Journal of *Sensor and Actuator Networks*

MDPI

Perspective

Agents and Robots for Reliable Engineered Autonomy: A Perspective from the Organisers of AREA 2020

Rafael C. Cardoso [1,*], Angelo Ferrando [2,*], Daniela Briola [3,*], Claudio Menghi [4,*] and Tobias Ahlbrecht [5,*]

1 Department of Computer Science, The University of Manchester, Manchester M13 9PL, UK
2 Department of Computer Science, Bioengineering, Robotics and Systems Engineering (DIBRIS), University of Genova, 16145 Genova, Italy
3 Department of Informatics, Systems and Communication (DISCO), University of Milano Bicocca, 20126 Milan, Italy
4 Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, L-4365 Luxembourg, Luxembourg
5 Department of Informatics, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany
* Correspondence: rafael.cardoso@manchester.ac.uk (R.C.C.); angelo.ferrando@dibris.unige.it (A.F.); daniela.briola@unimib.it (D.B.); claudio.menghi@uni.lu (C.M.); tobias.ahlbrecht@tu-clausthal.de (T.A.)

Abstract: Multi-agent systems, robotics and software engineering are large and active research areas with many applications in academia and industry. The First Workshop on Agents and Robots for reliable Engineered Autonomy (AREA), organised the first time in 2020, aims at encouraging cross-disciplinary collaborations and exchange of ideas among researchers working in these research areas. This paper presents a *perspective* of the organisers that aims at highlighting the latest research trends, future directions, challenges, and open problems. It also includes feedback from the discussions held during the AREA workshop. The goal of this perspective is to provide a high-level view of current research trends for researchers that aim at working in the intersection of these research areas.

Keywords: multi-agent systems; robotics; software engineering; verification and validation; human–agent interaction

Citation: Cardoso, R.C.; Ferrando, A.; Briola, D.; Menghi, C.; Ahlbrecht, T. Agents and Robots for Reliable Engineered Autonomy: A Perspective from the Organisers of AREA 2020. *J. Sens. Actuator Netw.* **2021**, *10*, 33. https://doi.org/10.3390/jsan10020033

Academic Editor: Thomas Newe

Received: 15 March 2021
Accepted: 13 May 2021
Published: 14 May 2021

1. Introduction

The robotics market is dramatically changing. Robots are more and more used to replace humans in their activities. For example, robots can be used in emergency search and rescue scenarios to reduce risks for humans rescuers. To operate in unpredictable environments, robots often need to be autonomous. Autonomous robots can perform their tasks with a high degree of autonomy without any human supervision. In addition, robots need to operate in a *reliable* manner to avoid failures that can have catastrophic consequences and lead to the loss of human life. The design of such robotic applications is complex since it requires engineers to consider different requirements related to different research domains.

The design of robotic applications requires *multi-agent solutions*. Robots are no longer only used in industrial applications, where robots operate in highly controllable and predictable environments. They are also used in an increasing number of domains, where the environment is often unpredictable and agents can have unexpected behaviours. For example, in an emergency search and rescue scenario the environment in which the robots operate is unpredictable: the structure of the buildings where robots are deployed may not be known in advance, and humans can have unpredictable reactions in emergencies. Robots in these applications often benefit from (or require) some level (semi or full) of autonomy. In addition, the missions the robots need to achieve are more and more complex and require multiple robots, with different capabilities, to collaborate. Thus, multi-agent solutions are required.

Verification and Validation (V&V) aims at checking whether software behaves as expected. The distributed and autonomous nature of multi-agent systems poses a novel set of challenges for V&V. For example, the autonomous behaviour responsible for decision-making should (ideally) be extensively verified since these systems are expensive to produce and are often deployed in safety-critical situations. However, the autonomous behaviour of these systems is often unpredictable: it depends on the environmental conditions in which the system operates, and often changes at runtime. Thus, autonomous robotic systems are introducing a novel set of challenges that need to be addressed by novel V&V techniques.

Software Engineering (SE) refers to a branch of computer since that aims at supporting rigorous software development. Engineering multi-agent systems requires systematic and rigorous techniques that allow developing systems that meet their requirements. Multi-agent systems are instances of complex distributed systems. They require engineers to define the software architecture, to design the agent behaviours (e.g., through models), the protocols (if any) that agents should use to communicate, and how and when agents need to collaborate. Selecting and defining all of these components is often a difficult and complex activity, since it requires cross-disciplinary skills, and knowledge of multi-agent solutions, and the verification and validation to be used to support software design.

Finally, since multi-agent systems usually need to interact with people, engineers need to consider *human–agent interactions* as a key feature during the software design. Human–robot interactions are complex to be designed. Humans can (negatively) affect the behaviour of the robots, especially when humans and robots are collaborating for achieving certain goals. Human behaviour can trigger unexpected software reactions. Software components must be designed to adapt and modify their behaviours and to effectively support unexpected human actions.

Therefore, the design of complex robotic applications requires combining solutions coming from different research areas: multi-agent systems, verification and validation, software engineering, and human–agent interaction. This paper presents a *perspective* from the organisers of the first workshop on Agents and Robots for reliable Engineered Autonomy (https://area2020.github.io/ accessed: 14 May 2021) (AREA 2020). The goal of this workshop was to attract researchers from these areas, to support the exchange of ideas, and the cross-fertilisation and collaboration among the different research communities. This perspective presents some of the latest research trends and promising solutions in each of these areas. It is based on the research experience of the authors, and some of the discussions held during the AREA workshop. As such, it does not aim to be a complete and detailed review of the work done in these research areas, but it aims to be an initial read for researchers that aim at working in the intersection of these areas based on a speculative view of the organisers of AREA 2020.

This perspective paper is organised as follows. Section 2 concerns *multi-agent programming*. It discusses the use of programming languages designed for multi-agent systems for developing decision-making in robots, listing some of the tools for agents and robots individually and how these have been combined by the community. Section 3 concerns *verification and validation*. It presents works performed in the verification and validation with a special interest on multi-agent systems and robotic applications. Section 4 concerns *software engineering*. It describes research work in SE with a special interest in multi-agent systems. Section 5 concerns *human–agent interactions*. It describes research works that concern the development of robotic applications that consider human–agent interactions in the software design. Finally, Section 6 concludes our perspective. It summarises and discusses the challenges identified for each of the research areas we considered, and links the findings of the different sections.

2. Multi-Agent Programming

In this section, we will present some techniques that support the use of agent programming for the development of robotic applications. We will then present some challenges that prevent the effective use of agent programming for developing robotic systems.

One of the most popular models for agent programming languages is the Belief–Desire–Intention (BDI) model [1,2]. In the BDI model, there are three major concerns to model the agent behaviours: *beliefs*—knowledge that the agent has about the world, itself, and other agents; *desires*—goals that the agent wants to achieve; and *intentions*—recipes on how to achieve goals. Some examples of BDI agent programming languages include Jason [3] (used in many of the applications we cite in our perspective paper), JaCaMo [4,5] (Jason combined with two other technologies to provide first-class support for programming agents, environments, and organisations), and GWENDOLEN [6] (a more bare-bones agent language made specifically to support formal verification of agent programs).

The Robot Operating System (ROS) [7] is the de facto standard for the development of software for robotic applications. ROS supports the development of robotic applications through ROS nodes. An ROS node is a process that performs some computation. An ROS application is made by several nodes (representing components and subsystems of the robot) that communicate with one another following the publisher/subscriber model. The main advantage of ROS is its interoperability with different robot manufacturers and models. Since robot manufacturers provide support for executing the ROS, developers do not have to learn the firmware of each robot model.

An approach combining JaCaMo (in particular, the agent and environment layers) with ROS is presented in [8]. Their approach uses environment artifacts to implement ROS nodes and to provide actions for agents to publish and subscribe to nodes. Since environment artifacts in JaCaMo are implemented in Java, the approach makes use of the *rosjava* package (http://wiki.ros.org/rosjava accessed: 14 May 2021) (an implementation of some of ROS core features in Java) to make the artifacts able to interact with ROS. However, *rosjava* is not directly maintained by the ROS community. Therefore, it requires additional time to be supported after each new ROS release.

A Jason based integration with ROS is introduced in [9]. Their approach modifies Jason's reasoning cycle to support agents with the ability to subscribe and publish in topics from ROS nodes. Extra code for ROS (written in C++) is required for the integration as well.

In [10,11], Jason is linked with ROS through their SAVI ROS BDI architecture. This architecture is implemented in Java (using the *rosjava* package, and as such has the same disadvantage present in [8]) and mainly introduces a state synchronisation module that acts as a mediator between ROS and Jason by managing perceptions, incoming and outgoing messages, and actions being sent by the agent.

Differently from all of the above approaches, the authors in [12] propose an interface for programming autonomous agents in ROS that works without any changes to ROS or to any of the supported agent languages (Jason and GWENDOLEN). This is achieved through the *rosbridge* library [13], which allows code written in other languages (ROS has native support for C++ and Python only) to communicate with ROS topics through the WebSocket protocol.

Perspective of the Authors

As noted in recent agent programming reviews and surveys [14–16], there are still many open challenges that prevent agent programming languages to be used in the robotic domain. Some of these challenges include:

- the limited set of features provided by existing agent-based languages;
- immature methodologies and tools;
- no significant advantages for developers to change to agent programming since most applications can be implemented in more contemporary programming languages;

- the lack of quantitative and qualitative comparisons with other agent languages and other programming paradigms that guide developer in the selection of the most suitable language for their needs;
- the limited integration of agent-based technologies with other techniques, e.g., techniques coming from Artificial Intelligence (AI).

Additionally, there are other challenges that are more specifically related to the use of agent programming languages in robotic systems development.

One of these challenges is the limited support for high frequency data that coming from sensors, which are commonly used to represent "beliefs." When high-frequency data representing beliefs come from sensors, then the agent has to spend a large amount of time reasoning on these new perceptions. This reduces the performance of the robots in computing the new actions to be executed. A common solution to this problem is to use filters that limit the amount of data that is perceived, either by decreasing the frequency or limiting data based on its content and what would be interesting to the agent. However, these filters are often domain-specific and have to be tweaked based on the application.

Another challenge is the compatibility of agent languages with popular robotic frameworks such as ROS. An increasing number of languages are being extended by the community to work with ROS; however, these often modify either ROS or the agent language (or sometimes both) which can discourage new developers from using them.

An earlier survey [17] on agent languages for programming autonomous robots has identified four major challenges:

1. support for agent languages in robotic frameworks;
2. effectively managing sensor data into beliefs;
3. support for real-time reactivity;
4. synchronising robots while executing their plans.

As previously discussed, a significant amount of research has been conducted for addressing challenges 1 and 2. However, more work is needed to support additional agent-based languages, and more sophisticated and effective filtering techniques to manage sensor data. Less research has been conducted to address challenges 3 and 4, since these challenges do not always appear in robotic applications. For example, in a scenario in which a robot has to inspect a nuclear facility, the robot should be able to handle and adapt to failures. We believe that research on real-time reactivity of robots and the effective synchronisation of robots for executing their plans, on the one hand, will benefit from the additional support provided for existing agent-based languages and the more sophisticated and effective management, on the other hand, may highlight limitations of these frameworks and pave the way for further additional research.

3. Verification and Validation

As discussed in our introduction, reliability is very important in the design of robotic applications. However, multi-agent applications are posing a new set of challenges for the verification and validation (V&V) activities. In this section, we are considering both static verification techniques for MAS and dynamic verification techniques. Static formal verification techniques, such as Model Checking and Theorem Proving, usually check whether the system meets its requirements. Requirements are usually represented using formal specifications, a.k.a. properties, the system is usually represented using models. Dynamic formal verification techniques, such as Runtime Verification techniques, usually monitor the system execution and check whether observed behaviours meet the system requirements.

In this section, we describe the latest developments in the context of formal verification and validation of MAS (Section 3.1) and robotic systems (Section 3.2). Some of the works we present in this section were also discussed by a recent survey on formal verification applied to autonomous systems and robotic applications [18]. Then, we will argue (Section 3.3) that, as also argued by [19], robotic applications and autonomous systems pose a new set of challenges for formal verification and validation techniques.

3.1. Multi-Agent Systems

For MAS, V&V techniques usually check the behaviour of a set of agents collaborating or competing amongst themselves to achieve certain goals.

3.1.1. Model Checking

Model-checking exhaustively verifies a system against a formal property. It returns a Boolean verdict: *true* if the property is satisfied, and *false* and a counterexample, if it is not. Model-checking techniques are usually compute-intensive since the implemented procedures have a high temporal complexity because all the behaviours of the system need to be analysed for proving that the property is satisfied.

In [20], a model checker for verifying MAS, called MCMAS, is proposed. MCMAS supports temporal, epistemic and strategic properties. In its standard version, MCMAS requires to know the number of agents at design time. In [21], a parametric extension (MCMAS-P) to handle scenarios where the number of components cannot be determined at design time is presented, while, in [22], a more expressive extension (MCMAS-SL) is proposed to support strategy logic. MCMAS has been used in many different works for verifying MAS. In particular, in [23], where an analysis is carried out on the verification problem of synchronous perfect recall multi-agent systems with imperfect information. While the general problem is known to be undecidable, [23] shows that, if the agents' actions are public, then verification is decidable.

In [24], the authors propose a method for, and implement a working prototype of, an ethical extension to a rational agent governing an unmanned aircraft. Differently from [20], this work is focused on verifying BDI agents, defined using the agent language Ethan, an extension of GWENDOLEN. The resulting ethical agent is verified in AJPF [25], a model checker for agent programs. Differently from MCMAS, which verifies an abstract model of the MAS (i.e., a Concurrent Game Structure—CGS), AJPF verifies the source code of the MAS application. Furthermore, MCMAS assumes that properties are expressed using Alternating-Time Temporal Logic (ATL), which allows for reasoning on agents' strategies, while AJPF assumes that properties are expressed in Linear Temporal Logic (LTL) (enriched with epistemic logic operators).

In [26], the authors proposed the VERMILLION framework. VERMILLION targets a broad class of avionics systems that is amenable to analysis using formal methods. It extends the BDI model to incorporate learning, safety, determinism, and real-time response, and represents the abstract formal model using the Z language [27]. Compared to MCMAS, VERMILLION performs formal verification on an abstract model of the system, and not to the source code of the MAS. This requires engineers to build the abstract model of the MAS before running the verification framework.

Autonomous platoons are a typical example of MAS which are subject to extensive research. Formal verification of autonomous platoons has been considered for example in [28]. In this work, the authors proposed a reconfigurable multi-agent architecture to ensure the safety of the platoon, and specifically to guarantee a certain inter-vehicle distance among the different vehicles of the platoon. The authors proposed a model for the platoon that enables vehicles to join and leave the platoon, and verified whether this model ensures the satisfaction of a set of safety properties. Safety properties are formally verified using the Uppaal model checker [29]. Additionally, this work proposes to use the Webots simulator (https://github.com/cyberbotics/webots accessed: 14 May 2021) to simulate certain behaviors of the model.

Similarly, in [30], the authors applied formal verification to the model of the system and the actual implementation to ensure that autonomous decision-making agents in vehicle platoons never violate some safety requirements. In addition, in this work, the model checking procedure relies on the Uppaal model checker: the models of the agents are translated into timed automata, which are verified in Uppaal.

In [31], the authors present a new technique for model checking the logic of knowledge and commitments (CTLKC+). The proposed technique is fully-automatic and reduction-

based. It reduces the problem of model checking CTLKC+ specifications to the problem of model checking an ARCTL [32] specifications. ARCTL is an existing logic that is supported by an existing model checker that relies on the NuSMV symbolic model checker [33].

3.1.2. Runtime Verification

There are various runtime verification techniques available in literature for MAS. In [34], the authors present a framework to verify agent interaction protocols at runtime. The formalism used in this work allows using variables to represent complex MAS behaviours. In [35], the authors extended their approach by supporting the usage of multiple monitors. Specifically, the global specification, which is used to represent the global protocol, is translated into partial decentralised specifications—one for each agent of the MAS. In [36,37], other works on runtime verification of agent interactions are proposed for the JADE platform. Specifically, in [36], the authors propose a framework called Multi-agent Runtime Verification (MARV). In this framework, requirements of MAS interaction during runtime are defined, such as availability and trustability. Differently from the other works, the requirements are expressed using natural language. The translation to a more formal representation is seen as a future work and not supported yet. Considering interactions in JADE, in [37], we may find a different approach which is partially obtained at runtime. In fact, the proposed method is performed in a semi-runtime way, where logs of messaging events are kept, and an algorithm for converting these logs to Time Petri Net as runtime program models is used.

When agents are dynamically adaptable, we may find application of runtime verification as presented in [38], where a runtime verification framework for dynamic adaptive MAS (DAMS-RV) based on an adaptive feedback loop is presented.

In [39], the authors propose a framework that combines model checking and runtime verification for analysing MAS. In this framework, the agents are first verified statically (using the AJPF [25] model checking), and, then, they are validated at runtime, through runtime verification using an extension of the work proposed in [34].

3.2. Robotic Applications

This section analyses works related to the formal V&V of robotic systems.

3.2.1. Model Checking

In [40], the authors propose an approach to formally verify an autonomous decision-making planner/scheduler system for an assisted living environment with the Care-O-bot robotic assistant. This is done by converting the robot house planner/scheduler rules into a multi-agent modelling language, i.e., Brahms model [41], and then, by translating this model into the PROMELA [42] language, which is then verified using the SPIN model checker [43]. Differently from the works we presented in the previous section, in this work, the model to be verified concerns scheduling rules used by an actual robot, rather than the reasoning process of an abstract agent.

In [44], the authors verified a formal model that describes mobile robot protocols operating in a discrete space is proposed. This formal model is then verified using the DiVinE model checker [45]. In [46], the authors propose an approach to verify real-time properties of ROS systems related to the communication between ROS nodes. Specifically, the authors analysed the source code of the Kobuki robot. Verification is performed by using the Uppaal model checker.

In [47], the authors analysed a collision avoidance protocol for multi-robot systems based on control barrier functions. The authors formally verified the properties of the collision avoidance framework. They showed that their controller formally guarantees collision free behaviour in heterogeneous multi-agent systems by applying slight changes to the desired controller via safety barrier constraints.

Finally, formal methods are also used to check whether the tasks of a robotic application can be scheduled with respect to a given hardware platform (e.g., [48,49]). For example, some of these works considered components specified in GenoM [50] (a middleware for

robotic development similar to ROS) and automatically translate them into FIACRE [51], a formal language for timed systems.

3.2.2. Human–Robot Interaction

Building reliable software systems involving human–robot interactions poses significant challenges for formal verification. In [52], the authors propose a risk analysis methodology for collaborative robotic applications, which relies on well-known standards, and use formal verification techniques to automate the traditional risk analysis methods. In [53], the authors propose an innovative methodology, called SAFER-HRC, is presented. This methodology is centred around the logic language TRIO and the companion bounded satisfiability checker Zot [54], to assess the safety risks in a Human–Robot Collaboration (HRC) application.

3.2.3. Runtime Verification

Runtime Verification approaches for robotic applications are also discussed in the scientific literature. RobotRV [55] is a data-centred real-time verification approach for robotic systems. Within this approach, a domain-specific language named RoboticSpec is designed to specify the complex application scenario of the robot system.

Another runtime verification framework, called ROSMonitoring [56,57], allows the verification of ROS-based systems. In ROSMonitoring, runtime monitors are automatically synthesised from high-level specifications and used to verify formal properties against message passing amongst ROS nodes. The advantage of this framework is its being formalism-agnostic and portable. Indeed, the formal part of the monitors is decoupled and can be easily replaced.

3.2.4. Machine Learning

Machine learning is widely relevant for designing robotic applications. In [58], the authors consider the problem of formally verifying the safety of an autonomous robot equipped with a neural network controller that processes LiDAR images to produce control actions. The contributions are: (i) the definition of a framework for formally proving safety properties of autonomous robots equipped with LiDAR scanners; (ii) the notion of imaging-adapted partitions along with a polynomial-time algorithm for processing the workspace into such partitions; and (iii) a Satisfiability Modulo Convex (SMC)-based algorithm combined with an SMC-based pre-processing for computing finite abstractions of neural network controlled autonomous systems.

3.3. Perspective of the Authors

A lot of research was done on formal verification of MAS and Robotic applications. However, many challenges still need to be addressed. In the following, we will discuss two of these challenges.

- *scalability.* Many approaches suffer from scalability issues [59]. Researchers should find more efficient ways to verify the system under analysis especially when the number of agents and robots increases. Indeed, MAS and robotic applications are intrinsically distributed, and we expect the number of robots and agents of future robotic applications to increase over time. As previously mentioned, scalability issues are less relevant for dynamic verification approaches, such as runtime verification that only verify subsets of system executions. We believe that combining static and dynamic verification may be a valuable direction to address this challenge;
- *verification of ML components.* ML components are mode and more used in safety-critical scenarios (e.g., Robotic applications). However, the behaviours of machine learning components are usually not understandable by humans. Indeed, the behaviour of ML components is not defined a priori by humans, but ML components learn their behaviours from a set of training data. This poses the challenge of understanding if a ML component ensures the satisfaction of safety properties. While in

the past years many works have been proposed to enhance learning algorithms with formal methods, a lot of work needs to be done to make these approaches applicable in practice. For example, for MAS applications, some works have been proposed for single Reinforcement Learning agents, but few of them considered Multi-Agent Reinforcement Learning.

4. Software Engineering

This section provides a brief overview on some of the software engineering (SE) techniques that aim to support the development of reliable *multi-agent* and *robotic* applications. Researchers are extending, adapting, and creating new SE techniques to meet the needs of robotic applications. However, there are still many challenges that prevent the effective and efficient development of multi-agent systems and the community requires novel SE solutions. We introduce the overall main problems by giving an overview of them, citing some of the existing and known approaches and solutions, and discussing promising research trends and open problems.

Specifically, in this section, we discuss rigorous and systematic techniques that allow the specification of MAS requirements (Section 4.1), effective and efficient techniques that support testing MAS (Section 4.2), and simulation tools that enable reproducing the MAS and robotic behaviour with reasonable accuracy (Section 4.3). Finally, we will discuss our perspectives (Section 4.4).

4.1. Requirement Specification

The specification of the requirements of a multi-agent application is critical during software development. In MAS, requirements specification often concerns the definition of the task, also known as mission [60], what the application should achieve, and how to make the requirement executable by the MAS. Compared with conventional software, the presence of multiple agents makes the requirement specification phase more complex and error-prone since engineers need to precisely identify the different agents and identify the tasks they need to execute [61]. To support engineers in the specification of the requirements of the MAS application, several tools were proposed in the literature, such as natural languages, logic-based languages, pattern-based languages, domain-specific languages, and goal-modelling techniques.

In the following, we summarise some of the solutions proposed in these areas and evaluate how these solutions were used within the papers presented in the AREA 2020 workshop (https://area2020.github.io/ accessed: 14 May 2021), since they provide good examples of research covering different aspects of MASs. Specifically, Table 1 summarises the requirement specification technique used for each of the papers presented in the AREA 2020 workshop. Each row contains a requirement specification technique, i.e., natural languages, logic-based languages, pattern-based languages, domain-specific languages, and goal-modelling. Each column contains the reference to one of the papers presented in the AREA 2020 workshop. The cell at the intersection between a row of one requirement specification technique, and a column, of one paper presented in the AREA 2020 workshop, indicates the requirement specification technique used in that paper. For example, the marker at the intersection between the row labelled as "logic-based languages" and the column labelled with the reference [62] indicates that a logic-based language is the requirement specification technique used in [62].

Table 1. Requirements specification technique used by the papers published in AREA 2020 ([63] is not reported in the table since it does not consider MAS requirements).

	[62]	[64]	[65]	[66]	[67]	[68]	[69]	[70]	[10]
Natural Language						✓			
Logic-based	✓	✓							
Pattern-based									
Domain-specific			✓	✓	✓			✓	✓
Goal-modelling									
Demonstrations							✓		

4.1.1. Natural Languages

In many contexts, requirements are initially expressed in natural language. This is a common case in many industrial applications (e.g., [71,72]). Natural languages offer significant benefits, they are easy to understand, and they support effective communication among different stakeholders. Several works considered the role of natural languages in the requirement specifications of multi-agent and robotic systems. For example, in [73], the authors proposed an approach to teach agents to communicate with humans in natural language. In [74], the authors analyse how to utilise and extend the Software Requirements Specifications model (IEEE Std 830-2009) to support the specification of requirements of multi-agent systems. In [75], the authors apply techniques of natural language processing for identifying the requirements and goals of multi-agent systems.

One paper presented at the AREA workshop assumed that the requirements of the MAS are specified using natural language. Specifically, in [68], the authors propose different types of interactions between an MAS and the final users who might benefit from communication-intensive, voice-based interactions.

4.1.2. Logic-Based Languages

Many researchers specify the requirements of the MAS in a logic-based language (e.g., [18,19,76–78]). Logic-based languages, such as LTL or CTL, assume that requirements are expressed using a set of atomic propositions that express relevant statements on the multi-agent system, combined with logical operators. One of the main advantages of using logic-based language is the availability of automated tools that support verification (e.g., model checking) and synthesis.

Two papers presented at the AREA workshop assumed that the requirements of the MAS are specified using logic-based languages. In [62], the authors use the TRIO [79] logical-language to specify the requirements of the MAS. TRIO is a first-order logical language. It provides temporal operators to constrain the values of some propositions at different time instants. In [64], the authors use the logical-based language provided by Uppaal [80] to specify the requirements of the MAS. Uppaal allows specifying missions through an extension of the CTL logic, which is a subset of TCTL. Specifically, it allows the specification of properties constraining a proposition to hold globally (resp. eventually) for every execution or querying whether there exists an execution such that a proposition holds globally (resp. eventually).

4.1.3. Pattern-Based Languages

Pattern-based languages are a common solution to solve recurrent problems of many domains, including robotics [60,81], cyber-physical systems [82], self-adaptive systems [83,84], machine learning [85], IoT [86], and multi-agent systems [84,87–89]. Existing design patterns in the field of multi-agent systems were classified in a recent survey [88].

By analysing the papers presented at the AREA workshop, we noticed that none of the published papers used pattern-based languages to specify the requirements of the multi-agent system.

4.1.4. Domain-Specific Languages

Several Domain-Specific Languages (DSL) for MAS were proposed in the literature (e.g., [90–92]). For MAS, DSLs usually provide constructs that enable engineers to model agents, the task they need to execute, and their interactions. Some of the recent DSLs for multi-agent systems are reported in the following.

In [93], the authors present a DSL for multi-robot application, based on the robotic mission specification patters [60]. In [94], the authors apply the DESIRE specification framework [95] on a case study on multi-agent systems. In [91], the authors propose a DSL for MAS. They also use the language and their graphical tool support for developing an MAS using a model-driven development approach.

Four papers presented at the AREA workshop proposed a DSL for the specification of a robot's core behaviours. In [65], the authors propose the use of Capability Analysis Tables (CATs). CATs provide a tabular representation that connects the inputs, outputs, and the behaviours of an MAS. Differently from other tools, e.g., logic-based languages, CATs are more understandable by non-expert users. We also considered the language used for specifying requirements by [66] as a domain-specific language. In this work, the authors assume that the requirement concerns reaching some specific states of the competence-aware systems used to model the MAS.

In [67], the authors use Jadescript [96] to specify the MAS and its requirements. Jadescript is a novel agent-oriented programming language compiled to Java. In [70], the authors consider the Planning Domain Definition Language (PDDL) [97] to specify the task to be performed by the MAS. PDDL is a DSL that is proposed to standardise automated planning languages. It enables the definition of the domain and the problem. The domain definition allows users to define predicates and operators (a.k.a. actions). The problem definition defines the objects of the problem instance, its initial state, and the goal. In [10], the authors use the Jason language to express the requirements of the MAS application. Jason is an agent-oriented programming language based on the BDI software model.

4.1.5. Goal-Modelling Techniques

Several goal-modelling techniques (e.g., Tropos [98], Gaia [99], Mobmas [100], INGENIAS [61]) support the development of multi-agent applications. These techniques enable users to identify the goals and the agents of the application. They also usually support the decomposition of goals into subgoals, and subgoals into tasks, and the assignment of tasks to agents.

By analysing the papers presented at the AREA workshop, we noticed that none of the published papers used goal-modelling techniques to specify the requirements of the multi-agent system.

4.1.6. Demonstrations

Many approaches use demonstrations to train the agents of a multi-agent application to perform their tasks [101–104]. Demonstration approaches usually require a human to demonstrate to the agent the task to be performed. Then, the agent learns and repeats the task.

One paper presented at the AREA workshop used demonstrations to specify the system goals. Specifically, in [69], the authors proposed a semi-supervised learning approach from demonstrations through program synthesis. Within this approach, a human operator specifies the goals by demonstrating to the agents how to perform the task. The MAS automatically infers high-level goals from the demonstration, synthesises a computer program based on the demonstrations, and learns behavioural models for predictive control.

The analysis of the papers presented at the AREA workshop (see Table 1) shows that, despite the many approaches proposed in the literature, it seems that there is still no consensus on the strategy to be used to specify MAS requirements. Most of the papers (5) used domain-specific languages for specifying the MAS requirements, followed by logic-based languages (2), natural languages (1), and demonstrations (1). Many research papers often do not explicitly discuss the reason that motivates the usage of a given

specific language for the requirement specification. For example, many papers assume that requirements are expressed using logic-based languages (a formalism that easily supports the development of research prototypes). We believe that this is often dictated by the fact that these languages have formal semantics and are supported by verification and synthesis tools that can be reused for verifying and synthesising plans of the MAS. Others use DSLs. We believe that this choice is dictated by the need of providing solutions closer to the needs of the final users.

4.2. MAS Testing

Testing robotic and multi-agent systems is a complex activity. It starts from the unit testing level, where the units can be the single agent functionalities, to the system integration level. At the system integration level, many aspects, such as the concurrent execution of the agents, the environment integration, the control over the coordination protocol and communication management and modalities, are considered. Testing all these aspects is inherently complex, and becomes even harder when tools do not provide appropriate support.

Agent development frameworks, such as JADE [105] or Jason, support developing and testing of MAS and robotic applications. However, each of these frameworks comes with some limitations. These limitations are particularly relevant for the design and development of MAS that need to be executed in a physical distributed environment, deployed over many machines and used by humans. A concrete testing and maintenance support that covers all of these requirements is still missing. In addition, the performances of V&V techniques are not sufficient to support the requirement of distributed environments, and cannot be easily integrated in the running environment.

In this section, we summarise a set of works that considered the problem of testing MAS and robotic applications. These works have been selected based on the authors' knowledge and experience. Specifically, in this section, we considered works that are: (1) exploring current support for testing MASs, (2) analysing applications of standard testing techniques from the software engineering community, and (3) reporting how V&V approaches have been exploited in the field.

4.2.1. Support for Testing MASs

A starting point for testing an MAS is provided by the development framework itself, allowing and supporting the agent internal state inspection and the messages exchange supervision. Both JADE and Jason offer such kind of tools (the Mind Inspector in Jason, the Introspector agent, and the Sniffer in JADE), which are usually necessary to perform a manual first check of the behaviours of agents and of the overall MAS, or that can be the used to perform more automated tests and verification.

Some studies related to JADE are, for example, [106,107]. In [106], the authors proposed a solution, based on mock agents, that is presented to perform testing of a single role of an agent under successful and exceptional scenarios; in [107], the authors proposed a framework developed on JADE [105] for building and running MAS test scenarios. This framework relies on the use of aspect-oriented techniques to monitor the autonomous agents during tests and control the test input of asynchronous test cases.

In [108], the authors proposed an approach for enabling DevOps activities (that is, collaborative programming features to achieve fast and continuous deployment of complex systems) in a new framework based on JaCaMo-web IDE [109] (which is a tool related to JaCaMo (http://jacamo.sourceforge.net/ accessed: 14 May 2021), and consequently to Jason). This extension to the IDE enables for interactive facilities, such as the automatic access to the updates made to the components (agents), and the possibility to execute tests on temporary running instances (allowing the framework to check the compatibility of new changes using the real scenario, since tests are performed while the programmer types, without affecting running instances). In addition, this extension provides facilities for preventing conflicts when developers attempt to edit a resource simultaneously, and

for managing versions. Thus, this is a concrete step toward offering real testing facilities to AOP.

There are some other works related to MAS testing that do not consider JADE or Jason. The SUnit framework [110] provides a model based approach based on an extension of Junit. The eCAT testing framework [111,112] supports continuous testing and automated test case generation. In [113], the authors proposed a tracing method supported by a tool implementation to capture and analyse dynamic runtime data collected by logging the behaviours of a set of agents. These solutions are ad-hoc solutions with limited practical adoption, compared to JADE or Jason. They usually rely on specific formal languages that make their use difficult, in particular in industrial applications.

4.2.2. Applications of Standard Testing Techniques from Software Engineering

Standard model-driven testing techniques coming from the SE domain (e.g., [114]) are usually difficult to be used in multi-agent applications. One of the main limitations of these techniques is that they require a model of the multi-agent application. While there are standard approaches for modelling object-oriented or service-oriented systems, standard models for MAS are less mature. Therefore, MAS design is still often performed by relying on ad-hoc solutions, which need to be standardised. Therefore, we believe that there is room for exploiting SE techniques in the Agent-Oriented Software Engineering (AOSE) area [115].

Some examples where standard SE testing approaches or techniques have been adopted for MASs platforms are reported in the following.

The BEAST methodology [116] is an example of agile testing methodology for multi-agent systems based on Behaviour Driven Development (BDD). It automatically generates test cases skeletons from BDD scenarios specifications. The BEAST framework enables testing MASs based on JADE or JADEX [117] platforms.

In [118], mutation testing is used to test Jason specifications. The authors propose a set of mutation operators for Jason, and present a mutation testing framework for individual Jason agents based on these mutation operators. In [119], the authors proposed a property-based testing (a particular form of model based testing) framework for MASs specified in Jason. Specifically, the authors proposed to replace a subset of the agents by a QuickCheck [120] state machine. This state machine interacts with the other agents by sending messages and modifying the environment, and judging whether the remaining real agents are correctly implemented by examining the messages sent to any replaced agent, and the belief perceptions that they receive.

4.2.3. Exploitation and Integration of V&V Approaches

There are many studies to check that the interaction between agents conforms to a formal specification (which is as a part of the testing activity). This is a very complex task since it involves the need to formally model and verify the protocol (as described in Section 3), and a concrete way to oversee the runtime execution, mixing together both a theoretical and an applied aspect.

A concrete example of the integration of a V&V technique directly into a development platform is proposed in [121,122], where an extension of the JADE Sniffer is used to create a monitor able to verify at runtime the MAS execution with respect to a global protocol specified using the Attribute Global Types formalism [123–125]. Since this is a JADE agent, it can be directly used in any JADE MAS, provided that the global protocol (if any) is translated into the requested language. Similarly, in [121], an extension of Jason is proposed to achieve the same monitoring.

In [39], we can find a work presenting the combination of formal verification and validation in the context of MAS verification, integrated into the MCAPL [126] framework. The verification part is obtained through the verification of the BDI agent, implemented in Gwendolen [6], against a formal model of the environment, while the corresponding validation is achieved through runtime verification, where monitors are used to verify at runtime that the real environment does not violate the assumptions made by the model checker.

4.3. Simulation Tools

Physically distributed systems are often needed in industrial and academic solutions, such as unmanned vehicles, logistic, ambient intelligent systems.

4.3.1. Simulation Tools for MASs

JADE is largely adopted in industry, due to its simplicity in modelling agents' tasks, its support for agent communication, and its extensive community support. JADE suffers from scalability issues [127,128]. In addition, it provides limited support for dynamically discovering new platforms that join the MAS at run-time. This limitation forces the usage of p2p communication, which is quite common in real world applications ([129]). However, differently from Jason, JADE was integrated within existing simulation platforms.

Before deploying the MAS, developers need to test their solutions. To be representative of a real situation, the physical environment must contain agents representing physical entities, such as vehicles, computers, and production systems. Testing these systems is usually done by first relying on simulations. We can simulate the behaviour of an MAS by relying on some stub entities, instead of using the actual MAS components. Stubs implement a logic which is similar but usually simpler than the one that will be executed by the actual agents. For example, stubs may abstract and simplify the interaction protocols used by the different agents to communicate.

Some simulators for MASs exist, and we will present them in this section, but they deal with the simulation in different ways.

In [130], the authors propose DMASF, a Python distributed simulator for large scale MASs (made of billions of agents). In this simulator, agents are implemented as specific entities using the simulator language. This means that a large number of agents can be simulated (we speak of numbers that are usually not manageable with JADE nor Jason, even using a simulation setup with many machines). However, the agents have to be re-implemented using the language of the simulator. This prevents user from testing actual code executed by agents of the MAS. Netlogo (http://ccl.northwestern.edu/netlogo/index.shtml accessed: 14 May 2021) is another simulator that can handle MAS applications with a high number of agents. However, similarly to DMASF, the logic of the MAS agents has to be re-implemented using the input language of the simulator. We do not provide a deeper analysis of these types of simulators since, in the rest of this paper, we will focus on simulators that can support JADE specifications.

The JREP platform [131] integrates JADE and Repast Simphony [132]. It solves some limitations of a similar approach presented in [133] that supports an older version of Repast and was limited by the focus on supply chain performance analysis and by an inefficient polling strategy affecting performances. The JREP platform offers an MAS development platform exploiting the JADE support for modelling the internal agent behaviour and the Repast support for simulating an environment where entities can interact in a simulation. JREP is a new development platform, where JADE agents need to be modified to implement new interfaces to be able to interact with the Repast environment. In this way, a bidirectional interface between the JADE agents and the Repast running environment is achieved, but the resulting JADE MAS is no longer able to run independently outside of the JREP platform.

In [134,135], the authors proposed the SAJaS API. This API has to be used with Repast Simphony to create, or improve, MAS based simulations enhanced with JADE-based features. Then, the "MAS Simulation to Development (MASSim2Dev)" code conversion tool transforms an MAS, developed using the SAJaS API, to a "standalone" standard JADE MAS. This tool is useful in scenarios where JADE developers need to perform tests in a simulator before distributing their JADE MAS. The architectural design of the JADE framework is based on Repast. However, since it is "JADE-like" environment, it is simpler to generate the JADE standard implementation. Unfortunately, MASSim2Dev can not manage the JADE blocking functions, and does not allow Ticker and Waker behaviours due to problems with time management.

4.3.2. Simulation Tools for Robots

Robotics systems also need to be extensively tested before deployment. Testing these systems is even more complex since robots interact with their physical environments through sensors and actuators. Therefore, to test these systems, simulators must provide reliable and accurate simulators between the robots and their environment. The variety of the environments in which the robots operate (e.g., very deep sea, disaster areas, no gravity scenarios and so on), makes the creation of accurate simulators even more complex.

In the following, we report a few examples of simulators present in the literature.

USARSim (Urban Search and Rescue Simulation) [136] is a general-purpose multi-robot simulator environment used as the simulation engine for the Virtual Robots Competition within the Robocup initiative, and has been often adopted in research activities. It presents an interface with Player [137] (a popular middleware used to control many different robots), and, thanks to this interface, the code developed within USARSim can be transparently moved to real platforms without any change (and vice versa). This simulator provides is relatively accurate: there is a close correspondence between results obtained within the simulation and the one obtained by the corresponding physical system.

MORSE (Modular Open Robots Simulation Engine) [138] is an open-source application that can be used in different contexts for the testing and verification of robotics systems. It is completely modular and can interact with any middleware used in robotics. In addition, it does not impose a format to which programmers must adapt. MORSE is designed to handle the simulation of several robots simultaneously, as a distributed application where the robotics software being evaluated can run on the same or a different computer as the simulation one. The evaluated components are executed on the target hardware and interact with the simulator with the very same protocols as the ones of the actual robots, sensors, and actuators, to make the shift from simulations to actual experiments transparent.

Gazebo [139] is a 3D dynamic multi-robot environment simulator. It is developed starting from the well known Player/Stage project [137], with the goal of enabling simulating dynamic outdoor environments and providing realistic sensor feedback, while still modelling robots as dynamic structures composed of rigid bodies connected via joints. The hardware simulated in Gazebo is designed to accurately reflect the behaviour of its physical counterpart: consequently, a client program shows an interface that is identical to the one that will be executed on the final robot. This makes Gazebo to be seamlessly inserted into the development process of a robotic system. Nowadays, it is largely adopted by the robotic community, and has a large and active supporting community.

4.4. Perspective of the Authors

Many SE approaches for MAS and robotic applications were proposed in the literature. However, many challenges still need to be addressed. In the following, we will discuss three of these challenges.

- *lack of clear guidance for the selection of the specification language to be used for the requirement specification.* The analysis of the paper presented in Section 4.1 showed the absence of a consensus on the strategy to be used to specify MAS requirements. Given the limited number of papers analysed (10), we cannot make any general claim on our observations, which should be confirmed by more extensive and in-depth studies. However, we believe that all the formalisms proposed in the literature for requirement specifications offer pro and cons, and that the research community should spend some effort in understanding when and how to use them and providing guidelines that can be used in research and practical works. We believe that our observations can pave the way for discussions and further studies on the requirement specification of multi-agent systems.
- *lack of mature testing tools for MAS and robotic applications.* The works summarised in Section 4.2 are some examples of SE techniques that support testing MAS and robotic applications. However, these techniques are supported by research prototypes that are still not mature enough to be used in industrial settings. Therefore, we believe

that more effort is needed to implement and develop mature tools that can be used in industrial applications.

- *lack of use of industrial simulators.* As discussed in Section 4.3, there are many simulators for JADE MASs. However, these simulators are still not ready for industrial usages. In addition, while there are many platforms for simulating robotics systems, the continuous innovation of available solutions from the robotic community (e.g., new sensors and actuators) is asking for more accurate simulators. It is also necessary to precisely document the usage scenarios and assumptions of each simulator, to enable developers to quickly find the best simulation platform for their needs. For this reason, we believe that research should work with integrating research solutions with real industrial products.

5. Human–Agent Interaction

Building reliable applications is of primary importance when agents and robots need to interact with humans. Robots or humans could be directly (negatively) affected by an agent's behaviour, e.g., when humans and agents are working together to achieve a goal.

To ensure reliable interactions, humans and agents need to anticipate each other's actions and reactions to some degree. They need to communicate and understand each other, as well as develop a shared understanding of their environment. In addition, humans need to trust the autonomous system.

We discuss three main approaches for realising reliable applications based on human–agent interaction, namely (i) building it right from the ground up, (ii) analysing existing interactions, and (iii) adjusting the users' expectations when necessary. In addition, the verification of human–robot interaction was discussed in Section 3.

5.1. Interaction Design

Agent interaction is usually based on interaction protocols. If humans are part of the system, the means of interaction between a user and an autonomous component have to be designed before the deployment of the application. Some approaches have been proposed to design human–robot interaction protocols.

Interaction Design Patterns (IDPs) [140] have been proposed to capture workable solutions for human–agent interaction. As design patterns are rather descriptive in nature, they allow for more flexibility in how they are actually implemented. In [141], five design patterns for eliciting self-disclosure are presented, as self-disclosure is an important part of getting acquainted, which leads to more trust and helps with long-term interactions. In their paper, children were the target audience. IDPs are also used in the framework of [142] to specify how to communicate explanations in order to improve the performance of mixed human–agent teams.

In [143], 18 guidelines for interactions between humans and AI are proposed from the perspective of the human–computer interaction field and evaluated through a user study. The guidelines include making clear what the system can do, how well it can do it, considering social norms, supporting efficient interaction, giving explanations, and adapting to the current user.

In [144], planning agents are used to coordinate in disaster response scenarios so that humans can choose to be guided by an agent. The authors also propose some guidelines for interaction design: agents should always be able to respond to the needs of the users, interactions should be rather simple with limited options, and interaction should enable transfer of control between the autonomous agent and its user. In [145], the authors proposed *adjustable autonomy*, which enables a (human) controller to switch unmanned aerial vehicles operation between manual control and autonomy, enabling them to supervise multiple entities at once and only assume control when necessary.

5.2. Modelling Mixed Human–Agent Systems

If a system can be modelled, it can be simulated (or even verified), it is easier to reason about it and reach a deeper understanding of the system behaviour. Modelling human–

agent interaction is a challenging endeavour. On the one hand, we have the autonomous nature of the agent, while, on the other hand, human behaviour tends to be unpredictable.

In [146], the authors proposed an approach to synthesise control protocols. An unmanned aerial vehicle and the operator are modelled as Markov Decision Processes (MDP), interacting via synchronised actions. MDPs allow modelling uncertainties regarding the behaviour of the human operator. The model is then augmented to a stochastic two-player game to account for non-determinism.

Another modelling approach for autonomous systems (e.g., autonomous driving) is proposed in [147]. They combine discrete event simulation with system dynamics models to simulate the effects of different system designs.

In [148], an entire robot swarm interacting with humans is considered. Challenges include which control type to use in which situation (assigning and controlling a leader, using the environment, controlling only parameters, or assigning behaviours), and also visualisation techniques to help the user understand a swarm's behaviour.

The modelling approaches for interaction design presented in this section are all created for a specific use case. More work is required to find general-purpose models that can be used in different domains.

5.3. Trust and Transparency

As systems get more and more autonomous, anticipating their behaviour becomes more challenging. Transparency can help users understand what their system is doing. In [149], the authors use a tiered transparency model based on situation awareness to show that more agent transparency leads to better trust calibration of the human operator without necessarily increasing their workload. This is further developed in [150], where the authors argue that bidirectional transparency is important. Thus, agents have to be designed to understand the plans, reasoning and future projections of their users.

In [151], the author argues that trust (in an autonomous system) at least requires a framework for recourse, the system's ability to give explanations, and verification and validation of the system.

In [150], the authors discuss further challenges regarding trust, namely how to quantify trust, how to model its evolution, how to create a logic that allows for specifications including trust, and also how to verify whether a system satisfies such a specification.

5.4. Behaviour Explanations

There are still many cases where full autonomy is not yet achievable. In these cases, humans have to take on a supervisory role. In these cases, humans may want to implicitly perform some kind of (mostly informal) verification of the agents' autonomous behaviour. One way of achieving this is giving autonomous systems the ability to explain their actions, their decisions and reasoning. While the previous paragraph handled how to enable users to get expectations which are more justified, some expectations might still be unreasonable. Behaviour explanations are one way of realigning these expectations, or even finding flaws in the autonomous logic of the system.

In [152], the author draws on the large body of work on explanations in the social sciences to infer properties of good explanations that humans will accept. They are contrastive (explaining why something happened instead of something else), selected (giving only relevant and important information) and social (considering the needs and background of the recipient).

In [153], a mechanism for answering "Why?"-questions about an agent's behaviour is implemented for the GWENDOLEN language. The questions are answered by identifying causal factors in the trace of the program, i.e., choice points, where another decision wouldn't have led to the result that needs to be explained.

A general perspective on explanations in AI is taken in [154] with a focus on systems incorporating machine learning. They give a formal definition of explainability, distinguishing it from interpretability and transparency of learning algorithms. Among others, they give possible reasons for making a system explainable (since each system does not

need to be, e.g., if users have no way of reacting to an explanation), possible groups of target recipients, ways to create interpretations and the relations between these questions.

In [155], the authors take another stance on explanation. They consider explanation as a model reconciliation problem. Explaining means making the mental models of the agent and the user converge, leading to a shared understanding of their world, so that the plan of the agent appears optimal.

5.5. Perspective of the Authors

Many researchers had considered human–agent interaction problems and designed solutions for these problems. However, many challenges still need to be addressed. In the following, we will discuss three of these challenges:

- *Making human–agent interaction more reliable*. There is an increasing need for making human–agent interaction more reliable. This problem can and has to be tackled from many different research angles. Interaction design, employing foremost guidelines and design patterns have laid the foundation for reliable interaction. However, reliability is mostly targeted implicitly, which leaves a need for the incorporation of an explicit notion of reliable interaction.
- *Providing modelling formalisms that effectively enable modelling human–agent systems*. Modelling human–agent systems requires new ways of specifying formerly informal concepts, such as trust, transparency, and maybe even more exotic concepts (for a machine) such as honesty and loyalty. Of course, the ability to model such systems is closely linked to being able to verify them.
- *Making systems more understandable*. Finally, making systems more understandable, e.g., by explaining them, requires many different parts coming together. In the concrete case of improving reliability, challenges include making sure users correctly understand what they are told, systems explaining their actions truthfully and users being able to verify that, or agents being able to understand why users perceive them as unreliable and act upon that.

6. Conclusions

In this perspective paper, we have discussed the applicability of agent-based programming in robotics, presented an overview of the landscape in the verification and validation of MAS and robot systems, analysed the use of software engineering in MAS, and described the latest research in human–agent interaction. Combining knowledge coming from these research areas may lead to innovative approaches that solve complex problems related to the development of autonomous robotic systems, and there is growing interest in solutions that are at the intersection of these research areas. The AREA workshop was a successful event attended by researchers working in these areas. It was an exciting event that enabled sharing ideas, open problems, and solutions and fostering cross-disciplinary collaborations among researchers. In this work, we used the papers and discussions from this workshop to analyse some of the aspects covered in this perspective.

Our perspective provided a high-level view of current research trends. We also identified a set of challenges for each of the areas we considered. For multi-agent programming, the challenges we identified include among others, the limited set of features provided by existing agent-based languages, immature methodologies and tools, and the limited integration of agent-based technologies with other techniques. For verification and validations, the challenges include the scalability of the proposed techniques and the verification of ML components. For software engineering, the challenges include the lack of clear guidance for the selection of the specification language to be used for expressing requirements, the lack of mature testing tools for MAS and robotic applications and the lack of use of industrial simulators within research works. Finally, for human–agent interactions, the challenges include the still inadequate reliability of human–agent interaction systems, the still immature modelling support for these systems, and the lack of techniques able to

make these systems more understandable. We believe that this perspective can be useful for researchers that aim at working at the intersection of these research areas.

We hope that the different research communities improve their collaboration efforts, so that the best proposals from different areas can be combined to create new and existing solutions and tools to be exploited both in academia and in industry.

Author Contributions: All authors (R.C.C., A.F., D.B., C.M. and T.A.) contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: Rafael C. Cardoso and Angelo Ferrando's work in this research was supported by UK Research and Innovation, and EPSRC Hubs for "Robotics and AI in Hazardous Environments": EP/R026092 (FAIR-SPACE), EP/R026173 (ORCA), and EP/R026084 (RAIN). Claudio Menghi is supported by the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant No 694277).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
AOSE	Agent-Oriented Software Engineering
BDD	Behaviour Driven Development
BDI	Belief-Desire-Intention
CAT	Capability Analysis Table
DSL	Domain-Specific Language
IDP	Interaction Design Pattern
MAS	Multi-Agent System
MDP	Markov Decision Process
PDDL	Planning Domain Definition Language
ROS	Robot Operating System
SE	Software Engineering

References

1. Bratman, M.E. *Intentions, Plans, and Practical Reason*; Harvard University Press: Cambridge, MA, USA, 1987.
2. Rao, A.S.; Georgeff, M. BDI Agents: From Theory to Practice. In Proceedings of the 1st International Conference Multi-Agent Systems (ICMAS), San Francisco, CA, USA, 12–14 June 1995; pp. 312–319.
3. Bordini, R.H.; Hübner, J.F.; Wooldridge, M. *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2007.
4. Boissier, O.; Bordini, R.H.; Hübner, J.F.; Ricci, A.; Santi, A. Multi-agent Oriented Programming with JaCaMo. *Sci. Comput. Program.* **2013**, *78*, 747–761, doi:10.1016/j.scico.2011.10.004.
5. Boissier, O.; Bordini, R.; Hubner, J.; Ricci, A. *Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo*; Intelligent Robotics and Autonomous Agents Series; MIT Press: Cambridge, MA, USA, 2020.
6. Dennis, L.A.; Farwer, B. Gwendolen: A BDI Language for Verifiable Agents. In *Workshop on Logic and the Simulation of Interaction and Reasoning*; AISB: London, UK, 2008.
7. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A. ROS: An open-source Robot Operating System. In Proceedings of the Workshop on Open Source Software at the International Conference on Robotics and Automation, Kobe, Japan, 12–17 May 2009.
8. Wesz, R. Integrating Robot Control into the Agentspeak(L) Programming Language. Master's Thesis, Pontificia Universidade Catolica do Rio Grande do Sul, Porto Alegre, Brazil, 2015.
9. Morais, M.G.; Meneguzzi, F.R.; Bordini, R.H.; Amory, A.M. Distributed fault diagnosis for multiple mobile robots using an agent programming language. In Proceedings of the 2015 International Conference on Advanced Robotics (ICAR), Istanbul, Turkey, 27–31 July 2015; pp. 395–400, doi:10.1109/ICAR.2015.7251486.
10. Onyedinma, C.; Gavigan, P.; Esfandiari, B. Toward Campus Mail Delivery Using BDI. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 127–143, doi:10.4204/EPTCS.319.10.

11. Onyedinma, C.; Gavigan, P.; Esfandiari, B. Toward Campus Mail Delivery Using BDI. *J. Sens. Actuator Netw.* **2020**, *9*, 56, doi:10.3390/jsan9040056.
12. Cardoso, R.C.; Ferrando, A.; Dennis, L.A.; Fisher, M. An Interface for Programming Verifiable Autonomous Agents in ROS. In *Multi-Agent Systems and Agreement Technologies*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 191–205.
13. Crick, C.; Jay, G.; Osentoski, S.; Pitzer, B.; Jenkins, O.C. Rosbridge: ROS for Non-ROS Users. In *Robotics Research: International Symposium ISRR*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 493–504.
14. Logan, B. An agent programming manifesto. *Int. J. Agent-Oriented Softw. Eng.* **2018**, *6*, 187–210.
15. Bordini, R.H.; Seghrouchni, A.E.F.; Hindriks, K.V.; Logan, B.; Ricci, A. Agent programming in the cognitive era. *Auton. Agents Multi Agent Syst.* **2020**, *34*, 37, doi:10.1007/s10458-020-09453-y.
16. Cardoso, R.C.; Ferrando, A. A Review of Agent-Based Programming for Multi-Agent Systems. *Computers* **2021**, *10*, 16, doi:10.3390/computers10020016.
17. Ziafati, P.; Dastani, M.; Meyer, J.J.; van der Torre, L. Agent Programming Languages Requirements for Programming Autonomous Robots. In *Programming Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 35–53.
18. Luckcuck, M.; Farrell, M.; Dennis, L.A.; Dixon, C.; Fisher, M. Formal specification and verification of autonomous robotic systems: A survey. *ACM Comput. Surv. CSUR* **2019**, *52*, 1–41.
19. Farrell, M.; Luckcuck, M.; Fisher, M. Robotics and integrated formal methods: Necessity meets opportunity. In *International Conference on Integrated Formal Methods*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 161–171.
20. Lomuscio, A.; Qu, H.; Raimondi, F. MCMAS: An open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.* **2017**, *19*, 9–30.
21. Kouvaros, P.; Lomuscio, A. Parameterised verification for multi-agent systems. *Artif. Intell.* **2016**, *234*, 152–189.
22. Čermák, P.; Lomuscio, A.; Murano, A. Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. In Proceedings of the AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015; Volume 29.
23. Belardinelli, F.; Lomuscio, A.; Murano, A.; Rubin, S. Verification of Multi-agent Systems with Imperfect Information and Public Actions. In Proceedings of the AAMAS, São Paulo, Brazil, 5–8 May 2017; Volume 17, pp. 1268–1276.
24. Dennis, L.; Fisher, M.; Slavkovik, M.; Webster, M. Formal verification of ethical choices in autonomous systems. *Robot. Auton. Syst.* **2016**, *77*, 1–14.
25. Dennis, L.A.; Fisher, M.; Webster, M.P.; Bordini, R.H. Model checking agent programming languages. *Autom. Softw. Eng.* **2012**, *19*, 5–63.
26. Kashi, R.N.; D'Souza, M. *Vermillion: A Verifiable Multiagent Framework for Dependable and Adaptable Avionics*; Technical Report; IIIT-Bangalore: Bengaluru, India, 2018.
27. Meyer, B.; Baudoin, C. *Méthodes de Programmation*, 1st ed.; Eyrolles: Paris, France, 1978.
28. Karoui, O.; Khalgui, M.; Koubâa, A.; Guerfala, E.; Li, Z.; Tovar, E. Dual mode for vehicular platoon safety: Simulation and formal verification. *Inf. Sci.* **2017**, *402*, 216–232.
29. Bengtsson, J.; Larsen, K.G.; Larsson, F.; Pettersson, P.; Yi, W. UPPAAL—A Tool Suite for Automatic Verification of Real-Time Systems. In *Workshop on Verification and Control of Hybrid Systems*; Springer: Berlin/Heidelberg, Germany, 1995; Volume 1066, pp. 232–243, doi:10.1007/BFb0020949.
30. Kamali, M.; Dennis, L.A.; McAree, O.; Fisher, M.; Veres, S.M. Formal verification of autonomous vehicle platooning. *Sci. Comput. Program.* **2017**, *148*, 88–106.
31. Al-Saqqar, F.; Bentahar, J.; Sultan, K.; Wan, W.; Asl, E.K. Model checking temporal knowledge and commitments in multi-agent systems using reduction. *Simul. Model. Pract. Theory* **2015**, *51*, 45–68.
32. Pecheur, C.; Raimondi, F. Symbolic Model Checking of Logics with Actions. In *Model Checking and Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4428, pp. 113–128, doi:10.1007/978-3-540-74128-2_8.
33. Cimatti, A.; Clarke, E.M.; Giunchiglia, F.; Roveri, M. NUSMV: A New Symbolic Model Checker. *Int. J. Softw. Tools Technol. Transf.* **2000**, *2*, 410–425.
34. Ancona, D.; Ferrando, A.; Mascardi, V. Parametric Runtime Verification of Multiagent Systems. In Proceedings of the AAMAS, São Paulo, Brazil, 8–12 May 2017; Volume 17, pp. 1457–1459.
35. Ferrando, A.; Ancona, D.; Mascardi, V. Decentralizing MAS Monitoring with DecAMon. In Proceedings of the Conference on Autonomous Agents and MultiAgent Systems, São Paulo, Brazil, 8–12 May 2017; ACM: New York, NY, USA, 2017; pp. 239–248.
36. Bakar, N.A.; Selamat, A. Runtime verification of multi-agent systems interaction quality. In *Asian Conference on Intelligent Information and Database Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 435–444.
37. Roungroongsom, C.; Pradubsuwun, D. Formal Verification of Multi-agent System Based on JADE: A Semi-runtime Approach. In *Recent Advances in Information and Communication Technology 2015*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 297–306.
38. Lim, Y.J.; Hong, G.; Shin, D.; Jee, E.; Bae, D.H. A runtime verification framework for dynamically adaptive multi-agent systems. In Proceedings of the International Conference on Big Data and Smart Computing (BigComp), Hong Kong, China, 18–20 January 2016; pp. 509–512.
39. Ferrando, A.; Dennis, L.A.; Ancona, D.; Fisher, M.; Mascardi, V. Verifying and Validating Autonomous Systems: Towards an Integrated Approach. In *Runtime Verification RV*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2018; Volume 11237, pp. 263–281.

40. Webster, M.; Dixon, C.; Fisher, M.; Salem, M.; Saunders, J.; Koay, K.L.; Dautenhahn, K.; Saez-Pons, J. Toward reliable autonomous robotic assistants through formal verification: A case study. *IEEE Trans. Hum.-Mach. Syst.* **2015**, *46*, 186–196.
41. Sierhuis, M.; Clancey, W.J. Modeling and Simulating Work Practice: A Method for Work Systems Design. *IEEE Intell. Syst.* **2002**, *17*, 32–41, doi:10.1109/MIS.2002.1039830.
42. Holzmann, G. *Spin Model Checker, the: Primer and Reference Manual*, 1st ed.; Addison-Wesley Professional: Boston, MA, USA, 2003.
43. Holzmann, G.J. The Model Checker SPIN. *IEEE Trans. Softw. Eng.* **1997**, *23*, 279–295, doi:10.1109/32.588521.
44. Bérard, B.; Lafourcade, P.; Millet, L.; Potop-Butucaru, M.; Thierry-Mieg, Y.; Tixeuil, S. Formal verification of mobile robot protocols. *Distrib. Comput.* **2016**, *29*, 459–487.
45. Barnat, J.; Brim, L.; Cerná, I.; Moravec, P.; Rockai, P.; Simecek, P. DiVinE—A Tool for Distributed Verification. In *Computer Aided Verification, CAV*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4144, pp. 278–281.
46. Halder, R.; Proença, J.; Macedo, N.; Santos, A. Formal verification of ROS-based robotic applications using timed-automata. In Proceedings of the IEEE/ACM FME Workshop on Formal Methods in Software Engineering (FormaliSE), Buenos Aires, Argentina, 27–27 May 2017; pp. 44–50.
47. Wang, L.; Ames, A.; Egerstedt, M. Safety barrier certificates for heterogeneous multi-robot systems. In Proceedings of the 2016 American Control Conference (ACC), Boston, MA, USA, 6–8 July 2016; pp. 5213–5218.
48. Foughali, M.; Berthomieu, B.; Dal Zilio, S.; Ingrand, F.; Mallet, A. Model checking real-time properties on the functional layer of autonomous robots. In *International Conference on Formal Engineering Methods*; Springer: Cham, Switzerland, 2016; pp. 383–399.
49. Foughali, M.; Berthomieu, B.; Dal Zilio, S.; Hladik, P.E.; Ingrand, F.; Mallet, A. Formal verification of complex robotic systems on resource-constrained platforms. In Proceedings of the IEEE/ACM International FME Workshop on Formal Methods in Software Engineering (FormaliSE), Gothenburg, Sweden, 27 May–3 June 2018; pp. 2–9.
50. Fleury, S.; Herrb, M.; Chatila, R. GenoM: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robot and Systems, Innovative Robotics for Real-World Applications, Grenoble, France, 11 September 1997; pp. 842–849, doi:10.1109/IROS.1997.655108.
51. Berthomieu, B.; Bodeveix, J.; Filali, M.; Garavel, H.; Lang, F.; Peres, F.; Saad, R.; Stöcker, J.; Vernadat, F. *The Syntax and Semantics of FIACRE*; Technical Report, Deliverable number F.3.2.11 of project TOPCASED; LAAS-CNRS: Toulouse, France, 2009.
52. Vicentini, F.; Askarpour, M.; Rossi, M.G.; Mandrioli, D. Safety assessment of collaborative robotics through automated formal verification. *IEEE Trans. Robot.* **2019**, *36*, 42–61.
53. Askarpour, M.; Mandrioli, D.; Rossi, M.; Vicentini, F. SAFER-HRC: Safety analysis through formal verification in human–robot collaboration. In *International Conference on Computer Safety, Reliability, and Security*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 283–295.
54. Pradella, M. A User's Guide to Zot. *arXiv* **2009**, arXiv:0912.5014.
55. Wang, R.; Wei, Y.; Song, H.; Jiang, Y.; Guan, Y.; Song, X.; Li, X. From offline towards real-time verification for robot systems. *IEEE Trans. Ind. Inform.* **2018**, *14*, 1712–1721.
56. Ferrando, A.; Cardoso, R.C.; Fisher, M.; Ancona, D.; Franceschini, L.; Mascardi, V. ROSMonitoring: A Runtime Verification Framework for ROS. In *Towards Autonomous Robotic Systems*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 387–399.
57. Ferrando, A.; Kootbally, Z.; Piliptchak, P.; Cardoso, R.C.; Schlenoff, C.; Fisher, M. Runtime Verification of the ARIAC Competition: Can a Robot be Agile and Safe at the Same Time? In Proceedings of the Italian Workshop on Artificial Intelligence and Robotics, Online, 25–27 November 2020; Volume 2806, pp. 7–11.
58. Sun, X.; Khedr, H.; Shoukry, Y. Formal verification of neural network controlled autonomous systems. In *ACM International Conference on Hybrid Systems: Computation and Control*; Association for Computing Machinery: Montreal, QC, Canada, 2019; pp. 147–156.
59. Askarpour, M.; Menghi, C.; Belli, G.; Bersani, M.M.; Pelliccione, P. Mind the gap: Robotic Mission Planning Meets Software Engineering. In *FormaliSE@ICSE 2020: International Conference on Formal Methods in Software Engineering*; ACM: New York, NY, USA, 2020; pp. 55–65.
60. Menghi, C.; Tsigkanos, C.; Pelliccione, P.; Ghezzi, C.; Berger, T. Specification Patterns for Robotic Missions. *IEEE Trans. Softw. Eng.* **2019**, *1*, doi:10.1109/TSE.2019.2945329.
61. Pavón, J.; Gómez-Sanz, J.J.; Fuentes, R. The INGENIAS methodology and tools. In *Agent-Oriented Methodologies*; IGI Global: Hershey, PA, USA, 2005; pp. 236–276.
62. Askarpour, M.; Rossi, M.; Tiryakiler, O. Co-Simulation of Human-Robot Collaboration: from Temporal Logic to 3D Simulation. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 1–8, doi:10.4204/EPTCS.319.1.
63. Halvari, T.; Nurminen, J.K.; Mikkonen, T. Testing the Robustness of AutoML Systems. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 103–116, doi:10.4204/EPTCS.319.8.
64. Lestingi, L.; Askarpour, M.; Bersani, M.; Rossi, M. Statistical Model Checking of Human-Robot Interaction Scenarios. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319; pp. 9–17, doi:10.4204/EPTCS.319.2.

65. Edwards, V.; McGuire, L.; Redfield, S. Establishing Reliable Robot Behavior using Capability Analysis Tables. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 19–35, doi:10.4204/EPTCS.319.3.
66. Basich, C.; Svegliato, J.; Wray, K.H.; Witwicki, S.J.; Zilberstein, S. Improving Competence for Reliable Autonomy. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 37–53, doi:10.4204/EPTCS.319.4.
67. Iotti, E.; Petrosino, G.; Monica, S.; Bergenti, F. Exploratory Experiments on Programming Autonomous Robots in Jadescript. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 55–67, doi:10.4204/EPTCS.319.5.
68. Ancona, D.; Bassano, C.; Chessa, M.; Mascardi, V.; Solari, F. Engineering Reliable Interactions in the Reality-Artificiality Continuum. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 69–80, doi:10.4204/EPTCS.319.6.
69. Smith, S.C.; Ramamoorthy, S. Semi-supervised Learning From Demonstration Through Program Synthesis: An Inspection Robot Case Study. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 81–101, doi:10.4204/EPTCS.319.7.
70. Stringer, P.; Cardoso, R.C.; Huang, X.; Dennis, L.A. Adaptable and Verifiable BDI Reasoning. In *Agents and Robots for Reliable Engineered Autonomy*; Electronic Proceedings in Theoretical Computer Science; Open Publishing Association: The Hague, The Netherlands, 2020; Volume 319, pp. 117–125, doi:10.4204/EPTCS.319.9.
71. Lami, G.; Gnesi, S.; Fabbrini, F.; Fusani, M.; Trentanni, G. *An Automatic Tool for the Analysis of Natural Language Requirements*; Technical Report; CNR Information Science and Technology Institute: Pisa, Italy, 2004.
72. Ambriola, V.; Gervasi, V. Processing natural language requirements. In Proceedings of the International Conference Automated Software Engineering, Incline Village, NV, USA, 1–5 November 1997; pp. 36–45.
73. Lazaridou, A.; Potapenko, A.; Tieleman, O. Multi-agent communication meets natural language: Synergies between functional and structural language learning. *arXiv* **2020**, arXiv:2005.07064.
74. Slhoub, K.; Carvalho, M.; Bond, W. Recommended practices for the specification of multi-agent systems requirements. In Proceedings of the IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, NY, USA, 19–21 October 2017; pp. 179–185.
75. Moreno, J.C.G.; López, L.V. Using Techniques Based on Natural Language in the Development Process of Multiagent Systems. In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 269–273.
76. Elkholy, W.; El-Menshawy, M.; Bentahar, J.; Elqortobi, M.; Laarej, A.; Dssouli, R. Model checking intelligent avionics systems for test cases generation using multi-agent systems. *Expert Syst. Appl.* **2020**, *156*, 113458, doi:10.1016/j.eswa.2020.113458.
77. Menghi, C.; Garcia, S.; Pelliccione, P.; Tumova, J. Multi-robot LTL Planning Under Uncertainty. In *Formal Methods*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 399–417.
78. Lacerda, B.; Lima, P.U. Designing petri net supervisors for multi-agent systems from LTL specifications. In *International Conference on Autonomous Agents and Multiagent Systems-Volume 3*; International Foundation for Autonomous Agents and Multiagent Systems: Taipei, Taiwan, 2011; pp. 1253–1254.
79. Ghezzi, C.; Mandrioli, D.; Morzenti, A. TRIO: A logic language for executable specifications of real-time systems. *J. Syst. Softw.* **1990**, *12*, 107–123.
80. Behrmann, G.; David, A.; Larsen, K.G. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems*; Springer: Berlin/Heidelberg, Germany; Bertinoro, Italy, 2004; pp. 200–236.
81. Menghi, C.; Tsigkanos, C.; Berger, T.; Pelliccione, P. PsALM: Specification of Dependable Robotic Missions. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), Montreal, QC, Canada, 25–31 May 2019; pp. 99–102, doi:10.1109/ICSE-Companion.2019.00048.
82. Boufaied, C.; Menghi, C.; Bianculli, D.; Briand, L.; Parache, Y.I. Trace-Checking Signal-based Temporal Properties: A Model-Driven Approach. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 21–25 September 2020; pp. 1004–1015.
83. Arcaini, P.; Mirandola, R.; Riccobene, E.; Scandurra, P. MSL: A pattern language for engineering self-adaptive systems. *J. Syst. Softw.* **2020**, *164*, 110558.
84. Musil, A.; Musil, J.; Weyns, D.; Bures, T.; Muccini, H.; Sharaf, M. Patterns for self-adaptation in cyber-physical systems. In *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 331–368.
85. Washizaki, H.; Uchida, H.; Khomh, F.; Guéhéneuc, Y.G. Studying software engineering patterns for designing machine learning systems. In Proceedings of the International Workshop on Empirical Software Engineering in Practice (IWESEP), Tokyo, Japan, 13–14 December 2019; pp. 49–495.
86. Washizaki, H.; Ogata, S.; Hazeyama, A.; Okubo, T.; Fernandez, E.B.; Yoshioka, N. Landscape of architecture and design patterns for iot systems. *Internet Things J.* **2020**, *7*, 10091–10101.
87. Garcia, A.; Silva, V.; Chavez, C.; Lucena, C. Engineering multi-agent systems with aspects and patterns. *J. Braz. Comput. Soc.* **2002**, *8*, 57–72.

88. Juziuk, J.; Weyns, D.; Holvoet, T. Design patterns for multi-agent systems: A systematic literature review. In *Agent-Oriented Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 79–99.
89. Dastani, M.; Testerink, B. Design patterns for multi-agent programming. *Int. J. Agent-Oriented Softw. Eng.* **2016**, *5*, 167–202.
90. Challenger, M.; Kardas, G.; Tekinerdogan, B. A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Softw. Qual. J.* **2016**, *24*, 755–795.
91. Challenger, M.; Demirkol, S.; Getir, S.; Mernik, M.; Kardas, G.; Kosar, T. On the use of a domain-specific modeling language in the development of multiagent systems. *Eng. Appl. Artif. Intell.* **2014**, *28*, 111–141.
92. Bauer, B.; Müller, J.P.; Odell, J. Agent UML: A formalism for specifying multiagent software systems. *Int. J. Softw. Eng. Knowl. Eng.* **2001**, *11*, 207–230.
93. García, S.; Pelliccione, P.; Menghi, C.; Berger, T.; Bures, T. High-level mission specification for multiple robots. In Proceedings of the ACM SIGPLAN International Conference on Software Language Engineering, Athens, Greece, 20–22 October 2019; pp. 127–140.
94. Brazier, F.M.T.; Dunin-Keplicz, B.; Jennings, N.R.; Treur, J. Formal Specification of Multi-Agent Systems: A Real-World Case. In Proceedings of the First International Conference on Multiagent Systems, San Francisco, CA, USA, 12–14 June 1995; The MIT Press: Cambridge, MA, USA, 1995; pp. 25–32.
95. Van Langevelde, I.; Philipsen, A.; Treur, J. Formal Specification of Compositional Architectures. In *ECAI'92: European Conference on Artificial Intelligence*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1992; pp. 272–276.
96. Bergenti, F.; Monica, S.; Petrosino, G. A scripting language for practical agent-oriented programming. In *ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 62–71.
97. Aeronautiques, C.; Howe, A.; Knoblock, C.; McDermott, I.D.; Ram, A.; Veloso, M.; Weld, D.; SRI, D.W.; Barrett, A.; Christianson, D.; et al. *PDDL | The Planning Domain Definition Language*; Technical Report; Yale Center for Computational Vision and Control: New Haven, CT, USA, 1998.
98. Giunchiglia, F.; Mylopoulos, J.; Perini, A. The tropos software development methodology: Processes, models and diagrams. In *International Workshop on Agent-Oriented Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 162–173.
99. Wooldridge, M.; Jennings, N.R.; Kinny, D. The Gaia methodology for agent-oriented analysis and design. *Auton. Agents Multi-Agent Syst.* **2000**, *3*, 285–312.
100. Tran, N.; Beydoun, G.; Low, G. Design of a peer-to-peer information sharing MAS using MOBMAS (ontology-centric agent oriented methodology). In *Advances in Information Systems Development*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 63–76.
101. Nicolescu, M.N.; Mataric, M.J. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, Melbourne, Australia, 14–18 July 2003; pp. 241–248.
102. Verstaevel, N.; Boes, J.; Nigon, J.; d'Amico, D.; Gleizes, M.P. Lifelong machine learning with adaptive multi-agent systems. In Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART 2017), Porto, Portugal, 24–26 February 2017; Volume 2, pp. pp–275.
103. Wang, X.; Klabjan, D. Competitive multi-agent inverse reinforcement learning with sub-optimal demonstrations. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 5143–5151.
104. Le, H.M.; Yue, Y.; Carr, P.; Lucey, P. Coordinated multi-agent imitation learning. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 1995–2003.
105. Bellifemine, F.L.; Caire, G.; Greenwood, D. *Developing Multi-Agent Systems with JADE*; Wiley: Hoboken, NJ, USA, 2007.
106. Coelho, R.; Kulesza, U.; von Staa, A.; Lucena, C. Unit Testing in Multi-Agent Systems Using Mock Agents and Aspects. In *SELMAS'06: International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*; ACM: New York, NY, USA, 2006; pp. 83–90, doi:10.1145/1138063.1138079.
107. Coelho, R.; Cirilo, E.; Kulesza, U.; von Staa, A.; Rashid, A.; Lucena, C. JAT: A Test Automation Framework for Multi-Agent Systems. In Proceedings of the IEEE International Conference on Software Maintenance, Paris, France, 2–5 October 2007; pp. 425–434.
108. Amaral, C.J.; Kampik, T.; Cranefield, S. A Framework for Collaborative and Interactive Agent-Oriented Developer Operations. In *AAMAS'20: Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems*; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA, 2020; pp. 2092–2094.
109. Amaral, C.J.; Hübner, J.F. Jacamo-Web is on the Fly: An Interactive Multi-Agent System IDE. In *Engineering Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 246–255.
110. Tiryaki, A.M.; Öztuna, S.; Dikenelli, O.; Erdur, R.C. SUNIT: A Unit Testing Framework for Test Driven Development of Multi-Agent Systems. In *Agent-Oriented Software Engineering VII*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 156–173.
111. Nguyen, C.D.; Perini, A.; Tonella, P. Automated Continuous Testing of MultiAgent Systems. In Proceedings of the European Workshop on Multi-Agent Systems (EUMAS), Hammamet, Tunisia, 13–14 December, 2007.
112. Nguyen, C.D.; Perini, A.; Tonella, P.; Miles, S.; Harman, M.; Luck, M. Evolutionary Testing of Autonomous Software Agents. In *AAMAS'09: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems—Volume 1*; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA, 2009; pp. 521–528.

113. Lam, D.N.; Barber, K.S. Debugging Agent Behavior in an Implemented Agent System. In *Programming Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 104–125.
114. Zhang, Z.; Thangarajah, J.; Padgham, L. Model Based Testing for Agent Systems. In *AAMAS'09: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems—Volume 2*; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA, 2009; pp. 1333–1334.
115. Padmanaban, R.; Thirumaran, M.; Suganya, K.; Priya, R.V. AOSE Methodologies and Comparison of Object Oriented and Agent Oriented Software Testing. In *ICIA-16: Proceedings of the International Conference on Informatics and Analytics*; ACM: New York, NY, USA, 2016; doi:10.1145/2980258.2982111.
116. Carrera, Á.; Iglesias, C.; Garijo, M. Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. *Inf. Syst. Front.* **2014**, *16*, 169–182, doi:10.1007/s10796-013-9438-5.
117. Braubach, L.; Pokahr, A.; Lamersdorf, W. Jadex: A BDI-Agent System Combining Middleware and Reasoning. In *Software Agent-Based Applications, Platforms and Development Kits*; Birkhäuser Basel: Basel, Switzerland, 2005; pp. 143–168.
118. Huang, Z.; Alexander, R.; Clark, J. Mutation Testing for Jason Agents. In *Engineering Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 309–327.
119. Benac Earle, C.; Fredlund, L.Å. A Property-Based Testing Framework for Multi-Agent Systems. In *AAMAS'19: Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems*; International Foundation for Autonomous Agents and Multiagent Systems: Richland, SC, USA, 2019; pp. 1823–1825.
120. Claessen, K.; Hughes, J. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. *SIGPLAN Not.* **2000**, *35*, 268–279, doi:10.1145/357766.351266.
121. Briola, D.; Mascardi, V.; Ancona, D. Distributed Runtime Verification of JADE and Jason Multiagent Systems with Prolog. In Proceedings of the Conference on Computational Logic, Torino, Italy, 16–18 June 2014; Volume 1195, pp. 319–323.
122. Ancona, D.; Briola, D.; Ferrando, A.; Mascardi, V. MAS-DRiVe: A Practical Approach to Decentralized Runtime Verification of Agent Interaction Protocols. In Proceedings of the Workshop "From Objects to Agents" Co-Located with 18th European Agent Systems Summer School (EASSS 2016), Catania, Italy, 29–30 June 2016; Volume 1664, pp. 35–43.
123. Mascardi, V.; Ancona, D. Attribute Global Types for Dynamic Checking of Protocols in Logic-based Multiagent Systems. *Theory Pract. Log. Program.* **2013**, *13*, 4–5.
124. Mascardi, V.; Briola, D.; Ancona, D. On the Expressiveness of Attribute Global Types: The Formalization of a Real Multiagent System Protocol. In *AI*IA 2013: Advances in Artificial Intelligence—XIIIth International Conference of the Italian Association for Artificial Intelligence*; Springer:Berlin/Heidelberg, Germany, 2013; Volume 8249, pp. 300–311.
125. Ancona, D.; Briola, D.; Ferrando, A.; Mascardi, V. Runtime verification of fail-uncontrolled and ambient intelligence systems: A uniform approach. *Intell. Artif.* **2015**, *9*, 131–148, doi:10.3233/IA-150084.
126. Dennis, L.A. The MCAPL Framework including the Agent Infrastructure Layer and Agent Java Pathfinder. *J. Open Source Softw.* **2018**, *3*. doi:10.21105/joss.00617.
127. Mengistu, D.; Tröger, P.; Lundberg, L.; Davidsson, P. Scalability in Distributed Multi-Agent Based Simulations: The JADE Case. In Proceedings of the Second International Conference on Future Generation Communication and Networking Symposia, Hinan, China, 13–15 December 2008; Volume 5, pp. 93–99, doi:10.1109/FGCNS.2008.158.
128. Lo Piccolo, F.; Bianchi, G.; Salsano, S. Measurement Study of the Mobile Agent JADE Platform. In Proceedings of the International Symposium on on World of Wireless, Mobile and Multimedia Networks, Buffalo-Niagara Falls, NY, USA, 26–29 June 2006; pp. 638–646.
129. Briola, D.; Micucci, D.; Mariani, L. A platform for P2P agent-based collaborative applications. *Softw. Pract. Exp.* **2019**, *49*, 549–558, doi:10.1002/spe.2657.
130. Aprameya Rao, I.V.; Jain, M.; Karlapalem, K. Towards Simulating Billions of Agents in Thousands of Seconds. In *AAMAS'07: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*; ACM: New York, NY, USA, 2007.
131. Gormer, J.; Homoceanu, G.; Mumme, C.; Huhn, M.; Muller, J.P. JREP: Extending Repast Simphony for JADE Agent Behavior Components. In Proceedings of the IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, Lyon, France, 22–27 August 2011; Volume 2, pp. 149–154, doi:10.1109/WI-IAT.2011.120.
132. North, M.; Howe, T.; Collier, N.; Vos, J. Repast Simphony runtime system. In Proceedings of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms, Chicago, IL, USA, 13–15 October 2005.
133. Yoo, M.J.; Glardon, R. Combining JADE and Repast for the Complex Simulation of Enterprise Value-Adding Networks. In *Agent-Oriented Software Engineering IX*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 243–256.
134. Cardoso, H.L. SAJaS: Enabling JADE-Based Simulations. In *Transactions on Computational Collective Intelligence XX*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 158–178, doi:10.1007/978-3-319-27543-7_8.
135. Lopes, J.; Cardoso, H. From simulation to development in MAS a JADE-based approach. In Proceedings of the ICAART—International Conference on Agents and Artificial Intelligence, Lisbon, Portugal, 10–12 January 2015; Volume 1, pp. 75–86.
136. Carpin, S.; Lewis, M.; Wang, J.; Balakirsky, S.; Scrapper, C. USARSim: A robot simulator for research and education. In Proceedings of the IEEE International Conference on Robotics and Automation, Rome, Italy, 10–14 April 2007; pp. 1400–1405.
137. Brian P. Gerkey, R.T.V.; Howard, A. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In Proceedings of the International Conference on Advanced Robotics, Coimbra, Portuga, 30 June–3 July 2003; pp. 317–323.

138. Echeverria, G.; Lassabe, N.; Degroote, A.; Lemaignan, S. Modular open robots simulation engine: MORSE. In Proceedings of the IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 46–51.
139. Koenig, N.; Howard, A. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, 28 September–2 October 2004; pp. 2149–2154.
140. Kahn, P.H.; Freier, N.G.; Kanda, T.; Ishiguro, H.; Ruckert, J.H.; Severson, R.L.; Kane, S.K. Design patterns for sociality in human–robot interaction. In Proceedings of the ACM/IEEE International Conference on Human Robot Interaction, Amsterdam , The Netherlands, 12–15 March 2008; pp. 97–104.
141. Ligthart, M.; Fernhout, T.; Neerincx, M.A.; van Bindsbergen, K.L.; Grootenhuis, M.A.; Hindriks, K.V. A child and a robot getting acquainted-interaction design for eliciting self-disclosure. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, Montreal, QC, Canada, 13–17 May 2019; pp. 61–70.
142. Neerincx, M.A.; van der Waa, J.; Kaptein, F.; van Diggelen, J. Using perceptual and cognitive explanations for enhanced human–agent team performance. In *International Conference on Engineering Psychology and Cognitive Ergonomics*; Springer:Berlin/Heidelberg, Germany, 2018; pp. 204–214.
143. Amershi, S.; Weld, D.; Vorvoreanu, M.; Fourney, A.; Nushi, B.; Collisson, P.; Suh, J.; Iqbal, S.; Bennett, P.N.; Inkpen, K.; et al. Guidelines for human-AI interaction. In Proceedings of the Chi Conference on Human Factors in Computing Systems, Glasgow, UK, 4–9 May 2019; pp. 1–13.
144. Ramchurn, S.D.; Wu, F.; Jiang, W.; Fischer, J.E.; Reece, S.; Roberts, S.; Rodden, T.; Greenhalgh, C.; Jennings, N.R. Human–agent collaboration for disaster response. *Auton. Agents Multi-Agent Syst.* **2016**, *30*, 82–111.
145. Orsag, M.; Haus, T.; Tolić, D.; Ivanovic, A.; Car, M.; Palunko, I.; Bogdan, S. Human-in-the-loop control of multi-agent aerial systems. In Proceedings of the 2016 European Control Conference (ECC), Aalborg, Denmark, 29 June–1 July 2016; pp. 2139–2145.
146. Feng, L.; Wiltsche, C.; Humphrey, L.; Topcu, U. Synthesis of human-in-the-loop control protocols for autonomous systems. *IEEE Trans. Autom. Sci. Eng.* **2016**, *13*, 450–462.
147. Cummings, M.; Clare, A. Holistic modelling for human-autonomous system interaction. *Theor. Issues Ergon. Sci.* **2015**, *16*, 214–231.
148. Kolling, A.; Walker, P.; Chakraborty, N.; Sycara, K.; Lewis, M. Human interaction with robot swarms: A survey. *IEEE Trans. Hum.-Mach. Syst.* **2015**, *46*, 9–26.
149. Selkowitz, A.; Lakhmani, S.; Chen, J.Y.; Boyce, M. The effects of agent transparency on human interaction with an autonomous robotic agent. In *Human Factors and Ergonomics Society Annual Meeting*; SAGE Publications Sage CA: Los Angeles, CA, USA, 2015; Volume 59, pp. 806–810.
150. Schaefer, K.E.; Straub, E.R.; Chen, J.Y.; Putney, J.; Evans, A.W., III. Communicating intent to develop shared situation awareness and engender trust in human–agent teams. *Cogn. Syst. Res.* **2017**, *46*, 26–39.
151. Winikoff, M. Towards trusting autonomous systems. In *International Workshop on Engineering Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 3–20.
152. Miller, T. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.* **2019**, *267*, 1–38.
153. Koeman, V.J.; Dennis, L.A.; Webster, M.; Fisher, M.; Hindriks, K. The "Why did you do that?" Button: Answering Why-questions for end users of Robotic Systems. In *International Workshop on Engineering Multi-Agent Systems*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 152–172.
154. Rosenfeld, A.; Richardson, A. Explainability in human–agent systems. *Auton. Agents Multi-Agent Syst.* **2019**, *33*, 673–705.
155. Chakraborti, T.; Sreedharan, S.; Zhang, Y.; Kambhampati, S. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. In Proceedings of the International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; pp. 156–163.