An Analysis of Multi Agent Systems Agent Based Programming

Ali-Khusein
School of Computing, First Moscow State University, Russia.
alikhusein60@gmail.com

Correspondence should be addressed to Ali-Khusein : alikhusein60@gmail.com

Article Info
Journal of Computing and Natural Science (http://anapub.co.ke/journals/jcns/jcns.html)
Doi: https://doi.org/10.53759/181X/JCNS202303017
Received 23 October 2022; Revised from 10 January 2023; Accepted 15 April 2023.
Available online 05 October 2023.

Abstract – The effectiveness of agent-based modeling as a simulation modeling methodology has resulted in its application in diverse settings, including the resolution of pragmatic business challenges, in recent times. The domain of symbolic artificial intelligence, which investigates intelligent and self-governing entities, is preoccupied with the mechanisms by which these entities arrive at determinations regarding their conduct in reaction to, or in expectation of, stimuli from the external environment. The scope of the methods employed encompasses a diverse array of techniques, spanning from negotiations to agent simulations, as well as multi-agent argumentation and planning. The present article scrutinizes the utilization of agent-based computing in multi-agent systems and provides an all-encompassing analysis of the relevant literature. This study delves into the examination of both traditional and contemporary agent programming languages, including their respective extensions, comparative analyses, and instances of their application in published literature.

Keywords – Agent-Based Model, Multi-Agent System, Procedural Reasoning System, Agent-Based Simulation, Object-Oriented Programming.

I. INTRODUCTION

The utilization of multi-agent systems can provide a viable solution to problems that would otherwise be deemed insurmountable for a solitary agent or a cohesive system. Intelligence can be approached algorithmically through techniques like search or reinforcement learning, or through more systematic, functional, or procedural methods. While there may be certain similarities between a Multi-Agent System (MAS) and Agent-Based Model (ABM), it is important to note that the two are not entirely synonymous. The primary objective of an ABM is not to address tangible pragmatic or technical challenges, but rather to acquire explanatory comprehension of the collective conduct of agents, who are not necessarily endowed with cognitive abilities, by adhering to fundamental principles, as frequently observed in natural systems. The terminology "agent-based model" is predominantly employed within the scientific community, whereas "multi-agent system" is more frequently utilized in the engineering and technological domains. The MAS in **Fig 1** has the potential to offer an effective approach in various domains such as online trade, disaster response, target monitoring, and social structure modelling.

Multi-Agent Systems (MASs) constitute a firmly established subfield within the domain of Artificial Intelligence (AI). Despite being a comparatively nascent field in contrast to more traditional research domains, Multi-Agent Systems (MASs) boast a substantial historical background. The agent technology, in 1995, was acknowledged as a swiftly evolving study domain and among the most rapidly expanding information technology domains. This assertion remains valid in contemporary times, as there exists a plethora of scholarly articles, instruments, and symposiums dedicated to promoting the progression of research in this field. However, the utilization of MASs is not as extensive as their potential warrants. With regards to the programming of MAS agents, as stated by Baldoni, Baroglio, Micalizio, and Tedeschi [1], the primary factor is the lack of motivation for key developers to transition to contemporary Agent-based Programming Languages (APL). This is due to the fact that the behaviors that can be conveniently programmed are adequately uncomplicated to be executed in conventional languages with minimal additional coding time. The proliferation of disorganized options presents a challenge for non-expert users in selecting APL and tools.

A smart agent could be defined as a computerized entity, which possesses the ability to make decision in a cognitive and rational manner, execute autonomous decisions, engage in social collaboration with other agents as needed, exhibit reactive behavior by perceiving and responding appropriately to its operating context, and ultimately take proactive action to attain its key objectives. An agent-oriented system (agent-based system) is a type of system in which the primary entities are agents that are regarded as first-class abstractions. From a computational standpoint, analogous reasoning can be applied. Through the use of a comparative analogy, it can be posited that agents bear a similar relationship to AOP (agent-oriented programming) languages as an object does to OOP (object-oriented programming) languages. In the context of APL, an agent serves as fundamental units, and the creation of programs involves the specification of their behaviors (i.e., the reasoning processes of an agent), goals (i.e., the objectives an agent seeks to attain), and interoperation (i.e., the manner in which agents cooperate to accomplish a task).

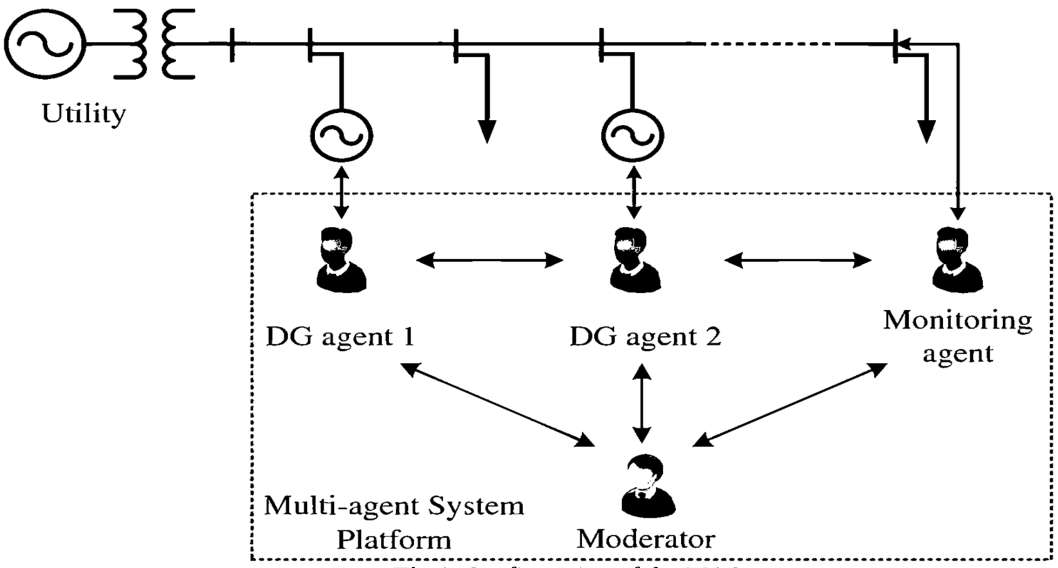

Fig 1. Configuration of the MAS

Agents are appropriate for scenarios that involve distributed or concurrent processing, as well as applications that require communication among multiple components. The utilization of agent technology could potentially prove advantageous in programs designed to analyze and make decisions predicated on data obtained from a network. Agents are deemed highly compatible with automation applications due to their ability to sustain their state of process and their surrounding environment. Furthermore, autonomous agents possess the capability to execute their assigned tasks without external intervention, rendering them advantageous for a diverse range of applications such as robotics, workflow administration, and plant/process automation. The utilization of the reasoning cycle in agents facilitates the explication of decisions made by the agent, thereby constituting an additional advantage of agent-oriented programming.

The objective of this paper is to furnish a thorough and all-encompassing account of the recent advancements in APL. This is aimed at enabling readers with diverse levels of acquaintance with the domain to gain a better comprehension of the present state of the field and its prospective future prospects. Our focus lies on the recent APL, including the frameworks and platforms employed in the advancement of Multi-Agent Systems (MASs). In addition to incorporating both theoretical and practical investigations, our discourse also briefly encompasses novel extensions, extant comparisons, and implementations of agent-based programming.

Unlike prior surveys and assessments of APLs, our study considers not only the introduction of new APLs but also the extension and juxtaposition of present ones. Furthermore, we examine the theoretical and practical aspects of APLs, thereby illuminating the disparity between them. In contrast to other reviews that have focused solely on specific aspects of APLs such as engineering, platforms, literature on agent-based simulations, or a single agency model, namely the Belief-Desire-Intention (BDI) model, our review aims to encompass the entire class of APLs.

The remainder of the paper has been organized as follows: Section II presents a brief overview of agent-based programming. Section III focuses on providing the methodology employed for this article. In Section IV, a discussion of agent-based programming for MAS is presented, in which key concepts are discussed: agent programming languages, agent programming languages comparison, and agent-based applications. Section V provides a brief summary of the key discussions in the article. Section VI draws final remarks to the article as well as future research directions.

II. BRIEF OVERVIEW OF AGENT-BASED PROGRAMMING

Agent-oriented programming, which is a subset of Object-oriented programming, was first introduced in 1993 [2]. The notion of an agent's cognitive state is presented and explicated, encompassing the agent's cognitive faculties, decision-making processes, and competencies. The present study expounds on the speech act theory, which is widely employed to delineate agent communications in distinct modern APL. Additionally, the study elucidates the applicability of the speech act theory to agent models in the AGENT-) interpreters that are applied in the LPL (lisp programming language). Over the past few decades, different cognitive and reasoning frameworks have been established in the field of agent-based programming. Subsequently, an examination is conducted on the Procedural Reasoning System (PRS), the Behavior Driven Inference (BDI), and the Situation Calculus. These frameworks have been pivotal in the evolution of different APL in the past and persist in their significance in contemporary times.

The Procedural Reasoning System (PRS) [3] is a system that is able to conduct reasoning concerning the processes that are types of knowledge that are procedural in nature. The system is implemented in Lisp. Upon acquiring knowledge of these techniques, a system agent can deliberately select the intentions to pursue for the purpose of achieving their goals. In contrast to conventional programming languages, the aforementioned procedures are solely executed when they are deemed necessary to achieve a specific objective or react to a particular occurrence. Partial hierarchical planning diverges from conventional AI planners by engaging with the environment as a component of its reasoning process, as opposed to devising a plan for a fixed environment.

AgentSpeak(L), which is the language reviewed by d'Inverno and Luck [4], is widely recognized as one of the most prominent languages that utilizes the BDI architecture. AgentSpeak(L), an agent-oriented programming language based on abstract logic, was initially introduced by Najjar et al. [5]. Subsequent works by Bordini, Hubner, and other scholars further developed and formalized the language. The PRS (as depicted in **Fig 2**) provides the agents in AgentSpeak(L) with the capability to utilize a pre-existing plan library. PRS plans comprise the subsequent components: For a plan to achieve its intended purpose, it must consist of three essential components: (i) a goal, which denotes the desired outcome of the plan; (ii) a context, which outlines the necessary environmental conditions for the plan to succeed; and (iii) a body, which encompasses the procedural aspect of the plan and may comprise a series of actions and sub-goals.

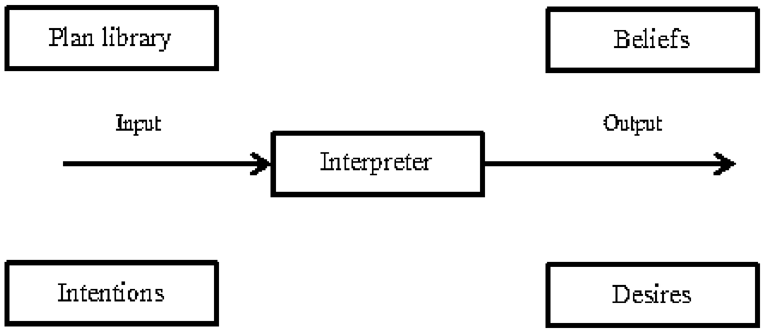

Fig 2. An illustration of PRS

The Belief-Desire-Intention (BDI) model is a cognitive framework that aids decision making by letting you choose the best action to take to accomplish your goals. A person's intentions, desires, and beliefs are the three major mental states that define their agency. What an agent thinks it knows about its surroundings, other agents, and itself is its set of beliefs. Desire is the condition or result that the agent hopes to achieve. In contrast, what an agent intends to do, or intends to do, is a set of actions that will lead to the intended outcome. Cognitive, volitional, and reflective states of mind are the relevant mental dispositions. Input information from the environment, including sensors, is received by a belief revision function, as shown in **Fig 3**.

The purpose of the event is to refresh the foundation of trust. Based on the belief and intention bases, this revision may provide new possibilities that satisfy current needs. A filter updates the intents basis by taking into account both the current belief base and the current desires base. In the end, the agent decides on a specific purpose to carry out. Numerous agent programming languages have made heavy use of the BDI paradigm because of its status as the de facto standard framework for agency. AgentSpeak(L) represents a language, which can understand agent programming as horn-clause logic programs and acts as an abstraction of applied BDI models. Many more APLs have been built on top of the theoretical framework that supports this language.

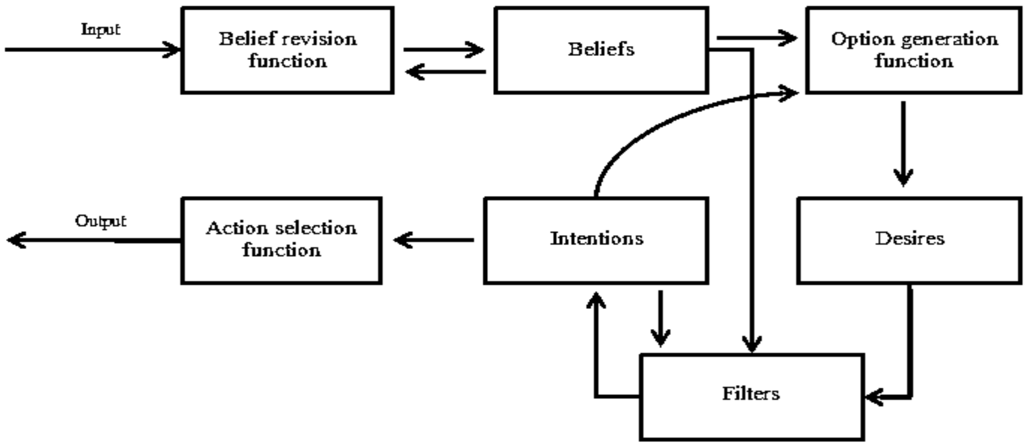

Fig 3. The BDI Models

A first-order language, known as a Situation Calculus, can be utilized to depict alterations in a dynamic ecosystem. A circumstance can be defined as a primary linguistic unit that delineates an action sequence. The first phase can be identified as a state where no events or occurrences have taken place. Similar to state transition systems, the execution of the function do (a, s) generates a scenario in which the state (s) has the potential to become the successor to a prior state. The utilization of Situation Calculus has been employed to depict the progression of the world in response to actions taken within it, due to the notable importance of dynamic environments in the realm of agent-based programming.

The aforementioned models hold significant importance in the previous agent-oriented programming. However, it is worth noting that there exist several other models that have also impacted these languages, as elaborated in Section IV. Certain agent languages integrate elements from different programming models, such as imperative, procedural, functional, actor-oriented, concurrent, and object-oriented, among others, to establish a novel, hybrid paradigm. Agent languages, which incorporate elements from different paradigms, are also considered, however, this analysis will not involve a comparative or in-depth examination of such languages.

The utilization of agent-oriented programming has been observed in diverse real-world scenarios, encompassing but not restricted to: the decentralized administration of electric power systems; the control of room assignment; the management of automatic machine-to-machine application (such as traffic rerouting); and the identification of privacy breaches. Section IV, (D) presents a discussion on the current endeavors to incorporate APLs in practical application programming.

III. METHODOLOGY

A thorough examination of scholarly literature pertaining to agent programming languages over a period of five years (2015-2020) was undertaken. The search conducted using Google Scholar yielded a total of 400 outcomes, accounting for duplicate entries. The examination was limited to the first 10 pages, encompassing 100 items, for each of the specified phrases. Our search strategy involved querying databases and search engines using specific keywords related to Agent-based model, multi-agent system, procedural reasoning system, agent-based simulation, object-oriented programming. Following the removal of duplicate entries, a total of 250 documents remained. In addition to the aforementioned entries, 16 additional sources were incorporated into the dataset. These sources were primarily derived from dated references that may not have been readily accessible through search engines, but were nonetheless cited in various scholarly papers and sources. Half of the external submissions, specifically eight, were deemed noteworthy applications with advanced stages of development prior to 2015 and subsequent revisions spanning from 2017 to 2020.

IV. AGENT-BASED PROGRAMMING FOR MAS

This section pertains to a comprehensive analysis of all the studies that were incorporated in our literature review. The present discourse commences with an exposition on agent programming and its associated languages, followed by an examination of contemporary evaluations in the realm of agent-oriented programming, and culminates with a succinct survey of pertinent applications.

Agent Programming Languages

Within the framework of Agent-Oriented Programming (AOP), software programs are regarded as agents. Agents exhibit characteristics similar to those of objects in Object-Oriented Programming (OOP) in that they possess a cognitive state and react to external stimuli by executing certain actions or modifying their internal state. It is commonly assumed that specific entities are classified as intelligent agents, as they engage in interactions with each other to attain their goals and exhibit social conduct. Programming languages that incorporate agent-oriented programming (AOP) are commonly referred to as "agent programming languages." Agent programming languages are utilized by various systems such as Agent-0, 3APL, 2APL, Jason, JACK, and GOAL. The vocabularies under consideration encompass significant notions such as belief, desire, and intention (BDI). These concepts serve as proxies for the agent's currently held beliefs, desired outcomes, and present actions, respectively. The formalizations in modal logics offer the syntax and semantics for a BDI model. Consequently, the theoretical definition and verification of agent programs can be accomplished with the aid of logic.

The BDI paradigm has been utilized in agent-based simulation (ABS) as well. ABS aims to enhance comprehension of how global features can emerge from a system of local interacting processes, as opposed to multi-agent systems. Mason, Repast, and GAMA are exemplars of Agent-Based Simulation (ABS) platforms. While certain ABS platforms do provide a structure for constructing models utilizing the BDI paradigm, it is not typical for ABS systems to employ the agent programming languages mentioned above. The utilization of a BDI model by agents allows for the manifestation of complex behavior beyond that of basic reactive models, without the added computational demands of cognitive architectures. BDI models are frequently preferred by domain experts as they offer a simpler means for them to express their expertise and also facilitate the explanation of behavior. Othman et al. [6] present a thorough investigation and evaluation of techniques for integrating BDI models into ABS. The authors underscore the benefits of BDI models in the implementation of descriptive agents, which possess more intricate models than reactive agents. These advantages are reiterated from prior discussions.

As mentioned to in Section II, it is evident that the BDI model of agency is the prevailing approach, having been employed about 7 in 15 languages. Most of programing languages, which have been developed and executed, have been applied using Java, probably due to the potential advantages offered by Java's cross-platform functionality facilitated by its Java Virtual Machine

General-Purpose APLs

Raees, Khan, Mustafa Abbasi, Ahmed, Fazilat, and Ahmed [7] present a comprehensive analysis of agent-based programming from the perspective software engineering. The poll results indicated that a significant challenge was encountered in facilitating comprehension of AOP among APL programmers, given its fundamentally distinct nature from the cognitive processes employed by most programmers. Collier, Russell, and Lillis [8] conducted an investigation into the correlation between AgentSpeak(L) and OOP in order to mitigate the cognitive disparity and establish a connection. This study proposes the introduction of a novel agent programming language, namely ASTRA. The Chromar notation was first presented by [9]. It is a rule-based system that employs stochastic semantics to generate a Markov chain that functions in a continuous time framework. Chromar is endowed with enhanced expressive capacity owing to its utilization of Haskell as the underlying framework, thereby rendering it highly compatible with the abundance of type definitions. Chromar employs first-order abstractions, whereby rules are capable of elucidating the conduct of a solitary agent as well as the synchronized conduct of multiple actors.

The cognitive agents in GOAL, a declarative agent programming language, are directed in their actions through the utilization of knowledge base beliefs and objectives. Although both the GOAL language and the BDI model place significant emphasis on beliefs and desires/goals, the former is primarily oriented towards adhering to pre-established procedures. Notwithstanding the fact that Prolog is frequently utilized to articulate the agent's knowledge, including rules, the agent programs are scripted using the GOAL syntax. The BDI agent model constitutes the fundamental basis for Jason, a programming language that extends the capabilities of AgentSpeak(L). Upon the occurrence of specific events within Jason, the agents proceed to take action on their immediate environment by implementing pre-established plans from their respective plan libraries. One of Jason's notable features is its capability to integrate Prolog-like rules into the belief base of an agent.

The JaCaMo platform encompasses three distinct levels of abstraction, namely Moise, CArtAgO, and Jason, each signifying a unique technology. The programming language utilized for writing code at the agent level is Jason, while CArtAgO is accountable for managing the ecosystem, and Moise is responsible for managing the business. JaCaMo effectively integrates the three applications by establishing a semantic relationship between concepts at distinct abstraction levels, namely the environment, agent, and organization. Upon completion of the project, the JaCaMo Multi-Agent System (MAS) development platform was obtained. Consequently, the advancement of complex MAS can be facilitated. The platform provides exceptional and advanced assistance in the development of agents, environments, and organizations. With the aim of developing autonomous agent programs that can undergo independent verification, Gwendolen was earlier founded as a smaller Jason subset.

However, it has since progressed into a distinct syntax and semantic. The system's emphasis on facilitating agent verification from the foundation up has resulted in the exclusion of several advanced features of AgentSpeak(L) and BDI, although it does provide support for the fundamental aspects. Although there exists a substantial body of literature on the verification of agent programs and multi-agent systems (MAS), it is important to note that these topics fall beyond the purview of the present article. The utility of Gwendolen for constructing MASs is not solely contingent upon its verifiability.

The JADE open source platform can be utilized for constructing peer-to-peer agent-oriented applications. Aside from agent-based abstractions, the system incorporates a model for task execution and composition, facilitates asynchronous message passing for peer-to-peer agent communication, and includes yellow page service from the subscribe/publish discovery method. Several programming languages, such as Jason and JaCaMo, utilize the distribution architecture facilitated by JADE. This feature enables JADE-based systems to be transferred between computers that operate on different operating systems, thus promoting portability. JADEL (JADE Language) was first introduced by van Steen and Tanenbaum [10] as an extension of JADE. It enables agent construction and MAS in addition of JADE without requiring direct Java application. Therefore, Jadescript was introduced as an extension of JADE by Petrosino, Iotti, Monica, and Bergenti [11]. The expressive syntax of Jadescript is significantly influenced by modern scripting languages, with the aim of enhancing its readability and aligning agent scripts with the conventions of pseudocode.

Jadex enables the development of autonomous software agents that possess self-awareness by utilizing Java programming language and XML markup language. The BDI-based agent abstraction offers several advantages, including agent runtime architecture, various simulation support, interaction styles, automatic overlay system construction, and a comprehensive set of runtime tools. The article referenced as [12] presents an accessible Java-oriented platform, known as LightJason, specifically designed for BDI agent-based simulation and programming. The LightJason system is structured using a logic-oriented, which integrates different enhanced capabilities, such multi-rule and –plan definition, lambda-expressions, explicit repair actions, multi-variable assignments, parallel execution, and thread-safe variables. This language is based on AgentSpeak(L), which serves as the underlying logic language for the LightJason platform. The LightJason system has been developed from its foundational principles, with its design influenced by the AgentSpeak(L) and Jason methodologies.

In [13] present PLACE (Planning-based Language for Agents and Computational Environments), a language designed for Agent-Oriented Programming (AOP) that enables agents to utilize Artificial Intelligence (AI) planning. The planning process of PLACE is executed utilizing a Hierarchical Task Network (HTN) planner, while its syntactic configuration bears resemblance to that of BDI. In contrast to other AOP languages, PLACE's actions incorporate temporal estimates, thereby requiring a planner capable of managing temporal intricacies. In case of an unfavorable outcome, the agents situated in PLACE have the ability to correct the situation by modifying their actions accordingly. To address the potential disruption of a plan caused by unforeseen circumstances, a plan repair mechanism is employed. The proposal for PLASA, a programming language designed for synchronous agents, is introduced in [14]. PLASA exhibits platform independence, facilitating the deployment of collaborative applications across a diverse range of real-world robotic systems and dynamic environments. The PLASA paradigm is essentially a form of synchronous robot movement that involves a modification of the Wait-Look-Compute-Move framework, as initially presented in [15]. Due to its advanced level of complexity, this programming language has the potential to serve as a means for humans to specify the desired actions for robots to execute.

The Relational Model Multi-Agent System (RMAS) is an appropriate technique for incorporating control and reasoning into Cyber-Physical Systems (CPS) due to its database-centric nature. It is advisable to commence the implementation of RMAS by integrating the Matlab development ecosystem with the SQLite database language. The primary objective of SARL is to furnish a programming language that is highly extensible and encompasses fundamental key concepts that effectively facilitate Aspect-Oriented Programming (AOP). The purpose of the language is to offer abstract concepts for various functionalities such as concurrent processing, distributed data storage, interactive programming, decentralized control, responsiveness, autonomy, and dynamic reconfiguration. The approach employed does not rely on a specific model, but instead generates a Domain-Specific Language (DSL) that functions as its foundational framework.

ABMS, Robotics, and Others

Agent-based modeling and simulation (ABMS) is a novel technique to model and simulate complex systems that encompass self-replicating and autonomous agents along with their interactions. Agents demonstrate conduct that is typically defined by fundamental principles and participate in exchanges that influence the conduct of other agents with whom they engage. Through the process of individually modeling each agent, it becomes possible to observe the extent to which their unique characteristics and behaviors influence the collective behavior of the system. In certain models, it is often observed that systems exhibit self-organization through a "bottom-up" approach, wherein the system is constructed agent-by-agent and interaction-by-interaction. Emergent patterns, structures, and behaviors arise as a result of agent interactions, which were not initially intended. Agent-based simulation is a simulation approach that distinguishes itself from other methods such as discrete-event simulation and system dynamics by emphasizing the modeling of agent heterogeneity within a population and the emergence of self-organization. Agent-based modeling is a suitable approach for modeling social systems due to its ability to enable agents to acquire knowledge from their experiences, modify their conduct in reaction to fresh data, and adjust to their environment.

Agent-based modeling has the potential to provide significant benefits to a wide range of fields and areas of study. The modeling of agent behavior has numerous applications, such as the prediction of epidemic spread and bio-warfare threat, comprehension of consumer purchasing patterns, simulation of the adaptive immune system, analysis of the collapse of ancient societies, examination of the engagement of forces in military settings, and various other purposes. Certain agent-based models are designed to be concise yet advanced, omitting extraneous details about a system to facilitate comprehension of a social phenomenon or conduct. Conversely, other agent-based models are comprehensive in scope, utilizing copious amounts of data, undergoing validation, and yielding results intended to inform policymaking and decision-making. The increasing popularity of agent-based modelling applications can be attributed to several factors, including the enhanced availability of data at increasingly granular levels, advancements in computing capabilities, and the development of specialized tools tailored to this modelling approach.

The increasing prevalence of agent-based modeling tracks in conferences and workshops, as well as the growing number of peer-reviewed publications in academic journals across various disciplines and in modeling and simulation journals, along with the expanding job market for individuals with expertise in agent-based modeling and the attention from funding agencies, all indicate a burgeoning interest in this field. According to the Winter Simulation Conference's website in [16], a total of 27 papers presented at the most recent conference featured the term "agent" in either their titles or abstracts.

In order to construct an agent-based model, the initial step involves the identification of its constituent components, followed by their modeling and subsequent coding. The structure of a basic agent-based model is illustrated in **Fig. 4**. In order to operationalize the model, it is necessary to have a computational engine that is capable of simulating the behaviors and interactions of agents. The provision of this functionality is facilitated by a toolkit, programming language, or other implementation that enables the modeling of agents. The term "running" an agent-based model refers to the iterative execution of agents' activities and interactions. While frequently simulated using time-stepped, activity-based, or discrete-event simulation architectures, alternative modeling approaches can also be employed to represent this process.

As noted by Krzywicki and Wobcke [17], there exists a disparity between the methodologies employed in AOSE (Agent-Oriented Software Engineering) and the enhancement of ABMS (Agent-Based Modeling and Simulation). The authors present an Agent-Oriented Software Engineering (AOSE) approach in [18] to address the aforementioned issue. This approach is referred to as the PEABS (Process for Developing Efficient Agent-Based Simulators). The INGENIAS methodology was employed to model the system's requirements and design its architecture. The utilization of an adaptation framework facilitates the ability of creators of ABMS to conduct highly efficient simulations on extensive datasets. Chaouch and Tamali [19] present an alternative approach to constructing agent-based models by introducing a novel cognitive agent infrastructure, which is oriented on the BDI model and incorporated into the GAMA simulation language.

Singh, Padgham, and Logan [20] combine BDI with ABMS, which is characterized by its flexibility and user-friendliness, even for individuals without specialized expertise. Sánchez, Coma, Aguelo, and Cerezo [21] present a framework that facilitates the integration of BDI cognitive agents within an ABMS system. This study represents an additional effort to establish a linkage between BDI and ABMS. The objective of Caillou, Gaudou, Grignard, Truong, and Taillandier [22] is to encompass any model based on BDI infrastructure with ABMS, thereby rendering it more comprehensive than citation. A comprehensive comprehension of the activities, which every agent may undertake within a simulated ecosystem and the percepts, which refer to environmental observations and events that are pertinent to the agent, is all that is required.

ALLEGRO, which stands for ALGOL within PREGO represents a simulation formalism that employs belief infrastructure for stochastic areas. It presents itself as a viable option to GOLOG for high-degree management of robots. In [23], the YAGI (Yet another GOLOG Interpreter) is presented as a prototype of a programming language designed for the purpose of programming robots and agents, utilizing actions and the situation calculus as its foundation. YAGI has bindings for two of the primary robotics frameworks, namely Fawkes and Robot Operating System (ROS). The Cognitive Affective Agent Programming Framework (CAAF) is expounded upon by Kaptein, Broekens, Hindriks, and Neerincx [24]. This framework is rooted in the emotion’s belief-desire theory and allows cognitive agents to perform emotion computations. The semantics presented by the authors offer insights into the fundamental code structures of the agents. The utilization of these fundamental components enables a software developer to construct a cognitive agent within an agent-based software system, thereby facilitating the automated computation of emotions during simulation processes.

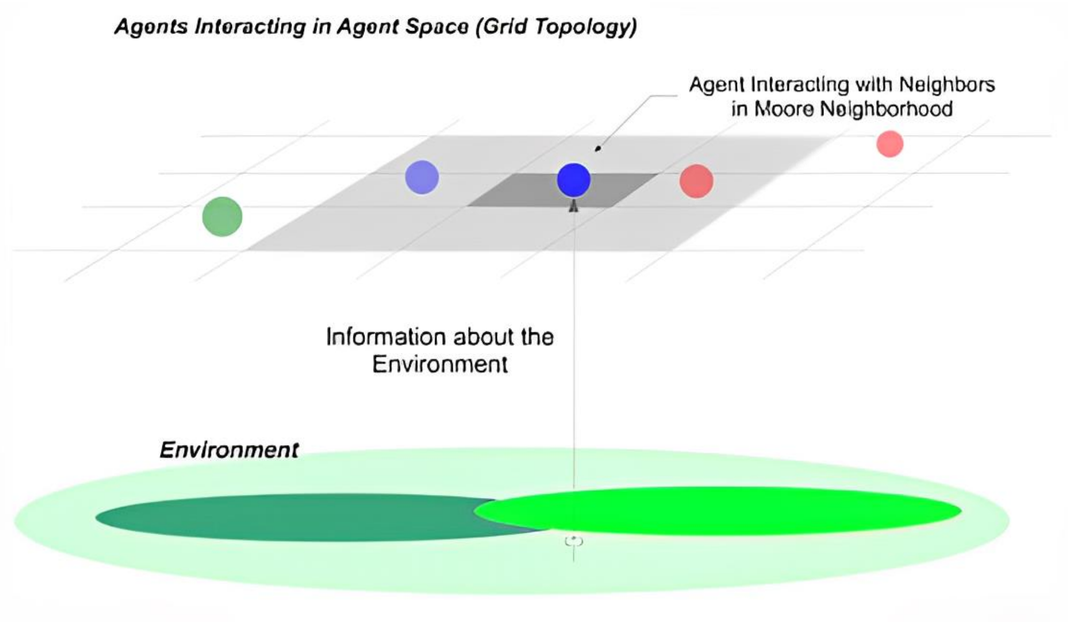
Agents Interacting in Agent Space (Grid Topology)
Agent Interacting with Neighbors in Moore Neighborhood
Information about the Environment
Environment

Fig 4. The Configuration of a Conventional Agent-Based Model

Agent Programming Languages Extensions

The previous section encompassed literature that pertains to newly developed agent programming languages (APLs) within the timeframe of 2015-2023, as well as works that showcase the most significant APLs of 2015 that are currently being upheld. In this section, we will examine the influential papers that have expanded the limits of APLs. The alterations made to established APLs are commonly referred to as "APL extensions." These modifications can range from basic feature additions to the development of a novel APL tailored to a distinct and unprecedented situation. The authors present a modified iteration of their MASCEM (Multi-Agent System for Competitive Electricity Market) in [25]. The developers aim to streamline the incorporation of supplementary models through the utilization of the enhanced and updated MASCEM simulator. The adaptability of MASCEM to the dynamic and challenging nature of energy markets is attributed to its structural implementation choices, which facilitate the accommodation of various instruments and methods. The latest expansion of MASCEM will employ ontologies specifically for the purpose of enhancing player interactions.

The extension of PEABS, known as TABSAOND, has been documented by García-Magariño, Palacios-Navarro, and Lacuesta [26]. The key differentiation between TABSAOND and PEABS lies in the fact that the former pertains to the strategic formulation and implementation of judgments in situations that lack a predetermined outcome. Furthermore, simulators are currently utilized as both software applications and online resources accessible through the internet. A conservative synchronization approach is made available for the SARL computer language and Janus runtime system, as documented in [27]. The utilization of the Janus system for agent-oriented programming that involves time is not feasible without the provision of a dedicated synchronization technique. This is due to the fact that Janus does not make any assumptions regarding the event sequence transmitted by an agent.

Anjum, Sun, Wang, and Orchard [28] propose novel programming constructs to enable the integration of EMIA, an advanced emotion model based on rules, with the 2APL agent language. Through a reconsideration of 2APL's syntax, semantics, and decision-making process, we have successfully integrated them. This amalgamation enables the agent to simulate responses to practical situations with a heightened level of realism in relation to the emotions it conveys. The ARGO system is a variant of the Jason framework that is employed for the development of robotic agents utilizing perception filters and Javino programming. In their publication, Leask and Logan [29] showcase the utilization of procedural reflections in the APL. This feature enables the encoding of both the agent's programs and its deliberation strategy within a single programming language, streamlining the implementation of specific stages in the BDI agent's deliberation cycle.

The recent extension has enabled agents developed using Jason and Gwendolen to engage with ROS, thereby facilitating the development of self-governing agents that can effectively manage and execute strategic decisions within robotic systems based on ROS. The rosbridge library enables the exchange of information between the environment and ROS nodes. The interface acts as an intermediary between the agent and ROS. The main differentiation of their approach from other endeavors aimed at enhancing traditional APLs to accommodate ROS is its self-sufficiency, as it does not require any additional modifications to either APLs or ROS. This feature renders their technique adaptable and transferable across various iterations of these systems and tools. Likewise, da Silva Medeiros, Julio, de Almeida, and Bastos [30] proposes a model that integrates Jason with ROS within an embedded system. It also introduces a novel structural design that enables direct communication between ROS and the agent.

The article [31] presents a proposed paradigm for a BDI agent-oriented programming model that utilizes reinforcement learning. Additionally, an application of this paradigm is discussed, which is based on the Jason APL. The technique facilitates the development of BDI agents that possess actions, which can be either acquired by the agent during the design and engineering phases or explicitly programmed.

Agent Programming Languages Comparison

The preceding two sections of literature demonstrate that the agent-based programming community offers various Agent Programming Languages (APLs) for developing Multi-Agent Systems (MAS). Regrettably, the evaluation of language proficiency is occasionally lacking or inadequate. Previous research has been conducted to compare agent-oriented languages with other paradigms, specifically actor-based languages. The study conducted by the authors provides evidence that agent languages, such as Jason, possess comparable capabilities to lightweight languages like actors.

Ho and Nakadai [32] have presented a set of criteria that can be utilized to evaluate specialized modeling languages designed for Multi-Agent Systems (MASs), encompassing both existing and prospective languages. The assessment primarily centers on language and its associated tools, and consequently presents both qualitative and quantitative findings. The authors establish a correlation between the pseudocodes of a prevalent technique utilized for addressing disseminated constraint satisfaction issues and a comparable strategy executed in JADEL. The study presented by van Haeringen, Gerritsen, and Hindriks investigates and compares parallel architectures designed to facilitate the implementation of multi-agent simulations on high-performance computing infrastructures, such as parallel clusters. Sun et al. [34] conduct a systematic evaluation of ABMS methods, wherein they differentiate between the concepts of model behavior complexity and model structure complexity. They also elucidate the nonlinear relationship between these two concepts. Subsequently, a comparative analysis is conducted to evaluate the expenses and advantages associated with the utilization of intricate models, which are predominantly grounded on empirical evidence, as opposed to rudimentary models that are primarily based on theoretical frameworks.

Agent-Based Applications

This part of the article enumerates a number of most recent applications that employ agent-oriented programming. The primary objective of this review is to demonstrate the diverse range of application areas where agents could prove to be advantageous. It is important to note that the list presented is not comprehensive and does not serve as the principal focus of our study. The MAPC (Multi-Agent Programming Contest) is a globally recognized yearly competition, which has been taking place since 2005. It is designed to test the skills of participants in the field of multi-agent programming. Further information about the contest can be found in [35]. The objective of this initiative is to incite research in the domain of multi-agent programming through the presentation of intricate benchmark scenarios that necessitate synchronized action. These scenarios can be employed to evaluate and contrast multi-APL, tools, and platforms. In recent years, various agent-based platforms and programming languages, including JaCaMo, Jason, and GOAL, have been utilized for implementation purposes.

Kerr et al. [36] have proposed the utilization of agent-based models for the purpose of simulating and assessing the transmission of COVID-19. A research domain exclusively dedicated to the utilization of agent-based technologies within the energy sector exists. The utilization of MAS technologies for Microgrid control, optimization, and market distribution is demonstrated in [37]. Additional resources are available for interested readers, including a comprehensive survey on the utilization of Multi-Agent Systems (MAS) in the management and regulation of Microgrids, as well as a thorough examination of the current advancements in the application of MAS to address energy optimization challenges. The article by Christensen and Salmon [38] showcases an implementation of Agent-Based Modeling and Simulation (ABMS) to investigate the correlation between alterations and human activities in land-cover/land-use. The objective of this reference is to provide a scientific basis for informed decision-making in the realm of land planning and utilization. The Repast modelling platform serves as the basis for the implementation of the model. Zolfagharipoor and Ahmadi [39] introduce FlowLogo, a modeling environment that is interactive in nature and is utilized for the development of coupled agent-based groundwater models. The authors also provide illustrations to demonstrate the workings of FlowLogo. FlowLogo is a software application that has been developed using NetLogo. It is notable for being the initial integrated software that provides a simple approach to depicting the actions of agents that change in response to groundwater conditions. A comprehensive examination of agent-based modeling and simulation (ABMS) tools and applications is available in [40]. In [41], a comprehensive methodological guide is presented for utilizing BDI agents within a social simulation. In addition, a review of the present tools and methodologies for implementing BDI agents in such simulations is provided.

Agents possess the potential to construct self-driven IoT models owing to their context-awareness, self-adaptation, and distributed nature. More information on the utilization of microservices as agents in IoT can be found. Due to their inherent complexity, dynamism, situatedness, and autonomy, various types of systems, whether natural or artificial, are most effectively analyzed through an agent-oriented method. The Internet of Things (IoT) and Multi-Agent Systems (MAS) exhibit a profound conceptual interrelation with agents and Self-Organizing (SO) systems. Given the extensive array of concerns, ABC has been employed as a means of modeling, designing, and simulating Internet of Things (IoT) applications and systems, with the aim of systematically expediting their development.

The utilization of agent-based modeling enables the capture of essential characteristics of Service Oriented (SO) and Internet of Things (IoT) systems, in a manner that is independent of specific technologies and with varying levels of detail. The agent model implicitly integrates the autonomy, proactivity, and situativity of self-organizing systems (SOs), while additional significant properties of SOs may be clearly-illustrated using agent-oriented concepts. The article by Berger, Häckel, and Häfner [42] presents a framework in which the functions of self-organizing (SO) systems are described in terms of objectives, the working strategy of SO systems is described on the basis of behaviors and the elements that contribute to the augmentation of SO systems (such as actuators, sensors, knowledge bases) are defined in terms of bindable agent systems. Nevertheless, these investigations employ diverse techniques to distinguish SOs/agents from one another. As demonstrated in [43], every agent or service object (SO) is assigned a specific role derived from a scenario-specific repository, such as a smart driver-support, smart car, and smart road networks in the transport sector. This key role regulates the default communications paradigms, goals, and behaviours of the agent or SO. Additionally, Amirat, Hock-Koon, and Oussalah [44] illustrate that the plans and goals of the SO/agent are represented through templates that reflect their respective functionalities. The other contributions do not refer to pre-established categories or structures.

In the study referenced as [45], individual agents or SOs are directed by a self-model represented as an automaton. This model reacts to various inputs, which are represented as messages, originating from other agents and the surrounding environment. The article by Ferguson and Bargh [46] encodes the motivations for actions and responses of SOs in their behaviors, which are influenced by both internal and external events as well as design objectives encapsulated in state-based tasks. Savaglio, Ganzha, Paprzycki, Bădică, Ivanović, and Fortino [47] propose a method for dynamically defining the self-state of a SO/agent by integrating real-time sensor data, location, and computing unit status. The agent-oriented service-oriented (SO) models that have been evaluated in this study are adept at facilitating the initial phases of development analysis due to their capacity to extract the crucial elements of SOs from the intricacies of low-level specifics or particular implementation constraints. It is necessary to conduct research in order to accurately depict the interactions between cyberphysical agents and SOs within everyday physical contexts, in addition to providing appropriate descriptions of agent-based SOs on their own. Agent-based models in the IoT context serve as a useful foundation for subsequent agent-oriented simulations and programming.

V. DISCUSSION

Agent-based modeling (ABM) is a computational modeling approach that represents a system as a collection of simulated individuals, known as agents, each endowed with decision-making capabilities. Each agent assesses its environment autonomously and behaves based on its own predetermined set of regulations. Agents have the capacity to perform diverse actions, including but not limited to generating, utilizing, or vending, as per the requirements of the system they are representing. The utilization of computational power in agent-based modeling enables the exploration of dynamics that are beyond the scope of pure mathematical methods. This modeling approach is distinguished by the repeated occurrence of competitive interactions among agents. Fundamentally, an agent-based model comprises a group of agents and their interconnections. Agent-based models, despite their simplicity, have the potential to exhibit intricate behavioral patterns that can offer valuable insights into the underlying dynamics of the systems they depict. It is plausible that agents could undergo evolutionary changes, thereby resulting in the emergence of behaviors that were not previously anticipated. Sophisticated ABM regularly integrates evolutionary algorithms, neural networks, or other learning methodologies to accomplish realistic learning and adaptability simulations.

The concept of ABM is primarily characterized by a mindset rather than a mere instrumentality. The ABM methodology entails the deconstruction of intricate systems into their constituent elements. It has been observed that certain scientists have erroneously inferred that differential equation modeling can serve as a viable substitute for agent-based modeling. However, it should be noted that the latter is essentially a compilation of differential equations, each of which characterizes the behavior of a specific constituent element within the system. The utilization of Agent-Based Modeling (ABM) is analogous to microscopic modeling, while macroscopic modeling serves as a substitute. The ABM methodology is currently experiencing increased adoption, thus prompting an opportune moment to reassess its worth and practical implementation. The present research aims to address these concerns by initially providing a synopsis and classification of the benefits of Agent-Based Modeling (ABM) and subsequently elucidating these benefits through a series of empirical investigations. Upon completion of the reading material, the audience will possess a comprehensive comprehension regarding the appropriate timing and rationale behind the implementation of Account-Based Marketing (ABM). ABM's success can be attributed to its ease of implementation. Once an individual is acquainted with the concept of agent-based modeling, writing code for it becomes effortless. The apparent ease of the approach may engender the erroneous inference that the fundamental tenets are equally uncomplicated to acquire. Notwithstanding its technological simplicity, the conceptual complexity of ABM is profound. Due to this unique combination, the employment of ABM is frequently incorrect.

The study has demonstrated that there is a diverse array of methodologies for agent-oriented programming, encompassing both established techniques such as BDI, as well as innovative approaches like simulation or planning-based methods. Several endeavors have been made to integrate the principles of agent-oriented programming with those of other models, particularly object-oriented programming, across various programming languages. The significant obstacle in broadening the society of programmers engaged in agent-oriented programming is the limited acquaintance with its concepts, which are distinct from other widely adopted paradigms. Incorporating certain concepts from alternative paradigms into agent-based programming languages has the potential to enhance their accessibility to individuals who may harbor apprehensions towards programming. Notwithstanding the utility of hybrid languages in certain contexts, "pure" approaches remain necessary for comprehensive treatment of all abstractions involved in agent programming, including but not limited to agents, environments, organizations, and interactions.

Despite the existence of various qualitative comparisons such as ideas and characteristics in the literature, providing an equitable evaluation of the attributes found in these languages is difficult owing to the utilization of different agency models. Further research is required to ascertain the fundamental components of agent-oriented programming and their correlation with the diverse agency models currently employed in prevalent programming languages. Due to the fact that the majority of agent programming languages are developed within academic settings, it can be challenging to conduct quantitative evaluations, such as performance comparisons, due to the iterative nature of their development cycles. It is imperative to establish standards that are specific to agents, which could be easily employed by the society to assess the caliber of novel programming languages and the effectiveness of language improvements.

The linguistic characteristics and proficiencies of a majority of languages can be observed through the numerous instances that are present within those languages. Although these instances may serve as useful aids for acquiring proficiency in the language, they frequently fail to convince novice users of the language's practical applicability in real-world scenarios. Notably, the agent community possesses a set of advanced scenarios that are readily available through the annual Multi-Agent Programming Contest (MAPC). These scenarios can be utilized to conduct comprehensive assessments of agent programming languages.

The limitations and challenges associated with BDI agent programming has been elucidated in recent scholarly investigations, such as [48]. It is crucial to enhance the functionality of current APLs in order to facilitate the widespread implementation of an agent technology. The researchers challenge the conclusions of prior surveys that suggest a lack of refined methodologies and tools as the primary factor, albeit a contributing factor, for the limited APLs adoption. Larsen [49] posits that there inadequacy in incentives for designers to transition to AOP, given that the functionalities exhibited by extant applications in the literature could be readily realized in conventional programming languages composed of minimal additional effort. In [50] present a succinct analysis of the historical and contemporary landscape of agent programming, with a particular focus on approaches based on the BDI model. The integration of artificial intelligence methodologies into agent programming languages is regarded as a pivotal progression towards the widespread adoption of AOP. Consequently, it is a significant matter for further investigation.

VI. CONCLUSION AND FUTURE RESEARCH

The domain of artificial intelligence that investigates agent-based programming is thriving. This survey study has categorized both established and emerging contributions into four distinct groups, namely: agent-oriented programming languages, their cross-language comparisons, exemplars, and extensions of applications designed using these languages. The contents of each submission were succinctly summarized and the primary discoveries were emphasized. In addition to exploring state-of-the-art techniques, we provided a brief survey of the prevalent agent-based programming languages presently under development. Annually, numerous proposals for novel language features and even complete languages are put forth; however, the overwhelming majority of these proposals fail to progress beyond the formal description phase. The paucity of dependable evaluation of the limited range of executed techniques is apparent. A crucial prerequisite for the progression of agent-based programming involves the evaluation of innovative methodologies against the current standard.

Over the course of five years subsequent to the initial publication of studies on APLs, a number of advanced models, frameworks, and platforms have been established. Each of these contributions has not only facilitated the advancement of agent-based literature but also expanded its scope of potential applications. As highlighted in references used in this article, the primary challenge associated with current APLs is not related to their feature repertoire, but rather pertains to their ease of use. When the immediate advantages of acquiring a new language are not readily discernible, individuals tend to lose motivation. The study focused on analyzing APLs that exhibit robustness and eloquence, yet pose considerable challenges in terms of usability, including the absence of a well-defined toolkit, comprehensive documentation, and objective assessments vis-à-vis alternative programming languages. To widen the scope of APL utilization beyond the agent community, it is imperative that forthcoming research endeavors focus on addressing the existing usability challenges.

Data Availability
No data was used to support this study.

Conflicts of Interests
The author(s) declare(s) that they have no conflicts of interest.

Funding
No funding was received to assist with the preparation of this manuscript.

Ethics Approval and Consent to Participate
The research has consent for Ethical Approval and Consent to participate.

Competing Interests

There are no competing interests.

References

[1]. M. Baldoni, C. Baroglio, R. Micalizio, and S. Tedeschi, “Accountability in multi-agent organizations: from conceptual design to agent programming,” Auton. Agent. Multi. Agent. Syst., vol. 37, no. 1, 2023.
[2]. Haldorai, A. Ramu, and S. A. R. Khan, Eds., “Business Intelligence for Enterprise Internet of Things,” EAI/Springer Innovations in Communication and Computing, 2020, doi: 10.1007/978-3-030-44407-5.
[3]. E. Begoli, “Procedural Reasoning System (PRS) architecture for agent-mediated behavioral interventions,” in IEEE SOUTHEASTCON 2014, 2014.
[4]. M. d’Inverno and M. Luck, “Computational Architecture for BDI Agents,” in Springer Series on Agent Technology, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 155–165.
[5]. A. Najjar et al., “Model transformations from the SARL agent-oriented programming language to an object-oriented programming language,” Int. J. Agent-oriented Softw. Eng., vol. 7, no. 1, p. 37, 2019.
[6]. N. B. Othman et al., “SUMMIT: A multi-modal agent-based co-simulation of urban public transport with applications in contingency planning,” Simul. Model. Pract. Theory, vol. 126, no. 102760, p. 102760, 2023.
[7]. M. Raees, T. A. Khan, K. Mustafa Abbasi, A. Ahmed, S. Fazilat, and I. Ahmed, “Context-aware services using MANETs for long-distance vehicular systems: A cognitive agent-based model,” Sci. Program., vol. 2021, pp. 1–12, 2021.
[8]. R. W. Collier, S. Russell, and D. Lillis, “Reflecting on agent programming with AgentSpeak(L),” in PRIMA 2015: Principles and Practice of Multi-Agent Systems, Cham: Springer International Publishing, 2015, pp. 351–366.
[9]. Haldorai and U. Kandaswamy, “Intelligent Spectrum Handovers in Cognitive Radio Networks,” EAI/Springer Innovations in Communication and Computing, 2019, doi: 10.1007/978-3-030-15416-5.
[10]. M. van Steen and A. S. Tanenbaum, “A brief introduction to distributed systems,” Computing, vol. 98, no. 10, pp. 967–1009, 2016.
[11]. G. Petrosino, E. Iotti, S. Monica, and F. Bergenti, “A description of the jadescript type system,” in Lecture Notes in Computer Science, Cham: Springer International Publishing, 2022, pp. 206–220.
[12]. R. C. Cardoso and A. Ferrando, “A review of agent-based programming for multi-agent systems,” Computers, vol. 10, no. 2, p. 16, 2021.
[13]. M. A. Hashmi, M. U. Akram, and A. El Fallah-Seghrouchni, “PLACE: Planning based language for agents and computational environments,” in Engineering Multi-Agent Systems, Cham: Springer International Publishing, 2018, pp. 142–158.
[14]. “PLASA: Programming language for synchronous agents,” 2018.
[15]. M. Alcántara, A. Castañeda, D. Flores-Peñaloza, and S. Rajsbaum, “The topology of look-compute-move robot wait-free algorithms with hard termination,” Distrib. Comput., vol. 32, no. 3, pp. 235–255, 2019.
[16]. “Winter simulation conference 2023,” Winter Simulation Conference 2023, 01-Aug-2016. [Online]. Available: https://meetings.informs.org/wordpress/wsc2023/. [Accessed: 04-Jun-2023].
[17]. A. Krzywicki and W. Wobcke, “Empirical software engineering for evaluating adaptive task-oriented personal assistants: a case study in human/machine event extraction and coding,” Int. J. Agent-oriented Softw. Eng., vol. 7, no. 2, p. 184, 2022.
[18]. W. Wobcke and A. Krzywicki, “Empirical software engineering for evaluating adaptive task-oriented personal assistants: a case study in human/machine event extraction and coding,” Int. J. Agent-oriented Softw. Eng., vol. 7, no. 2, p. 184, 2022.
[19]. Haldorai and A. Ramu, “The Impact of Big Data Analytics and Challenges to Cyber Security,” Advances in Information Security, Privacy, and Ethics, pp. 300–314, 2018, doi: 10.4018/978-1-5225-4100-4.ch016.
[20]. D. Singh, L. Padgham, and B. Logan, “Integrating BDI agents with agent-based simulation platforms,” Auton. Agent. Multi. Agent. Syst., vol. 30, no. 6, pp. 1050–1071, 2016.
[21]. Y. Sánchez, T. Coma, A. Aguelo, and E. Cerezo, “ABC-EBDI: An affective framework for BDI agents,” Cogn. Syst. Res., vol. 58, pp. 195–216, 2019.
[22]. P. Caillou, B. Gaudou, A. Grignard, C. Q. Truong, and P. Taillandier, “A simple-to-use BDI architecture for agent-based modeling and simulation,” in Advances in Intelligent Systems and Computing, Cham: Springer International Publishing, 2017, pp. 15–28.
[23]. A. Ferrein, C. Maier, C. Mühlbacher, T. Niemueller, G. Steinbauer, and S. Vassos, “Controlling logistics robots with the action-based language YAGI,” in Intelligent Robotics and Applications, Cham: Springer International Publishing, 2016, pp. 525–537.
[24]. F. Kaptein, J. Broekens, K. V. Hindriks, and M. Neerincx, “CAAF: A Cognitive Affective Agent Programming Framework,” in Intelligent Virtual Agents, Cham: Springer International Publishing, 2016, pp. 317–330.
[25]. G. Santos et al., “Multi-agent simulation of competitive electricity markets: Autonomous systems cooperation for European market modeling,” Energy Convers. Manag., vol. 99, pp. 387–399, 2015.
[26]. I. García-Magariño, G. Palacios-Navarro, and R. Lacuesta, “TABSAOND: A technique for developing agent-based simulation apps and online tools with nondeterministic decisions,” Simul. Model. Pract. Theory, vol. 77, pp. 84–107, 2017.
[27]. S. Galland, S. Rodriguez, and N. Gaud, “Run-time environment for the SARL agent-programming language: the example of the Janus platform,” Future Gener. Comput. Syst., vol. 107, pp. 1105–1115, 2020.
[28]. A. Anjum, F. Sun, L. Wang, and J. Orchard, “A novel neural network-based symbolic regression method: Neuro-encoded expression programming,” arXiv [cs.NE], 2019.
[29]. S. Leask and B. Logan, “Programming agent deliberation using procedural reflection,” Fundam. Inform., vol. 158, no. 1–3, pp. 93–120, 2018.
[30]. L. da Silva Medeiros, R. E. Julio, R. M. A. de Almeida, and G. S. Bastos, “Enabling real-time processing for ROS2 embedded systems,” in Studies in Computational Intelligence, Cham: Springer International Publishing, 2019, pp. 477–528.
[31]. A. R. Panisson and R. H. Bordini, “Towards a computational model of argumentation schemes in agent-oriented programming languages,” in 2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT), 2020.
[32]. F. Ho and S. Nakadai, “Preference-based multi-objective multi-agent path finding,” Auton. Agent. Multi. Agent. Syst., vol. 37, no. 1, 2023.
[33]. E. S. van Haeringen, C. Gerritsen, and K. V. Hindriks, “Emotion contagion in agent-based simulations of crowds: a systematic review,” Auton. Agent. Multi. Agent. Syst., vol. 37, no. 1, 2023.
[34]. Z. Sun et al., “Simple or complicated agent-based models? A complicated issue,” Environ. Model. Softw., vol. 86, pp. 56–67, 2016.
[35]. “Multi-agent programming contest,” Multiagentcontest.org. [Online]. Available: https://multiagentcontest.org/. [Accessed: 04-Jun-2023].
[36]. C. C. Kerr et al., “Covasim: an agent-based model of COVID-19 dynamics and interventions,” bioRxiv, 2020.
[37]. M. W. Khan and J. Wang, “The research on multi-agent system for microgrid control and optimization,” Renew. Sustain. Energy Rev., vol. 80, pp. 1399–1411, 2017.
[38]. C. Christensen and J. Salmon, “An agent-based modeling approach for simulating the impact of small unmanned aircraft systems on future battlefields,” J. Def. Model. Simul. Appl. Methodol. Technol., vol. 19, no. 3, pp. 481–500, 2022.
[39]. M. A. Zolfagharipoor and A. Ahmadi, “Agent-based modeling of participants’ behaviors in an inter-sectoral groundwater market,” J. Environ. Manage., vol. 299, no. 113560, p. 113560, 2021.
[40]. J. Afara, V. Ajila, H. Macdonell, and P. Dobias, “Use of agent-based modeling to model intermediate force capabilities in (counter)mobility crowd scenarios,” J. Def. Model. Simul. Appl. Methodol. Technol., p. 154851292211417, 2022.
[41]. C. Adam and B. Gaudou, “BDI agents in social simulations: a survey,” Knowl. Eng. Rev., vol. 31, no. 3, pp. 207–238, 2016.

[42]. S. Berger, B. Häckel, and L. Häfner, “Organizing self-organizing systems: A terminology, taxonomy, and reference model for entities in cyber-physical production systems,” Inf. Syst. Front., vol. 23, no. 2, pp. 391–414, 2021.
[43]. V. Joevivek et al., “Spatial and temporal correlation between beach and wave processes: implications for bar–berm sediment transition,” Frontiers of Earth Science, vol. 12, no. 2, pp. 349–360, Jun. 2017, doi: 10.1007/s11707-017-0655-y.
[44]. A. Amirat, A. Hock-Koon, and M. C. Oussalah, “Object-oriented, component-based, agent-oriented and service-oriented paradigms in software architectures,” in Software Architecture 1, Chichester, UK: John Wiley & Sons, Ltd, 2014, pp. 1–53.
[45]. I. Schoon and J. Heckhausen, “Conceptualizing individual agency in the transition from school to work: A social-ecological developmental perspective,” Adolesc. Res. Rev., vol. 4, no. 2, pp. 135–148, 2019.
[46]. M. J. Ferguson and J. A. Bargh, “How social perception can automatically influence behavior,” Trends Cogn. Sci., vol. 8, no. 1, pp. 33–39, 2004.
[47]. C. Savaglio, M. Ganzha, M. Paprzycki, C. Bădică, M. Ivanović, and G. Fortino, “Agent-based Internet of Things: State-of-the-art and research challenges,” Future Gener. Comput. Syst., vol. 102, pp. 1038–1053, 2020.
[48]. R. H. Bordini, A. El Fallah Seghrouchni, K. Hindriks, B. Logan, and A. Ricci, “Agent programming in the cognitive era,” Auton. Agent. Multi. Agent. Syst., vol. 34, no. 2, 2020.
[49]. J. B. Larsen, “Going beyond BDI for agent-based simulation,” J. Inf. Telecommun., vol. 3, no. 4, pp. 446–464, 2019.
[50]. Anandakumar, H., and R. Arulmurugan. "Supervised, unsupervised and reinforcement learning-A detailed perspective." J Dyn Control Syst 11 (2019): 429-433.