: A modern approach to distributed artificial intelligence**,” by Gerard Weiss.

### B. DATA SOURCES

The following universities and courses were surveyed in the analysis:

#### 1) BRAZILIAN UNIVERSITIES

- **University of São Paulo (USP)**: Course on *Multi-Agent Systems*.
- **Federal University of Pernambuco (UFPE)**: Course on *Intelligent Agents*.
- **Federal University of Santa Catarina (UFSC)**: Course on *Multi-Agent Systems*.
- **University of Brasília (UnB)**: Course on *Multi-Agent Systems*.
- **Federal University of Bahia (UFBA)**: Course on *Autonomous Agents and Multi-Agent Systems*.
- **Federal University of Maranhão (UFMA)**: Course on *Multi-Agent Systems*.
- **Federal University of Rio Grande do Sul (UFRGS)**: Course on *Autonomous Agents and Multi-Agent Systems*.

#### 2) INTERNATIONAL UNIVERSITIES

- **University of Western Australia**: Course on *Intelligent Agents*.
- **KU Leuven**: Course on *Multi-Agent Systems*.
- **Technical University of Denmark**: Course on *Artificial Intelligence and Multi-Agent Systems*.
- **University of Bologna**: Course on *Multi-Agent Systems*.
- **Maastricht University**: Course on *Agents and Multi-Agent Systems*.
- **University of Amsterdam**: Course on *Multi-Agent Systems*.
- **Norwegian University of Science and Technology (NTNU)**: Course on *Multi-Agent Systems and Game Theory*.
- **University of Oslo**: Course on *Multi-Agent Systems*.
- **University of Auckland**: Course on *Intelligent Software Agents*.
- **University of Algarve**: Course on *Multi-Agent Systems*.
- **University of Lisboa**: Course on *Multi-Agent Systems*.
- **University of Minho**: Course on *Agents and Multi-Agent Systems*.
- **University of Porto**: Course on *Multi-Agent Systems*.
- **Barcelona School of Informatics (FIB)**: Course on *Introduction to Multi-Agent Systems*.

- **Swiss Federal Institute of Technology Lausanne (EPFL):** Course on *Intelligent Agents*.
- **University of Liverpool:** Course on *Multi-Agent Systems*.
- **University of Southampton:** Course on *Intelligent Agents*.
- **Carnegie Mellon University:** Course on *Multi-Agent Systems*.
- **Harvard University:** Course on *Multi-Agent Systems*.

**C. COMPARATIVE USAGE OF MAS BOOKS IN COURSES**

The following table summarizes the usage of the three main textbooks in Brazilian and international universities, along with their global usage (a combination of both Brazilian and international data).

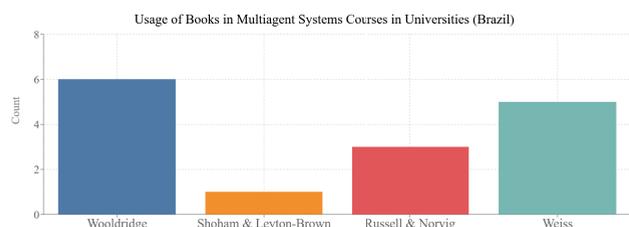
**TABLE 1. Comparative usage of the prevalent MAS books by region: Local (Brazil), International (Intl.), global.**

Book	Brazil (%)	Intl. (%)	Global (%)
Wooldridge	85.71%	78.95%	80.77%
Shoham & Leyton-Brown	14.29%	36.84%	30.77%
Russell & Norvig	42.86%	57.89%	53.85%
Weiss	71.43%	15.79%	30.77%

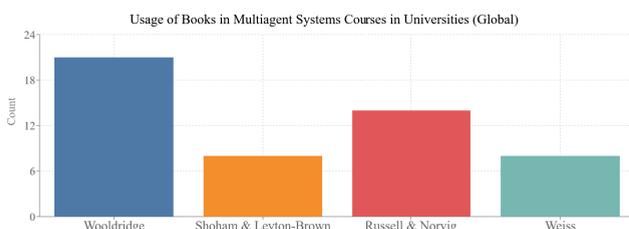
**D. ANALYSIS AND GRAPHICS**

Our analysis shows that *Wooldridge* is used in the majority of courses related to multi-agent systems. Although *Russell and Norvig*'s book is a common reference for AI courses, it does not appear in all syllabi. Also, the books by *Shoham and Leyton-Brown* and *Weiss* show different levels of usage in different regions.

For a graphical representation, please refer to Figure 6.



**FIGURE 5. Usage of books in multi-agent system courses in brazilian universities (Source: authors).**



**FIGURE 6. Usage of books in multi-agent system courses globally (Source: authors).**

**REFERENCES**

- [1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.
- [2] R. Gasser and M. N. Huhns, *Distributed Artificial Intelligence*, vol. 2. San Mateo, CA, USA: Morgan Kaufmann, 2014.
- [3] R. E. Fikes, "A commitment-based framework for describing informal cooperative work," *Cognit. Sci.*, vol. 6, no. 4, pp. 331–347, Dec. 1982.
- [4] Y. Müller, "Decentralized artificial intelligence," in *Decentralised AI*, 1st ed., Y. Demazeau and J.-P. Müller, Eds., Jul. 199., pp. 3–13.
- [5] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*, vol. 1. Reading, MA, USA: Addison-Wesley, 1999.
- [6] H. Van Dyke Parunak, "Applications of distributed artificial intelligence in industry," *Found. Distrib. Artif. Intell.*, vol. 2, no. 1, p. 18, 1996.
- [7] M. Van Steen and A. S. Tanenbaum, *Distributed Systems*. Scotts Valley, CA, USA: CreateSpace Independent Publishing Platform, 2017.
- [8] T. Berners-Lee, D. Dimitroyannis, A. J. Malinckrodt, and S. McKay, "World wide web," *Comput. Phys.*, vol. 8, no. 3, pp. 298–299, 1994.
- [9] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Sci. Amer.*, vol. 284, no. 5, pp. 34–43, May 2001.
- [10] J. Webber, S. Parastatidis, and I. Robinson, *REST in Practice: Hypermedia and Systems Architecture*. Sebastopol, CA, USA: O'Reilly Media, 2010.
- [11] V. Mascardi, "Fantastic MASs and where to find them: First results and lesson learned," in *Proc. Int. Workshop Eng. Multi-Agent Syst.*, 2023, pp. 233–252.
- [12] B. C. Neuman, "Scale in distributed systems," in *Readings in Distributed Computing Systems*, T. Lee Casavant and M. Singhal, Eds., Los Alamitos, CA, USA: IEEE CS Press, 1994, pp. 463–489.
- [13] F. Bergenti, "A discussion of two major benefits of using agents in software development," in *Proc. Int. Workshop Eng. Societies Agents World*. Berlin, Germany: Springer, 2002, pp. 1–12.
- [14] P. Leitão, U. Inden, and C.-P. Rückemann, "Parallelising multi-agent systems for high performance computing," in *Proc. 3rd Int. Conf. Adv. Commun. Comput. (INFOCOMP)*, vol. 6, 2013, p. 1.
- [15] A. Rousset, B. Herrmann, C. Lang, and L. Philippe, "A survey on parallel and distributed multi-agent systems for high performance computing simulations," *Comput. Sci. Rev.*, vol. 22, pp. 27–46, Nov. 2016.
- [16] J. R. Chen, S. R. Wolfe, and S. D. Wragg, "A distributed multi-agent system for collaborative information management and learning," in *Proc. 9th Int. Conf. Inf. Knowl. Manage. (CIKM)*. New York, NY, USA: Association for Computing Machinery, 2000, pp. 382–388. [Online]. Available: <https://doi.org/10.1145/354756.354844>
- [17] C.-J. Su and C.-Y. Wu, "JADE implemented mobile multi-agent based, distributed information platform for pervasive health care monitoring," *Appl. Soft Comput.*, vol. 11, no. 1, pp. 315–325, Jan. 2011.
- [18] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Reading, MA, USA: Addison-Wesley, 2003.
- [19] B. V. Selic, *On the Semantic Foundations of Standard UML 2.0*. Berlin, Germany: Springer, 2004, pp. 181–199.
- [20] B. Selic, "UML 2: A model-driven development tool," *IBM Syst. J.*, vol. 45, no. 3, pp. 607–620, 2006.
- [21] R. Perrey and M. Lycett, "Service-oriented architecture," in *Proc. Symp. Appl. Internet Workshops*, 2003, pp. 116–119.
- [22] E. Newcomer and G. Lomow, *Understanding SOA With Web Services*. Reading, MA, USA: Addison-Wesley, 2005.
- [23] D. Garlan and M. Shaw, "An introduction to software architecture," in *Advances in Software Engineering and Knowledge Engineering*. Singapore: World Scientific, 1993, pp. 1–39.
- [24] K. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *Proc. 11th ACM Symp. Operating Syst. Princ.*, 1987, pp. 123–138.
- [25] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, N. Tatbul, S. Zdonik, and M. Stonebraker, "Monitoring streams—A new class of data management applications," in *Proc. 28th Int. Conf. Very Large Databases*. Burlington, MA, USA: Morgan Kaufmann, 2002, pp. 215–226.
- [26] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003.
- [27] S. Krakowiak, "Middleware architecture with patterns and frameworks," Grenoble, France, Tech. Rep., 2007.
- [28] D. Weyns, *Architecture-Based Design of Multi-Agent Systems*. Berlin, Germany: Springer, 2010.
- [29] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems With JADE*, vol. 7. Hoboken, NJ, USA: Wiley, 2007.

- [30] M. F. Wood and S. A. DeLoach, "An overview of the multiagent systems engineering methodology," in *Proc. Int. Workshop Agent-Oriented Softw. Eng.* Berlin, Germany: Springer, 2000, pp. 207–221.
- [31] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, "Multi-agent oriented programming with JaCaMo," *Sci. Comput. Program.*, vol. 78, no. 6, pp. 747–761, Jun. 2013.
- [32] M. Wooldridge and N. R. Jennings, "Agent theories, architectures, and languages: A survey," in *Proc. Int. Workshop Agent Theories, Architectures, Lang.* Berlin, Germany: Springer, 1994, pp. 1–39.
- [33] A. S. Rao and M. Georgeff, "BDI agents: From theory to practice," in *Proc. ICMAS*, 1995, pp. 312–319.
- [34] Y. Labrou, "Standardizing agent communication," in *Multi-Agent Systems and Applications* (ECCAI Advanced Course on Artificial Intelligence). Berlin, Germany: Springer, 2001, pp. 74–97.
- [35] S. Li and M. M. Kocar, *Agent Communication Language*. New York, NY, USA: Springer, 2013.
- [36] R. Kamdar, P. Paliwal, and Y. Kumar, "A state of art review on various aspects of multi-agent system," *J. Circuits, Syst. Comput.*, vol. 27, no. 11, Oct. 2018, Art. no. 1830006.
- [37] P. D. O'Brien and R. C. Nicol, "Fipa-towards a standard for software agents," *BT Technol. J.*, vol. 16, no. 3, pp. 51–59, 1998.
- [38] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent communication language," in *Proc. 3rd Int. Conf. Inf. Knowl. Manage.*, 1994, pp. 456–463.
- [39] J. R. Searle and J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*, vol. 626. Cambridge, U.K.: Cambridge Univ. Press, 1969.
- [40] S. Răileanu, F. D. Anton, T. Borangiu, and S. Anton, "Design of high availability manufacturing resource agents using jade framework and cloud replication," in *Service Orientation in Holonic and Multi-Agent Manufacturing*. Berlin, Germany: Springer, 2018, pp. 201–215.
- [41] M. F. Bloss, R. M. da Silva, and H.-M. Wee, "A framework for designing supply chain disruptions management considering productive systems and carrier viewpoints," *Int. J. Prod. Res.*, vol. 56, no. 15, pp. 5045–5061, Aug. 2018.
- [42] N. R. Jennings, K. Sycara, and M. Wooldridge, "A roadmap of agent research and development," *Auto. Agents Multi-Agent Syst.*, vol. 1, no. 1, pp. 7–38, 1998.
- [43] H. Lieberman, "Letizia: An agent that assists web browsing," *IJCAI*, vol. 1995, pp. 924–929, Aug. 1995.
- [44] O. Etzioni, "Moving up the information food chain: Deploying softbots on the world wide web," in *Proc. Nat. Conf. Artif. Intell.*, Aug. 1996, pp. 1322–1326.
- [45] L. Ardissono, C. Barbero, A. Goy, and G. Petrone, "An agent architecture for personalized web stores," in *Proc. 3rd Annu. Conf. Auto. Agents*, Apr. 1999, pp. 182–189.
- [46] D. Greenwood and M. Calisti, "Engineering web service-agent integration," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 2, Oct. 2004, pp. 1918–1925.
- [47] I. Dickinson and M. Wooldridge, "Agents are not (just) web services: Considering bdi agents and web services," in *Proc. Workshop Service-Oriented Comput. Agent-Based Eng.*, Utrecht, The Netherlands, 2005, pp. 1–9.
- [48] A. Ricci, E. Denti, and M. Piunti, "A platform for developing SOA/WS applications as open and heterogeneous multi-agent systems," *Multiagent Grid Syst.*, vol. 6, no. 2, pp. 105–132, Jun. 2010.
- [49] J. Hendler, "Agents and the semantic web," *IEEE Intell. Syst.*, vol. 16, no. 2, pp. 30–37, Mar. 2001.
- [50] M. N. Huhns, "Agents as web services," *IEEE Internet Comput.*, vol. 6, no. 4, pp. 93–95, Jul. 2002.
- [51] X. Bai, G. Dai, D. Xu, and W.-T. Tsai, "A multi-agent based framework for collaborative testing on web services," in *Proc. 4th IEEE Workshop Softw. Technol. Future Embedded Ubiquitous Syst., 2nd Int. Workshop Collaborative Comput., Integr., Assurance*, Apr. 2006, pp. 205–210.
- [52] B. Hirsch and J. W. P. Ng, "Education beyond the cloud: Anytime-anywhere learning in a smart campus environment," in *Proc. Int. Conf. Internet Technol. Secured Trans.*, Dec. 2011, pp. 718–723.
- [53] C. Muldoon, "An agent framework for ubiquitous services," Ph.D. dissertation, Univ. College Dublin, Dublin, Ireland, 2007.
- [54] A. Thiele, S. Kaiser, T. Konnerth, and B. Hirsch, "MAMS service framework," in *Proc. Int. Workshop Service-Oriented Computing: Agents, Semantics, Eng.* Berlin, Germany: Springer, 2009, pp. 126–142.
- [55] D. I. Tapia, S. Rodríguez, J. Bajo, and J. M. Corchado, "FUSION@, a SOA-based multi-agent architecture," in *Proc. Int. Symp. Distrib. Comput. Artif. Intell. (DCAI)*. Berlin, Germany: Springer, Sep. 2008, pp. 99–107.
- [56] A. Casals, A. E. Fallah-Seghrouchni, O. Negrioni, and A. Othmani, "Exposing agents as web services in JADE," in *Proc. Int. Workshop Eng. Multi-Agent Syst.* Cham, Switzerland: Springer, 2019, pp. 340–350.
- [57] A. Tolc and A. M. Uhrmacher, "Agents: Agenthood, agent architectures, and agent taxonomies," in *Agent-Directed Simulation and Systems Engineering*. Weinheim, Germany: Wiley-VCH Verlag GmbH & Co. KGaA, 2009, pp. 75–109.
- [58] M. Paletta, "A scouting-based multi-agent system model to deal with service collaboration in cloud computing," in *Systems and Software Development, Modeling, and Analysis: New Perspectives and Methodologies*. Hershey, PA, USA: IGI Global, 2014, p. 282.
- [59] A. K. Chopra, S. H. Christie, and M. P. Singh, "Multiagent foundations for distributed systems: A vision," in *Proc. Int. Workshop Eng. Multi-Agent Syst.* Cham, Switzerland: Springer, 2021, pp. 62–71.
- [60] A. K. Chopra, "Interaction-oriented software engineering: Programming abstractions for autonomy and decentralization," *AI Commun.*, vol. 35, no. 4, pp. 381–391, Sep. 2022.
- [61] D. B. Lange, "Mobile objects and mobile agents: The future of distributed computing?" in *Proc. Eur. Conf. Object-Oriented Program.* Berlin, Germany: Springer, 1998, pp. 1–12.
- [62] H. F. Ahmad, "Multi-agent systems: Overview of a new paradigm for distributed systems," in *Proc. 7th IEEE Int. Symp. High Assurance Syst. Eng.*, Oct. 2002, pp. 101–107.
- [63] V. Mascardi, D. Weyns, and A. Ricci, "Engineering multi-agent systems: State of affairs and the road ahead," *ACM SIGSOFT Softw. Eng. Notes*, vol. 44, no. 1, pp. 18–28, Mar. 2019.
- [64] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia methodology for agent-oriented analysis and design," *Auto. Agents Multi-Agent Syst.*, vol. 3, no. 3, pp. 285–312, 2000.
- [65] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An agent-oriented software development methodology," *Auto. Agents Multi-Agent Syst.*, vol. 8, no. 3, pp. 203–236, May 2004.
- [66] L. Padgham and M. Winikoff, "Prometheus: A methodology for developing intelligent agents," in *Proc. Int. Workshop Agent-Oriented Softw. Eng.* Berlin, Germany: Springer, 2002, pp. 174–185.
- [67] I. Zinnikus, C. Hahn, and K. Fischer, "A model-driven, agent-based approach for the integration of services into a collaborative business process," in *Proc. 7th Int. Joint Conf. Auto. Agents Multiagent Syst.*, 2008, pp. 241–248.
- [68] G. Kardas, A. Goknil, O. Dikenelli, and N. Y. Topaloglu, "Model driven development of semantic web enabled multi-agent systems," *Int. J. Cooperat. Inf. Syst.*, vol. 18, no. 2, pp. 261–308, Jun. 2009.
- [69] C. Hahn, S. Jacobi, and D. Raber, "Enhancing the interoperability between multiagent systems and service-oriented architectures through a model-driven approach," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Intell. Agent Technol.*, vol. 2, Aug. 2010, pp. 415–422.
- [70] G. Cabri, L. Leonardi, and M. Puviani, "Service-oriented agent methodologies," in *Proc. 16th IEEE Int. Workshops Enabling Technol., Infrastructure Collaborative Enterprises (WETICE)*, Jun. 2007, pp. 24–29.
- [71] A. Ciorcea, O. Boissier, A. Zimmermann, and A. M. Florea, "Give agents some REST: A resource-oriented abstraction layer for internet-scale agent environments," in *Proc. 16th Conf. Auto. Agents MultiAgent Syst.*, May 2017, pp. 1502–1504.
- [72] A. Ricci, A. Ciorcea, S. Mayer, O. Boissier, R. H. Bordini, and J. F. Hübner, "Engineering scalable distributed environments and organizations for MAS," in *Proc. 18th Int. Conf. Auto. Agents MultiAgent Syst.*, May 2019, pp. 790–798.
- [73] S. Willmott, J. L. Dale, B. Burg, P. Charlton, and P. O. Brien, "Agentcities: A worldwide open agent network," *Agentlink News*, vol. 8, pp. 13–15, Jan. 2001.
- [74] R. D. O'Reilly, "A distributed architecture for the monitoring and analysis of time series data," Ph.D. thesis, Univ. College Cork, Cork, Ireland, Tech. Rep., 2015.
- [75] A. Ciorcea, O. Boissier, and A. Ricci, "Beyond physical mashups: Autonomous systems for the web of things," in *Proc. 8th Int. Workshop Web Things*, Oct. 2017, pp. 16–20.
- [76] A. Casals, A. E. Fallah-Seghrouchni, A. A. F. Brandão, C. E. Pantoja, and J. Viterbo, "Resource-dependent contextual planning in AmI," *Proc. Comput. Sci.*, vol. 151, pp. 485–492, Jan. 2019.

- [77] F. Zambonelli, N. R. Jennings, and M. Wooldridge, "Developing multiagent systems: The Gaia methodology," *ACM Trans. Softw. Eng. Methodol.*, vol. 12, no. 3, pp. 317–370, Jul. 2003.
- [78] S. A. DeLoach, "The MaSE methodology," in *Methodologies and Software Engineering for Agent Systems*. Berlin, Germany: Springer, 2004, pp. 107–125.
- [79] S. A. DeLoach and J. C. Garcia-Ojeda, "The O-MASE methodology," in *Handbook on Agent-Oriented Design Processes*. Berlin, Germany: Springer, 2014, pp. 253–285.
- [80] R. Fabian, *Data-Oriented Design*. Stockport, U.K.: Self-Published, 2020.
- [81] B. Meyer, *Object-Oriented Software Construction*, vol. 2. Englewood Cliffs, NJ, USA: Prentice-Hall, 1997.
- [82] T. Mironov, L. Motaylenko, D. Andreev, I. Antonov, and M. Aristov, "Comparison of object-oriented programming and data-oriented design for implementing trading strategies backtester," in *Proc. Int. Sci. Practical Conf.*, vol. 2, 2021, pp. 124–130.
- [83] K. Fedoseev, N. Askarbekuly, E. Uzbekova, and M. Mazzara, "Application of data-oriented design in game development," *J. Phys. Conf. Ser.*, vol. 1694, 2020, doi: 10.1088/1742-6596/1694/1/012035.
- [84] R. Nystrom, *Game Programming Patterns*. Seattle, WA, USA: Genever Benning, 2014.
- [85] J. Gregory, *Game Engine Architecture*, 3rd ed., Boca Raton, FL, USA: CRC Press, 2017.
- [86] I. Crnkovic and M. P. H. Larsson, *Building Reliable Component-Based Software Systems*. Norwood, MA, USA: Artech House, 2002.
- [87] D. Wiebusch and M. E. Latoschik, "Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems," in *Proc. IEEE 8th Workshop Softw. Eng. Architectures Realtime Interact. Syst. (SEARIS)*, Mar. 2015, pp. 25–32.
- [88] D. Baron, *Hands-On Game Development Patterns With Unity*. Birmingham, U.K.: Packt Publishing Ltd, 2019.
- [89] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
- [90] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.
- [91] A. N. Kumar, R. K. Raj, S. G. Aly, M. D. Anderson, B. A. Becker, R. L. Blumenthal, E. Eaton, S. L. Epstein, M. Goldweber, P. Jalote, D. Lea, M. Oudshoorn, M. Pias, S. Reiser, C. Servin, R. Simha, T. Winters, and Q. Xiang, "Computer science curricula 2023," Assoc. Comput. Machinery, New York, NY, USA, Tech. Rep., 2024.
- [92] A. Shoulson, F. Moreno, M. Jones, R. Mead, and N. I. Badler, "Parameterizing behavior trees," in *Proc. Int. Conf. Motion Games*. Berlin, Germany: Springer, 2011, pp. 144–155.
- [93] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth, "The essence of P2P: A reference architecture for overlay networks," in *Proc. 5th IEEE Int. Conf. Peer-Peer Comput.*, Aug. 2005, pp. 11–20.
- [94] J. Thönes, "Microservices," *IEEE Softw.*, vol. 32, no. 1, p. 116, Jan./Feb. 2015.



**ARTHUR CASALS** (Member, IEEE) received the B.S. degree in computer engineering from the Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, Brazil, in 2004, and the Ph.D. degree in informatique, télécommunications et électronique from Sorbonne Université, Paris, France, in 2022. He is currently pursuing the Ph.D. degree in computer engineering with the Escola Politécnica, USP, São Paulo, Brazil. From 2005 to 2018, he assumed different roles in the industry, all related to technology development and its applications. His research interests include multi-agent systems, distributed systems, and software engineering.



**ANAROSA A. F. BRANDÃO** received the bachelor's and master's degrees in mathematics from the University of São Paulo, in 1990 and 1994, respectively, and the Ph.D. degree in computer science from the Pontifical Catholic University of Rio de Janeiro, in 2005. She received the "Livro Docente" title from the University of São Paulo, in 2021. Since December 2008, she has been a Professor with the Department of Computer Engineering and Digital Systems, Escola Politécnica, Universidade de São Paulo. In 2008, she became the mother of Leonardo. Currently, she is the President of the Inclusion and Belonging Committee, Escola Politécnica, from 2024 to 2026. From October 2022 to February 2024, she held the position of Coordinator of the Computer Engineering Concentration Area within the Graduate Program in Electrical Engineering at the Escola Politécnica, USP. Her research interests include computer science, with a focus on artificial intelligence and informatics in education. Her work primarily centers around agent and multi-agent systems and accessible web-based learning systems. She has been representing the Escola Politécnica at the Brazilian Computing Society (SBC), since March 2021. She also contributes as a member of the Advisory Committee for the WESAAC series of workshops—Workshop on School of Agent Systems, their Environments, and Applications. She was the Editor-in-Chief of *Brazilian Journal of Informatics in Education* (RBIE) (CEIE-SBC, from 2022 to 2024).

• • •