Autonomous Agents and Multi-Agent Systems (2021) 35:1
https://doi.org/10.1007/s10458-020-09478-3

Logic-based technologies for multi-agent systems: a systematic literature review

Roberta Calegari[1] · **Giovanni Ciatto**[2] · **Viviana Mascardi**[3] · **Andrea Omicini**[2]

Abstract
Precisely when the success of artificial intelligence (AI) sub-symbolic techniques makes them be identified with the whole AI by many non-computer-scientists and non-technical media, symbolic approaches are getting more and more attention as those that could make AI amenable to human understanding. Given the recurring cycles in the AI history, we expect that a revamp of technologies often tagged as "classical AI"—in particular, *logic-based* ones—will take place in the next few years. On the other hand, agents and *multi-agent systems* (MAS) have been at the core of the design of intelligent systems since their very beginning, and their long-term connection with *logic-based technologies*, which characterised their early days, might open new ways to engineer *explainable intelligent systems*. This is why understanding the current status of *logic-based technologies for MAS* is nowadays of paramount importance. Accordingly, this paper aims at providing a comprehensive view of those technologies by making them the subject of a *systematic literature review* (SLR). The resulting technologies are discussed and evaluated from two different perspectives: the MAS and the logic-based ones.

Keywords SLR · Logic-based technologies · MAS

✉ Roberta Calegari
roberta.calegari@unibo.it

Giovanni Ciatto
giovanni.ciatto@unibo.it

Viviana Mascardi
viviana.mascardi@unige.it

Andrea Omicini
andrea.omicini@unibo.it

[1] ALMA–AI Interdepartmental Center of Human Centered AI, Alma Mater Studiorum–Università di Bologna, Bologna, Italy

[2] Dipartimento di Informatica – Scienza e Ingegneria (DISI), Alma Mater Studiorum–Università di Bologna, Bologna, Italy

[3] Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS), Università di Genova, Genoa, Italy

Published online: 19 October 2020

1 Introduction

While discussions about the balance between risks and benefits of artificial intelligence (AI) take a significant space in public interest, industry seems to have finally joined (and, maybe, surpassed) academia in valuing AI as one of the pillars of the next industrial revolution. Public institutions have followed, too: for instance, the European Commission already invested 1.5 billion euros in AI for 2018–2020, and many more are planned beyond 2020.[1] Furthermore, the "American Artificial Intelligence Initiative: Year One Annual Report", delivered on February 2020 [334], witnesses the record amounts of AI research and development investment, and the same trend is observed in China.[2]

When industry, politics, and society focus on new technologies, the promise of a future astonishing progress is no longer enough, since industry leaders, decision makers, and tax payers call for immediate and visible benefits. That is why, following the recent success of deep learning, and the reviving popularity of many other AI techniques, in the next decade academia is going to devote non-negligible research efforts to *intelligent systems*, and in particular to *intelligent system engineering*.

Given the current status of AI technologies—mostly successful in well-delimited application scenarios—, a key issue for intelligent system engineering is *integration* of the diverse AI techniques: in software engineering terms, not just how to integrate diverse technologies, but also (perhaps mainly) how to preserve *conceptual integrity* when highly-heterogeneous approaches—bringing about manifold abstractions of various nature—are put to work together.

The most straightforward and generally-acknowledged way to address the above issue is by using *agents* and *multi-agent systems* (MAS). In fact, this is the way in which AI has been taught worldwide in the last decades, based on the most used AI textbook [305]: there, the agent abstraction is adopted as the conceptual architecture to frame the many different AI techniques, and MAS abstractions provide sound foundation for the design of intelligent systems. Also, the long-term research activity in fields like agent-oriented computing (AOC)—about MAS programming languages and technologies—and agent-oriented software engineering (AOSE)—about MAS methods, tools, and methodologies—actually makes agents and MAS the most suitable candidates to work as the sources for well-founded technologies and methodologies for intelligent system engineering.

Nowadays, a clear dichotomy is emerging from within the AI area. Whereas the overwhelming success of sub-symbolic techniques like deep learning has basically wiped out most objections against AI, public concern about the forthcoming role of intelligent systems in human society has raised new issues, such as the need of explaining the behaviour of intelligent agents, and make it understandable for humans—particularly when AI plays a critical role within human organisations. The fact is, the overall approach to XAI (eXplainable Artificial Intelligence [28] would seemingly require the extensive adoption of symbolic techniques (possibly integrated with sub-symbolic and non-symbolic ones) in order to reach its main goals—such as observability, interpretability, explainability, and accountability. As a result, exactly when the general focus of AI is on sub-symbolic techniques—as those that have made AI work in real-world applications—, symbolic approaches are beginning to be carefully scrutinised—as those that could make AI also amenable to human understanding.

[1] https://ec.europa.eu/digital-single-market/en/artificial-intelligence, last accessed in April 2020.

[2] https://www.technologyreview.com/2019/12/05/65019/china-us-ai-military-spending/, last accessed in April 2020.

Among them, *logic-based techniques* are not just the oldest ones, but possibly represent the most straightforward path towards human understanding: if for no other reason, for the fact that logic—as the study of correct reasoning—targets first of all human cognitive processes. Also, logic-based approaches are since long time at the core of many successful agent-based models and technologies. Generally speaking, cognitive agents in intelligent MAS straightforwardly exploit logic-based models and technologies for rational process, knowledge representation, expressive communication, and effective coordination. Overall, MAS and logic-based technologies have already crossed their paths in the last decades, and can be expected to do so more and more in the years to come.

Altogether, this is why this paper focuses on logic-based approaches in MAS: as they are to be counted among the most promising techniques for building understandable and explainable intelligent systems. More precisely, given the unavoidable push towards the engineering of intelligent systems, here we explicitly target logic-based *technologies*: we mean to understand and represent the current status of the available logic-based technologies in the MAS field that are actually usable in the engineering of intelligent systems.

Of course, we acknowledge that other papers in the past have already (and often brilliantly) surveyed MAS technologies as well as logic-based approaches within MAS. What motivates a new study on the subject, however, it is not just the renewed interest and public investment on AI, or, our focus on technologies. Instead, to the best of our knowledge, no systematic review has been performed on the subject to date. And, the requirement of verifiability and reproducibility of scientific literature can be satisfied by SLR (systematic literature review) only—particularly nowadays, when AI research is more and more subject to the public scrutiny, precisely for the overwhelming interest it has raised. Then, the goal of this paper is to provide an exhaustive and organised view of the available logic-based technologies for MAS, by performing a carefully-designed and implemented SLR on the subject.

Accordingly, the paper is organised as follows. Section 2 introduces the method used for the SLR, discussing the technical details from the fundamental research questions down to the technical operations actually leading to the material discussed in the paper. Section 3 presents and organises the technologies devised out according to a MAS-based perspective, and provides a brief description for each one. Section 4 collects and displays information from the sources and the selected technologies in form of tables and word clouds. Section 5 provides for an overall view on logic-based technologies for MAS, offering some interpretation criteria on their current state of the art and perspectives. Lastly, Sect. 6 concludes the paper with some final remarks.

2 Method

2.1 Premises

Making a review *systematic* requires first of all that the main concepts are defined clearly enough not just to make its scope precise, but mostly to make the review itself reproducible—to some extent. This does not mean of course that notions like agents and multi-agent systems are to be re-defined here *ex novo*, or, with more detail than is typically used in the literature: but, at least, it means that the acceptations of terms used in the remainder of this paper have to be well-founded and understandable.

So, when including/excluding MAS technologies in/from our survey, we refer to the well-known definition of agents by Wooldridge and Jennings [356], or, to a basically-equivalent

definition of agents and MAS [280]: agents are computational abstractions encapsulating control along with a criterion to drive control (a task, a goal), whereas MAS are collections of agents interacting (communicating, coordinating, competing, cooperating) in a computational system. Even less problematic is to define what is logic: dating back to Aristotle—whose logic is the instrument for human knowledge [301]—logic studies the way we draw conclusions and express ourselves, and deals with how to formalise it [256]. Since several sorts of logic exists and are adopted in general in the broad context of computational logic, in the following we stick to those logics clearly identifiable in the literature, in terms of name, formal definition, and reference paper(s).

Yet, the main issue here does not concern either MAS or logic. Instead, it comes from our basic choice to focus on logic-based *technologies* for MAS, and from its motivation as well. In fact, a key point is for agent-based technologies to actually *work* in the engineering of intelligent systems: correspondingly, in the remainder of this survey we choose to include only logic-based MAS technologies that are both scientifically *identifiable* and technically *available*. This means, first of all, that there must exist (at least) a scientific publication that clearly defines and describes the technology (and so, covering both its agent- and logic-based aspects); then, that the technology itself should be clearly mentioned in the reference publication, as well as technically accessible (e.g., from a web site or a public repository) at the time of the survey.

More precisely, in this paper we consider as *logic-based MAS technology* any agent-oriented software architecture, framework, or language that (i) involves some clearly-defined logic model, and (ii) comes with some actual technological reification. Indeed, we argue that technological aspects are far from being marginal, as technology is "the application of scientific knowledge for practical purposes",[3] and now more than ever the engineering of intelligent systems is in dire need of dependable technologies. Clearly enough, technologies may involve logics at several different levels and in disparate ways, in the same way as they may address diverse abstractions of MAS. Within the scope of our work, in particular, we are mostly interested in technologies *exposing* logic and agent-orientation at the application level—i.e., making logic explicit to their intended users. Consider for instance Jason [42], a well-known MAS specimen of technology inspired by the Belief-Desire-Intention (BDI) architecture [295–297]—and hence by the BDI logic—, written in Java. There, BDI logic pervades Jason design, implementation, and API, and is clearly exposed to Jason users. Similarly, Golog [228]—a Prolog-based technology for MAS programming—would have been considered to be a logic-based technology even if it were written in Java, as logic is required by users to build MAS through Golog.

Accordingly, in the remainder of this survey we choose to include only those logic-based technologies for MAS that (i) can be associated with a sound scientific contribution describing some clearly-identifiable software system, framework, or toolkit; (ii) are in the form of executable software, which can be actually reached, downloaded, and tested on some modern execution platform—so that they may actually be used and work in real-world intelligent systems, at least in principle; (iii) can be found via URLs or references from their primary works, or, straightforwardly via web search on the authors' information—otherwise, any link between the model from the primary work and the technology would be debatable, and possibly arbitrary; (iv) the URL does not belong to some web archive service only—which would clearly witness the fact that the technology is in some state of abandon.

[3] https://www.oxfordreference.com/page/scienceandtech/science-and-technology, last accessed in July 2020.

2.2 Research questions

Every SLR [56,263] is built around some fundamental research question. According to Higgins and Green [176], "A *systematic review* attempts to identify, appraise and synthesise all the empirical evidence that meets pre-specified eligibility criteria to answer a given research question. Researchers conducting systematic reviews use explicit methods aimed at minimising bias, in order to produce more reliable findings that can be used to inform decision making."

Our primary research question essentially expresses our main research goal here. In fact, we are interested in understanding

(G) What is the role played by logic-based technologies in MAS nowadays?

In order to make it possible to conduct a SLR based on this (quite general and possibly generic) question, it is necessary to re-articulate the research goal in terms of more detailed (secondary) research questions:

(Q1) Which logic-based technologies for MAS are actually available?
(Q2) Which aspects of MAS technologies are affected by logic-based technologies, and to what extent?
(Q3) Which MAS abstractions/issues/features are covered by logic-based technologies?
(Q4) Which sorts of logics are actually exploited to frame logic-based MAS technologies?
(Q5) Which logic-based technologies for MAS are effectively used in real-world domains?

The above questions drive the process of search, by articulating the way in which papers and technologies are looked for, examined, and possibly selected for the review.

2.3 Sources

The electronic sources used in the search process are of two sorts: (i) general *bibliographic databases* and (ii) *specific sources*—such as conferences, symposia, and workshops—on the core topics. The former are searched *intensively* via specific queries, whereas the latter are explored *extensively*, year by year, volume by volume, paper by paper. This is because small, specialised workshops are often missing from the major bibliographic databases used for our query-based intensive search; instead, one can typically count on the complete indexing of major conferences and international journals. Accordingly, the extensive scrutiny of all the papers published in the specialised workshops on topics near to MAS and logic is mostly intended to *complement* the intensive search performed throughout the bibliographic engines, so as to result in the most exhaustive search possible overall.

In particular, in the following we extensively consider all the papers published in

- the International Workshop on Declarative Agent Languages and Technologies (DALT)—10 events (2003–2012)
- the International Workshop on Engineering Multi-Agent Systems (EMAS)—7 events (2013–2019), including DALT since 2013
- the International Workshop on Computational Logic in Multi-Agent Systems (CLIMA)—15 events (2000–2014)
- the Workshop on Logics for Intelligent Agents and Multi-agent Systems (WLIAMAS @ IAT)—3 events (2008–2010)
- the Workshop on Logical Aspects of Multi-Agent Systems (LAMAS)—8 events (2002, 2010–2015, 2017)

and check them for the required content and references. The aforementioned sources have to be searched extensively given that their coverage by the most common sources for scientific literature is often incomplete at best.

In detail, the sources used for intensive search (via keyword-based queries) are the following:

(i) Google Scholar—http://scholar.google.com
(ii) IEEE Xplore—http://ieeexplore.ieee.org
(iii) ScienceDirect—http://www.sciencedirect.com
(iv) SpringerLink—http://www.springerlink.com
(v) DBLP—http://dblp.uni-trier.de
(vi) ACM Digital Library—http://dl.acm.org.

Each of these sources is queried with the following combinations of keywords:

(KW1) multi agent system logic technology
(KW2) multi agent system logic implementation
(KW3) multi agent system logic industry
(KW4) multi agent system logic verification
(KW5) multi agent system logic agent programming
(KW6) multi agent system logic agent reasoning
(KW7) multi agent system logic society
(KW8) multi agent system logic environment

and the first hundred results returned are considered as potential candidates: the occurrence of a logic-based approach for MAS is first checked for each paper, then, in case it is found, the existence of a technology is looked for and possibly verified.

2.4 Papers classification

Papers resulting from the above search activity are classified according to three main classes:

primary papers—i.e., those papers *introducing* and describing a technology of interest
auxiliary papers—i.e., those papers *mentioning* a technology of interest—meaning that the mentioned technology has been employed in some way in the development of the contribution of the paper
secondary papers—i.e., those papers *surveying* technologies of interest.

Given their obvious relevance to a systematic survey, secondary papers are not only used as the direct source for technologies of interest; instead, their extensive and organised bibliographic references is recursively searched for new possible items of interest. In particular, we extensively search references from the following secondary works: [23,44,147,245,246,248,277,307,357].

It is worth pointing out that the aforementioned secondary works are similar in purpose to this work, yet they are not systematic literature reviews. This work, in turn, has a wider scope, an updated background, and it is based on a structured and reproducible method. Furthermore, in this paper, we provide a technological assessment of each technology with an unprecedented level of detail. In particular, Baldoni et al. [23] provides a historical overview of the relationship between declarative languages and multi-agent systems, with no ambition of completeness, and no interest in the analysis of technologies. Conversely, this paper aims at understanding and representing the current state of the available logic-based technologies for MAS: so, our SLR focuses on those technologies that are actually usable in the engineering of intelligent systems.

3 Technology synopsis: the MAS perspective

Agents are not alone in MAS. They interact with other agents, and with the surrounding environment as well: a shared view exists in the literature that agent, society, and environment can be taken as the three basic categories to interpret and model the structure and dynamics of non-trivial MAS. This view, proposed by Omicini [273] and backed up—either implicitly or explicitly—by other MAS models—e.g., [286,350]—, infrastructures—e.g., [298,344]—, and methodologies—e.g., [91,104,166]—drives the organisation of this section. Indeed, the works that meet the selection criteria discussed in Sect. 2, are presented according to the following schema:

- Agent
  - Programming, reasoning & planning
  - Agent reliability & verification
- Society
  - Organisational reasoning & coordination
  - Argumentation
  - Reliability & verification of MAS
- Environment
  - Agents in the Semantic Web (distributed cognition)
  - Situated interaction & coordination

Technologies belonging to each sub-category are presented in chronological order and accompanied by a short description, including *primary references*, *URL* referencing the projects home pages, and other *related papers*, if any. In particular, each technology is described from two different yet complementary perspectives: the MAS and the logical ones. On the one side, the MAS perspective aims at providing an answer to research questions (Q2) and (Q3). On the other side, the logical perspective aims at answering to research question (Q4). Furthermore, each technology description also includes a brief technical assessment aimed at addressing research questions (Q1), and (Q5).

Many technologies are based on, or integrate, more than one logic-based approach. In those situations, we classify a technology according to the "most distinguishing" approach—i.e., the approach for which the technology is mainly known and referenced for in the literature. For instance, let us consider again Jason as an example. To support agent programmers in modelling agent beliefs and deductive reasoning mechanisms, Jason includes a full-fledged Prolog interpreter. Prolog interpreters are considered the main software tool within the scope of logic programming. However, Jason is rarely described as a technology based on logic programming (LP) [234], despite being technically able to handle logic programs. Indeed, the BDI paradigm is far more characterising for Jason than LP: hence, we categorise Jason as a BDI logic technology.

3.1 Agent

From the MAS perspective, most technologies falling under the "Agent" category, are *logic-based agent programming languages*. Whereas almost all these languages allow for the specification and implementation of societal aspects—via communication actions—and interaction with the environment—via perception and external actions—, their focus is on

the description of individual agents, which motivates the inclusion in this section. These technologies are listed in the "Programming, reasoning, and planning" sub-category, and include languages for agents characterised by beliefs (and, depending on the language, other mentalistic notions such as desires, goals, plans, commitments)—such as ASTRA, AGENT-0, AgentSpeak(L) and its Jason implementation, DyLOG, AFAPL, GOAL, JACK, Jadex, eXAT, Rodin, JADL, 3APL and 2APL, TeleoR and Golog-like languages; languages where deliberation is the most characterising feature—such as SHOP, AgentSpeak(ER); languages that extend logic programming with communication, concurrent execution and/or time management—such as CaseLP/DCaseLP, Concurrent MetateM, DALI, Go!, Mozart/Oz. We also include in this sub-category RASP, because of its attention to modelling agent knowledge and reasoning, jDALMAS, which extends the reasoning mechanism with deontic abstractions, and DRAGO, which bases agent knowledge and reasoning on ontologies.

Finally, it is widely acknowledged that many areas of application of MAS clearly require dependable computational systems—and, in particular, that such systems should be verifiable. The last sub-category under the agent perspective is hence devoted to technologies for the (automatic or semi-automatic, static or dynamic) verification of agent-based systems. In the "Agent reliability & verification" sub-category we discuss only one technology, MCAPL, which is more oriented (although not exclusively oriented) towards model checking of individual agents. More technologies are presented in the societal extension of this category.

From the logic perspective, the technologies in the "Agent" category largely emphasise the impact of logic programming on agents specification. In fact, most of them either extend or rely on LP to describe agents behaviours, beliefs, or semantics. This is unsurprising, as the connection between agents and LP has been well understood in the literature since the dawn of the agent-oriented programming research. From the seminal works by Kowalski [212], Dell'Acqua et al. [116], Kowalski and Sadri [213], Bozzano et al. [49] up to the recent work by Calegari et al. [61], logic programming is considered by many scientists as quite a natural choice for specifying, implementing, and verifying rational agents. Among the dozens of surveyed works exploiting LP and its variants, only some of them respect the strict acceptation of (available) technology adopted here.

3.1.1 Programming, reasoning, and planning

AGENT-0 (1991)

Primary references: [318,319]
Other references describing / exploiting the technology: [180,336,357]
URL: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/agents/aop/0.html

MAS perspective AGENT-0 is a framework for programming MAS. Programming an agent in AGENT-0 amounts at defining its commitment rules which are decisions to perform an action, not to pursue a goal. The conditions under which a commitment can be made are expressed in terms of mental conditions—involving what the agent believes to be true and which commitments it believes to hold—, and message conditions—namely logical combinations of message patterns `(From, Type, Content)`. There, `Type` may be one among some predefined performatives, as `INFORM`, `OFFER`, `REQUEST`. Agents communicate through message passing and an agent capability is a relation between that agent mental state and its environment. Complex mental categories like desires, goals, intentions and plans are not supported by the language.

Logical perspective AGENT-0 is based on a particular, ad-hoc, modal logic. Modal logic [70,134] is a suitable and powerful formalism to describe agents as intentional systems, as observed by Wooldridge and Jennings [356] in the early 90s. More precisely, AGENT-0 is an interpreted language based on a propositional temporal logic developed by Thomas et al. [336]. It is a simple point-based temporal language, with facts associated with points in time and possibly nested time-stamped beliefs. The logical language has modal operators representing knowledge, belief, desire, commitment, and ability. According to Wooldridge et al. [357], it represents the foundation for the AGENT-0 semantics: however, it is worth noting that AGENT-0 temporal operators lack a formal logic-based semantics. Given that AGENT-0 has no formal semantics, it is not surprising that also the temporal component of the language is presented in an intuitive, informal way.

Technological notes: The AGENT-0 technology is still available, even if it dates back to 1991. It consist of a Lisp [252] module, originally intended for the Allegro Common Lisp system, and which does not include license information. The Lisp module can still be loaded on modern Common Lisp platforms. However, following the provided instructions, a runtime error prevents its actual execution.

Concurrent MetateM (1993)

Primary references: [142,146]
Other references describing / exploiting the technology: [29,46,144,145,180]
URL: http://cgi.csc.liv.ac.uk/~michael/TLBook/MetateM-System

MAS perspective MetateM is a concrete framework for programming MAS through modal and temporal logics. Concurrent MetateM is the multi-agent extension of MetateM: it combines the features supported by MetateM with a model of concurrent computation, providing an approach to the modelling of complex reactive systems. A Concurrent MetateM system contains a number of concurrently executing agents which communicate through message passing.

Logical perspective MetateM interpreter implements an "on the basis of the past, do the future" behaviour based upon the methodology of temporal logic as an executable imperative language presented by Moszkowski [259]. The rules which drive the agent behaviour are temporal logic formulae of the form: "antecedent about the past, consequent about the future". The methodology finds its root in the work presented by Gabbay [155], stating that any temporal logic formula can be rewritten in terms of one or more $Antecedent \rightarrow Consequent$ rules. These rules make up the agent specifications and dictate the agent behaviours and, consequently, the MAS dynamics.

Technological notes: Despite the many implementations discussed for MetateM in the literature, we were able to find only an old, not maintained, Java-based implementation of Concurrent MetateM that requires Java Virtual Machine (JVM) 6+. It is distributed under the terms of the GNU General Public License, together with some code examples, which we used to test it. The available implementation still works, even on the most recent version of the JVM (https://www.java.com/, last accessed in April 2020).

AgentSpeak(L) (1996) & Jason (2006)

Primary references: [42,294]
Other references describing / exploiting the technology: [42,45,106,201,207,255,258,284,299]
URL: http://jason.sourceforge.net

MAS perspective Jason is an interpreter for an extended version of AgentSpeak(L), a logic-based agent-oriented programming language introduced by Rao [294] and based on the Beliefs-Desires-Intentions (BDI) architecture and logic presented by Rao and Georgeff [295–297]. Through Jason, developers can model and implement autonomous agents in a goal-oriented way, by writing the plans an agent may choose to adopt when reacting to the many events steering the dynamics of a MAS. All that Jason agents can do occurs due to events, including perception, belief-base update, and message-passing, as well as goal instantiation or satisfaction. The extended AgentSpeak(L) language exploited by Jason essentially provides a practical means of manipulating agent knowledge and activities through a Prolog-like syntax.

Logical perspective The Jason technology is based on the BDI logic. Furthermore, the Jason interpreter comes with an internal, Prolog-like, logic programming engine implemented in Java—supporting *annotated* logic terms, among the other features. The Jason operational semantics has been defined by Vieira et al. [343]. Based on the BDI logics, Jason expects agents to be defined in terms of beliefs—i.e., the knowledge an agent has about the world—, desires—i.e., the states of the world an agent is willing to pursue or maintain—, and intentions—i.e., the actions in which an agent means to be involved in a given moment. Notice that the Jason technology is aimed neither at formally inspecting MAS nor at automatically deriving MAS properties. Conversely, BDI logic is exploited as an effective programming language for AOP.

Technological notes: Jason is an open source technology that requires JVM 8+. It is available under the GNU Lesser General Purpose license at http://jason.sourceforge.net. It is alive and actively developed, as highlighted by the intense activity on its official GitHub repository and by the many resources available on its homepage. It is also available as a Maven artifact, along with its dependencies, thus giving proof of its maturity as a JVM technology. We tested Jason exploiting the many examples available on the project homepage, showing that the technology works even with the most recent versions of JVM.

Golog (1997), ConGolog (2000) & IndiGolog (2009)

Primary references: [113,114,228]
Other references describing / exploiting the technology: [46,117,245,254,283]
URL: http://www.cs.toronto.edu/cogrobo/main/systems

MAS perspective In Golog-like programming languages, the agent dynamic world can be modelled by exploiting an explicit representation of the initial state of the system, and the effect of actions on that world. The representation is characterised by axioms provided by users. Golog-like languages allow the developer to program autonomous agents that perceive their environment and plan while operate. ConGolog provides support for concurrent programming, while IndiGolog specialises in providing agents with the ability to sense their environment and plan accordingly.

Logical perspective Both Golog and its successors extend or are implemented in Prolog (SWI-Prolog,[4] ECLiPSE-Prolog[5]). The formalism adopted by these languages is rooted in the Situation Calculus. They consist of Prolog-based meta-interpreters that support a custom action language defined by overriding or defining Prolog operators.

[4] https://www.swi-prolog.org, last accessed in April 2020.

[5] https://eclipseclp.org, last accessed in April 2020.

Technological notes: We only succeeded in accessing the Golog base code, as ConGolog- and IndiGolog-related resources are no longer available on the project homepage. Golog sources consist of Prolog files supposedly targeting both SWI- and ECLiPSE-Prolog. Sources include a custom copyright notice that allows them to be used for research purposes. However, due to the lack of adequate information on how to run the code, we were not able to assess whether the technology actually works or not.

SHOP (1999)

Primary references: [266]
Other references describing / exploiting the technology: [150,171,172]
URL: https://www.cs.umd.edu/projects/shop
https://github.com/shop-planner/shop3

MAS perspective SHOP (Simple Hierarchical Ordered Planner) and its variants—SHOP2 by Nau et al. [267] and JSHOP2 by Ilghami [197]—are domain-independent automated-planning systems based on ordered task decomposition, which is a type of Hierarchical Task Network (HTN) planning—see Erol et al. [135] for details. SHOP is based on an action language with a Lisp-like syntax. In particular, SHOP complies with the PDDL specification [253], which is a standardisation effort for planning languages. Since it is based on PDDL, SHOP allows developers to express the "physics" of a domain: making it possible to declare domain predicates and available actions, together with their structure and effects.

As an ordered task decomposition planner, SHOP produces plans where tasks appear in the same order in which they should be executed (total-ordering requirement in SHOP, partial-ordering requirement in SHOP2). This limits the degree of uncertainty about the world, thus reducing the complexity of the reasoning. In addition to the usual HTN methods and operators, SHOP and its extensions can define axioms, mix symbolic/numeric conditions, and call external functions.

Although SHOP is not described as an agent-related technology, it has been employed in some agent-oriented works, referenced above in "Other references".

Logical perspective Being PDDL-compliant, SHOP is based on predicate logic. As with most AI planners, a logical atom in SHOP consists of a predicate name followed by a list of arguments, and a state of the world consists of a collection of ground logical atoms. As with most HTN planners, a task in SHOP consists of a task name followed by a list of arguments. Although tasks are similar syntactically to logical atoms, they are different semantically since they denote activities that need to be performed rather than logical conditions. However, SHOP implements an inference mechanism based on Horn clauses [190], with support to negation as failure. A transition from a state to another one is accomplished by an instance of an operator whose precondition is a logical consequence of the current state. The knowledge base contains domain-specific knowledge that SHOP may exploit to improve planning in that domain. It consists of axioms, methods, and operators.[6]

Technological notes: Most SHOP-related technologies lay unmaintained on the SourceForge repository accessible from the project homepage. However, a recent reboot of the SHOP project—namely, SHOP3—is available and actively maintained as a Common Lisp project on GitHub. This is distributed under the terms of the Mozilla Public License. We successfully installed the technology into our Common Lisp environment, and loaded the provided Lisp modules on it. However, due to the lack of available examples, we were not able to test the effectiveness of SHOP3.

[6] https://common-lisp.net/, last accessed in April 2020.

CaseLP (1999), DyLOG (2004) & DCaseLP (2007)

Primary references: [18,19,244,247]
Other references describing / exploiting the technology: [289]
URL: http://person.dibris.unige.it/mascardi-viviana/Software/DCaseLP.html

MAS perspective CaseLP (Complex Application Specification Environment Based on Logic Programming) is a LP-based prototyping environment for MAS, used to develop prototypes in collaboration with the Italian Railways and the Elsag company (now Leonardo s.p.a.) in 1999, for planning goods transportation. DCaseLP [247] is its distributed version (from which the "D" comes from), based on JADE [34], tuProlog [122] and Jess [177]. DyLOG [18]—later renamed "DYnamics in LOGic"[7]—is a high-level LP language for modelling rational agents, based on a modal theory of actions and mental attitudes, where modalities are used for representing actions, while beliefs model the agent internal state. DCaseLP and DyLOG were exploited together to support the design of agent interaction protocols in a methodologically-correct way [19].

Logical perspective CaseLP, DCaseLP, and DyLOG extend, or are implemented in, Prolog. At the technological level, DCaseLP consists of three Java archives, which allow agents specified in declarative languages—including tuProlog and Jess—to be integrated and run in the same JADE environment.

DyLOG is implemented in SICStus Prolog.[8]

Technological notes: The DCaseLP technology lays unmaintained on the project homepage. It consists of an archive containing both source and compiled Java code including none of the dependencies declared on the project homepage—namely, tuProlog, Jess, and ArgoUML. We were not able to find any license-related information for this technology. We also tried to run DCaseLP, but we failed due to the impossibility of retrieving the correct version of the dependencies from the Web. Notice that, since no specific version of the dependencies is indicated in the available documentation, we tested several combinations.

Mozart/Oz (1999)

Primary references: [303]
Other references describing / exploiting the technology: [81,87,175,327]
URL: http://mozart.github.io

MAS perspective The Mozart Programming System is an advanced development platform for intelligent, distributed applications. It implements Oz 3, the latest language of the Oz family [175] of multi-paradigm high-level languages based on the concurrent constraint model. Oz supports declarative programming, object oriented programming, constraint programming, and concurrency. Based upon Oz, Mozart supports multi-paradigm development.

Even though Mozart is not explicitly described as an agent-related technology, it has been exploited by some agent-oriented works, listed in the "Other references" field above.

Logical perspective The Oz programming language combines the logic and functional programming paradigms to support concurrent programming via logical constraint-based inference. The Mozart language and runtime explicitly target search problems through constraint propagation, similarly to what most constraint logic programming (CLP) [200] solvers

[7] http://www.di.unito.it/~alice/dynamicslogic/, last accessed in April 2020.

[8] https://sicstus.sics.se, last accessed in April 2020.

do. In particular, such sorts of problems can be written in a very concise way in Mozart, letting its syntax and declarative semantics abstract away the details of search.

Technological notes: Mozart is an actively-maintained project whose sources and releases are available on GitHub, under the terms of an *ad-hoc* open source license (https://mozart2.org/license-info/license.html, last accessed in April 2020). It consists of a native application, which supports all major operative systems (i.e. MacOS, Linux, and Windows). We successfully attempted to install and run the Mozart compiler, as well as testing how it works with some snippets of Oz code from the Mozart documentation.

GOAL (2000)

Primary references: [183]
Other references describing / exploiting the technology: [46,111,112,179–181,202,208]
URL: http://goalapl.atlassian.net/wiki/spaces/GOAL

MAS perspective GOAL (Goal-Oriented Agent Language) is an agent-oriented programming language inspired by the UNITY concurrent language [69]. Like UNITY, a set of actions which execute in parallel constitutes a program. However, unlike UNITY, which is based on variables assignment, GOAL incorporates complex notions such as belief, goal, and agent capabilities which operate on high-level information instead of simple values. GOAL agents make decisions based on their beliefs and goals. However, unlike other agent programming languages, GOAL focuses on providing a higher level of declarativeness to agents developers. GOAL agent goals describe what an agent wants to achieve, not how to achieve it—the desire is deliberated via automated reasoning. GOAL supports intra-agent communication, event processing (e.g., perceptions and messages), decision making, symbolic knowledge manipulation exploiting common knowledge representation technologies such as Prolog and OWL/SPARQL [210], in addition to reinforcement learning, and planning. It provides an IDE fully integrated into Eclipse making common editor features available.

Logical perspective GOAL is based on the BDI logic. However, similarly to other agent programming languages like Jason, GOAL mostly focuses on effective agent programming supported by a sound semantics, rather than exploiting BDI logic for MAS formal analysis. The syntax of the GOAL language is Prolog-inspired, therefore intrinsically logical. However, similarly to AgentSpeak(L), a number of constructs are allowed by the GOAL language, supporting goals, beliefs, rules, actions, and procedural knowledge.

Technological notes: GOAL is an actively-maintained project deeply intertwined with the Eclipse IDE. Indeed, while the GOAL code base is private, its binaries are only available as part of a plug-in for the Eclipse IDE. Apparently, there is no supported way to execute GOAL agents outside the IDE. The GOAL plug-in for Eclipse is distributed under the terms of the GNU Public License. The plug-in comes with some example projects we used for testing GOAL agents, after successfully installing its latest version on a fresh Eclipse IDE setup. Our tests were successful, thus proving GOAL is a working technology.

JACK Intelligent Agents (2001)

Primary references: [192]
Other references describing / exploiting the technology: [111,180,246,353]
URL: http://www.agent-software.com.au/products/jack

MAS perspective JACK is a cross-platform environment developed by the AOS Group, an Australian research oriented company, with the goal of bringing intelligent agents into the industrial playground. It allows the developers to build, run, and integrate commercial-grade multi-agent systems, where agents are inspired by BDI architecture and logics.

A JACK MAS is implemented in an extended variant of Java. Extensions provide the means for manipulating agents plans, events, beliefs, interactions, in addition to ordinary Java computations. However, the most common way to implement a JACK agent goals and beliefs is to rely on object-oriented technology. In this sense, multiple programming paradigms and styles are fruitfully harmonised in JACK to create an industry-ready MAS technology.

Logical perspective JACK is inspired by BDI logics. Similarly to Jason and GOAL, however, the BDI architecture described by Rao [294] is used as a basis to design and implement the JACK computational model. Indeed, in order to facilitate the work of developers—in particular those with an AI background—JACK also supports logical variables and cursors. These can be used to perform complex queries to an agent belief base. Accordingly, the semantics of a JACK program has been described as "midway between the semantics of logic programming (with the addition of type checking in Java style) and embedded SQL" [192].

Technological notes: JACK is closed source, thus we were not able to inspect it from a technological point of view.

Agent Factory Agent Programming Language (AFAPL, 2002).

Primary references: [87]
Other references describing / exploiting the technology: [44,46,180,232,257]
URL: http://sourceforge.net/projects/agentfactory/

MAS perspective AFAPL is another BDI agent programming language, aimed at designing and developing intentional agents. Agent Factory is the tool implementing the AFAPL language. The tool supports the many aspects of MAS developments, including distribution—which is supported exploting the JADE platform. Among the peculiarities of AFAPL, the notion of commitment is the most notable. Agent behaviour is dictated by the goals agents commit to, therefore the developers plan the behaviour of the agents by declaring the conditions for the adoption of the commitments in an explicit way.

Logical perspective AFAPL has been deliberately designed to support "heterogeneous logic-based agent architectures with the goal of providing a common tool set that can be adapted to different agent models, ranging from custom Java agents, through to reactive agent architectures, and finally to high-level agent programming languages" [304]. However, for what concerns agents execution semantics, AFAPL leverages on the BDI architecture as well. In particular, its computational model is inspired to the AgentSpeak(L) one. AFAPL developers can exploit first-order logic expressions to represent both agent beliefs and behaviours.

Technological notes: The AFAPL technology consists of an unmaintained Java project not including license information. A pre-compiled Java archive is available on the project home page. However, given the lack of any documentation describing how to run the provided binary, we were not able to further asses the technology.

DALI (2002)

Primary references: [97]
Other references describing / exploiting the technology: [36,98,99,101]
URL: http://github.com/AAAI-DISIM-UnivAQ/DALI

MAS perspective DALI is an agent programming language based on the active logic programming language [94] designed to make logical specifications of agents executable. A DALI agent is composed by reactive rules to face the events that can occur in the environment, as well as proactive rules, aimed at allowing the agent to perform actions on its own initiative. The reactive and proactive behaviour of a DALI agent is triggered by external or internal events, which can refer to either the present or the past.

Logical perspective The DALI language is kept as close as possible to the syntax and semantics of the plain Horn-clause language. DALI inference engine is based on an extended form of resolution. The semantics of a DALI program is named "evolutionary semantics" and is defined in terms of a modified program where reactive and proactive rules are reinterpreted in terms of standard Horn Clause rules.

Technological notes: The DALI technology consists of a Prolog system for SICStus Prolog (no specific version is indicated), and lays unmaintained on GitHub. It is distributed under the terms of the Apache License. To assess DALI from a technological point of view, we downloaded an evaluation version of SICStus Prolog. Despite the DALI sources can be successfully loaded in the SICStus engine, we were unable to verify whether the technology actually works due to the lack of adequate information on how to start the examples contained in the DALI code base.

eXAT (2003)

Primary references: [123]
Other references describing / exploiting the technology: [124]
URL: http://github.com/gleber/exat

MAS perspective The *erlang eXperimental Agent Tool* (eXAT) is another platform for agent programming, providing high-level abstractions and libraries to support both behavioural and intelligent aspects of an agent in a single platform. It is based on the Erlang[9] programming language, and provides programmers with a full-fledged framework supporting the design of agent reasoning, behaviours, interaction through a single, coherent technological tool. According to the authors, this is made possible by means of a set of modules strongly tied together. Roughly speaking, modules include: an Erlang-based expert system engine, an execution environment for agent behaviours based on object-oriented finite-state automata, and a module for supporting FIPA-ACL-compliant message passing.

Logical perspective eXAT is based on the Erlang language, which combines logical and functional paradigms, extending the syntax and the semantics of Prolog. Agent programs can be expressed via Erlang functions that can have multiple clauses (like Prolog). These features are used in eXAT to endow agents with automatic reasoning capabilities. For this purpose, eXAT includes an Erlang-based rule production system—namely, ERESYE—aimed at building agent-based expert systems. ERESYE enables the creation of multiple concurrent rule production engines, each with its own facts, composing the mental state of an agent. The platform can be also used to implement coordination among Erlang processes since each engine can behave as a Linda tuple space [160].

Technological notes: eXAT is currently available on GitHub as an unmaintained Erlang project. Only the source code of eXAT is available, since no pre-compiled binary is provided. It is distributed under the terms of the GNU General Public License. We attempted compilation on a freshly-installed Erlang runtime, exploiting the build automation configuration provided. However, the compilation was unsuccessful due to a missing dependency.

[9] http://www.erlang.org, last accessed in April 2020.

Go! (2004)

Primary references: [81]
Other references describing / exploiting the technology: [82,83]
URL: http://github.com/fgmccabe/go

MAS perspective Go! is a multi-paradigm language with a declarative subset of function and relation definitions, an imperative subset that includes action procedure definitions and program structuring mechanisms. Go! is strongly typed and multi-threaded. Threads in a single Go! invocation can communicate either by thread-to-thread message communication or by synchronised access/update of shared data, as dynamic relations. Threads in different Go! invocations can only communicate by using messages.

Logical perspective Although Go! has many features in common with Prolog, some Prolog features that limit readability—like "cut"—are absent from Go!. Instead, it supports higher level programming constructs, like single solution calls, and if rules. Furthermore, functional relationships can be defined through functions. Agents' threads can also use shared dynamic relations acting as Linda-style tuple spaces. The language is also suitable for representing ontologies exploiting its integration of logic, functional and imperative programming.

Technological notes: Go! is available on GitHub, although it has not been maintained. It is distributed under the terms of the GNU General Public License. It consists of an articulated project including sources written in different programming languages, without any binary. However, due to the lack of information on how to compile the code or how to start Go!, further evaluations could not be performed.

Event-B & the Rodin Platform (2004)

Primary references: [325]
Other references describing / exploiting the technology: [2,165]
URL: http://www.event-b.org
https://sourceforge.net/projects/rodin-b-sharp

MAS perspective Event-B is a formal method with tool support that enables a step-wise development of reactive distributed systems. It specifically addresses the situated nature of MAS. To ensure their correctness and structure their development, Event-B allows developers to rigorously specify complex agent interactions and verify their correctness and safety via formal reasoning. The Rodin tool is intended to support construction and verification of Event-B models.

Logical perspective Event-B is centred around the notion of events (transitions). It is based on first-order logic and a typed set-theory. The models described with Event-B are built by means of two basic constructs: contexts and machines. Contexts contain the static parts of a model whereas machines contain its dynamic parts. Rodin relies on the Event-B method which proposes to use the basic mathematical language (first-order logic, elementary set theory) for writing formal specifications. In particular Rodin exploits predicate logic to formalise and develop transition—intended as events—in MAS.

Technological notes: The Rodin technology consists in a custom variant of the Eclipse IDE. Its source is publicly available on SourceForge, as well as on some pre-compiled binaries, consisting of a self-contained setup of the aforementioned Eclipse-based IDE. We were unable to find any license-related information. We succeeded in installing and launching the pre-compiled IDE. However, due to the lack of availability of examples and benchmarks, we were unable to further evaluate the technology.

DRAGO (2005)

Primary references: [312]
URL: http://drago.fbk.eu/index.html

MAS perspective The DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies) technology addresses the problem of reasoning with multiple ontologies interconnected by semantic mappings. The agent reasoning process results from a combination of semantic mappings of local reasoning chunks performed in a single OWL ontology [33].

Logical perspective DRAGO exploits description logic (DL) [15] and semantic tuple centres. The DRAGO technology is theoretically grounded on the formal framework of the distributed description logics [47], which extends standard DL with semantic mappings, which can be seen as a bridge rule between different ontologies. This enables the representation of multiple semantically connected ontologies in which the reasoning procedure—according to the distributed contextual reasoning paradigm—is based on distributed tableau techniques, an extension of standard DL tableau.

Technological notes: The DRAGO technology consists of a number of unmaintained pre-compiled jars available on the project home page. They target the JVM 5+ platform, and they are distributed with no license information. We successfully tested DRAGO on newer versions of the JVM, and against the provided example ontologies. Therefore, it can be considered as a working technology.

Jadex (2005)

Primary references: [288]
Other references describing / exploiting the technology: [111,180,246,248]
URL: http://www.activecomponents.org
https://github.com/actoron/jadex

MAS perspective Jadex is a software framework for the creation of goal oriented agents following the BDI model. Its main goal is to build up a rational agent layer that sits on top of the JADE infrastructure. Jadex enables the construction of intelligent agents leveraging on sound software engineering foundations. Wherever applicable, object-oriented concepts and technologies are exploited, supporting the description of beliefs, intentions, desires, and plans both in Java and XML. Moreover, the Jadex reasoning engine tries to overcome traditional limitations of BDI systems by introducing explicit goals.

Logical perspective Jadex reasoning engine is based on BDI logic, and combines it with state-of-the-art software engineering techniques involving XML and Java. The initial state of an agent is determined among other things by the beliefs, goals, and the library of known plans. Jadex uses a declarative and a procedural approach to define the components of an agent. Jadex does not enforce a logic-based representation of beliefs, even if it is a viable option. Moreover, ordinary Java objects of any kind can be contained in agent belief bases, allowing for instance developers to reuse classes generated by ontology modelling tools or database mapping layers.

A more declarative way of accessing beliefs and belief sets is provided by queries, which can be specified in an OQL-like language [66]. The beliefs are used as input for the reasoning engine by specifying certain belief states, e.g. as preconditions for plans or creation conditions for goals. The engine monitors the beliefs for relevant changes, and automatically adjusts goals and plans accordingly.

Technological notes: Jadex is an open source technology requiring JVM 8. It is available under the GNU General Public License both on the project homepage and on GitHub. It is alive and actively developed, as demonstrated by the intense activity on its official GitHub repository and by the many resources available through its homepage. It is also available as a Maven artifact, along with all its dependencies; thus proving its maturity as a JVM technology. While testing the last release of Jadex, we met some problems with the build automation system provided. Apparently, some dependency is missing. Even though we might be just a temporary problem, we did not test Jadex further.

JADL & JIAC Framework (2006)

Primary references: [211,240]
Other references describing / exploiting the technology: [46,147,239]
URL: http://www.jiac.de/agent-frameworks/jiac-v

MAS perspective JADL (JIAC Agent Description Language) is the core of an extensive agent framework called JIAC, and has originally been proposed by Sesseler [313]. JIAC [239] consists of a methodology supported by (i) a Java-based runtime environment, (ii) some tools for creating agents, and (iii) some extensions which range from web-service-connectivity to an OWL-to-JADL translator. A JIAC MAS consists of a number of platforms, each of which has its own directory facilitator and agent management system. Each platform hosts different agents which are specified via the JADL declarative language and consist of a set of ontologies, rules, plan elements, and initial goal states. AgentBeans are Java classes containing methods which can be called directly from within JADL, allowing agents to interact with the real world. JADL provides constructs to describe plans as well as ontologies, FIPA-based speech acts, a (procedural) scripting part for (complex) actions.

Logical perspective JIAC is a comprehensive agent framework including logic concepts like ontology language, trinary propositional calculus, first-order logic, situation calculus, reaction rules, and JADL as a logic-based agent programming language. JADL is based on three-valued predicate logic, thereby providing an open world semantics. It comprises four main elements: plan elements, rules, ontologies, and services. JADL makes it possible to use the concept of uncertainty using a situation calculation that features a three valued logic. The third truth value is added for predicates that cannot be evaluated, with the information available to a particular agent. Thus, a predicate can be explicitly evaluated as unknown. JADL makes it possible to define knowledge bases which most of the rest of the language is based on. Every object that the language refers to needs to be defined in an ontology. Differently from Jason, GOAL, 2APL, the planning mechanism supported by JADL is a first principle planning, with no pre-compiled plan library.

Technological notes: JADL and JIAC sources and binaries are available through and ad-hoc Maven repository reachable from the JIAC home page. The last version of the software has been released in 2018, thus we consider the project is still maintained. The whole JIAC project requires the JVM 7+ platform and it is distributed under the terms of the Apache License. It consists of a very mature Java project. Thanks to the availability of several JADL examples and documentation resources, we succeeded in testing JIAC, which can thus be considered as a working technology.

2APL (2008) & 3APL (1999)

Primary references: [108,182]
Other references describing / exploiting the technology: [109–111,130,180,298,360]
URL: http://apapl.sourceforge.net
http://www.cs.uu.nl/3apl

MAS perspective The 3APL language provides programming constructs to implement individual agents directly in terms of beliefs, goals, plans, actions, and practical reasoning rules. Beliefs, plans, rules for revising plans and declarative goals are the basic building blocks of 3APL agents. Declarative goals describe the state an agent wants to reach and can be used to program pro-active behaviour. Plans form the procedural part of an agent and can be executed by the agent in order to achieve its goals. Rules enable the generation of plans on the basis of goals (and beliefs).

2APL (A Practical Agent Programming Language) is a BDI-based agent-oriented programming languages that extends 3APL by Hindriks et al. [182]. It integrates declarative beliefs and goals with events and plans, and provides practical programming constructs to enable the generation, repair, and execution of plans based on beliefs, goals, and events. The operational semantics is given via transitions between configurations. The configuration of an individual agent consists of its beliefs, goals, events, plans, and practical reasoning rules, while the configuration of a MAS consists of the configurations of the individual agents, the state of the external shared environments, and the agents' access relation to environment. 2APL is equipped with an execution platform and an Eclipse plug-in editor. The platform provides a graphical interface where a user can load, execute, and debug 2APL multi-agent programs using different execution modes and several debugging/observation tools. The platform can be used in the stand-alone mode or in distributed mode using JADE. The adoption of JADE allows a multi-agent program to be run on different machines connected in a network.

Logical perspective 3APL is a multi-agent programming language that presents a mixture of logic programming and rule based systems. The main components of a 3APL agent can be expressed in a declarative way Belief and goal query expressions are either the atomic formulae or a well-formed formulae constructed from atoms and logical connectors.

Conversely, the BDI-oriented multi-agent programming language 2APL enables the implementation of an agent's beliefs in terms of logical facts and rules. 2APL, as other logic-based agent programming languages, is based on a combination of imperative programming with logic-based knowledge bases. Goals in 2APL are logical formulae that the agents exploit in the execution of a plan, via logical entailment.

Technological notes: Both 3APL and 2APL technologies are still available on their respective homepages, although not maintained any more. In both cases the source code is not available, and we could not find any license-related information, although we were able to access some pre-compiled binaries. They consist of some executable jars targetting JVM 6+. Thanks to the availability of both examples and documentation, we were able to test the 2APL technology that still works on JVM 6, 7, and 8. The same is not true for 3APL, for which we encountered runtime errors on all version of the JVM ranging from 6 to 13.

jDALMAS (2009)

Primary references: [185]
Other references describing / exploiting the technology: [184]
URL: http://sourceforge.net/projects/jdalmas

MAS perspective jDALMAS is a general-level Prolog implementation of the abstract DALMAS—Deontic Action-Logic Multi-Agent System—architecture by Hjelmblom and Odelstad [185]. The jDALMAS technology enables the specification of agents' possible actions in specific states. There, an action is permissible if it is not prohibited by the DALMAS normative system. In the deliberation phase, when an agent needs to choose the next action, DALMAS allows it to select an act out of a set of feasible acts. This may lead to a new state, depending on the state of the system when the act is performed. Thus, the choice

of act is determined by the combination of the DALMAS preference structure and its deontic structure.

Logical perspective The technology represents the implementation of the DALMAS abstract architecture. The DALMAS architecture is regulated by a normative system consisting of norms, which are expressed in an algebraic notation based on the Kanger–Lindahl theory of normative positions [233]. The Kanger–Lindahl theory is based on Kanger's deontic action-logic [205], as a mix of action logic in terms of action operator *do* and deontic logic in terms of *shall* or *may*.

Technological notes: The jDALMAS technology lays unmaintained on the project home page. Only an articulated Java code base is provided, with no clear information describing the license of the software or how to compile and use it. For this reason, we made no further assessments.

RASP (2010)

Primary references: [100]
Other references describing / exploiting the technology: [95,96,130]
URL: http://www.dmi.unipg.it/formis/raspberry

MAS perspective RASP (Resourced Answer Set Programming) extends Answer Set Programming (ASP, by Lifschitz [231]) by supporting declarative reasoning on production and consumption of resources. Although RASP is not, strictly speaking, an agent-oriented paradigm, Costantini and Formisano [96] exploit it to extend (logic) agents reasoning capabilities in a resource-oriented scenario.

Logical perspective RASP combines answer set semantics with quantitative reasoning and with various forms of preferences and conditional preferences on resource usage. Raspberry [100] is a prototypical implementation of a translator from ground RASP to ASP: it leaves unchanged the ASP portion of the input program and translates RASP-rules (those involving atoms specifying amounts of resources) into suitable ASP fragments.

Technological notes: The Raspberry technology is still available, even though not maintained, on the project home page. Only binaries are available, requiring the Linux platform, and no license information is provided. We tested Raspberry based on the examples provided: the technology still works perfectly. It is worth to be mentioned that the ASP (https://potassco.org/clingo, last accessed in April 2020) sources generated Raspberry require an ASP solver to be executed. To this purpose, we exploited Clingo.

TeleoR (2015)

Primary references: [84]
Other references describing / exploiting the technology: [80,257,269]
URL: http://staff.itee.uq.edu.au/pjr/HomePages/QulogHome.html

MAS perspective TeleoR extends the Teleo-Reactive (TR) rule based robotic agent programming language by Nilsson [269]. In both TR and TeleoR programs consist of Guard–Action rules grouped into parametrised procedures. The Guard is a deductive query to a set of percept facts generated from sensors. Actions may be primitive robotic actions, single call to a program procedure, which can be a recursive call, or an update to the belief store. TeleoR also supports reactive task and sub-task behaviours of robotic agents, besides wait/repeat re-start of failed actions, and support for agents multi-tasking.

Logical perspective The TeleoR declarative programming language exploits logical rules for information representation. In particular, each TR node represents both percepts and beliefs through logical facts and rules. Furthermore, TeleoR is based on the Qu-Prolog system[10] logic/functional programming language, which extends Prolog with a flexible type system supporting higher-order functional terms and functions. These are exploited in TeleoR for belief storage, inference, and deliberation.

Technological notes: The TeleoR technology is available as part of the QuLog system on the project homepage. Only the source code is distributed and actively maintained, even if license information is missing. The provided code requires the Qu-Prolog system, for which pre-compiled binaries are not available. Our attempts to compile Qu-Prolog were unsuccessful, so we did not evaluate TeleoR further.

ASTRA (2015)

Primary references: [88]
Other references describing / exploiting the technology: [139,299]
URL: http://astralanguage.com
http://github.com/remcollier/astra

MAS perspective Similarly to Jason, ASTRA allows developers to define agent plans in a declarative way; yet, unlike Jason, it offers a tighter integration with the Java language and runtime. Java objects and types can in fact directly be used within ASTRA scripts.

Logical perspective ASTRA exploits BDI logic. ASTRA is an implementation of the AgentSpeak(TR) language (not to be confused with AgentSpeak(ER) presented in the sequel), that is, a logic-based agent programming language combining AgentSpeak(L) with teleo-reactive functions designed and implemented by the ASTRA's developers.

Technological notes: The ASTRA technology essentially consists of a plug-in for the Eclipse IDE. Even if the Java source code of ASTRA can be found unmaintained on GitHub, we were able to install the ASTRA plug-in on a fresh new Eclipse setup. Links for installation are available on the ASTRA homepage. We successfully tested ASTRA against the code snippets available on the project official documentation.

AgentSpeak(ER) & Jason-ER (2019)

Primary references: [299]
Other references describing / exploiting the technology: [45,88]
URL: http://github.com/agentspeakers/jason-er

MAS perspective AgentSpeak(ER) is an extension of AgentSpeak(L) featuring plan encapsulation, i.e. the possibility to define plans that fully encapsulate the strategy to achieve the corresponding goals, integrating both the proactive and the reactive behaviour. AgentSpeak(ER) agents expose "g-plans" that can be repeatedly re-executed until a given goal-condition about the beliefs of some agent is met. AgentSpeak(ER) comes with two prototype implementations, one based on ASTRA by Collier et al. [88] and the other one based on Jason. Jason-ER[11] provides the syntactical and semantic extensions needed by Jason to support AgentSpeak(ER).

Logical perspective The logical perspective is similar to the Jason one with some enhancements. The Jason-ER extension turns out to bring some benefits to agent programming based

[10] http://staff.itee.uq.edu.au/pjr/HomePages/QuPrologHome.html, last accessed in April 2020.

[11] https://github.com/agentspeakers/jason-er, last accessed in April 2020.

on the BDI logic, namely: (i) improving the readability of the agent code, reducing fragmentation and increasing modularity; (ii) promoting a cleaner goal-oriented programming style by enforcing encapsulation of reactive plans within goal plans while still permitting purely reactive behaviour; (iii) improving intention management, enforcing a one-to-one relation between intentions and goals—so every intention is related to a single (top-level) goal.

Technological notes: We were able to find only the Jason-ER implementation, as a fork of the aforementioned Jason code base. As such, it consists of a Java-based project licensed through the GNU Lesser General Public License and targetting JVM 8+. Despite the many examples inherited from Jason code base, only one has been added in Jason-ER to exemplify the novel "g-plans" feature—and that we successfully tested.

3.1.2 Agent reliability & verification

MCAPL (2012), Gwendolen & AIL

Primary references: [120]
Other references describing / exploiting the technology: [118–121]
URL: http://mcapl.sourceforge.net/

MAS perspective Designed by Dennis et al. [120], the Model Checking Agent Programming Language (MCAPL) consists of tools for prototyping BDI agent programming languages and for model checking programs written in such languages via an interface to the Java Path Finder model checker (JPF, [170]). MCAPL consists of (i) the Agent Infrastructure Layer (AIL), which provides a common abstract base for the creation of concrete interpreters for BDI agent programming languages like Gwendolen by Dennis and Farwer [119], and (ii) Agent JPF (AJPF), an extended version of JPF, with a property specification language appropriate for model-checking any AIL-based MAS. An extension to AJPF to generate models also suitable for non-agent model-checkers was developed by Dennis et al. [121].

Although nothing prevents MCAPL from being used in a real MAS context, its most widespread usage is for model checking properties of individual BDI-style agents; for this reason, it is presented in this section rather than in Sect. 3.2.3.

Logical perspective The MCAPL interface allows a user to specify simple properties in a temporal and modal logic and then check that these properties hold using the JPF. Formulas in the MCAPL property specification language have an LTL (Linear Temporal Logic)-based [287] semantics defined by the MCAPL interface.

Technological notes: The MCAPL technology lays unmaintained on a SourceForge repository, which is accessible from the project homepage. Both the Java source code and the corresponding pre-compiled jars are provided, along with the GNU General Public License, and a number of MAS examples based on AIL. The AIL examples can be run directly or model-checked via MCAPL. We only succeeded in the first activity, since MCAPL fails to load the JPF dependency provided, because it is apparently too old to work with modern versions of JVM.

3.2 Society

Society plays a central role in MAS technologies, since it represents the ensemble where the collective behaviours of the MAS are *coordinated* towards the achievement of the overall system goals. Along this line, coordination models glue agents together by governing agent interaction in a MAS, and pave the way towards social intelligence. The selected technologies

in the "Organisational reasoning & coordination" sub-category are ALIAS, 2CL, TuCSoN, ReSpecT, and AORTA.

Within an agent society, agents can enter into argumentation processes to reach agreements and dynamically adapt to changes. Dispute and conflict need to be managed in order to achieve a common agreement and to establish the winner argument in case of conflicts. Many technologies exist for solving reasoning tasks on the abstract argumentation frameworks by Dung [129]; problems of this kind are intractable and this calls for efficient algorithms and solvers: the International Competition on Computational Models of Argumentation (ICCMA), whose fourth edition will be run in 2021,[12] provides a forum for empirical comparison of solvers. Most solvers are based on logic-based programming or logical frameworks including ASP and SAT-solvers, as discussed by Gaggl et al. [156]. The literature on argumentation solvers overlaps to some extent that on agent- and logic-based technologies for argumentation: under the "Argumentation" sub-category, the SLR only includes those argumentation frameworks that are explicitly tagged as agent-based and that can be considered the founders of a given logical model: DeLP, Aspartix, ASPIC+, Dung-o-Matic, TOAST, and SPINdle.

Finally, MAS should be verifiable, also under the social perspective, i.e. in terms of verification of interaction and communication protocols. The "MAS reliability & verification" sub-category thus includes MCK, SALMA, SCIFF, MCMAS, and Trace Expressions.

3.2.1 Organisational reasoning & coordination

ALIAS (1999)

Primary references: [76]
Other references describing / exploiting the technology: [75,77]
URL: http://lia.disi.unibo.it/research/ALIAS

MAS perspective The ALIAS architecture introduces abduction in a logic multi-agent environment. It implements a distributed protocol to coordinate the reasoning of all the abductive agents in the system. The global knowledge is represented by a set of abducted hypotheses stored on a Linda-like tuple space—that is, a mechanism to coordinate agent reasoning.

Logical perspective ALIAS is built on top of abductive logic programming [204]. In ALIAS, agents are equipped with hypothetical reasoning capabilities, obtained by means of abduction. The union of their knowledge bases ends up to generate a global knowledge base, which can change in a dynamic fashion, as a consequence of agent movements, and which is maintained coherent as it would be if it was owned by a single entity.

Technological notes: The ALIAS technology lays unmaintained on the project homepage. The last version (v. 3), is distributed with no license-related information. It leverages on a JVM-base Prolog interpreter—namely, Jinni [332]—that is not available any more. Thus, we were not able to assess the technology any further.

TuCSoN (1999) & ReSpecT (2001)

Primary references: [275,276]
Other references describing / exploiting the technology: [64,264,264,265,265,278,279]
URL: http://tucson.unibo.it
http://respect.apice.unibo.it

[12] http://argumentationcompetition.org, last accessed in April 2020.

MAS perspective TuCSoN is a standard middleware technology based on Java, and on tuProlog [122] engine—that is, a first-order logic technology. Thanks to TuCSoN, both Java and Prolog agents—possibly distributed over the network—can interact via logic tuple *centres*. The tuple centre model by Omicini and Denti [275] is the core of the TuCSoN coordination infrastructure. Tuple centres are essentially LINDA tuple spaces containing *logic* tuples, observable through *logic* templates. However, tuple centres are not mere containers of tuples. They can be *programmed* by agents through the ReSpecT language ReSpecT is in fact a logic-based coordination language aimed at providing for social intelligence in MAS.

TuCSoN supports multiple ReSpecT tuple centres, which can be spatially distributed and connected via linkability [278]. Accordingly, tuple centres can be locally deployed within a physically-distributed environment as discussed by Casadei and Omicini [64], with a possibly huge number of spatial containers and physical devices. Furthermore, tuple centres embody the laws for local coordination, ruling the social behaviour of the agents interacting through them.

It is worth to be mentioned that the current version of TuCSoN is the result of many years of active development by many different people pursuing different goals—as demonstrated by the multiple variants of TuCSoN and ReSpecT described in this section. Thus, as discussed in Ciatto et al. [79], the TuSoW project [78] can be considered a notable rebooting attempt for TuCSoN, focusing on supporting modern mainstream technologies and platforms.

Logical perspective TuCSoN coordination media are tuple centres based on Prolog. The logical feature of TuCSoN tuple centres makes it possible to spread intelligence through the system where needed, for example by exploiting cognitive agents. Coordination media in TuCSoN and ReSpecT (and, in particular, situated ReSpecT) are logic-based tuple centres powered by tuProlog. They can be adopted both for the communication language (logic tuples), and for the behaviour specification language (ReSpecT). Basically, reactions and resources in ReSpecT are defined as Prolog-like rules.

Technological notes: TuCSoN and ReSpecT are part of the same technology. Both their source code and pre-compiled jars are available on the TuCSoN GitHub page (https://github.com/TuCSoN-Coord/TuCSoN, last accessed in April 2020), reachable from the projects home pages. They consist of a Java project requiring JVM 8+, and they are distributed under the terms of the GNU Lesser Public License. We successfully tested TuCSoN and ReSpecT against the provided examples.

2CL (2010)

Primary references: [26]
Other references describing / exploiting the technology: [22,24]
URL: http://www.di.unito.it/~alice/2CL

MAS perspective 2CL is at the same time a methodology for the design of protocols and a toolkit enabling their engineering. Interactions among agents are represented in 2CL by exploiting commitment-based interaction protocols [148], which normally consist of sets of actions with a shared meaning. From the point of view of an agent, the meaning of an action is completed by the context in which it is used. In other words, the context shapes the behaviour of the agent in that the agent decides which actions to take depending on it.

Usually, commitment-based protocols take into account constitutive rules and not regulative ones. 2CL is indeed based on the notion of behaviour-oriented commitment protocols, which account both for the constitutive and the regulative specifications and that explicitly

foresee a representation of the latter based on constraints among commitments. In particular, the language 2CL makes it possible to write these regulative specifications.

Logical perspective From the theoretical perspective, the 2CL tool is an extension of the enhanced commitment machine by Winikoff et al. [354], which allows all the possible executions of a business protocol to be explored, showing all the violations. Its implementation is done in tuProlog, and the software interprets a 2CL business protocol specification by means of a parser written in Java.

Technological notes: The 2CL technology lays unmaintained on the project home page. It consists of an archive containing pre-compiled jars requiring the JVM 7+ platform, no license information is provided. We successfully tested the 2CL technology against the available examples, and on newer versions of the JVM as well.
It is worth to be mentioned, however, that 2CL is the only commitment-based model—among the many related works we encountered in the early phases of this survey, like, e.g., the work from Chopra and Singh [74]—which also includes some runnable technology available to date.

AORTA (2014)

Primary references: [201]
Other references describing / exploiting the technology: [222]
URL: http://www2.compute.dtu.dk/~ascje/AORTA
https://github.com/andreasschmidtjensen/aorta

MAS perspective AORTA (Adding Organisational Reasoning to Agents) provides organisational reasoning capabilities to agents implemented in existing agent programming languages—like, for instance, Jason. AORTA assumes a pre-existing organisation, is independent from the agent, and focuses on reasoning rules that specify how the agent reasons about the specification. The organisation is separated from the agent, meaning that the architecture of the agent is independent from the organisational model.

Logical perspective AORTA makes use of tuProlog, so the contents of the agent's knowledge bases can be translated into Java objects supported by tuProlog. Logic programming and object oriented paradigms are exploited in a synergic integration. AORTA has been implemented on the top of Jason.

Technological notes: The AORTA technology consists of an unmaintained Java project available on GitHub. The repository also includes some pre-compiled jars and some examples, but it carries no license information. The provided jars require the JVM 7+ platform, and they have been successfully tested against the provided examples.

3.2.2 Argumentation

DeLP (2004)

Primary references: [158]
Other references describing / exploiting the technology: [333]
URL: http://tweetyproject.org/w/delp/index.html
http://lidia.cs.uns.edu.ar/DeLP

MAS perspective Defeasible Logic Programming (DeLP) combines results of logic programming and defeasible argumentation. In particular DeLP is based on the "pure" defeasible model introduced by Nute [272]. The ability to draw tentative conclusions and retract them based on further evidence—which characterises non-monotonic and defeasible logic—is also a relevant feature for agents which deal with uncertainty. DeLP provides the possibility of representing information in the form of weak rules in a declarative manner, and a defeasible argumentation inference mechanism for warranting the entailed conclusions. To solve conflicts, DeLP lets agents perform a dialectical process in which all arguments in favour and against a conclusion are considered, and arguments regarded as ultimately undefeated are considered warranted.

Logical perspective All the technologies for argumentation are based on deontic and defeasible logic. DeLP is based on logic programming: it can be considered as a logic-based approach to structured argumentation.

Technological notes: The DeLP technology is currently part of the Tweety Project (http://tweetyproject.org, last accessed in April 2020) – that is, "a collection of various Java libraries that implement approaches to different areas of artificial intelligence"—, and it consists of a publicly available web service intended for in-browser usage. Technically, DeLP is a technology requiring JVM 12+ whose source and compiled code is distributed through Maven Central, under the terms of the GNU Lesser General Public License. However, the DeLP service is currently offline and we were not able to run another instance locally due to some implementation issues. For this reason, we did not evaluate DeLP further.

Aspartix (2008)

Primary references: [131]
Other references describing / exploiting the technology: [39,347]
URL: http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage

MAS perspective Aspartix is an agent argumentation framework that formalise statements together with a relation denoting rebuttals between them. Its semantics gives an abstract handle to solve the inherent conflicts between statements by selecting admissible subsets of them. Argumentation frameworks can support the decision making of an agent within a modular architecture for agents.

Logical perspective Aspartix is an ASP-based approach to find the justified arguments of an argumentation framework with respect to different acceptable extensions for a broad range of formalisations of Dung's argumentation framework. It relies on a fixed disjunctive datalog [1] program which takes an instance of an argumentation framework as input, and uses an answer-set solver for computing the type of extension specified by the user.

Technological notes:
The Aspartix technology is available as a Web Service, through the project home page, even if no source code is provided. The software is distributed under the terms of the MIT License. Even if we cannot estimate the maintenance level of the project, we are able to state the Web Service has been lastly updated in 2018. As a Web Service, Aspartix can be virtually used on any platform. Finally, the Aspartix Web Service comes with some predefined argumentation framework examples, which we successfully tested.

ASPIC+ & Dung-o-Matic (2008) & TOAST (2012)

Primary references: [348]
Other references describing / exploiting the technology: [39,40]
URL: http://tweetyproject.org/downloads/index.html
http://toast.arg.tech
http://www.arg-tech.org/index.php/projects/dung-o-matic

MAS perspective The ASPIC+ argumentation framework by Prakken [293] provides structure to agent arguments, while still allowing an abstract framework to be derived and, ultimately, evaluated using established acceptability semantics. The corresponding technology allow agents to resolve their conflicting arguments through a semantics of their choice, other than specifying priorities or assumption over the rules they leverage upon for reasoning.

Logical perspective ASPIC+ can be considered as the most common structured argumentation approach to Dung's abstract argumentation. Dung-O-Matic is an engine implemented in Java that reflects the ASPIC+ framework. TOAST is a web service easing the exploitation of ASPIC+, based on Dung-O-Matic. It allows a structured argumentation system to be processed into arguments and attacks from which a Dung-style framework can be derived and evaluated.

Technological notes: While the ASPIC+ technology is available as part of the Tweety Project—and, as such, it targets the JVM 12+ platform, and its source and compiled code is available on Maven Central, under the terms of the GNU Lesser General Public License—, the TOAST technology is currently up and running a Web Service—thus targetting virtually all platforms, even if no source code or license information are provided. Conversely, we were not able retrieve any source or compiled code relative to Dung-O-Matic. We successfully tested TOAST against the examples provided in the user guide. Similarly, we successfully tested ASPIC+ through the Java examples provided as part of the Tweety Project.

SPINdle (2009)

Primary references: [220]
Other references describing / exploiting the technology: [12,335]
URL: https://sourceforge.net/projects/spindlereasoner
http://spindle.data61.csiro.au/spindle/documentation.html

MAS perspective SPINdle is an open-source Java-based defeasible logic reasoner capable to perform efficient and scalable reasoning on defeasible logic theories. These technologies tackle the way the Semantic Web affects knowledge and information interchange among intelligent agents in multi-agent systems, as well as reasoning interoperability. SPINdle can be used as a standalone theory prover and can be embedded into any applications as a defeasible logic rule engine. It allows agents to issues queries, on a given knowledge base or a theory generated on the fly by other applications, and automatically produces the conclusions of its consequences. The theory can also be represented using XML, for agent communication.

Logical perspective SPINdle is based on the "pure" defeasible model introduced by Nute [272]. Defeasible knowledge can be expressed and conflicts can be solved according to standard defeasible logic [271]. The theory can also be represented using RuleML,[13] an XML-based standard language that enables users to use different types of rules (such as

[13] http://ruleml.org/index.html, last accessed in April 2020.

derivation rules, facts, queries, integrity constraints) to represent different kinds of elements according to their needs.

Technological notes: The SPINdle technology consists of a Java project currently hosted on SourceForge. Both the source code and the pre-compiled binaries are available, targetting the JVM 7+ platform, even if the project is unmaintained since 2017. SPINdle is distributed under the terms of the GNU General Public License. We successfully tested SPINdle against the `.dlf` files provided as part of the pre-compiled binaries.

3.2.3 MAS reliability & verification

MCK (2004)

Primary references: [157]
Other references describing / exploiting the technology: [268]
URL: http://cgi.cse.unsw.edu.au/~mck/pmck
http://cgi.cse.unsw.edu.au/~mck/mckform/

MAS perspective MCK is a model checker for the logic of knowledge: it supports several different ways of defining knowledge given a description of a MAS and the observations made by the agents. It automates the MAS formal analysis focusing on how the states of information of agents change over time. MCK can automatically verify whether a specifications expressed in a formal logic of knowledge, probability and time holds in such a system.

Logical perspective MCK allows verification of temporal-epistemic logics using various Ordered Binary Decision Diagram (OBDD, [54]) and SAT based techniques [203]. The MCK input is a system design (represented as a program) and a question (encoded in a formal language that can express properties concerning time, probability and knowledge). MCK supports several different semantics for the agent epistemic state, depending on whether an agent determines what it knows from just its current observation, its current observation plus the current time, or its history of observations (interpreted either synchronously or asynchronously). The temporal dimension is supported by both linear and branching time temporal expressiveness.

Technological notes: The MCK technology is only available as a web service accessible from the project home page. No source or binary code is provided, nor any license information. We were not able to estimate the date of the last update to the Web Service. However, we successfully tested the MCK technology against the examples provided on the Web Service web page.

MCMAS (2006)

Primary references: [236]
Other references describing / exploiting the technology: [68,133,235,237,285]
URL: http://vas.doc.ic.ac.uk/software/mcmas

MAS perspective MCMAS is a model checker for MAS specified through various agent-based logics. The symbolic representation of state spaces and the algorithms for the computation of epistemic operators encoding private and group knowledge and Alternating-time Temporal Logic (ATL, [8]) exploit Ordered Binary Decision Diagrams (OBDDs). Model

checking of basic fairness conditions are supported, as well as the generation of counterexamples and witness executions. MCMAS can be used from a shell or via a graphical interface based on Eclipse.[14]

Logical perspective MCMAS model checker includes temporal operators, too. MCMAS supports a rich set of specifications, including Computation Tree Logic (CTL, [85]) operators, epistemic operators, ATL, and notions pertaining to correct behaviour.

Technological notes: The MCMAS technology is available through the project home page, after providing personal information. Both the source code and pre-compiled binaries are available, even if they are distributed with no license information. The code base consists of C++ sources targeting the Linux platform. We were able to successfully test MCMAS against some system descriptions examples contained into the MCMAS manual.

SCIFF (2005)

Primary references: [3]
Other references describing / exploiting the technology: [4,9,55,72,73,147,338,341]
URL: http://lia.deis.unibo.it/research/sciff

MAS perspective SCIFF is a model-checker mainly used to verify the compliance of agents to interaction protocols. SCIFF offers variables (e.g., to model time) and constraints on variables occurring in hypotheses and expectations; social goals, defined as predicates, can express the social aim or outcome of some agent interaction, or can be used to start an abductive derivation in the more classical tradition of abductive logic programming. The ability to generate positive and negative expectations, beside making hypotheses, and the concepts of fulfilment and violation of expectations are the core ingredients of dynamic checking of protocol compliance.

Logical perspective SCIFF is based on abductive logic programming. It is an abductive proof procedure mainly used to verify the compliance of agents to interaction protocols. In order to allow such application, SCIFF extends IFF by Fung and Kowalski [154].

Technological notes: The SCIFF technology currently lays unmaintained on the project home page. It consists of a number of unlicensed Prolog sources targeting either SICStus- or SWI-Prolog. We succeeded in loading the SCIFF Prolog modules on a freshly installed SWI-Prolog system, but we did not perform any further assessment due to the lack of available examples.

SALMA (2014)

Primary references: [217]
Other references describing / exploiting the technology: [218]
URL: http://www.salmatoolkit.org
https://github.com/salmatoolkit/salma

MAS perspective SALMA (Simulation and Analysis of Logic-based Multi-Agent models) is a simulation and statistical model checking tool for MAS. The simulated system can be specified exploiting logical axioms based upon the Golog situation calculus. The framework for MAS modelling and verification makes it possible to reason about the interaction of agents with each other and with their (physical) environment.

[14] A tool variant, MCMAS-SLK, can be downloaded at http://swmath.org/software/24778, last accessed in April 2020.

Logical perspective SALMA extends the classical situation calculus and linear temporal logic so as to address the specific requirements of multi-agent simulation models, in order to simulate and (statistically) model-check them. The system model exploits a first-order logic structure, the simulation is coupled with a statistical model-checker that uses a first-order variant of time-bounded linear temporal logic for describing properties.

Technological notes: The SALMA technology can be currently found unmaintained on GitHub. It consists of an unlicensed project requiring the Python 3 platform. No ready-to-use package is provided, and SALMA requires a number of dependencies to be executed. We were not able to retrieve all the necessary dependencies. Therefore, we did not evaluate SALMA further.

Trace Expressions (2016)

Primary references: [10]
Other references describing / exploiting the technology: [9,11,52,140]
URL: http://rmlatdibris.github.io
https://github.com/RMLatDIBRIS/compiler

MAS perspective Trace Expressions, named Global Types in the original paper by Ancona et al. [9], are a compact and expressive formalism for modelling "expected behaviours of the system" based on a set of operators to denote finite and infinite traces of events. They have been used to model and verify, among the others, MAS, distributed systems, and data types. RML [149] is a rewriting-based and system agnostic Domain Specific Language for Runtime Verification which decouples monitoring from instrumentation by allowing users to write specifications and to synthesise monitors from them, independently of the system under scrutiny. RML compiles down to Trace Expressions.

Logical perspective The Trace Expressions runtime verification engine is developed in SWI-Prolog, and takes advantage of its native support for cyclic terms and coinductive logic programming by Simon et al. [320].

Technological notes: The RML technology is an actively maintained Java based project currently hosted on GitHub. The source code requires the JVM 8+ platform and it is distributed under the terms of the Apache License. Despite no pre-compiled binary is available, the project includes a build automation tool which eases compilation and usage. We successfully tested RML against the example files contained into the project repository.

3.3 Environment

When direct interaction and explicit communication do not fit the needs and constraints for agent interaction, mediated interaction and environment-based coordination may come into play. The difference between works on coordination and interaction classified under the "Society" perspective and works listed in this section, is that in the latter coordination/interaction artefacts are meant as runtime abstractions encapsulating and providing coordination/interaction services. Those services can be exploited as the basic building blocks for designing and developing suitable working environments for heterogeneous MAS, supporting MAS coordination for collaboration or competition.

When agents are immersed in a knowledge-intensive environment (KIE), the cognition process goes beyond that of the individual agent, and distributed cognition processes may take place, promoting the idea of *intelligent environment* [187]. Thus, the environment concept is

extended, and it is not just limited to situated action—which motivates the classification of Semantic Web works in the environment abstraction.

Moving from Hendler [174], the "Agents in the Semantic Web" sub-category lists Onto2JaCaMo, AgentOWL, and EMERALD, which exploit semantic web technologies to interoperate. Another technology in the "Environment" category is situated ReSpecT, a technologies for "Situated Interaction & Coordination". It emphasises the situated component of interaction, thus related to the environment. Inbetween the two categories lays LPaaS (Logic Programming as a Service), a framework that supports the distribution of logic knowledge in the environment, and where artefacts can actively participate in the agent cognitive process. In LPaaS, in fact, artefacts play a supporting role in terms of knowledge repositories (in the form of environment structure and properties) and embed the reasoning process enabled and constrained by the knowledge they embody.

3.3.1 Agents in the semantic web

AgentOWL (2006)

Primary references: [186]
URL: http://agentowl.sourceforge.net

MAS perspective AgentOWL supports RDF/OWL ontology models in JADE MAS. It uses Jena[15] for ontology model manipulation and allows to model the agent knowledge using OWL and to exchange messages with OWL and SPARQL as content language. Ontologies are modelled with Protégé[16] following the CommonKADS knowledge and engineering methodology [310]; the MAS design is based on AUML [30] and MAScommonKADS [196].

Logical perspective Being the logic underlying the standard languages for the semantic web—OWL DL, OWL Lite, OWL Full, OWL 2—, description logic underlies, up to some extent, all those technologies used for "agents in the semantic web". In AgentOWL, OWL is mainly used to represent the knowledge of an individual agent.

Technological notes: The Agent-OWL technology consists of an unmaintained project containing both Java sources and binaries, requiring the JVM 6+ platform. Its code base includes a GNU Lesser General Public License. We failed to start the pre-compiled version of Agent-OWL on most JVM versions, namely, version 6, 7, 9, and 10. We succeeded in starting it on other versions of JVM up to version 13, even if a number of exceptions made us classify Agent-OWL as a non-working technology.

EMERALD (2010)

Primary references: [215]
Other references describing / exploiting the technology: [216]
URL: http://lpis.csd.auth.gr/systems/emerald

MAS perspective EMERALD is a MAS framework built on top of JADE; it aims at supporting interoperable reasoning among agents in the Semantic Web, by using third-party trusted reasoning services. In EMERALD, every agent can exchange its position justification arguments with any other agent, with no need for all agents to conform to the same kind of rule paradigm or logic. EMERALD provides a knowledge-based agent module based on Jess by Jess by Hill [177].

[15] https://jena.apache.org/documentation/ontologyd, last accessed in April 2020.

[16] https://protege.stanford.edu, last accessed in April 2020.

Logical perspective EMERALD uses OWL as the language to model common knowledge that agents need to share in order to interoperate.

Technological notes: The EMERALD technology consists of an unmaintained Java project, requiring JVM 6+ and including no license information. Only pre-compiled binaries are currently available, which we successfully started and executed against the examples provided along with them.

Onto2JaCaMo, 2017

Primary references: [151]
Other references describing / exploiting the technology: [152]
URL: http://www.inf.pucrs.br/linatural/wordpress/recursos-e-ferramentas/onto2jacamo

MAS perspective Onto2JaCaMo supports MAS development in compliance with a well-defined AOSE methodology defined by the authors themselves in Freitas et al. [152]. The tool offers a combined support of forward and reverse engineering, such that changes in one artefact can always be merged into the other without compromising consistency or losing changes; this approach is referred to as "round-trip engineering".

Logical perspective Onto2JaCaMo uses OWL as the language to model common knowledge that agents need to share in order to interoperate.

Technological notes: The Onto2JaCaMo technology consists of a pre-compiled Java archive containing a plug-in for the Eclipse-IDE (no version specified), to be manually installed. We assessed the plug-in against a freshly installed version of the Eclipse IDE. However, the provided plug-in seems incompatible with the last version of Eclipse. For this reason, we did not assess the technology any further.

3.3.2 Situated interaction & coordination

Situated ReSpecT (2008)

Primary references: [64,65]
Other references describing / exploiting the technology: [264,265,275]
URL: http://respect.apice.unibo.it

MAS perspective The ReSpecT event model—which already accounts for situatedness issues by defining an event in terms of its source cause, event target and time at which the event occurred—has been extended in Situated ReSpecT introducing generic environmental resources as a new kind of sources / targets of events so as to account for situatedness and to coordinate a system for sensing and controlling environmental properties.

Logical perspective Situated ReSpecT extends the ReSpecT coordination language for programming tuple centres to govern interactions between agents and environment.

Technological notes: The Situated ReSpecT technology is currently integrated with the aforementioned TuCSoN technology.

3.3.3 Semantic web & situated interaction and coordination

LPaaS (2018) & LVLP (2016)

Primary references: [59,60]
Other references describing / exploiting the technology: [61,274]
URL: http://lpaas.apice.unibo.it

MAS perspective LPaaS [60] and LVLP [59] are two models that—once specialised in the MAS context [274]—answer the need of intelligent environment and situated distributed cognition. LPaaS allows logic-based services to be embedded in the environment and made available to agents *as a service*. LVLP exploits the notion of *labelled variable* to allow for domain-specific logic-based computations. Altogether, LPaaS & LVLP models and architectures—and the corresponding technology—support the distribution of (micro)intelligence in pervasive MAS [274].

Logical perspective LPaaS in the MAS context exploits logic programming (and Prolog in particular) for the distribution of intelligence chunks placed where and when needed to locally tackle the specific reasoning needs in complex distributed systems.

Technological notes: The LPaaS technology consists of a web service implemented in Java and requiring the JVM 8+ platform. Currently, the service is not deployed on a ready-to-use web page. In fact, only the source code is provided, by means of a GitLab repository. However, the technology can be launched through the provided build automation system. We successfully ran the LPaaS service following the instructions provided.

4 Analysis

4.1 Cloud of words

Figure 1 shows the word cloud generated from all the 271 papers in the survey. In order to give some meaning to this obviously limited, yet synthetic view of the literature, the ten most evident words in the cloud could be divided into three subcategories:

- *agent*, *action*, *plan*, *goal*,
- *model*, *logic*, *rules*, *state*, *belief*,
- *language*, *program/programming*.

In short, the picture could be interpreted as suggesting that logic-based technologies in MAS are mostly exploited to make agents act, plan, and achieve goals based on the expressive power of logics in modelling knowledge into rules, states, and beliefs, by providing a well-founded basis for programming MAS via logic languages.

The cloud also highlights the relatively-high frequency of other words, related to the *model* one, such as: *semantics*, *constraints*, *properties*, *ontology*, *variables*, *conditions*, *functions*—all of them basically reinforcing the (quite obvious) idea that logic can be essential in formalising MAS and their properties. Further scrutiny reveals that terms *time*, *environment*, and *event* also appear quite often. This hints at the close relationship between agents and environment, and with the events that occur over the time—typically modelled in terms of logics such as temporal logic.

Fig. 1 The cloud of words obtained by the main keywords extracted from the papers subject of the SLR

Other interesting words emerging from the cloud are *information*, *reasoning*, and *intelligence*, possibly reflecting one of the main purposes of the logic-based MAS technologies—namely, the design and development of intelligent MAS in knowledge-intensive systems. Finally, words like *social*, *interaction*, *protocol*, and *communication* highlight the social component of MAS as well as its connection with logic-based technologies.

4.2 Selected technologies: MAS & logic perspectives

In this subsection we further analyse the technologies selected by our SLR—reported in Table 1 in a joint MAS/logic view—, in order to provide additional information about the five research questions—from (Q1) to (Q5).

The most common usage of logic in MAS relates to agents programming and their reasoning/planning capability. Indeed, Table 1 also suggests that logic-based languages have the potential to play a prominent role as intelligence providers: the typical LP features—e.g., programs as logic theories, computation as deduction, programming with relations and inference—make logic languages a straightforward choice for building intelligent components.

A more detailed observation of Table 1 shows the MAS sub-areas where logic-based technologies are extensively exploited are: (i) verification of both agent behaviour and social

Table 1 Logic-based MAS technologies: joint MAS/logic view

		Agent			Society			Environment	
		Programming, reasoning	& planning	Agent reliability & Verification	Organisational reasoning & coordination	Argumentation	MAS reliability & verification	Agents in the semantic web	Situated interaction & coordination
	Restricted	PDDL							
	Predicate	Rodin							
	Logic	SHOP							
Logic programming		DALI			2CL		Trace Expr	LPaaS & LVLP	
		DCaseLP			AORTA				Situated respect
		eXAT			TuCSoN				
	"Standard"	Go!			Respect				
		JADL							
		Teleo-R							
	Situation	ConGolog							
	Calculus	Golog					SALMA		
		IndiGolog							
	ALP				ALIAS		SCIFF		
	ASP	RASP				Aspartix			
Modal logic		2APL/3APL		MCAPL					
		ASTRA							
		GOAL							
	BDI	Jack							
	Logic	Jadex							
		Jason							
		Jason(ER)							

Table 1 continued

	Agent			Society			Environment	
	Programming, reasoning	& planning	Agent reliability & Verification	Organisational reasoning & coordination	Argumentation	MAS reliability & verification	Agents in the semantic web	Situated interaction & coordination
Modal Temporal Logic	AGENT-0					MCMAS		
	METATEM							
Modal Epistemic Logic						MCK		
Deontic & defeasible logic	jDALMAS				Aspic+			
					DeLP			
					Spindle			
Description Logic	DRAGO						AgentOWL	
							EMERALD	
							Onto2Jacamo	

interaction (or MAS in the whole), and (ii) argumentation—the table only reports on the main "leading technologies" for each argumentation model.

For the former, logic-based techniques are widely recognised to be among the leading technologies for the verification of reactive systems, thanks to properties such as a solid logic ground to build upon, the possibility of providing interactions and communication with both a declarative and an operational semantics, and the chance to define logic-based integrity constraints. For the latter, as far as argumentation is concerned, although all the approaches implement defeasible logics (often with deontic extensions), a leading technology providing a shared set of abstractions is still missing, and each solution is instead typically tailored to a specific application domain. Generally speaking, the potential of logical approaches in norm-based systems and conflict resolution is widely acknowledged—also in terms of explainability of complex MAS.

As far as the environment modelling and engineering is concerned, most technologies target knowledge representation and, in particular, description logics. This is likely due to the environment being the most proper abstraction to encapsulate stateful (and semantic) information, and to work as a support for agent distributed cognition process in knowledge-intensive systems.

(Subsets of) predicate logic, logic programming, and action languages (e.g., situation calculus) have been mostly exploited for behavioural aspects of agents—most likely due to their natural procedural interpretation. On the other side, modal and epistemic logics have been mostly adopted in model-checking and verification technologies, presumably because they make it possible to express meaningful, desired specifications about what agents or societies do or know, while taking time into account.

Also, and quite naturally, one of the most successful logic-based architecture for agents—the BDI one—has never been exploited outside the boundaries of individual agents. This is likely due to the lack of intuitive interpretation of the desire and intention abstraction for both environments and societies.

As far as the simulation tools for MAS are concerned, the SLR also points out that they often do not rely upon a formal logic-based framework: the logic is instead exploited to define the operational semantics of the simulation, or, to express temporal/epistemic specifications in the case of model-checking joined to the simulation process—as in the case of SALMA. This is why we only included in the SLR the few simulation frameworks with a strong logical characterisation; other simulation frameworks exist, such as LEADSTO mentioned in Table 4, but an actually-downloadable implementation could not be found for them—and this was the reason for their exclusion.

The final remark is related to the lack of MAS technologies based on fuzzy or probabilistic logic. This is somehow surprising, given the huge interest of the MAS community towards Bayesian or Markovian approaches. Many theoretical works can be found in this area, yet none comes with a technology that is actually implemented and alive.

4.3 Technological analysis

In this subsection we deepen the analysis of the technologies selected by our SLR in order to provide additional insights about research questions (Q1), (Q3), and (Q4). Table 2 provides an overview of our analysis.

There, we analyse the selected technologies according to a number of technical dimensions, corresponding to the columns of Table 2. In particular: column **Fresh.** assesses the *freshness* of each technology, by indicating—when possible—the year of the visible update;

Table 2 Overview on the selected technologies and their analysis

Name	Fresh.	Code	Doc.	License	Target	Runs	Benchmark	Works
2APL	2012	–	3	None	JVM 6, 7, 8	Yes	*.mas	Yes
2CL	2013	–	1	None	JVM 7	Yes	*.2cl	Yes
3APL	2007	–	4	None	JVM 6	No	–	–
Agent-0	1993	1	1	None	(Allegro) Common Lisp	Yes	None	No
AFAPL2	2016	3	–	LGPL v2	JVM	?	None	?
AgentOWL[a]	2013	3	1	LGPL v2	JVM 8, 11+	Yes	Compiled demo	No
AIL	2018	3	1	LGPL v3	JVM 8+	Yes	*.ail	Yes
ALIAS[b]	2000	1	1	None	JVM | Prolog	No	–	–
AORTA	2015	4	–	None	JVM +7	Yes	*.ail	Yes
ASPARTIX[c]	2018	–	1	MIT	Web service	Yes	*.dl	Yes
ASPIC	2019	4	3	LGPL v3	JVM 12+	Yes	Compiled demo	Yes
Astra[e]	2018	3	5	GPL v3	JVM	Yes	Provided examples	Yes
ConGolog	–	–	–	?	?	?	?	?
DALI	2018	3	1	Apache v2	SICStus	Yes	?	?
DCaseLP/DyLog	2005	2	3	None	JVM	No	–	–
DeLP[d]	2018	4	3	LGPL v3	Web service	No	–	No
DRAGO	2006	2	2	None	JVM 5+	Yes	Any OWL ontology	Yes
EMERALD[a]	2010	–	3	None	JVM 6+	Yes	Compiled demo	Yes
eXAT	2012	4	3	GPL v3	Erlang	No	–	No
Go!	2015	2	–	GPL v2	?	?	?	?
GOAL[e]	2019	5	4	GPL v3	JVM 8	Yes	Template projects	Yes
Golog	1998	1	–	Ah hoc	SWI | ECLiPSe	Yes	–	?
IndiGolog	–	–	–	?	?	?	?	?
JACK	–	–	5	Proprietary	JVM	?	?	?
Jadex	2020	5	5	GPL v3	JVM 8+	Yes	Compiled demo	No
JADL/JIAC	2018	4	5	Apache v2	JVM 7+	Yes	Provided examples	Yes

Table 2 continued

Name	Fresh.	Code	Doc.	License	Target	Runs	Benchmark	Works
Jason	2020	5	5	LGPL v3	JVM 8+	Yes	Provided examples	Yes
Jason-ER	2019	5	–	LGPL v3	JVM 8+	Yes	Provided examples	Yes
jDALMAS	2016	3	–	None	JVM	No	–	–
LPaaS	2018	5	3	Apache v2	JVM 8+	Yes	None	?
MCAPL	2018	3	1	LGPL v3	JVM 8+	No	*.ail	No
MCK	–	–	3	None	Web service	Yes	Provided examples	Yes
MCMAS	2017	4	3	None	Native/Linux	Yes	From manual	Yes
Mozart	2019	5	5	Ad-hoc	Native	Yes	Provided examples	Yes
MetateM	2010	2	3	GPL v3	JVM 6+	Yes	*.sys	Yes
Onto2Jacamo[e]	2017	–	1	None	JVM	No	–	–
Raspberry	–	–	1	None	Native/Linux	Yes	*.rasp	Yes
RML	2019	5	3	Apache 2	JVM 8+	Yes	*.rml	Yes
Rodin	2020	4	?	None	JVM 8	Yes	None	?
SALMA[f]	2016	3	1	None	Python	No	–	–
SCIFF	2008	1	2	None	SWI | SICStus	Yes	None	?
SHOP	2020	4	2	MPL	Common Lisp	Yes	?	?
Spindle	2017	2	4	GPL v3	JVM 7+	Yes	*.dfl	Yes
Teleo-R/QLog	2019	3	3	None	QuProlog	No	–	–
TuCSoN/Respect (and variants)	2020	3	3	LGPL v3	JVM 8+	Yes	Compiled demo	Yes
TuSoW	2020	5	–	Apache 2	JVM 8+	Yes	None	?

Dashes represent missing values, whereas question marks represent unknown values
[a]Depends on some ancient Jade version, which is not provided
[b]Requires the Jinni Prolog interpreter for Java
[c]Requires the Clingo solver
[d]Depends on ancient tuProlog version, which is not provided
[e]Is a plug-in for (or customisation of) the Eclipse IDE
[f]Requires ECLiPSe Prolog and the PyCLP Python module

column **Code** provides a qualitative assessment of each technology *code base*, when available; column **Doc.** provides a qualitative assessment of each technology *documentation*, when available; column **License** indicates under which license each technology is provided, if any; column **Target** reports on the target *runtime platform(s)* each technology can be executed upon; column **Runs** shows our success in *executing* each technology—or loading it into its target runtime—, possibly after any necessary compilation step; column **Benchmark** points out whether each technology is distributed with some *benchmarks* or *examples*; column **Work** points out whether each technology *works*, i.e., if the aforementioned benchmarks / examples can be executed successfully.

More precisely, the freshness of a technology is defined by the *year* of its most recent release, or, source code modification. The code-base assessment consists of a (possibly missing) integer number ranging from 1 to 5:

- "1" means the code base is available as a bare archive, even if it appears to have no clear file organisation, and it is comes with no facility supporting the compilation (or, in general, usage) of the code;
- "2" means the code base is available as a bare archive, and it adheres to a well organised structure or it includes instructions on how to compile/use it
- "3" means the code base is available through some version control system[17] (VSC, henceforth), but poor support is provided for compilation or usage;
- "4" means the code base is available through some VCS and it comes with some build automation tool[18] as well, supporting compilation or usage;
- "5" means the code base is available through some VCS, it comes with some build automation tool, and it is also distributed through some official repository;

whereas a missing value denotes that no assessment can be drawn given the available information—e.g., because we were not able to access any code base.

Similarly, the documentation assessment consists of a (possibly missing) integer number ranging from 1 to 5 as well. In this case, however:

- "1" means that the only available form of documentation is some textual note briefly describing the technology or how manage the codebase;
- "2" means that some *structured* form of documentation exists describing the technology or how manage the codebase;
- "3" means that a *detailed manual* or some *API reference* are available, but *not both*;
- "4" means that *both* a detailed manual and some API reference are available;
- "5" means that a detailed manual, some API reference, and some *usage examples* or *tutorials* are available;

whereas a missing value denotes the total lack of any form of documentation.

Whenever available, the license of each technology is referenced in Table 2 as well, possibly leveraging on well-known license acronyms. For instance, as far as open source licenses are concerned, "GPL" refers to the GNU General Public License,[19] "LGPL" refers to the GNU *Lesser* General Public License,[20] "MIT" refers to the MIT License,[21] "MPL" refers to

[17] e.g., SVN, Git, Mercurial, etc.

[18] e.g., Make, Apache Ant, Apache Maven, Gradle, Pip, Npm, etc.

[19] https://www.gnu.org/licenses/gpl-3.0.html, last accessed in April 2020.

[20] https://www.gnu.org/licenses/lgpl-3.0.html, last accessed in April 2020.

[21] https://opensource.org/licenses/mit-license.php, last accessed in April 2020.

the Mozilla Public License,[22] whereas "Apache" refers to the Apache License.[23] The absence of licenses for a particular technology is pointed out as well, as it may have an impact on its users.[24]

The target runtime platform is another relevant aspect we analyse for each technology. It provides an intuition of which sorts of machines and devices could in principle be capable of running a given technology. As it clearly emerges from Table 2, the JVM platform is targeted by most technologies. This is why, in the particular case of JVM-based technologies, we try to assess the specific version(s) of the JVM they can run upon. Thus, the "JVM *N*+" notation indicates that a given technology is tested on all JVM versions ranging from *N* (included) to version 13 (included)—which is the most recent one at the time of writing—, and only executes without errors starting from version *N*. Of course, the JVM is not the only platform our selected technologies leverage upon. Some technologies require a compilation step targeting some *native* platform. So, for instance, in case only the *OS* operative system is supported, we write "Native / *OS*" to identify the target platform. Other technologies target the Common Lisp, Python, or Erlang runtimes, which come with several implementations supporting mainstream operative systems, similarly to what JVM does. There exist also technologies targeting specific, well-known, implementations of Prolog, such as SWI-Prolog, SICStus Prolog, or ECLiPSe-Prolog. In this case we simply write a short indicator of the target Prolog implementation ("SWI", "SICStus", or "ECLiPSe"), or some piped-separated combination of two or more indicators, in case more than one implementation are supported. Furthermore, a few technologies are available as web services. In those cases, we argue that the actual platform of the service implementation is not essential—this is why we simply denote the target platform as "Web Service". Finally, there are some technologies which are explicitly aimed at extending (or customising) the Eclipse IDE,[25] and are not meant to be used otherwise. In those cases, we indicate "JVM" as the target platform, and tag the technology through an *ad-hoc* footnote.

In order to test whether a technology runs or not, we simply launch it on its target platform—possibly, after performing all necessary compilation/configuration steps. If neither compilation/configuration nor launching produces error or crash, then we say the technology runs, otherwise it does not. Thus, a question mark in Table 2 in the **Runs** column may indicate either the total lack of any information on how to launch the technology, or, the impossibility of producing/accessing an executable to launch.

Finally, when we are able to run a given technology, we then test it to get further detail, so as to understand if it actually *works* or not. To do so, we first look for available *benchmarks* or examples into the running technologies code bases, documentation, or home pages. In case some benchmarks/example are available, we check if they can be run without producing error outputs or crashes. If they can, then the technology works, otherwise it does not. Thus, a missing value in Table 2 in the **Benchmark** or **Works** columns indicates that no assessment is needed because the technology does not run. Conversely, a question mark in the same columns denotes the impossibility to perform any further assessment due to lacking benchmarks, examples, or instructions on how to launch them.

By aggregating the data in Table 2, we can draw several interesting conclusions. Most relevant ones are summarised in Table 3. The most evident information is that only 12 out of 47 technologies have been actively maintained since 2019—i.e., 25.53% of the total. The

[22] https://www.mozilla.org/en-US/MPL/2.0, last accessed in April 2020.

[23] https://www.apache.org/licenses/LICENSE-2.0, last accessed in April 2020.

[24] https://choosealicense.com/no-permission, last accessed in April 2020.

[25] https://www.eclipse.org/ide, last accessed in April 2020.

Table 3 Statistics on selected technologies

	Absolute	Relative (%)	Meaning
N. selected tech	47	100	
Maintained since 2019	12	25.53	Fresh. $\geq$ 2019
Maintained since 2018	20	42.55	Fresh. $\geq$ 2018
Open source	26	55.32	License $\notin$ {none, ?}
Unlicensed	19	40.43	License = none
JVM-based	30	63.83	Target starts with JVM
Certainly runs	31	65.96	Runs = yes
Certainly works	21	44.68[a]	Works = yes
Codebase quality	16	34.04	Code $\geq$ 4
Documentation quality	9	19.15	Doc. $\geq$ 4

[a]Corresponding to 67.74% of the technologies which certainly run

percentage is lower than 50%, even if we enlarge the spectrum to technologies maintained since 2017. However, most technologies (65.96%) can still be run successfully—regardless of when they were last updated—, even though we are able to make them work in 67.74% of cases only.

Another interesting trait is that—except for JACK—all technologies that are explicitly licensed come with an open source license. These correspond to 55.32% of the total. The amount of unlicensed technologies is quite high as well, as it corresponds to the 40.43% of the total.

It is interesting to note how the JVM is by far the preferred platform for logic-based MAS technologies. Indeed, 63.83% of the selected technologies target some version of the JVM. Other recurring platforms are Lisp-, Prolog-, or native-based.

Finally, it is worth to be mentioned how—except for a few notable exceptions, such as Jason—poor care is given to technologies code bases and documentary resources. Indeed, considering the 1–5 ranges defined above, only 34.04% of the technologies come with a code base whose quality is greater than 3, whereas only 19.15% are scored similarly as far as documentation is concerned.

4.4 Rejected technologies

For the sake of completeness, Table 4 summarises the main technologies that were excluded from our SLR, along with the corresponding motivation. In particular, the table shows the instantiation of the criteria for MAS technologies inclusion defined in Sect. 2.1.

Let us recall the inclusion criteria we adopted in this SLR. Contributions are considered as logic-based MAS technologies if *all* the following conditions simultaneously hold:

(a) they can be clearly related to some well-known and shared definition of agent and MAS—e.g., [280,356];
(b) they stick to some logic which is clearly identifiable in the literature, in terms of name, formal definition, and reference paper(s);
(c) they are both scientifically identifiable (reference paper) and technically available (i.e., some software actually exists).

Table 4 Main technologies excluded from the survey, along with the motivation spanning between (i) "general purpose", (ii) "no logic", (iii) "technology not found"

Motivation for exclusion	Technology
General purpose	4QL [243], CCALC [163], CDF-Rules [164], Clingo [292], DR-Prolog [37], Inter4QL [199], InterProlog [62], Jess [177], JPF [170], KQML [143], Maude [86], MOCHA [7], OWL & SWRL [191], Pellet [321], PRISM [219], Progol [261], Protege [162], SICStus [63], Smodels [331], sparql [210], SPIN & Promela [188], SWI Prolog [352], TRACK–R [159], tuProlog [122], Xsb [309], YAP Prolog [93]
No logic	AGENTFLY [322], Aglets [221], CArtAgO [298], ConArg [38], cougaar [173], JaCaMo [41], JADE-JBossESB [346], JACKAL [92], Klaim [115], Magentix2 [328], MASON [238], Moise & S-Moise+ [193], Presage2 [71], RETSINA [329], SARL APL [302]
Technology not found	ABLE [349] , AgentStra [229], Ameli [137], ANA (Automated Negotiation Agent) [214], AOSDE [316], April [251], Argonaut [103], ARTIMIS [51], Baop [106], Camus [230], CASL (cognitive agents specification language) [315], CASP (Checking AgentSpeak Program) [43], Caso [107], Cast [359], CLAIM [132], CLELIA [14], CooL-AgentSpeak [249], Cupid [74], Dare [241], DASMAS, [282] DeLP-MAPOP/DeLP-POP [141], DESIRE [50], DIPLoMAT system [242], dMARS [126], Dribble [300] DR-NEGOTIATE/DR-DEVICE [323], EVOLP [324], Gama [222], Golem [53], IMPACT [13], InfoSleuth [32], Instql [189], ISLANDER/AMELI [136], JAPL [16], JADEL [35], JASDL [207], Jinni [332], iJADE stock advisor [302], KARO [195], KGP[308], Language-a [161], LEADSTO [48], LAIMA [345], MABLE [358], MAGE [317], MAGENTA [178], Maglog [260], MALLET [138], MANCaLog [314], Mobile Agent Reactive Spaces (MARS) [58], MiLog [153], Minerva [224], Mole [31], n-Jason [223], NegoPlan [250], nomprol [90], OMNI [125], OntoWEDSS [67], OPAL [270], OperA / Operetta [5], Ora4mas [194], Orwell, [118] OSCAR [290], PACT [102], Pclaim [169], PERSUADER [330], PLACA [337], PROSOCS [326], Rasa [181], RP-DeLP [6], Seagent [168], Semantic TuCSoN [265], Siebog [342], SOCS [339], Sonar [209], SyMPA-(CLAIM) [311], Tal [89], VerICS [268]

The aforementioned criteria find their reification into three main motivations for rejection, described below:

general purpose—the contribution may be exploited or referenced in other MAS-related (resp. logic-related) works or technologies, but it is not explicitly intended to target them;
no logic—the contribution does not stick to any clearly identifiable logic from the literature;
technology not found—no actual software resource, no evidence found that some close-source software actually exists.

Each one of the motivations alone is a sufficient condition for exclusion—so that a contribution is excluded as soon as one of the above conditions is met. So, for instance, all general-purpose Prolog engines (e.g., SICStus, SWI-Prolog, tuProlog, or ECLIPSE-Prolog) are excluded, as they do not intentionally refer to any specific MAS abstraction—criterion (a).

Also, JaCaMo [41] is excluded according to criterion (b). JaCaMo[26] builds on top of Jason, and integrates Jason Moise [193] for programming agent organisations, and CArtAgO [298] for programming shared environments. However, among these technologies, only Jason is

[26] http://jacamo.sourceforge.net, last accessed in April 2020.

logic-based: CArtAgO and Moise are Java-based technologies whose only connection with logical approaches is via their Jason wrappers. This is why, we choose to include Jason in our technological overview, while excluding JaCaMo, CArtAgO and Moise, despite the corresponding technologies are alive and functioning.

Finally, some well-known logic-based technologies are excluded according to criterion (c), as no actually-downloadable version of the tool could be found—for public link missing, dead link, or no download available. Among the others, technologies such as JASDL, n-Jason (missing link to the corresponding technology) and VerICS and SOCS (dead link) are notable examples of technologies excluded for that specific reason. At last, it is worth to be mentioned that a number of works concerning social commitments have been excluded as well, for similar reasons. The Cupid [74] model, in particular, is a notable example. Given their importance, however, a few comments about their absence are due, as it may seem astonishing. In fact, despite these are well-founded and impactful theoretical works, we were not able to find any runnable technology available to date, neither in the corresponding papers nor on the Web. This is the only reason why these models are mentioned in this section.

5 Discussion

The main question behind this work is (G) "What is the role of logic-based technologies in MAS nowadays?"—as mentioned in Sect. 3. To properly answer this question, we identify five more specific research questions—namely, (Q1), (Q2), (Q3), (Q4), and (Q5)—, and we deeply and systematically inspect the literature in order to support a detailed answer for each of them. In this section we further discuss the contribution from Sect. 3, along with the tables and data from Sect. 4, so as to summarise a brief and explicit response to each research question.

Generally speaking, the fundamental role of logic-based technologies in MAS nowadays is to address the need for intelligence that characterises agent-based abstractions—i.e., the cognitive abilities required by distributed intelligent system components. However, logic-based technologies can seldom be considered to be mature enough to tackle the requirements of industrial and real-word domains. In the following paragraphs, we deepen the discussion by focusing on the specific research questions.

(Q1) Which logic-based technologies for MAS are actually available? The available logic-based technologies for MAS are listed in Table 2. Among the 47 technologies we assess, 21 (i.e., 44.68%) are still alive and working. However, only 12 of them (i.e., 25.53%) show recent updates (since 2019). Also, in terms of the overall assessment of the technology—based on the criteria detailed in Sect. 4.3—, one may notice that only a relatively-small number of them meet basic technological standards.

(Q2) Which aspects of MAS technologies are affected by logic-based technologies, and to what extent? (Q3) Which MAS abstractions / issues / features are covered by logic-based technologies? As summarised by Table 1, logic-based technologies cover almost all the existing MAS abstractions, with a heterogeneous distribution.

In particular, 50% of technologies focus on *agent* programming, exploiting logic to describe the agent behaviour (at an adequate level of abstraction), as well as their goals and knowledge—often expressed through beliefs, desires, and intentions.

With respect to the *society* abstraction, there are about 10 technologies identified by this study. Among them, TuCSoN is arguably the most mature one. In TuCSoN, logic is exploited to

address coordination issues, as well as reasoning tasks. Other technologies that address agent societies issues are mostly related to defeasible reasoning and, in particular, argumentation. However, our assessment reveals that most of them are not mature enough for distributed MAS environments, although exceptions exist—e.g., ASPIC+. This line of research remains highly promising. It is worth noting that the adoption of a logic-based approach for argumentation purposes has many advantages, which all lead towards explainable systems.

The *environment* abstraction is the least affected by logic-based technologies: only few technologies deal with MAS environment and adopt logic at the same time. In the few identified technologies, logic is often exploited with the purpose of making the environment intelligent, or, to represent distributed knowledge, possibly exploiting ontologies. However, none of them can be considered actually ready to match the needs of real-world application domains. This is why this research line appears promising and deserves further attention.

Finally, many logic-based formal methods are used for MAS engineering, as well as for model checking and verification [355]. The corresponding technologies found in the SLR are reported under the "MAS/Agent Reliability & Verification" column of Table 1. However, more often than not, the corresponding technologies appear poorly maintained, thus revealing another potentially productive area for applied research.

(Q4) Which sorts of logics are actually exploited to frame logic-based MAS technologies? Most of the selected logic-based MAS technologies are related to logic programming—likely because of its wide diffusion, simplicity, and adaptability—or to the BDI logic—both for the cognitive abstractions that straightforwardly match the needs of rational gents, and for the vast amount of contributions and results on the topic by the MAS community. Furthermore, defeasible and deontic logics are exploited for dealing with uncertain or defeasible knowledge, as well as to address normative concerns. Finally, description logics are widely used in order to represent knowledge of the agents and of the whole system. It is worth noting that some logics, such as fuzzy logic, show great potential for dealing with uncertainty in MAS: nevertheless, none of the research works surveyed actually translate their theoretical contributions into some usable technology.

(Q5) Which logic-based technologies for MAS are effectively used in real-world domains? Among the 47 technologies surveyed, evidences of non-trivial real-world applications can only be found for IndiGolog, JACK, JIAC, and Agent Factory (AFPL). In particular: (i) IndiGolog has been used for programming robot controller, workflow management systems, and web service composition systems [46]; (ii) JACK has been used for building autonomous unmanned air vehicles and manufacturing systems, for simulating the human behaviour, or, for implementing decision support systems [353]; (iii) JIAC has been used in a number of projects that span from simulation and service execution to energy control and home automation [46]; (iv) AFPL has been used for the development of applications in various domains, including e-commerce, ubiquitous computing, mobile computing, robotics, wireless sensor networks, and mixed reality [46]. However, it is worth recalling that only JACK and JIAC are still actively maintained, as highlighted in Sect. 4.3.

Nevertheless, we believe that the amount of mature technologies that are potentially ready to be used in real-world domains is larger than those currently exploited—indeed, the analysis above is intentionally limited to the real-world applications explicitly described in the articles included in the SLR. By considering as mature technologies all the technologies in Table 2 that (i) have been maintained since at least 2 years, (ii) have a scored of at least 3 in their source code and documentation evaluation, and (iii) are still working, the set of *mature technologies*

can be extended to include also ASPIC+, ASTRA, GOAL, Jadex, Jason, MCMAS, Mozart, RML, and TuCSoN.

Final and general remarks This work is motivated by a broader research question—namely:

> Given the new challenges that are opening up in the field of intelligent systems and AI for logic-based technologies, do there exist technologies that can be considered ready enough? If not, what is missing?

To answer this question, we should first recall that modern intelligent systems:

(i) are deeply intertwined with domains like the Internet of (Intelligent) Things (Io(I)T) and Cyber-Physical Systems (CPS);
(ii) are therefore inherently distributed, thus demanding for robustness, efficiency, interoperability, portability, standardisation, situatedness, and real-time support;
(iii) need to reconcile and synthesise symbolic and sub-symbolic AI, exploiting the former to *explain* the latter so as to overcome fears and ethical issues posed by AI by providing for explainability, observability, interpretability, responsibility, and trustability—the scope of XAI.

In the remainder of this section we focus on the issues above, and extract further considerations from the data collected in Sect. 4.

Applicability to distributed domains such as IoT and CPS Often, the existing agent solutions applicable to the IoT and CPS are available only for a specific and limited set of devices. For instance, Agent Factory Micro Edition (AFME) [262] enables the execution of a deliberative agent on top of mobile phones with CLDC/MIDP profiles and Sun-SPOT sensor by means of TCP/IP and Zigbee protocols.

However, some technologies are, more than others, explicitly designed to support IoT domains and CPS. For example, the LPaaS architecture is designed to promote distributed intelligence for the IoT world—offering logic programming in terms of service, and explicitly addressing the requirements and issues of cloud and edge architectures. It exploits the Everything as a Service [27] metaphor to maximise availability and interoperability while promoting context-awareness. From the MAS point of view, LPaaS takes care of distributing knowledge as well as reasoning capabilities placed in the agent environment. Analogously, the situated coordination approach promoted by the TuCSoN/ReSpecT model and technology can be explicitly exploited to handle situatedness in MAS as a coordination issue. Also, TuCSoN provides the main abstractions for IoT environments: environmental resources can be sources of perceptions (like sensors), targets of actions (like actuators), or even both.

Finally, there are technologies that are not explicitly meant to address the IoT and CPS domains, but still let us suppose they would be easily portable to those domains—because of their standard compliance, interoperability, and portability features. Among the many, Jason supports interoperability with non-Jason agents via JADE through FIPA-ACL communication. Similarly, there are extensions to JACK that make it work in open systems. Finally, the Teleo-Reactive approach has been often exploited to facilitate the development of the IoT systems as a set of communicating Teleo-Reactive nodes. The software behaviour of the nodes is specified in terms of goals, perceptions, and actions over the environment, achieving higher abstraction than using general-purpose programming languages and therefore, enhancing the involvement of non-technical users in the specification process.

Symbolic and sub-symbolic integration With respect to the second requirement of AI—i.e., the need to reconcile and integrate symbolic and sub-symbolic techniques—none of the selected technologies has been experimented yet. However, we argue that portable and interoperable technologies might be more suitable for the integration. Anyway, the field is still unexplored and represents a frontier research domain.

Can existing technologies be labelled as ready? If not, what is missing? To recap, the role of logic-based technologies in MAS nowadays exhibits a huge potential for covering the vast majority of intelligent system abstractions. However, just a few among the technologies surveyed can be actually labelled as *ready-to-go*, in particular when considering the new challenges for symbolic technologies in AI.

In fact, even though 10% of the selected technologies can be considered as mature—in terms of cross-platform support, code quality, and ease of distribution in heterogeneous environments—, they often have not been tested in pervasive and real-world scenarios, yet. This implies, at least, that further research is required to ensure any technological barriers can be overcome. Furthermore, integration with sub-symbolic techniques remains a *nice-to-have* feature, but it is not actually a thing in any MAS technology, for the time being.

Nevertheless, the selected technologies are an excellent starting point for (i) highlighting the advantages of logic-based technologies, and (ii) broadening the scope of research towards the directions envisioned.

6 Conclusion

SLR originated in the medical research field [176], first of all as a tool to make some sense over the heterogeneity of methodologies, tools, scope, and size, which were severely affecting the overall coherence of the research results (not just methods) in most areas, there. SLR have spread to other research fields, too: in the ICT areas, some of the first SLR can be found in the software engineering field [206], for instance.

Whereas one might argue against the actual need of adopting the SLR approach everywhere—for instance, in the MAS field—yet it should be observed that the ever-growing articulation of most areas in computer science and AI in the last decade has led to a huge popularity and diffusion of surveys—on any possible topics, also given the global popularity of AI and CS nowadays. Most of the surveys around, however, while useful and interesting, typically suffer from the authors' bias [206], in particular when large research topics are addressed: as such, they typically fail in the essential aspects of *reproducibility* that should characterise any scientific work [291]. This is where SLR comes at a hand: by providing a technical tool to produce surveys that can be verified and reproduced, since it requires to make the methods used to build the survey explicit and repeatable.

Given the recent popularity of AI, and the potential role that logic-based technologies for MAS could play in the engineering of complex intelligent systems, we think that a SLR targeting MAS technologies based on any sort of logic could be of wide interest. Along this line, this paper presents a SLR on logic-based technologies for MAS.

By logic-based technologies for MAS we mean any agent-oriented software architecture, framework, or language (i) involving some clearly-defined logic model, and (ii) coming with some actual technological reification.

The paper follows the standard SLR method: we carried out a manual retrieval, filtering, analysis, and categorisation of huge number of papers, resulting in 271 documents, retrieved by repeating 8 queries on 6 electronic search engines (Google Scholar, IEEE Xplore, Sci-

enceDirect, SpringerLink, DBLP, ACM Digital Library) and 5 specific conference/workshop proceedings. Whereas search engines are chosen to make our survey as comprehensive as possible, queries are precisely designed to capture meaningful works w.r.t. the main goal (G) of our SLR, other than the 5 specific research questions it has been split into—namely, (Q1) up to (Q5). Our research questions aims at inspecting (i) which logic-based technologies for MAS are available today, (ii) what is the role of logic in these MAS technologies, and (iii) what is the technologies current state and maturity,

The methodological approach and the inclusion, exclusion, and analysis criteria adopted here are carefully designed and described in detail, keeping a tight focus on the *reproducibility* of the whole process. In particular, we only include works defining or exploiting logic-based MAS technologies according to the above definition and for which the existence of some software reification can be proven.

The technologies resulting from this systematic exploration are analysed and assessed from two different perspectives—namely, the MAS one and the logical one—, thus discussing which specific MAS- and logic-related aspect is tackled or exploited by each technology. As suggested by Omicini [273], we categorise the selected technologies on the three main abstractions in MAS—namely, agents, societies, and the environment—thus revealing an uneven distribution of logic-based technologies on MAS abstractions, and highlighting research opportunities on less popular abstractions—as the environment one.

We also perform a technical assessment of each technology, which enables a detailed discussion on the current state of logic-based MAS technologies. The outcome of the discussion highlights that, as far as logic-based technologies for MAS are concerned—except for some rare important success stories—there is still room for technological advancements. Quite often, in fact, despite the enormous technological effort clearly carried out by the MAS community in the last decades, the technologies cannot be considered mature and ready for use in the new challenging contexts required by AI. Our SLR identifies the most promising research areas, as well as the areas that mainly need technological progress.

Because of the scientific, well-founded, and reproducible approach of this SLR, we believe that it represents a reliable tool to assess the current state of the topic, and we hope it can be used to understand the future directions in this area.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement. This work has been partially supported by the H2020 Project "AI4EU" (G.A. 825619). One of the authors, Roberta Calegari, has been supported by project "CompuLaw", funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (G.A. 833647).

References

1. Abiteboul, S., & Hull, R. (1988). Data functions, datalog and negation (extended abstract). In H. Boral & P. Larson (Eds.), *International conference on management of data* (pp. 143–153). Chicago, IL: ACM. https://doi.org/10.1145/50202.50218.

2. Abrial, J. R., Butler, M., Hallerstede, S., Hoang, T. S., Mehta, F., & Voisin, L. (2010). Rodin: An open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, *12*(6), 447–466. https://doi.org/10.1007/s10009-010-0145-y.
3. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2005). The *S*CIFF abductive proof-procedure. In S. Bandini & S. Manzoni (Eds.), *AI*IA 2005: Advances in artificial intelligence: 9th Congress of the Italian Association for Artificial Intelligence, Milan, Italy, September 21–32, 2005. Proceedings, Lecture notes in artificial intelligence* (Vol. 3673, pp. 135–147). Berlin: Springer. https://doi.org/10.1007/11558590_14.
4. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., & Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Transactions on Computational Logic (TOCL)*, *9*(4), 29:1–29:43. https://doi.org/10.1145/1380572.1380578.
5. Aldewereld, H., & Dignum, V. (2011). OperettA: Organization-oriented development environment. In *Languages, methodologies, and development tools for multi-agent systems, Lecture notes in computer science* (Vol. 6822, pp. 1–18). Berlin: Springer. https://doi.org/10.1007/978-3-642-22723-3_1.
6. Alsinet, T., Béjar, R., Godo, L., & Guitart, F. (2014). RP-DeLP: A weighted defeasible argumentation framework based on a recursive semantics. *Journal of Logic and Computation*, *26*(4), 1315–1360. https://doi.org/10.1093/logcom/exu008.
7. Alur, R., Henzinger, T. A., Mang, F. Y., Qadeer, S., Rajamani, S. K., & Tasiran, S. (1998). MOCHA: Modularity in model checking. In *Computer aided verification, Lecture notes in computer science* (Vol. 1427, pp. 521–525). Berlin: Springer. https://doi.org/10.1007/BFb0028774.
8. Alur, R., Henzinger, T. A., & Kupferman, O. (2002). Alternating-time temporal logic. *Journal of the ACM*, *49*(5), 672–713. https://doi.org/10.1145/585265.585270.
9. Ancona, D., Drossopoulou, S., & Mascardi, V. (2012). Automatic generation of self-monitoring MASs from multiparty global session types in Jason. In Baldoni et al. [20] (pp. 76–95). https://doi.org/10.1007/978-3-642-37890-4_5.
10. Ancona, D., Ferrando, A., & Mascardi, V. (2016). Comparing trace expressions and linear temporal logic for runtime verification. In E. Ábrahám, M. M. Bonsangue, & E. B. Johnsen (Eds.), *Theory and practice of formal methods—Essays dedicated to Frank de Boer on the occasion of his 60th birthday, Lecture notes in computer science* (Vol. 9660, pp. 47–64). Berlin: Springer. https://doi.org/10.1007/978-3-319-30734-3_6.
11. Ancona, D., Ferrando, A., & Mascardi, V. (2017). Parametric runtime verification of multiagent systems. In K. Larson, M. Winikoff, S. Das, & E. H. Durfee (Eds.), *16th Conference on autonomous agents and multiagent systems, AAMAS 2017, São Paulo, Brazil, May 8–12, 2017* (pp. 1457–1459). São Paulo: ACM. http://dl.acm.org/citation.cfm?id=3091328.
12. Antoniou, G., Dimaresis, N., & Governatori, G. (2009). A modal and deontic defeasible reasoning system for modelling policies and multi-agent systems. *Expert Systems with Applications*, *36*(2, Part 2), 4125–4134. https://doi.org/10.1016/j.eswa.2008.03.009.
13. Arisha, K., Ozcan, F., Ross, R., Kraus, S., & Subrahmanian, V. (1998). IMPACT: The interactive maryland platform for agents collaborating together. In *International conference on multi agent systems (Cat. No. 98EX160)* (pp. 385–386). IEEE. https://doi.org/10.1109/ICMAS.1998.699225.
14. Arranz Matía, A. L., & Sanz-Bobi, M. A. (2005). CLELIA: A multi-agent system for publishing printed and electronic media. *Expert Systems with Applications*, *28*(4), 725–734. https://doi.org/10.1016/j.eswa.2004.12.029.
15. Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., & Nardi, D. (Eds.). (2003). *The description logic handbook: Theory, implementation, and applications*. Cambridge: Cambridge University Press.
16. Bahaj, M., & Soklabi, A. (2013). JAPL: The JADE agent programming language. *Journal of Emerging Technologies in Web Intelligence*, *5*(3), 272–278. https://doi.org/10.4304/jetwi.5.3.272-277.
17. Baldoni, M., & Endriss, U. (Eds.). (2006). *Declarative agent languages and technologies IV, Lecture notes in computer science* (Vol. 4327). Berlin: Springer. https://doi.org/10.1007/11961536.
18. Baldoni, M., Baroglio, C., Martelli, A., & Patti, V. (2003). Reasoning about self and others: Communicating agents in a modal action logic. In C. Blundo & C. Laneve (Eds.), *Theoretical computer science. ICTCS 2003, Lecture notes in computer science* (Vol. 2841, pp. 228–241). Berlin: Springer. https://doi.org/10.1007/978-3-540-45208-9_19.
19. Baldoni, M., Baroglio, C., Gungui, I., Martelli, A., Martelli, M., Mascardi, V., et al. (2005). Reasoning about agents' interaction protocols inside dcaselp. In *Declarative agent languages and technologies II* (pp. 112–131). Berlin: Springer. https://doi.org/10.1007/11493402_7.
20. Baldoni, M., Endriss, U., Omicini, A., & Torroni, P. (Eds.). (2005). *Declarative agent languages and technologies III, Lecture notes in computer science* (Vol. 3904). Berlin: Springer. https://doi.org/10.1007/11691792.

21. Baldoni, M., Son, T. C., van Riemsdijk, B. M., & Winikoff, M. (Eds.). (2008). *Declarative agent languages and technologies VI, Lecture notes in computer science* (Vol. 5397). Berlin: Springer. https://doi.org/10.1007/978-3-540-93920-7.
22. Baldoni, M., Baroglio, C., & Marengo, E. (2010). Constraints among commitments: Regulative specification of interaction protocols. In W. Faber, & N. Leone (Eds.), *Proceedings of the 25th Italian conference on computational logic, Rende, Italy, July 7–9, 2010, CEUR workshop proceedings* (Vol. 598). https://CEUR-WS.org. http://ceur-ws.org/Vol-598/paper04.pdf.
23. Baldoni, M., Baroglio, C., Mascardi, V., Omicini, A., & Torroni, P. (2010). Agents, multi-agent systems and declarative programming: Who, what, when, where, why, how? In A. Dovier & E. Pontelli (Eds.), *A 25 year perspective on logic programming. Achievements of the Italian Association for Logic Programming, GULP, LNAI: State-of-the-art survey (chapter 10)* (pp. 200–225). Berlin: Springer. https://doi.org/10.1007/978-3-642-14309-0_10.
24. Baldoni, M., Baroglio, C., Capuzzimati, F., Marengo, E., & Patti, V. (2012). A generalized commitment machine for 2CL protocols and its implementation. In Baldoni et al. [25] (pp. 96–115). https://doi.org/10.1007/978-3-642-37890-4_6.
25. Baldoni, M., Dennis, L., Mascardi, V., & Vasconcelos, W. (Eds.). (2012). *Declarative agent languages and technologies X, Lecture notes in computer science* (Vol. 7784). Berlin: Springer. https://doi.org/10.1007/978-3-642-37890-4.
26. Baldoni, M., Baroglio, C., Marengo, E., Patti, V., & Capuzzimati, F. (2014). Engineering commitment-based business protocols with the 2CL methodology. *Autonomous Agents and Multi Agent Systems*, *28*(4), 519–557. https://doi.org/10.1007/s10458-013-9233-1.
27. Banerjee, P., Friedrich, R., Bash, C., Goldsack, P., Huberman, B., Manley, J., et al. (2011). Everything as a service: Powering the new information economy. *IEEE Computer*, *44*(3), 36–43. https://doi.org/10.1109/MC.2011.67.
28. Barredo Arrieta, A., Díaz Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., et al. (2020). Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115. https://doi.org/10.1016/j.inffus.2019.12.012.
29. Barringer, H., Fisher, M., Gabbay, D., Gough, G., & Owens, R. (1990). MetateM: A framework for programming in temporal logic. In J. W. de Bakker, W. P. de Roever, & G. Rozenberg (Eds.), *Stepwise refinement of distributed systems models, formalisms, correctness, Lecture notes in computer science* (Vol. 430, pp. 94–129). Berlin: Springer. https://doi.org/10.1007/3-540-52559-9_62.
30. Bauer, B., Müller, J. P., & Odell, J. (2001). Agent UML: A formalism for specifying multiagent software systems. *International Journal of Software Engineering and Knowledge Engineering*, *11*(3), 207–230. https://doi.org/10.1142/S0218194001000517.
31. Baumann, J., Hohl, F., Rothermel, K., & Straßer, M. (1998). Mole—Concepts of a mobile agent system. *World Wide Web*, *1*(3), 123–137. https://doi.org/10.1023/A:1019211714301.
32. Bayardo, R. J, Jr., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., et al. (1997). InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. *ACM SIGMOD Record*, *26*(2), 195–206. https://doi.org/10.1145/253262.253294.
33. Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., et al. (2004). *OWL web ontology language reference*. https://www.w3.org/TR/owl-guide/.
34. Bellifemine, F., Bergenti, F., Caire, G., & Poggi, A. (2009). Jade—A java agent development framework. In Håkansson et al. [167] *3rd KES International Symposium, KES-AMSTA 2009, Uppsala, Sweden, June 3–5, 2009. Proceedings* (pp. 125–147). https://doi.org/10.1007/0-387-26350-0_5.
35. Bergenti, F. (2014). An introduction to the JADEL programming language. In *26th International conference on tools with artificial intelligence*. IEEE. https://doi.org/10.1109/ICTAI.2014.147.
36. Bevar, V., Costantini, S., Tocchio, A., & De Gasperis, G. (2012). A multi-agent system for industrial fault detection and repair. In *Advances on practical applications of agents and multi-agent systems* (pp. 47–55). Berlin: Springer. https://doi.org/10.1007/978-3-642-28786-2_5.
37. Bikakis, A., & Antoniou, G. (2005). DR-Prolog: A system for defeasible reasoning with rules and ontologies on the semantic web. In *20th National conference on artificial intelligence, Pittsburgh, Pennsylvania* (Vol. 5, pp. 1594–1595). https://doi.org/10.1109/TKDE.2007.29.
38. Bistarelli, S., & Santini, F. (2011). ConArg: A constraint-based computational framework for argumentation systems. In *23rd International conference on tools with artificial intelligence* (pp. 605–612). Boca Raton, FL: IEEE. https://doi.org/10.1109/ICTAI.2011.96.
39. Bistarelli, S., Rossi, F., & Santini, F. (2014). Enumerating extensions on random abstract-AFs with ArgTools, Aspartix, ConArg2, and Dung-O-Matic. In *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 8624, pp. 70–86). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-09764-0_5.

40. Black, E., Coles, A., & Bernardini, S. (2014). Automated planning of simple persuasion dialogues. In Bulling et al. [57] (pp. 87–104). https://doi.org/10.1007/978-3-319-09764-0_6.
41. Boissier, O., Bordini, R. H., Hübner, J., Ricci, A., & Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, *78*(6), 747–761. https://doi.org/10.1016/j.scico.2011.10.004. Special section on Agent-oriented design methods and programming techniques for distributed computing in dynamic and complex environments
42. Bordini, R. H., & Hübner, J. F. (2006). BDI agent programming in AgentSpeak using Jason. In F. Toni & P. Torroni (Eds.), *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 3900, pp. 143–164). Berlin: Springer. https://doi.org/10.1007/11750734_9.
43. Bordini, R. H., Fisher, M., Pardavila, C., Visser, W., & Wooldridge, M. (2003). Model checking multi-agent programs with CASP. In *Computer aided verification, Lecture notes in computer science* (Vol. 2725, pp. 110–113). Berlin: Springer. https://doi.org/10.1007/978-3-540-45069-6_10.
44. Bordini, R. H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A. E., Gómez-Sanz, J. J., Leite, J., et al. (2006). A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, *30*, 33–44.
45. Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). Programming multi-agent systems in AgentSpeak using Jason. In *Wiley series in agent technology*. Hoboken, NJ: Wiley. https://doi.org/10.5555/1197104.
46. Bordini, R. H., Dastani, M., Dix, J., & Fallah-Seghrouchni, A. E. (Eds.). (2009). *Multi-agent programming. Languages, tools and applications*. Boston: Springer. https://doi.org/10.1007/978-0-387-89299-3.
47. Borgida, A., & Serafini, L. (2003). Distributed description logics: Assimilating information from peer sources. In S. Spaccapietra, S. March, & K. Aberer (Eds.), *Journal on Data Semantics I, Lecture notes in computer science* (Vol. 2800, pp. 153–184). Berlin: Springer. https://doi.org/10.1007/978-3-540-39733-5_7.
48. Bosse, T., Jonker, C. M., van der Meij, L., & Treur, J. (2005). LEADSTO: A language and environment for analysis of dynamics by SimulaTiOn. In *Innovations in applied artificial intelligence, Lecture notes in computer science* (Vol. 3533, pp. 363–366). Berlin: Springer. https://doi.org/10.1007/11504894_51.
49. Bozzano, M., Delzanno, G., Martelli, M., Mascardi, V., & Zini, F. (1999). Logic programming and multi-agent systems: A synergic combination for applications and semantics. In K. R. Apt, V. W. Marek, M. Truszczynski, & D. S. Warren (Eds.), *The logic programming paradigm, Artificial intelligence* (pp. 5–32). Berlin: Springer. https://doi.org/10.1007/978-3-642-60085-2_1.
50. Brazier, F. M., Dunin-Keplicz, B. M., Jennings, N. R., & Treur, J. (1997). Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, *6*(1), 67–94. https://doi.org/10.1142/S0218843097000069.
51. Bretier, P., & Sadek, D. (1997). A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction. In *Intelligent agents III agent theories, architectures, and languages, Lecture notes in computer science* (Vol. 1193, pp. 189–203). Berlin: Springer. https://doi.org/10.1007/BFb0013586.
52. Briola, D., Mascardi, V., & Ancona, D. (2014). Distributed runtime verification of JADE multiagent systems. In D. Camacho, L. Braubach, S. Venticinque, & C. Badica (Eds.), *Intelligent distributed computing VIII—Proceedings of the 8th international symposium on intelligent distributed computing, IDC 2014, Madrid, Spain, September 3–5, 2014, Studies in computational intelligence* (Vol. 570, pp. 81–91). Berlin: Springer. https://doi.org/10.1007/978-3-319-10422-5_10.
53. Bromuri, S., Urovi, V., Contreras, P., & Stathis, K. (2008). A virtual e-retailing environment in GOLEM. In *4th International conference on intelligent environments*. IEE. https://doi.org/10.1049/cp:20081174.
54. Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691. https://doi.org/10.1109/TC.1986.1676819.
55. Bryl, V., Mello, P., Montali, M., Torroni, P., & Zannone, N. (2008). $\mathcal{B}$-Tropos. Agent-oriented requirements engineering meets computational logic for declarative business process modeling and verification. In Sadri and Satoh [306] (pp. 157–176). https://doi.org/10.1007/978-3-540-88833-8_9.
56. Budgen, D., & Brereton, P. (2006). Performing systematic literature reviews in software engineering. In *28th International conference on software engineering (ICSE 2006)* (pp. 1051–1052). New York, NY: ACM. https://doi.org/10.1145/1134285.1134500.
57. Bulling, N., van der Torre, L., Villata, S., Jamroga, W., & Vasconcelos, W. (Eds.). (2014). *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 8624). Berlin: Springer. https://doi.org/10.1007/978-3-319-09764-0.
58. Cabri, G., Leonardi, L., & Zambonelli, F. (2000). MARS: A programmable coordination architecture for mobile agents. *IEEE Internet Computing*, *4*(4), 26–35. https://doi.org/10.1109/4236.865084.

59. Calegari, R., Denti, E., Dovier, A., & Omicini, A. (2018). Extending logic programming with labelled variables: Model and semantics. *Fundamenta Informaticae*, *161*(1–2), 53–74. https://doi.org/10.3233/FI-2018-1695. Special Issue CILC 2016.
60. Calegari, R., Denti, E., Mariani, S., & Omicini, A. (2018). Logic programming as a service. *Theory and Practice of Logic Programming*, *18*(5–6), 846–873. https://doi.org/10.1017/S1471068418000364. Special issue "Past and present (and future) of parallel and distributed computation in (constraint) logic programming".
61. Calegari, R., Denti, E., Mariani, S., & Omicini, A. (2019). Logic programming as a service in multi-agent systems for the Internet of Things. *International Journal of Grid and Utility Computing*, *10*(4), 344–360. https://doi.org/10.1504/IJGUC.2019.10022135.
62. Calejo, M. (2004). InterProlog: Towards a declarative embedding of logic programming in Java. In *9th European conference on logics in artificial intelligence* (pp. 714–717). Lisbon: Springer. https://doi.org/10.1007/978-3-540-30227-8_64.
63. Carlsson, M., et al. (2020). *SICStus Prolog user's manual (release 4.6.0)*. Kista: Swedish Institute of Computer Science. https://sicstus.sics.se/sicstus/docs/latest4/pdf/sicstus.pdf.
64. Casadei, M., & Omicini, A. (2008). Situating A&A ReSpecT for pervasive environment applications. In L. J. B. Nixon, M. Bortenschlager, E. Simperl, & R. Tolksdorf (Eds.), *17th IEEE International workshops on enabling technologies: Infrastructures for collaborative enterprises. Workshop on coordination models and applications (CoMA 2008), IEEE WETICE 2008* (pp. 76–81). Rome: IEEE Computer Society. https://doi.org/10.1109/WETICE.2008.31.
65. Casadei, M., & Omicini, A. (2009). Situated tuple centres in ReSpecT. In S. Y. Shin, S. Ossowski, R. Menezes, & M. Viroli (Eds.), *24th Annual ACM symposium on applied computing (SAC 2009), ACM, Honolulu, Hawai'i, USA* (Vol. III, pp. 1361–1368). https://doi.org/10.1145/1529282.1529586.
66. Cattell, R. G. G., & Barry, D. K. (Eds.). (2000). *The object data standard: ODMG 3.0*. San Francisco, CA: Morgan Kaufmann Publishers.
67. Ceccaroni, L. (2001). *OntoWEDSS—An ontology-based environmental decision-support system for the management of wastewater treatment plants*. PhD thesis, Universitat Politècnica de Catalunya. Departament de Llenguatges i Sistemes Informàtics. https://upcommons.upc.edu/handle/2117/93950.
68. Čermák, P., Lomuscio, A., Mogavero, F., & Murano, A. (2014). MCMAS-SLK: A model checker for the verification of strategy logic specifications. In A. Biere & R. Bloem (Eds.), *Computer aided verification, Lecture notes in computer science* (Vol. 8559, pp. 525–532). Cham: Springer. https://doi.org/10.1007/978-3-319-08867-9_34.
69. Chandy, K. M. (1988). *Parallel program design: A foundation*. Boston, MA: Addison-Wesley Longman Publishing Co. Inc.
70. Chellas, B. F. (1980). *Modal logic: An introduction*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511621192.
71. Chen, M., Athanasiadis, D., Al Faiya, B., McArthur, S., Kockar, I., Lu, H., et al. (2017). Design of a multi-agent system for distributed voltage regulation. In *19th International conference on intelligent system application to power systems* (pp. 1–6). San Antonio, TX: IEEE. https://doi.org/10.1109/ISAP.2017.8071400.
72. Chesani, F., Gavanelli, M., Alberti, M., Lamma, E., Mello, P., & Torroni, P. (2006). Specification and verification of agent interaction using abductive reasoning. In *Lecture notes in computer science*. Berlin: Springer (pp. 243–264). https://doi.org/10.1007/11750734_14.
73. Chesani, F., Mello, P., Montali, M., & Torroni, P. (2009). Verifying a-priori the composition of declarative specified services. In M. Baldoni, C. Baroglio, J. Bentahar, G. Boella, M. Cossentino, M. Dastani, et al. (Eds.), *Multi-agent logics, languages, and organisations—Second international federated workshops, MALLOW'009, workshops proceedings, Turin, Italy* (Vol. 494, pp. 14–21). http://ceur-ws.org/Vol-494/.
74. Chopra, A. K., & Singh, M. P. (2015). Cupid: Commitments in relational algebra. In B. Bonet & S. Koenig (Eds.), *29th AAAI conference on artificial intelligence* (pp. 2052–2059). AAAI Press. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9938.
75. Ciampolini, A., Lamma, E., Stefanelli, C., & Mello, P. (1997). A coordination protocol for abductive logic agents. In *1997 IEEE International conference on intelligent processing systems (Cat. No.97TH8335)* (Vol. 1, pp. 143–148). IEEE. https://doi.org/10.1109/ICIPS.1997.672754.
76. Ciampolini, A., Lamma, E., Mello, P., & Torroni, P. (1999). The dynamic composition of abductive agents in *ALIAS*. In A. Brogi & P. Hill (Eds.), *2nd International workshop on component-based software development in computational logic (COCL'99), Paris, France*. http://www.di.unipi.it/~brogi/ResearchActivity/COCL99/proceedings/ciampolini.ps.
77. Ciampolini, A., Lamma, E., Mello, P., Toni, F., & Torroni, P. (2003). Cooperation and competition in alias: A logic framework for agents that negotiate. *Annals of Mathematics and Artificial Intelligence*, *37*(1), 65–91. https://doi.org/10.1023/A:1020259411066.

78. Ciatto, G., Rizzato, L., Omicini, A., & Mariani, S. (2019). TuSoW: Tuple spaces for edge computing. In *The 28th International conference on computer communications and networks (ICCCN 2019)*. Valencia: Institute of Electrical and Electronics Engineers. https://doi.org/10.1109/ICCCN.2019.8846916.
79. Ciatto, G., Di Marzo, S. G., Louvel, M., Mariani, S., Omicini, A., & Zambonelli, F. (2020). Twenty years of coordination technologies: COORDINATION contribution to the state of art. *Journal of Logical and Algebraic Methods in Programming*, *113*, 1–25. https://doi.org/10.1016/j.jlamp.2020.100531.
80. Clark, K., Hengst, B., Pagnucco, M., Rajaratnam, D., Robinson, P., Sammut, C., et al. (2016). A framework for integrating symbolic and sub-symbolic representations. In S. Kambhampati (Ed.), *25th International joint conference on artificial intelligence (IJCAI-16)* (pp. 2486–2492). IJCAI/AAAI Press. http://www.ijcai.org/Abstract/16/354.
81. Clark, K. L., & McCabe, F. G. (2004). Go!—A multi-paradigm programming language for implementing multi-threaded agents. *Annals of Mathematics and Artificial Intelligence*, *41*(2), 171–206. https://doi.org/10.1023/B:AMAI.0000031195.87297.d9.
82. Clark, K. L., & McCabe, F. G. (2004). Go! for multi-threaded deliberative agents. In Leite et al. [225] (pp. 54–75). https://doi.org/10.1007/978-3-540-25932-9_4.
83. Clark, K. L., & McCabe, F. G. (2006). Ontology oriented programming in Go!. *Applied Intelligence*, *24*(3), 189–204. https://doi.org/10.1007/s10489-006-8511-x.
84. Clark, K. L., & Robinson, P. J. (2015). Robotic agent programming in TeleoR. In *IEEE International conference on robotics and automation (ICRA)*. Seattle, WA: IEEE (pp. 5040–5047).
85. Clarke, E. M., & Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching-time temporal logic. In D. Kozen (Ed.), *Logics of programs, workshop, Yorktown Heights, New York, USA, May 1981, Lecture notes in computer science* (Vol. 131, pp. 52–71). Berlin: Springer. https://doi.org/10.1007/BFb0025774.
86. Clavel, M., Eker, S., Lincoln, P., & Meseguer, J. (1996). Principles of Maude. *Electronic Notes in Theoretical Computer Science*, *4*, 65–89. https://doi.org/10.1016/S1571-0661(04)00034-9.
87. Collier, R. W. (2002). *Agent factory: A framework for the engineering of agent-oriented applications*. PhD thesis, University College Dublin.
88. Collier, R. W., Russell, S. E., & Lillis, D. (2015). Reflecting on agent programming with AgentSpeak(L). In *PRIMA 2015: Principles and practice of multi-agent systems, Lecture notes in computer science* (pp. 351–366). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-25524-8_22.
89. Corapi, D., Russo, A., & Lupu, E. (2010). Inductive logic programming as abductive search. In M. Hermenegildo & T. Schaub (Eds.), *Technical communications of the 26th international conference on logic programming* (Vol. 7, pp. 54–63). Dagstuhl: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/LIPIcs.ICLP.2010.54.
90. Corapi, D., Sykes, D., Inoue, K., & Russo, A. (2011). Probabilistic rule learning in nonmonotonic domains. In J. A. Leite, P. Torroni, T. Ågotnes, G. Boella, & L. van der Torre (Eds.), *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 6814, pp. 243–258). Berlin: Springer. https://doi.org/10.1007/978-3-642-22359-4_17.
91. Cossentino, M., Gaud, N., Hilaire, V., Galland, S., & Koukam, A. (2010). ASPECS: An agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, *20*(2), 260–304. https://doi.org/10.1007/s10458-009-9099-4.
92. Cost, R. S., Finin, T., Labrou, Y., Lua, X., Peng, Y., Soboroff, I., et al. (1998). *Jackal: A Java-based tool for agent development*. Working papers of the AAAI-98 workshop on software tools for developing agents.
93. Costa, V. S., Rocha, R., & Damas, L. (2012). The YAP prolog system. *Theory and Practice of Logic Programming*, *12*(1–2), 5–34. https://doi.org/10.1017/S1471068411000512.
94. Costantini, S. (2014). Towards active logic programming. In A. Brogi & P. Hill (Eds.), *PLI workshop on COmponent-based software developement in computational logic (COCL'99)*. Paris. http://pages.di.unipi.it/brogi/cocl99.html.
95. Costantini, S. (2015). ACE: A flexible environment for complex event processing in logical agents. In M. Baldoni, L. Baresi, & M. Dastani. (Eds.), *Engineering multi-agent systems (EMAS 2015), Lecture notes in computer science* (Vol. 9318, pp. 70–91). Cham: Springer. https://doi.org/10.1007/978-3-319-26184-3_5.
96. Costantini, S., & Formisano, A. (2016). Augmenting agent computational environments with quantitative reasoning modules and customizable bridge rules. In N. Osman & C. Sierra (Eds.), *Autonomous agents and multiagent systems (AAMAS 2016), Lecture notes in computer science* (Vol. 10003, pp. 104–121). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-46840-2_7.
97. Costantini, S., & Tocchio, A. (2002). A logic programming language for multi-agent systems. In S. Flesca, S. Greco, G. Ianni, & N. Leone (Eds.), *Logics in artificial intelligence, Lecture notes in computer science* (Vol. 2424, pp. 1–13). Berlin: Springer. https://doi.org/10.1007/3-540-45757-7_1.

98. Costantini, S., & Tocchio, A. (2004). The DALI logic programming agent-oriented language. In J. J. Alferes & J. A. Leite (Eds.), *Logics in artificial intelligence, 9th European conference, JELIA 2004, Lecture notes in computer science* (Vol. 3229, pp. 685–688). Berlin: Springer. https://doi.org/10.1007/978-3-540-30227-8_57.
99. Costantini, S., Tocchio, A., & Tsintza, P. (2007). A heuristic approach to P2P negotiation. In Sadri and Satoh [306] (pp. 177–192). https://doi.org/10.1007/978-3-540-88833-8_10.
100. Costantini, S., Formisano, A., & Petturiti, D. (2010). Extending and implementing RASP. *Fundamenta Informaticae*, *105*(1–2), 1–33. https://doi.org/10.3233/FI-2010-356.
101. Costantini, S., De Gasperis, G., & Nazzicone, G. (2017). DALI for cognitive robotics: Principles and prototype implementation. In *International symposium on practical aspects of declarative languages* (pp. 152–162). Cham: Springer. https://doi.org/10.1007/978-3-319-51676-9_10.
102. Cutkosky, M. R., Engelmore, R. S., Fikes, R. E., Genesereth, M. R., Gruber, T. R., Mark, W. S., et al. (1993). PACT: An experiment in integrating concurrent engineering systems. *Computer*, *26*(1), 28–37. https://doi.org/10.1109/2.179153.
103. da Silva, D. M., & Vieira, R. (2007). Argonaut: Integrating Jason and Jena for context aware computing based on OWL ontologies. In *Workshop on agents, web services, and ontologies—Integrated methodologies (AWESOME'007), Durham, UK* (p. 19).
104. da Silva, V. T., Garcia, A. F., Brandão, A., Chavez, C., de Lucena, C. J. P., & Alencar, P. S. C. (2002). Taming agents and objects in software engineering. In A. F. Garcia, C. J. P. de Lucena, F. Zambonelli, A. Omicini, & J. Castro (Eds.), *Software engineering for large-scale multi-agent systems. SELMAS 2002, Lecture notes in computer science* (Vol. 2603, pp. 1–26). Berlin: Springer. https://doi.org/10.1007/3-540-35828-5_1.
105. Dalpiaz, F., Dix, J., & van Riemsdijk, M. B. (Eds.). (2014). *Engineering multi-agent systems, Lecture notes in computer science* (Vol. 8758). Paris: Springer. https://doi.org/10.1007/978-3-319-14484-9.
106. Dasgupta, A., & Ghose, A. K. (2010). BDI agents with objectives and preferences. In Omicini et al. [281] (pp. 22–39). https://doi.org/10.1007/978-3-642-20715-0_2.
107. Dasgupta, A., & Ghose, A. K. (2010). Implementing reactive BDI agents with user-given constraints and objectives. *International Journal of Agent-Oriented Software Engineering*, *4*(2), 141. https://doi.org/10.1504/IJAOSE.2010.032799.
108. Dastani, M. (2008). 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, *16*(3), 214–248. https://doi.org/10.1007/s10458-008-9036-y.
109. Dastani, M., & Steunebrink, B. R. (2009). Operational semantics for BDI modules in multi-agent programming. In Dix et al. [127] (pp. 83–101). https://doi.org/10.1007/978-3-642-16867-3_5.
110. Dastani, M., & van Zee, M. (2013). Belief caching in 2APL. In J. Leite, A. Omicini, P. Torroni, & P. Yolum (Eds.), *Declarative agent languages and technologies, LNAI* (Vol. 8245, pp. 117–136). Berlin: Springer. https://doi.org/10.1007/978-3-642-45343-4_7.
111. Dastani, M. M., Hindriks, K. V., Novák, P., & Tinnemeier, N. A. M. (2008). Combining multiple knowledge representation technologies into agent programming languages. In Baldoni et al. [21] (pp. 60–74). https://doi.org/10.1007/978-3-540-93920-7_5.
112. de Boer, F. S., Hindriks, K. V., van der Hoek, W., & Meyer, J. J. C. (2007). A verification framework for agent programming with declarative goals. *Journal of Applied Logic*, *5*(2), 277–302. https://doi.org/10.1016/j.jal.2005.12.014.
113. De Giacomo, G., Lespérance, Y., & Levesque, H. J. (2000). ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, *121*(1–2), 109–169. https://doi.org/10.1016/S0004-3702(00)00031-X.
114. De Giacomo, G., Lespérance, Y., Levesque, H. J., & Sardiña, S. (2009). Indigolog: A high-level programming language for embedded reasoning agents. In R. H. Bordini, M. Dastani, J. Dix, & A. E. Fallah-Seghrouchni (Eds.), *Multi-agent programming, languages, tools and applications*. Boston, MA: Springer (pp. 31–72). https://doi.org/10.1007/978-0-387-89299-3_2.
115. De Nicola, R., Ferrari, G. L., & Pugliese, R. (1998). KLAIM: A kernel language for agents interaction and mobility. *Transactions on Software Engineering*, *24*(5), 315–330. https://doi.org/10.1109/32.685256.
116. Dell'Acqua, P., Sadri, F., & Toni, F. (1998). Combining introspection and communication with rationality and reactivity in agents. In *Logics in artificial intelligence, European workshop, JELIA '98, Dagstuhl, Germany, October 12–15, 1998, proceedings* (pp. 17–32). Berlin: Springer. https://doi.org/10.1007/3-540-49545-2_2.
117. Demolombe, R., & Otermin Fernandez, A. M. (2006). Intention recognition in the situation calculus and probability theory frameworks. In Toni and Torroni [340] (pp. 358–372). https://doi.org/10.1007/11750734_20.
118. Dennis, L., Tinnemeier, N., & Meyer, J. J. (2009). Model checking normative agent organisations. In Dix et al. [127] (pp. 64–82). https://doi.org/10.1007/978-3-642-16867-3_4.

119. Dennis, L. A., & Farwer, B. (2008). Gwendolen: A BDI language for verifiable agents. In B. Löwe (Ed.), *AISB 2008 Symposium on logic and the simulation of interaction and reasoning* (Vol. 9, pp. 16–23). Aberdeen: The Society for the Study of Artificial Intelligence and Simulation of Behaviour, University of Aberdeen.
120. Dennis, L. A., Fisher, M., Webster, M. P., & Bordini, R. H. (2012). Model checking agent programming languages. *Automated Software Engineering*, *19*(1), 5–63. https://doi.org/10.1007/s10515-011-0088-x.
121. Dennis, L. A., Fisher, M., & Webster, M. (2013). Using agent JPF to build models for other model checkers. In Leite et al. [227] (pp. 273–289). https://doi.org/10.1007/978-3-642-40624-9_17.
122. Denti, E., Omicini, A., & Ricci, A. (2001). tuProlog: A light-weight Prolog for Internet applications and infrastructures. In I. V. Ramakrishnan (Ed.), *Practical aspects of declarative languages, Lecture notes in computer science* (Vol. 1990, pp. 184–198). Berlin: Springer. https://doi.org/10.1007/3-540-45241-9_13. 3rd International symposium (PADL 2001), Las Vegas, NV, USA, 11–12 March 2001. Proceedings.
123. Di Stefano, A., & Santoro, C. (2003). eXAT: An experimental tool for programming multi-agent systems in erlang. In G. Armano, F. D. Paoli, A. Omicini, & E. Vargiu (Eds.), *WOA 2003—Dagli oggetti agli agenti: sistemi intelligenti e computazione pervasiva, Pitagora Editrice Bologna, Villasimius, CA* (pp. 121–127).
124. Di Stefano, A., & Santoro, C. (2005). Using the Erlang language for multi-agent systems implementation. In *International conference on intelligent agent technology. IEEE/WIC/ACM 2005, Compiegne, France* (pp. 679–685). https://doi.org/10.1109/IAT.2005.141.
125. Dignum, V., Vázquez-Salceda, J., & Dignum, F. (2005). OMNI: Introducing social structure, norms and ontologies into agent organizations. In R. H. Bordini, M. M. Dastani, J. Dix, & A. El Fallah Seghrouchni (Eds.), *Programming multi-agent systems, Lecture notes in computer science* (Vol. 3346, pp. 181–198). Berlin: Springer. https://doi.org/10.1007/978-3-540-32260-3_10.
126. d'Inverno, M., Luck, M., Georgeff, M., Kinny, D., & Wooldridge, M. (2004). The dMARS architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, *9*(1–2), 5–53. https://doi.org/10.1023/B:AGNT.0000019688.11109.19.
127. Dix, J., Fisher, M., & Novák, P. (Eds.). (2009). *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 6214). Berlin: Springer. https://doi.org/10.1007/978-3-642-16867-3.
128. Dix, J., Leite, J., Governatori, G., & Jamroga, W. (Eds.). (2010). *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 6245). Berlin: Springer. https://doi.org/10.1007/978-3-642-14977-1.
129. Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, *77*(2), 321–357. https://doi.org/10.1016/0004-3702(94)00041-X.
130. Dyoub, A., Costantini, S., & De Gasperis, G. (2018). Answer set programming and agents. *The Knowledge Engineering Review*, *33*, e19. https://doi.org/10.1017/S0269888918000164.
131. Egly, U., Gaggl, S. A., & Woltran, S. (2008). Aspartix: Implementing argumentation frameworks using answer-set programming. In M. Garcia de la Banda & E. Pontelli (Eds.), *Logic programming* (pp. 734–738). Berlin: Springer. https://doi.org/10.1007/978-3-540-89982-2_67.
132. El Fallah-Seghrouchni, A., & Suna, A. (2004). CLAIM: A computational language for autonomous, intelligent and mobile agents. In M. M. Dastani & A. El Fallah Seghrouchni (Eds.), *Programming multi-agent systems (ProMAS 2003), Lecture notes in computer science* (Vol. 3067, pp. 90–110). Berlin: Springer. https://doi.org/10.1007/978-3-540-25936-7_5.
133. El-Menshawy, M., Bentahar, J., & Dssouli, R. (2010). Symbolic model checking commitment protocols using reduction. In Omicini et al. [281] (pp. 185–203). https://doi.org/10.1007/978-3-642-20715-0_11.
134. Emerson, E. A. (1990). Temporal and modal logic. In J. van Leeuwen (Ed.), *Formal models and semantics, Handbook of theoretical computer science* (pp. 995–1072). Amsterdam: Elsevier. https://doi.org/10.1016/B978-0-444-88074-1.50021-4.
135. Erol, K., Hendler, J. A., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. In B. Hayes-Roth & R. E. Korf (Eds.), *12th National conference on artificial intelligence* (Vol. 2, pp. 1123–1128). Seattle, WA: AAAI Press/The MIT Press. http://www.aaai.org/Library/AAAI/1994/aaai94-173.php.
136. Esteva, M., De La Cruz, D., & Sierra, C. (2002). ISLANDER: An electronic institutions editor. In *1st International joint conference on autonomous agents and multiagent systems (AAMAS 2002)* (pp. 1045–1052). ACM. https://doi.org/10.1145/545056.545069.
137. Esteva, M., Rosell, B., Rodriguez-Aguilar, J. A., & Arcos, J. L. (2004). AMELI: An agent-based middleware for electronic institutions. In *3rd International joint conference on autonomous agents and multiagent systems* (pp. 236–243). https://doi.org/10.1109/AAMAS.2004.10060.
138. Fan, X., Yen, J., Miller, M. S., & Volz, R. A. (2005). The semantics of MALLET—An agent teamwork encoding language. In Leite et al. [226] (pp. 69–91). https://doi.org/10.1007/11493402_5.

139. Fernández Díaz, Á., Benac Earle, C., & Fredlund, LÅ. (2019). Pitfalls of Jason concurrency. In Weyns et al. [351] (pp. 19–33). https://doi.org/10.1007/978-3-030-25693-7_2.
140. Ferrando, A., Dennis, L. A., Ancona, D., Fisher, M., & Mascardi, V. (2018). Recognising assumption violations in autonomous systems verification. In E. André, S. Koenig, M. Dastani, & G. Sukthankar (Eds.), *17th International conference on autonomous agents and multiagent systems, AAMAS 2018* (pp. 1933–1935). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. http://dl.acm.org/citation.cfm?id=3238028.
141. Ferrando, S. P., & Onaindia, E. (2012). Defeasible argumentation for multi-agent planning in ambient intelligence applications. In *11th International conference on autonomous agents and multiagent systems—Volume 1* (pp. 509–516). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems. https://dl.acm.org/doi/10.5555/2343576.2343649.
142. Finger, M., Fisher, M., & Owens, R. (1993). MetateM at work: Modelling reactive systems using executable temporal logic. In *Sixth international conference on industrial and engineering applications of artificial intelligence and expert systems*. Edinburgh: Gordon and Breach Publishers.
143. Finin, T., Fritzson, R., McKay, D., & McEntire, R. (1994). KQML as an agent communication language. In A. R. Nabil, B. K. Bharat, & Y. Yesha (Eds.), *3rd International conference on information and knowledge management* (pp. 456–463). New York, NY: ACM. https://doi.org/10.1145/191246.191322.
144. Fisher, M. (1992). A normal form for first-order temporal formulae. In D. Kapur (Ed.), *Automated deduction—CADE-11* (pp. 370–384). Berlin: Springer. https://doi.org/10.1007/3-540-55602-8_178.
145. Fisher, M. (2006). Implementing temporal logics: Tools for execution and proof. In Inoue et al. [198] (pp. 129–142). https://doi.org/10.1007/11750734_8.
146. Fisher, M. (2006). MetateM: The story so far. In R. H. Bordini, M. M. Dastani, J. Dix, & A. El Fallah Seghrouchni (Eds.), *Programming multi-agent systems (ProMAS 2005), Lecture notes in computer science* (Vol. 3862, pp. 3–22). Berlin: Springer. https://doi.org/10.1007/11678823_1.
147. Fisher, M., Bordini, R. H., Hirsch, B., & Torroni, P. (2007). Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence*, *23*(1), 61–91. https://doi.org/10.1111/j.1467-8640.2007.00295.x.
148. Fornara, N., & Colombetti, M. (2003). Defining interaction protocols using a commitment-based agent communication language. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems* (pp. 520–527). ACM, ACM Press. https://doi.org/10.1145/860575.860659.
149. Franceschini, L. (2019). RML: Runtime monitoring language—A system-agnostic DSL for runtime verification. In *3rd International conference on art, science, and engineering of programming*. New York, NY: ACM (pp. 28:1–28:3). https://doi.org/10.1145/3328433.3328462.
150. Freitas, A., Schmidt, D., Panisson, A., Meneguzzi, F., Vieira, R., & Bordini, R. H. (2014). Semantic representations of agent plans and planning problem domains. In Dalpiaz et al. [105] (pp. 351–366). https://doi.org/10.1007/978-3-319-14484-9_18.
151. Freitas, A., Bordini, R. H., & Vieira, R. (2017). Model-driven engineering of multi-agent systems based on ontologies. *Applied Ontology*, *12*(2), 157–188. https://doi.org/10.3233/AO-170182.
152. Freitas, A., Bordini, R. H., & Vieira, R. (2019). Designing multi-agent systems from ontology models. In D. Weyns, V. Mascardi, & A. Ricci (Eds.), *Engineering multi-agent systems, Lecture notes in computer science* (Vol. 11375, pp. 76–95). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-25693-7_5.
153. Fukuta, N., Ito, T., & Shintani, T. (2000). MiLog: A mobile agent framework for implementing intelligent information agents with logic programming. In *Pacific Rim international workshop on intelligent information agents* (pp. 113–123).
154. Fung, T. H., & Kowalski, R. A. (1997). The IFF proof procedure for abductive logic programming. *The Journal of Logic Programming*, *33*(2), 151–165. https://doi.org/10.1016/S0743-1066(97)00026-5.
155. Gabbay, D. (1987). Modal and temporal logic programming. In A. Galton (Ed.), *Temporal logics and their applications* (pp. 197–237). New York: Academic Press Professional, Inc. https://doi.org/10.5555/42251.42257.
156. Gaggl, S. A., Linsbichler, T., Maratea, M., & Woltran, S. (2018). Summary report of the second international competition on computational models of argumentation. *AI Magazine*, *39*(4), 77–79. https://doi.org/10.1609/aimag.v39i4.2781.
157. Gammie, P., & Van Der Meyden, R. (2004). MCK: Model checking the logic of knowledge. In R. Alur & D. A. Peled (Eds.), *International conference on computer aided verification, Lecture notes in computer science* (Vol. 3114, pp. 479–483). Boston, MA: Springer. https://doi.org/10.1007/978-3-540-27813-9_41.
158. Garcia, A. J., & Simari, G. R. (2004). Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming*, *4*(2), 95–138. https://doi.org/10.1017/S1471068403001674.

159. Garcia-Serrano, A., Vioque, D. T., Carbone, F., & Mendez, V. (2003). FIPA-compliant MAS development for road traffic management with a knowledge-based approach: The TRACK-R agents. In *Challenges open agent systems workshop*, Melbourne, Australia.
160. Gelernter, D. (1985). Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, *7*(1), 80–112. https://doi.org/10.1145/2363.2433.
161. Gelfond, M., & Lifschitz, V. (1993). Representing action and change by logic programs. *The Journal of Logic Programming*, *17*(2–4), 301–321. https://doi.org/10.1016/0743-1066(93)90035-F.
162. Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., et al. (2003). The evolution of Protégé: An environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, *58*(1), 89–123. https://doi.org/10.1016/S1071-5819(02)00127-1.
163. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., & Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence, 153*(1), 49–104. https://doi.org/10.1016/j.artint.2002.12.001. Logical formalizations and commonsense reasoning
164. Gomes, A. S., Alferes, J. J., & Swift, T. (2010). Implementing query answering for hybrid MKNF knowledge bases. In M. Carro & R. Peña (Eds.), *Practical aspects of declarative languages, Lecture notes in computer science* (Vol. 5937, pp. 25–39). Berlin: Springer. https://doi.org/10.1007/978-3-642-11503-5_4.
165. Graja, Z., Migeon, F., Maurel, C., Gleizes, M. P., & Kacem, A. H. (2014). A stepwise refinement based development of self-organizing multi-agent systems: Application to the foraging ants. In Dalpiaz et al. [105] (pp. 40–57). https://doi.org/10.1007/978-3-319-14484-9_3.
166. Hahn, C., Madrigal-Mora, C., & Fischer, K. (2009). A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, *18*(2), 239–266. https://doi.org/10.1007/s10458-008-9042-0.
167. Håkansson, A., Nguyen, N. T., Hartung, R. L., Howlett, R. J., & Jain, L. C. (Eds.). (2009). *Agent and multi-agent systems: Technologies and applications, Lecture notes in computer science, 3rd KES international symposium, KES-AMSTA 2009, Uppsala, Sweden, June 3–5, 2009. Proceedings* (Vol. 5559). Berlin: Springer. https://doi.org/10.1007/978-3-642-01665-3.
168. Halaç, T. G., Çetin, Ö., Ekinci, E. E., Erdur, R. C., & Dikenelli, O. (2009). Executing agent plans by reducing to workflows. In M. Baldoni, C. Baroglio, J. Bentahar, G. Boella, M. Cossentino, M. Dastani, et al. (Eds.), *2nd Multi-agent logics, languages, and organisations federated workshops* (Vol. 494). http://ceur-ws.org/Vol-494/ladspaper10.pdf.
169. Hashmi, A., & Fallah-Seghrouchni, A. E. (2009). Temporal planning in dynamic environments for P-CLAIM agents. In M. Baldoni, C. Baroglio, J. Bentahar, G. Boella, M. Cossentino, M. Dastani, et al. (Eds.), *2nd Multi-agent logics, languages, and organisations federated workshops* (Vol. 494). http://ceur-ws.org/Vol-494/ladspaper8.pdf.
170. Havelund, K., & Pressburger, T. (2000). Model checking JAVA programs using JAVA pathfinder. *International Journal on Software Tools for Technology Transfer (STTT)*, *2*(4), 366–381. https://doi.org/10.1007/s100090050043.
171. Hayashi, H., Cho, K., & Ohsuga, A. (2005). A new HTN planning framework for agents in dynamic environments. In J. Dix & J. Leite (Eds.), *Computational logic in multi-agent systems (CLIMA 2004), Lecture notes in computer science* (Vol. 3259). Berlin: Springer (pp. 108–133). https://doi.org/10.1007/978-3-540-30200-1_7.
172. Hayashi, H., Tokura, S., Hasegawa, T., & Ozaki, F. (2005). Dynagent: An incremental forward-chaining HTN planning agent in dynamic domains. In Baldoni et al. [20] (pp. 171–187). https://doi.org/10.1007/11691792_11.
173. Helsinger, A., Thome, M., & Wright, T. (2004). Cougaar: A scalable, distributed multi-agent architecture. In *International conference on systems, man and cybernetics* (Vol. 2, pp. 1910–1917). The Hague: IEEE. https://doi.org/10.1109/ICSMC.2004.1399959.
174. Hendler, J. (2001). Agents and the semantic web. *IEEE Intelligent Systems*, *16*(2), 30–37. https://doi.org/10.1109/5254.920597.
175. Henz, M., Smolka, G., & Würtz, J. (1993). Oz—A programming language for multi-agent systems. In R. Bajcsy (Ed.), *13th International joint conference on artificial intelligence* (pp. 404–409). Chambéry: Morgan Kaufmann Publishers Inc.
176. Higgins, J. P. T., & Green, S. (Eds.). (2008). *Cochrane handbook for systematic reviews of interventions, Cochrane book series*. New York: Wiley. https://doi.org/10.1002/9780470712184.
177. Hill, E. F. (2003). *Jess in action: Java rule-based systems*. Greenwich, CT: Manning Publications Co.
178. Himoff, J., Skobelev, P., & Wooldridge, M. (2005). MAGENTA technology. In *4th International joint conference on autonomous agents and multiagent systems* (pp. 60–66). ACM. https://doi.org/10.1145/1082473.1082805.

179. Hindriks, K. V. (2009). Programming rational agents in GOAL. In A. El Fallah Seghrouchni, J. Dix, M. Dastani, & R. H. Bordini (Eds.), *Multi-agent programming: Languages, tools and applications*. Boston, MA: Springer (pp. 119–157). https://doi.org/10.1007/978-0-387-89299-3_4.
180. Hindriks, K. V. (2014). The shaping of the agent-oriented mindset. In Dalpiaz et al. [105] (pp. 1–14). https://doi.org/10.1007/978-3-319-14484-9_1.
181. Hindriks, K. V., & van Riemsdijk, M. B. (2008). Using temporal logic to integrate goals and qualitative preferences into agent programming. In Baldoni et al. [21] (pp. 215–232). https://doi.org/10.1007/978-3-540-93920-7_14.
182. Hindriks, K. V., de Boer, F. S., van der Hoek, W., & Meyer, J. C. (1999). Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, *2*(4), 357–401. https://doi.org/10.1023/A:1010084620690.
183. Hindriks, K. V., de Boer, F. S., van der Hoek, W., & Meyer, J. J. C. (2000). Agent programming with declarative goals. In C. Castelfranchi & Y. Lespérance (Eds.) *Intelligent agents VII. Agent theories architectures and languages, 7th international workshop, ATAL 2000, Boston, MA, USA, July 7–9, 2000, proceedings, Lecture notes in computer science* (Vol. 1986, pp. 228–243). Berlin: Springer. https://doi.org/10.1007/3-540-44631-1_16.
184. Hjelmblom, M. (2008). *Deontic action-logic multi-agent systems in Prolog*. Tech. Rep. 30, Uppsala University, this report is originally a thesis work for degree of Master of Science in Computer Science at Uppsala University. The thesis was written and presented under the author's former name, Magnus Blom.
185. Hjelmblom, M., & Odelstad, J. (2009). jDALMAS: A Java/Prolog framework for deontic action-logic multi-agent systems. In Håkansson et al. [167] *3rd KES International symposium, KES-AMSTA 2009, Uppsala, Sweden, June 3–5, 2009. Proceedings* (pp. 110–119). https://doi.org/10.1007/978-3-642-01665-3_12.
186. Hluchý, L., Laclavik, M., Balogh, Z., & Babik, M. (2006). Agentowl: Semantic knowledge model and agent architecture. *Computers and Artificial Intelligence*, *25*(5), 421–439.
187. Hollan, J., Hutchins, E., & Kirsh, D. (2000). Distributed cognition: Toward a new foundation for human–computer interaction research. *ACM Transactions on Computer-Human Interaction (TOCHI)*, *7*(2), 174–196. https://doi.org/10.1145/353485.353487.
188. Holzmann, G. J. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, *23*(5), 279–295. https://doi.org/10.1109/32.588521.
189. Hopton, L., Cliffe, O., Vos, M.D., & Padget, J. (2009). InstQL: A query language for virtual institutions using answer set programming. In Dix et al. [127] (pp. 102–121). https://doi.org/10.1007/978-3-642-16867-3_6.
190. Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, *16*(1), 14–21. https://doi.org/10.2307/2268661.
191. Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M., et al. (2004). SWRL: A semantic web rule language combining OWL and RuleML. https://www.w3.org/Submission/SWRL/.
192. Howden, N., Rönnquist, R., Hodgson, A., & Lucas, A. (2001). JACK intelligent agents—Summary of an agent infrastructure. In *2nd International workshop on infrastructure for agents, MAS, and scalable MAS*.
193. Hubner, J. F., Sichman, J. S., & Boissier, O. (2002). Moise+: Towards a structural, functional, and deontic model for MAS organization. In *First international joint conference on autonomous agents and multiagent systems: Part 1 (AAMAS '02)* (pp. 501–502). New York, NY: ACM. https://doi.org/10.1145/544741.544858.
194. Hübner, J. F., Boissier, O., & Bordini, R. H. (2010). From organisation specification to normative programming in multi-agent organisations. In Dix et al. [128] (pp. 117–134). https://doi.org/10.1007/978-3-642-14977-1_11.
195. Hustadt, U., Dixon, C., Schmidt, R. A., Fisher, M., Meyer, J. J., & Van Der Hoek, W. (2000). Verification within the KARO agent theory. In J. Rash, W. Truszkowski, M. Hinchey, C. Rouff, & D. Gordon (Eds.), *Formal approaches to agent-based systems* (Vol. 1871, pp. 33–47). Berlin: Springer. https://doi.org/10.1007/3-540-45484-5_3.
196. Iglesias, C. A., Garijo, M., González, J. C., & Velasco, J. R. (1998). Analysis and design of multiagent systems using MAS-CommonKADS. In M. P. Singh, A. Rao, & M. J. Wooldridge (Eds.), *Intelligent agents IV agent theories, architectures, and languages* (pp. 313–327). Berlin: Springer. https://doi.org/10.1007/bfb0026768.
197. Ilghami, O. (2006). *Documentation for JSHOP2*. Technical Report CS-TR-4694, Department of Computer Science, University of Maryland.
198. Inoue, K., Satohand, F., & Toni, K. (Eds.). (2006). *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 4371). Berlin: Springer. https://doi.org/10.1007/978-3-540-69619-3.

199. Inter4QL. (2019). *Homepage*. http://4ql.org/a-downloads-inter4ql.html.
200. Jaffar, J., & Lassez, J. (1987). Constraint logic programming. In *14th ACM SIGACT-SIGPLAN symposium on principles of programming languages* (pp. 111–119). Munich: ACM. https://doi.org/10.1145/41625.41635.
201. Jensen, A. S., Dignum, V., & Villadsen, J. (2014). The AORTA architecture: Integrating organizational reasoning in Jason. In Dalpiaz et al. [105] (pp. 127–145). https://doi.org/10.1007/978-3-319-14484-9_7.
202. Jongmans, S. S. T. Q., Hindriks, K. V., & van Riemsdijk, M. B. (2010). Model checking agent programs by using the program interpreter. In Dix et al. [128] (pp. 219–237). https://doi.org/10.1007/978-3-642-14977-1_17.
203. Kacprzak, M., Lomuscio, A., Niewiadomski, A., Penczek, W., Raimondi, F., & Szreter, M. (2006). Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. *Fundamenta Informaticae*, *72*(1–3), 215–234.
204. Kakas, A. C., Kowalski, R. A., & Toni, F. (1992). Abductive logic programming. *Journal of Logic and Computation*, *2*(6), 719–770. https://doi.org/10.1093/logcom/2.6.719.
205. Kanger, S. (1971). New foundations for ethical theory. In R. Hilpinen (Ed.), *Deontic logic: Introductory and systematic readings, synthese library (Studies in epistemology, logic, methodology, and philosophy of science)* (Vol. 33, pp. 36–58). Dordrecht: Springer. https://doi.org/10.1007/978-94-010-3146-2_2.
206. Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., Linkman, S. (2009). Systematic literature reviews in software engineering—A systematic literature review. *Information and Software Technology, 51*(1), 7–15. https://doi.org/10.1016/j.infsof.2008.09.009. Special section—Most cited articles in 2002 and regular research papers
207. Klapiscak, T., & Bordini, R. H. (2008). JASDL: A practical programming approach combining agent and semantic web technologies. In Baldoni et al. [21] (pp. 91–110). https://doi.org/10.1007/978-3-540-93920-7_7.
208. Koeman, V. J., Hindriks, K. V., & Jonker, C. M. (2016). Using automatic failure detection for cognitive agents in Eclipse (AAMAS 2016 DEMONSTRATION). In M. Baldoni, J. P. Muller, I. Nunes, & R. Zalila-Wenkstern (Eds.), *Engineering multi-agent systems, Lecture notes in computer science* (Vol. 10093, pp. 59–80). Cham: Springer. https://doi.org/10.1007/978-3-319-50983-9_4.
209. Köhler-Bußmeier, M., & Wester-Ebbinghaus, M. (2009). SONAR: A multi-agent infrastructure for active application architectures and inter-organisational information systems. In L. Braubach, W. van der Hoek, P. Petta, & A. Pokahr (Eds.), *Multiagent system technologies, Lecture notes in computer science* (Vol. 5774, pp. 248–257). Berlin: Springer. https://doi.org/10.1007/978-3-642-04143-3_27.
210. Kollia, I., Glimm, B., & Horrocks, I. (2011). SPARQL query answering over owl ontologies. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer, et al. (Eds.), *The semantic web: Research and applications, Lecture notes in computer science* (Vol. 6643, pp. 382–396). Berlin: Springer. https://doi.org/10.1007/978-3-642-21034-1_26.
211. Konnerth, T., Hirsch, B., & Albayrak, S. (2006). JADL—An agent description language for smart agents. In Baldoni and Endriss [17] (pp. 141–155). https://doi.org/10.1007/11961536_10.
212. Kowalski, R. A. (1995). Logical foundations for multi-agent systems. In M. Alpuente & M. I. Sessa (Eds.), *Conference on declarative programming, GULP-PRODE'95, Marina di Vietri, Italy* (pp. 39–40).
213. Kowalski, R. A., & Sadri, F. (1999). From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, *25*(3–4), 391–419. https://doi.org/10.1023/A:1018934223383.
214. Kraus, S., Sycara, K., & Evenchik, A. (1998). Reaching agreements through argumentation: A logical model and implementation. *Artificial Intelligence*, *104*(1–2), 1–69. https://doi.org/10.1016/S0004-3702(98)00078-2.
215. Kravari, K., Kontopoulos, E., & Bassiliades, N. (2010). EMERALD: A multi-agent system for knowledge-based reasoning interoperability in the semantic web. In S. Konstantopoulos, S. Perantonis, V. Karkaletsis, C. D. Spyropoulos, & G. Vouros (Eds.), *Artificial intelligence: Theories, models and applications, Lecture notes in computer science* (Vol. 6040, pp. 173–182). Berlin: Springer. https://doi.org/10.1007/978-3-642-12842-4_21.
216. Kravari, K., Papatheodorou, K., Antoniou, G., & Bassiliades, N. (2011). Extending a multi-agent reasoning interoperability framework with services for the semantic web logic and proof layers. In N. Bassiliades, G. Governatori, & A. Paschke (Eds.), *Rule-based reasoning, programming, and applications* (pp. 29–43). Berlin: Springer. https://doi.org/10.1007/978-3-642-22546-8_4.
217. Kroiß, C. (2014). A statistical model checker for situation calculus based multi-agent models. In *2014 International conference on autonomous agents and multi-agent systems (AAMAS'14), IFAAMAS, Richland, SC, USA* (pp. 1567–1568). http://dl.acm.org/citation.cfm?id=2615731.2616065.

218. Kroiß, C., & Bureš, T. (2016). Logic-based modeling of information transfer in cyber-physical multi-agent systems. *Future Generation Computer Systems*, *56*(C), 124–139. https://doi.org/10.1016/j.future.2015.09.013.
219. Kwiatkowska, M., Norman, G., & Parker, D. (2002). PRISM: Probabilistic symbolic model checker. In T. Field, P. G. Harrison, J. Bradley, & U. Harder (Eds.), *Computer performance evaluation: Modelling techniques and tools, Lecture notes in computer science* (Vol. 2324, pp. 200–204). Berlin: Springer. https://doi.org/10.1007/3-540-46029-2_13.
220. Lam, H. P., & Governatori, G. (2009). The making of SPINdle. In A. Paschke, G. Governatori, & J. Hall (Eds.), *Rule interchange and applications (RuleML 2009), RuleML* (pp. 315–322). Berlin: Springer. https://doi.org/10.1007/978-3-642-04985-9_29.
221. Lange, D. B., & Mitsuru, O. (1998). *Programming and deploying Java Mobile Agents Aglets*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.
222. Larsen, J. B. (2019). Adding organizational reasoning to agent-based simulations in GAMA. In Weyns et al. [351] (pp. 242–262). https://doi.org/10.1007/978-3-030-25693-7_13.
223. Lee, J., Padget, J., Logan, B., Dybalova, D., & Alechina, N. (2014). N-Jason: Run-time norm compliance in AgentSpeak(l). In Dalpiaz et al. [105] (pp. 367–387). https://doi.org/10.1007/978-3-319-14484-9_19.
224. Leite, J. A., Alferes, J. J., & Pereira, L. M. (2002). MINERVA—A dynamic logic programming agent architecture. In *Intelligent agents VIII, Lecture notes in computer science* (Vol. 2333, pp. 141–157). Berlin: Springer. https://doi.org/10.1007/3-540-45448-9_11.
225. Leite, J. A., Omicini, A., Sterling, L., & Torroni, P. (Eds.). (2004). *Declarative agent languages and technologies I, Lecture notes in computer science* (Vol. 2990). Berlin: Springer. https://doi.org/10.1007/b97923.
226. Leite, J. A., Omicini, A., Torroni, P., & Yolum, P. (Eds.). (2005). *Declarative agent languages and technologies II, Lecture notes in computer science* (Vol. 3476). Berlin: Springer. https://doi.org/10.1007/b136890.
227. Leite, J. A., Son, T. C., Torroni, P., van der Torre, L., & Woltran, S. (Eds.). (2013). *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 8143). Berlin: Springer. https://doi.org/10.1007/978-3-642-40624-9.
228. Levesque, H. J., Reiter, R., Lespérance, Y., Lin, F., & Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, *31*(1–3), 59–83.
229. Li, S. (2007). AgentStra: An Internet-based multi-agent intelligent system for strategic decision-making. *Expert Systems with Applications*, *33*(3), 565–571. https://doi.org/10.1016/j.eswa.2006.05.018.
230. Liffiton, M. H., & Sakallah, K. A. (2007). Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, *40*(1), 1–33. https://doi.org/10.1007/s10817-007-9084-z.
231. Lifschitz, V. (1999). Action languages, answer sets, and planning. In K. R. Apt, V. W. Marek, M. Truszczynski, & D. S. Warren (Eds.), *The logic programming paradigm—A 25-year perspective, Artificial intelligence* (pp. 357–373). Berlin: Springer. https://doi.org/10.1007/978-3-642-60085-2_16.
232. Lillis, D., & Collier, R. W. (2010). ACRE: Agent communication reasoning engine. In O. Boissier, A. E. Fallah-Seghrouchni, S. Hassas, & N. Maudet (Eds.), *Proceedings of the multi-agent logics, languages, and organisations federated workshops (MALLOW 2010), CEUR workshop proceedings* (Vol. 627). https://CEUR-WS.org. http://ceur-ws.org/Vol-627/lads_1.pdf.
233. Lindahl, L. (2012). *Position and change: A study in law and logic, Synthese library* (Vol. 112). Berlin: Springer. https://doi.org/10.1007/978-94-010-1202-7.
234. Lloyd, J. W. (1987). *Foundations of logic programming* (2nd ed.). Berlin: Springer. https://doi.org/10.1007/978-3-642-83189-8.
235. Lomuscio, A., & Raimondi, F. (2006). The complexity of model checking concurrent programs against CTLK specifications. In Baldoni and Endriss [17] (pp. 29–42). https://doi.org/10.1007/11961536_3.
236. Lomuscio, A., & Raimondi, F. (2006). MCMAS: A model checker for multi-agent systems. In H. Hermanns & J. Palsberg (Eds.), *Tools and algorithms for the construction and analysis of systems. TACAS 2006, Lecture notes in computer science* (Vol. 3920, pp. 450–454). Berlin: Springer. https://doi.org/10.1007/11691372_31.
237. Lomuscio, A., Qu, H., & Raimondi, F. (2017). MCMAS: An open-source model checker for the verification of multi-agent systems. *International Journal on Software Tools for Technology Transfer*, *19*(1), 9–30. https://doi.org/10.1007/s10009-015-0378-x.
238. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, *81*(7), 517–527. https://doi.org/10.1177/0037549705058073.

239. Lützenberger, M., Küster, T., & Konnerth, T. (2013). JIAC V—A MAS framework for industrial applications. In *International conference on autonomous agents and multi-agent systems, IFAAMAS* (pp. 1189–1190). http://dl.acm.org/citation.cfm?id=2485136.
240. Lützenberger, M., Konnerth, T., & Küster, T. (2015). Programming of multiagent applications with JIAC. In P. Leitão & S. Karnouskos (Eds.), *Industrial agents: Emerging applications of software agents in industry* (pp. 381–398). Boston: Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-800341-1.00021-8.
241. Ma, J., Russo, A., Broda, K., & Clark, K. (2008). DARE: A system for distributed abductive reasoning. *Autonomous Agents and Multi-Agent Systems*, *16*(3), 271–297. https://doi.org/10.1007/s10458-008-9028-y.
242. Malheiro, B., & Oliveira, E. (2001). Argumentation as distributed belief revision: Conflict resolution in decentralised co-operative multi-agent systems. In P. Brazdil & A. Jorge (Eds.), Progress in artificial intelligence, lecture notes in computer science (Vol. 2258, pp. 205–218). Berlin: Springer. https://doi.org/10.1007/3-540-45329-6_22.
243. Maluszyński, J., & Szalas, A. (2011). Logical foundations and complexity of 4QL, a query language with unrestricted negation. *Journal of Applied Non-Classical Logics*, *21*(2), 211–232. https://doi.org/10.3166/jancl.21.211-232.
244. Martelli, M., Mascardi, V., & Zini, F. (1999). Specification and simulation of multi-agent systems in caselp. In M. C. Meo & M. V. Ferro (Eds.), *Conference on declarative programming, AGP'99, L'Aquila* (pp. 13–28).
245. Mascardi, V., Martelli, M., & Sterling, L. (2004). Logic-based specification languages for intelligent software agents. *Theory and Practice of Logic Programming*, *4*(4), 429–494. https://doi.org/10.1017/S1471068404002029.
246. Mascardi, V., Demergasso, D., & Ancona, D. (2005). Languages for programming BDI-style agents: An overview. In *6th Workshop "From objects to agents" (WOA 2005)* (Vol. 2005, pp. 9–15).
247. Mascardi, V., Martelli, M., & Gungui, I. (2007). DCaseLP: A prototyping environment for multi-language agent systems. In M. Dastani, A. E. Fallah-Seghrouchni, J. Leite, & P. Torroni (Eds.), *Languages, methodologies and development tools for multi-agent systems, Lecture notes in computer science, 1st international workshop, LADS 2007, Durham, UK, September 4–6, 2007* (Vol. 5118, pp. 139–155). Berlin: Springer. https://doi.org/10.1007/978-3-540-85058-8_9. Revised selected papers.
248. Mascardi, V., Hendler, J., & Papaleo, L. (2012). Semantic web and declarative agent languages and technologies: Current and future trends. In Baldoni et al. [25]. https://doi.org/10.1007/978-3-642-37890-4_12.
249. Mascardi, V., Ancona, D., Barbieri, M., Bordini, R. H., & Ricci, A. (2014). CooL-AgentSpeak: Endowing AgentSpeak-DL agents with plan exchange and ontology services. *Web Intelligence and Agent Systems*, *12*(1), 83–107. https://doi.org/10.3233/WIA-140287.
250. Matwin, S., Szpakowicz, S., Koperczak, Z., Kersten, G. E., & Michalowski, W. (1989). Negoplan: An expert system shell for negotiation support. *IEEE Intelligent Systems*, *4*(4), 50–62. https://doi.org/10.1109/64.43285.
251. McCabe, F. G., & Clark, K. L. (1995). April—Agent process interaction language. In M. J. Wooldridge & N. R. Jennings (Eds.), *Intelligent agents, Lecture notes in computer science* (Vol. 890, pp. 324–340). Berlin: Springer. https://doi.org/10.1007/3-540-58855-8_21.
252. McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM*, *3*(4), 184–195. https://doi.org/10.1145/367177.367199.
253. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., et al. (1998). *PDDL—The planning domain definition language*. https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf.
254. McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems*, *16*(2), 46–53. https://doi.org/10.1109/5254.920599.
255. Meneguzzi, F., & Luck, M. (2007). Composing high-level plans for declarative agent programming. In M. Baldoni, T. C. Son, M. B. van Riemsdijk, & M. Winikoff (Eds.), *Declarative agent languages and technologies V, Lecture notes in computer science* (Vol. 4897, pp. 69–85). Berlin: Springer. https://doi.org/10.1007/978-3-540-77564-5_5.
256. Metakides, G., & Nerode, A. (1996). Foreword. In *Studies in computer science and artificial intelligence* (Vol. 13, pp. vii–ix). Amsterdam: Elsevier. https://doi.org/10.1016/S0924-3542(96)80008-9.
257. Morales, J. L., Sánchez, P., & Alonso, D. (2014). A systematic literature review of the teleo-reactive paradigm. *Artificial Intelligence Review*, *42*(4), 945–964. https://doi.org/10.1007/s10462-012-9350-2.
258. Moreira, Á. F., Vieira, R., Bordini, R. H., & Hübner, J. F. (2005). Agent-oriented programming with underlying ontological reasoning. In Baldoni et al. [20] (pp. 155–170). https://doi.org/10.1007/11691792_10.

259. Moszkowski, B. C. (1986). *Executing temporal logic programs*. Cambridge: Cambridge University Press.
260. Motomura, S., Kawamura, T., & Sugahara, K. (2005). Maglog: A mobile agent framework for distributed models. In *17th IASTED PDCS international conference, Phoenix, AZ, USA* (pp. 414–420). http://www.actapress.com/Abstract.aspx?paperId=22336.
261. Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*(3–4), 245–286. https://doi.org/10.1007/BF03037227.
262. Muldoon, C., O'Hare, G. M. P., Collier, R., & O'Grady, M. J. (2006). Agent factory micro edition: A framework for ambient applications. In V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, & J. Dongarra (Eds.), *Computational science—ICCS 2006, Lecture notes in computer science* (Vol. 3993, pp. 727–734). Berlin: Springer. https://doi.org/10.1007/11758532_95.
263. Mulrow, C. D. (1994). Systematic reviews: Rationale for systematic reviews. *British Medical Journal*, *309*(6954), 597–599. https://doi.org/10.1136/bmj.309.6954.597.
264. Nardini, E., Omicini, A., & Viroli, M. (2011). Description spaces with fuzziness. In M. J. Palakal, C. C. Hung, W. Chu, & W. E. Wong (Eds.), *26th Annual ACM symposium on applied computing (SAC 2011), artificial intelligence and agents, information systems, and software development, ACM, Tunghai University, TaiChung, Taiwan* (Vol. II, pp. 869–876). https://doi.org/10.1145/1982185.1982375.
265. Nardini, E., Omicini, A., & Viroli, M. (2013). Semantic tuple centres. *Science of Computer Programming*, *78*(5), 569–582. https://doi.org/10.1016/j.scico.2012.10.004. Special section: Self-organizing coordination.
266. Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. In *16th International joint conference on artificial intelligence (IJCAI 1999)* (Vol. 2, pp. 968–973). Stockholm: Morgan Kaufmann Publishers Inc. https://www.ijcai.org/Proceedings/1999-2.
267. Nau, D. S., Au, T., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., et al. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379–404. https://doi.org/10.1613/jair.1141.
268. Niewiadomski, A., Penczek, W., & Szreter, M. (2004). VerICS 2004: A model checker for real time and multi-agent systems. In *International workshop on concurrency, specification and programming (CS&P'04)* (Vol. 170, pp. 88–99). Humboldt-Univeristaet, Berlino, Informatik-Berichte.
269. Nilsson, N. J. (2001). Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence*, *5*(B), 99–110.
270. Nowostawski, M., Purvis, M. K., Oliveira, M. D., & Cranefield, S. (2006). Institutions in the OPAL multi-agent system. *Journal of Intelligent and Fuzzy Systems*, *17*(3), 191–207.
271. Nute, D. (1988). Defeasible reasoning: A philosophical analysis in Prolog. In J. H. Fetzer (Ed.), *Aspects of artificial intelligence* (pp. 251–288). Dordrecht: Springer.
272. Nute, D. (1994). Defeasible logic. In D. M. Gabbay, C. J. Hogger, & J. A. Robinson (Eds.), *Handbook of logic in artificial intelligence and logic programming (chapter 7)* (Vol. 3, pp. 353–395). New York, NY: Oxford University Press, Inc.
273. Omicini, A. (2001). SODA: Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini & M. J. Wooldridge (Eds.), *Agent-oriented software engineering, Lecture notes in computer science, 1st international workshop (AOSE 2000), Limerick, Ireland, 10 June 2000* (Vol. 1957, pp. 185–193). Berlin: Springer. https://doi.org/10.1007/3-540-44564-1_12. Revised papers.
274. Omicini, A., & Calegari, R. (2019). Injecting (micro)intelligence in the IoT: Logic-based approaches for (M)MAS. In D. Lin, T. Ishida, F. Zambonelli, & I. Noda (Eds.), *Massively multi-agent systems II, Lecture notes in computer science (chapter 2), international workshop, MMAS 2018, Stockholm, Sweden, July 14, 2018* (Vol. 11422, pp. 21–35). Berlin: Springer. https://doi.org/10.1007/978-3-030-20937-7_2. Revised selected papers.
275. Omicini, A., & Denti, E. (2001). From tuple spaces to tuple centres. *Science of Computer Programming*, *41*(3), 277–294. https://doi.org/10.1016/S0167-6423(01)00011-9.
276. Omicini, A., & Zambonelli, F. (1999). Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, *2*(3), 251–269. https://doi.org/10.1023/A:1010060322135. Special issue: Coordination mechanisms for web agents.
277. Omicini, A., & Zambonelli, F. (2004). MAS as complex systems: A view on the role of declarative approaches. In Leite et al. [225] (pp. 1–17). https://doi.org/10.1007/978-3-540-25932-9_1.
278. Omicini, A., Ricci, A., & Zaghini, N. (2006). Distributed workflow upon linkable coordination artifacts. In P. Ciancarini & H. Wiklicky (Eds.), *Coordination models and languages, Lecture notes in computer science* (Vol. 4038, pp. 228–246). Berlin: Springer. https://doi.org/10.1007/11767954_15.
279. Omicini, A., Ricci, A., & Viroli, M. (2007). Timed environment for Web agents. *Web Intelligence and Agent Systems*, *5*(2), 161–175.

280. Omicini, A., Ricci, A., & Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems, 17*(3), 432–456. https://doi.org/10.1007/s10458-008-9053-x. Special issue on Foundations, advanced topics and industrial perspectives of multi-agent systems.
281. Omicini, A., Sardina, S., & Vasconcelos, W. (Eds.). (2010). *Declarative agent languages and technologies VIII, Lecture notes in computer science* (Vol. 6619). Berlin: Springer. https://doi.org/10.1007/978-3-642-20715-0.
282. Orgun, B., Dras, M., Cassidy, S., & Nayak, A. (2005). DASMAS: Dialogue based automation of semantic interoperability in multi agent systems. In T. Meyer & M. A. Orgun. (Eds.), *Australasian ontology workshop* (Vol. 58, pp. 75–82). Sydney: Australian Computer Society, Inc.
283. Pan, Y., Tu, P. H., Pontelli, E., & Son, T. C. (2005). Construction of an agent-based framework for evolutionary biology: A progress report. In J. Leite, A. Omicini, P. Torroni, & P. Yolum (Eds.), *Declarative agent languages and technologies* (pp. 92–111). Berlin: Springer. https://doi.org/10.1007/11493402_6.
284. Pantoja, C. E., Stabile, M. F., Lazarin, N. M., & Sichman, J. S. (2016) ARGO: An extended Jason architecture that facilitates embedded robotic agents programming. In M. Baldoni, J. P. Müller, I. Nunes, & R. Zalila-Wenkstern (Eds.), *Engineering multi-agent systems (EMAS 2016), Lecture notes in computer science* (Vol. 10093, pp. 136–155). Berlin: Springer. https://doi.org/10.1007/978-3-319-50983-9_8.
285. Pilecki, J., Bednarczyk, M. A., & Jamroga, W. (2014). Synthesis and verification of uniform strategies for multi-agent systems. In Bulling et al. [57] (pp. 166–182). https://doi.org/10.1007/978-3-319-09764-0_11.
286. Platon, E., Mamei, M., Sabouret, N., Honiden, S., & Parunak, H. V. D. (2007). Mechanisms for environments in multi-agent aystems: Survey and opportunities. *Autonomous Agents and Multi-Agent Systems*, *14*(1), 31–47. https://doi.org/10.1007/s10458-006-9000-7. Special issue on Environments for multi-agent systems.
287. Pnueli, A. (1977). The temporal logic of programs. In *18th Annual symposium on foundations of computer science* (pp. 46–57). Providence, RI: IEEE Computer Society. https://doi.org/10.1109/SFCS.1977.32.
288. Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine. In Håkansson et al. [167] (pp. 149–174). https://doi.org/10.1007/0-387-26350-0_6. 3rd KES International symposium, KES-AMSTA 2009, Uppsala, Sweden, June 3–5, 2009. Proceedings.
289. Pokorny, L. R., & Ramakrishnan, C. R. (2005). Modeling and verification of distributed autonomous agents using logic programming. In Leite et al. [225] (pp. 148–165). https://doi.org/10.1007/11493402_9.
290. Pollock, J. L. (1996). OSCAR: A general-purpose defeasible reasoner. *Journal of Applied Non-classical Logics*, *6*(1), 89–113.
291. Popper, K. R. (2002). *The logic of scientific discovery* (1st English edition: 1959). London: Routledge.
292. Potassco UoP the Potsdam Answer Set Solving Collection. (2019). *clingo and gringo*. https://potassco.org/clingo/.
293. Prakken, H. (2010). An abstract framework for argumentation with structured arguments. *Argument and Computation*, *1*(2), 93–124. https://doi.org/10.1080/19462160903564592.
294. Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde & J. W. Perram (Eds.), *Agents breaking away, Lecture notes in computer science* (Vol. 1038, pp. 42–55). Berlin: Springer. https://doi.org/10.1007/BFb0031845. 7th European workshop on modelling autonomous agents in a multi-agent world, MAAMAW '96 Eindhoven, The Netherlands, January 22–25, 1996 proceedings.
295. Rao, A. S., & Georgeff, M. P. (1991). Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics. In J. Mylopoulos & R. Reiter (Eds.), *12th International joint conference on artificial intelligence. Sydney, Australia, August 24–30, 1991* (pp. 498–505). Morgan Kaufmann. http://ijcai.org/Proceedings/91-1/Papers/077.pdf.
296. Rao, A. S., & Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In J. F. Allen, R. Fikes, & E. Sandewall (Eds.), *2nd International conference on principles of knowledge representation and reasoning (KR'91)* (pp. 473–484). Morgan Kaufmann Publishers Inc. https://doi.org/10.5555/3087158.3087205.
297. Rao, A. S., & Georgeff, M. P. (1993). A model-theoretic approach to the verification of situated reasoning systems. In R. Bajcsy (Ed.), *13th International joint conference on artificial intelligence. Chambéry, France, August 28–September 3, 1993* (pp. 318–324). Morgan Kaufmann. http://ijcai.org/Proceedings/93-1/Papers/045.pdf.
298. Ricci, A., Piunti, M., & Viroli, M. (2011). Environment programming in multi-agent systems: An artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, *23*(2), 158–192. https://doi.org/10.1007/s10458-010-9140-7. Special issue: Multi-agent programming.

299. Ricci, A., Bordini, R. H., Hübner, J. F., & Collier, R. (2019). *AgentSpeak(ER): Enhanced encapsulation in agent plans*. In Weyns et al. [351] (pp. 34–51). https://doi.org/10.1007/978-3-030-25693-7_3.
300. Riemsdijk, B., Hoek, W., & Meyer, J. J. C. (2003). Agent programming in Dribble: From beliefs to goals with plans. In *Formal approaches to agent-based systems, Lecture notes in computer science* (Vol. 2699, pp. 294–295). Berlin: Springer. https://doi.org/10.1007/978-3-540-45133-4_32.
301. Rijk, L. M. D. (2002). *Aristotle: Semantics and ontology. Volume I: General Introduction. The works on logic, Philosophia Antiqua* (Vol. 91). Leiden: Brill Academic Publishers.
302. Rodriguez, S., Gaud, N., & Galland, S. (2014). SARL: A general-purpose agent-oriented programming language. In *2014 IEEE/WIC/ACM International joint conferences on web intelligence (WI) and intelligent agent technologies (IAT)* (Vol. 3, pp. 103–110). IEEE. https://doi.org/10.1109/WI-IAT.2014.156.
303. Roy, P. V., & Haridi, S. (1999). Mozart: A programming system for agent applications. In D. De Schreye (Ed.), *International workshop on distributed and internet programming with logic and constraint languages, MIT Press, Las Cruces, New Mexico, USA, Roy Van Peter, Haridi Seif, Skarmeas Nikolaos, Clark L. Keith, Maamar Zakaria and Brueckner A. Sven, part of International conference on logic programming (ICLP'99)*.
304. Russell, S., Jordan, H., O'Hare, G. M. P., & Collier, R. W. (2011). Agent factory: A framework for prototyping logic-based AOP languages. In F. Klügl & S. Ossowski (Eds.), *Multiagent system technologies* (pp. 125–136). Berlin: Springer. https://doi.org/10.1007/978-3-642-24603-6_13.
305. Russell, S. J., & Norvig, P. (2002). *Artificial intelligence: A modern approach*, 2nd edn. Englewood Cliffs, NJ: Prentice Hall/Pearson Education International. http://vig.prenhall.com/catalog/academic/product/1,4096,0137903952,00.html.
306. Sadri, F., & Satoh, K. (Eds.). (2007). *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 5056). Berlin: Springer. https://doi.org/10.1007/978-3-540-88833-8.
307. Sadri, F., & Toni, F. (1999). *Computational logic and multi-agent systems: A roadmap*. Technical Report, Imperial College, London, UK: Department of Computing.
308. Sadri, F., Stathis, K., & Toni, F. (2006). Normative KGP agents. *Computational & Mathematical Organization Theory*, *12*(2–3), 101–126. https://doi.org/10.1007/s10588-006-9539-5.
309. Sagonas, K., Swift, T., & Warren, D. S. (1994). XSB as an efficient deductive database engine. *SIGMOD Record*, *23*(2), 442–453. https://doi.org/10.1145/191843.191927.
310. Schreiber, G. T., & Akkermans, H. (2000). *Knowledge engineering and management: The CommonKADS methodology*. Cambridge, MA: MIT Press. https://doi.org/10.5555/347025.
311. Seghrouchni, A. E. F., & Suna, A. (2005). Claim and Sympa: A programming environment for intelligent and mobile agents. In R. H. Bordini, M. Dastani, J. Dix, & A. El Fallah Seghrouchni (Eds.), *Multi-agent programming* (Vol. 15, pp. 95–122). Boston, MA: Springer. https://doi.org/10.1007/0-387-26350-0_4.
312. Serafini, L., & Tamilin, A. (2005). DRAGO: Distributed reasoning architecture for the semantic web. In A. Gómez-Pérez & J. Euzenat (Eds.), *The semantic web: Research and applications* (pp. 361–376). Berlin: Springer. https://doi.org/10.1007/11431053_25.
313. Sesseler, R. (2002). *Eine modulare architektur für dienstbasierte interaktion zwischen agenten*. Doctoral thesis, Technischen Universität Berlin.
314. Shakarian, P., Simari, G. I., & Schroeder, R. (2013). MANCaLog: A logic for multi-attribute network cascades. In M. L. Gini, O. Shehory, T. Ito, & C. M. Jonker (Eds.), *International conference on autonomous agents and multi-agent systems, IFAAMAS* (pp. 1175–1176). http://dl.acm.org/citation.cfm?id=2485129.
315. Shapiro, S., Lespérance, Y., & Levesque, H. J. (2002). The cognitive agents specification language and verification environment for multiagent systems. In *1st International joint conference on autonomous agents and multiagent systems (AAMAS 2002)* (Vol. 1, pp. 19–26). Bologna: ACM. https://doi.org/10.1145/544741.544746.
316. Shi, Z., Li, Y., Wang, W., Cao, H., & Jiang, T. (1998). AOSDE: An agent-oriented software development environment. In *International conference on multi agent systems (ICMAS 1998)*. IEEE. https://doi.org/10.1109/ICMAS.1998.699289.
317. Shi, Z., Zhang, H., Cheng, Y., Jiang, Y., Sheng, Q., & Zhao, Z. (2004). MAGE: An agent-oriented software engineering environment. In *3rd International conference on cognitive informatics*. Victoria: IEEE. https://doi.org/10.1109/COGINF.2004.1327482.
318. Shoham, Y. (1991). AGENT0: A simple agent language and its interpreter. In T. L. Dean & K. R. McKeown (Eds.), *Proceedings of the 9th national conference on artificial intelligence* (Vol. 2, pp. 704–709). AAAI Press/The MIT Press. http://www.aaai.org/Library/AAAI/1991/aaai91-110.php.
319. Shoham, Y. (1993). Agent-oriented programming. *Artificial Intelligence*, *60*(1), 51–92. https://doi.org/10.1016/0004-3702(93)90034-9.

320. Simon, L., Mallya, A., Bansal, A., & Gupta, G. (2006). Coinductive logic programming. In S. Etalle & M. Truszczynski (Eds.), *22nd International conference on logic programming (ICLP 2006), Lecture notes in computer science* (Vol. 4079, pp. 330–345). Berlin: Springer. https://doi.org/10.1007/11799573_25.
321. Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, *5*(2), 51–53. https://doi.org/10.1016/j.websem.2007.03.004.
322. Šišlák, D., Volf, P., Kopřiva, Š., & Pěchouček, M. (2008). AGENTFLY: A multi-agent airspace test-bed (demo paper). In *7th International joint conference on autonomous agents and multiagent systems, ACM, Estoril, Portugal* (pp. 1665–1666). http://www.ifaamas.org/Proceedings/aamas08/proceedings/pdf/demo/AAMAS08_demo5.pdf.
323. Skylogiannis, T., Antoniou, G., Bassiliades, N., Governatori, G., & Bikakis, A. (2007). DR-NEGOTIATE—A system for automated agent negotiation with defeasible logic-based strategies. *Data & Knowledge Engineering*, *63*(2), 362–380. https://doi.org/10.1016/j.datak.2007.03.004.
324. Slota, M., & Leite, J. (2008). EVOLP: An implementation. In Sadri and Satoh [306] (pp. 288–298). https://doi.org/10.1007/978-3-540-88833-8_16.
325. Snook, C., Butler, M., Edmunds, A., & Johnson, I. (2004). Rigorous development of reusable, domain-specific components, for complex applications. In J. Jurgens & R. France (Eds.), *3rd International workshop on critical systems development with UML, Lisbon, Portugal* (pp. 115–129).
326. Stathis, K., Kakas, A., Lu, W., Demetriou, N., Endriss, U., & Bracciali, A. (2004). PROSOCS: A platform for programming software agents in computational logic. In *17th European meeting on cybernetics and systems research, symposium "From agent theory to agent implementation" (AT2AI-4)* (Vol. II, pp. 523–528). Vienna: Austrian Society for Cybernetic Studies.
327. Suarez-Romero, J. A., Alonso-Betanzos, A., Guijarro-Berdinas, B., & Duran-Sanles, C. (2005). A tool for agent communication in Mozart/Oz. In *IEEE/WIC/ACM International conference on intelligent agent technology* (pp. 706–710). IEEE. https://doi.org/10.1109/IAT.2005.23.
328. Such, J. M., García-Fornes, A., Espinosa, A., & Bellver, J. (2013). Magentix2: A privacy-enhancing agent platform. *Engineering Applications of Artificial Intelligence*, *26*(1), 96–109. https://doi.org/10.1016/j.engappai.2012.06.009.
329. Sycara, K., Lu, J., Klusch, M., & Widoff, S. (1999). Matchmaking among heterogeneous agents on the Internet. In *AAAI spring symposium on intelligent agents in Cyberspace* (pp. 22–24).
330. Sycara, K. P. (1990). Persuasive argumentation in negotiation. *Theory and Decision*, *28*(3), 203–242. https://doi.org/10.1007/BF00162699.
331. Syrjänen, T., & Niemelä, I. (2001). The smodels system. In T. Eiter, W. Faber, & M. l. Truszczyński (Eds.), *International conference on logic programming and nonmonotonic reasoning, Lecture notes in computer science* (Vol. 2173, pp. 434–438). Berlin: Springer. https://doi.org/10.1007/3-540-45402-0_38.
332. Tarau, P. (1998). Jinni: A lightweight java-based logic engine for Internet programming. In K. Sagonas (Ed.), *International workshop on implementation technology for programming languages based on logic, Manchester, UK* (pp. 1–15).
333. Teresa, A., Ramón, B., Guitart, F., & Godo, L. (2013). Web based system for weighted defeasible argumentation. In Leite et al. [227] (pp. 155–171). https://doi.org/10.1007/978-3-642-40624-9_10.
334. The White House OSTP. (2020). *American artificial intelligence initiative: Year one annual report*. https://www.whitehouse.gov/wp-content/uploads/2020/02/American-AI-Initiative-One-Year-Annual-Report.pdf.
335. Thimm, M. (2010). Realizing argumentation in multi-agent systems using defeasible logic programming. In P. McBurney, I. Rahwan, S. Parsons, & N. Maudet (Eds.), *Argumentation in multi-agent systems (ArgMAS 2009), Lecture notes in computer science* (Vol. 6057, pp. 175–194). Berlin: Springer. https://doi.org/10.1007/978-3-642-12805-9_11.
336. Thomas, B., Shoham, Y., Schwartz, A., & Kraus, S. (1991). Preliminary thoughts on an agent description language. *International Journal of Intelligent Systems*, *6*(5), 497–508. https://doi.org/10.1002/int.4550060504.
337. Thomas, S. R. (1995). The PLACA agent programming language. In *Intelligent agents, Lecture notes in computer science* (Vol. 890, pp. 355–370). Berlin: Springer. https://doi.org/10.1007/3-540-58855-8_23.
338. Toni, F. (2005). Multi-agent systems in computational logic: Challenges and outcomes of the socs project. In Toni and Torroni [340] (pp. 420–426). https://doi.org/10.1007/11750734_26.
339. Toni, F. (2006). Multi-agent systems in computational logic: Challenges and outcomes of the SOCS project. In Toni and Torroni [340] (pp. 420–426). https://doi.org/10.1007/11750734_26.
340. Toni, F., & Torroni, P. (Eds.). (2005). *Computational logic in multi-agent systems, Lecture notes in computer science* (Vol. 3900). Berlin: Springer. https://doi.org/10.1007/11750734.

341. Torroni, P., Chesani, F., Mello, P., & Montali, M. (2009). Social commitments in time: Satisfied or compensated. In M. Baldoni, J. Bentahar, M. B. van Riemsdijk, & J. Lloyd (Eds.), *Declarative agent languages and technologies VII, Lecture notes in computer science* (Vol. 5948, pp. 228–243). Berlin: Springer. https://doi.org/10.1007/978-3-642-11355-0_14.
342. Vidaković, M., Ivanović, M., Stantić, D., & Vidaković, J. (2018). How research achievements can influence delivering of a course—Siebog agent middleware. In G. Jezic, Y. Chen-Burger, R. J. Howlett, L. C. Jain, L. Vlacic, & R. Šperka (Eds.), *Agents and multi-agent systems: Technologies and applications, smart innovation, systems and technologies* (pp. 110–120). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-92031-3_11.
343. Vieira, R., Moreira, Á., Wooldridge, M., & Bordini, R. H. (2007). On the formal semantics of speech-act based communication in an agent-oriented programming language. *Journal of Artificial Intelligence Research*, *29*(1), 221–267.
344. Viroli, M., Holvoet, T., Ricci, A., Schelfthout, K., & Zambonelli, F. (2007). Infrastructures for the environment of multiagent systems. *Autonomous Agents and Multi-Agent Systems, 14*(1), 49–60. https://doi.org/10.1007/s10458-006-9001-6. Special issue: Environment for multi-agent systems.
345. Vos, M. D., Crick, T., Padget, J., Brain, M., Cliffe, O., & Needham, J. (2005). LAIMA: A multi-agent platform using ordered choice logic programming. In Baldoni et al. [20] (pp. 72–88). https://doi.org/10.1007/11691792_5.
346. Vrba, P., Fuksa, M., & Klíma, M. (2014). JADE-JBossESB gateway: Integration of multi-agent system with enterprise service bus. In *International conference on systems, man, and cybernetics* (pp. 3663–3668). San Diego, CA: IEEE. https://doi.org/10.1109/SMC.2014.6974499.
347. Wallner, J. P., Weissenbacher, G., & Woltran, S. (2013). Advanced SAT techniques for abstract argumentation. In *Computational logic in multi-agent systems, Lecture notes in artificial intelligence* (Vol. 8143, pp. 138–154). Berlin: Springer. https://doi.org/10.1007/978-3-642-40624-9_9.
348. Walton, D., Reed, C., & Macagno, F. (2008). *Argumentation schemes*. Cambridge: Cambridge University Press. https://doi.org/10.1017/cbo9780511802034.
349. Wavish, P. (1992). Exploiting emergent behavior in multi-agent systems (abstract). *ACM SIGOIS Bulletin*, *13*(3), 16. https://doi.org/10.1145/152683.152701.
350. Weyns, D., Omicini, A., & Odell, J. J. (2006). Environment as a first class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems, 14*(1), 5–30. https://doi.org/10.1007/s10458-006-0012-0. Special issue on Environments for multi-agent systems
351. Weyns, D., Mascardi, V., & Ricci, A. (Eds.). (2019). *Engineering multi-agent systems, Lecture notes in computer science* (Vol. 11375). Stockholm: Springer. https://doi.org/10.1007/978-3-030-25693-7.
352. Wielemaker, J., Schrijvers, T., Triska, M., & Lager, T. (2012). SWI-Prolog. *Theory and Practice of Logic Programming*, *12*(1–2), 67–96. https://doi.org/10.1017/S1471068411000494.
353. Winikoff, M. (2009). JACK^{TM} intelligent agents: An industrial strength platform. In Håkansson et al. [167] (pp. 175–193). https://doi.org/10.1007/0-387-26350-0_7. 3rd KES International symposium, KES-AMSTA 2009, Uppsala, Sweden, June 3–5, 2009. Proceedings.
354. Winikoff, M., Liu, W., & Harland, J. (2005). Enhancing commitment machines. In Leite et al. [226] (pp. 198–220). https://doi.org/10.1007/11493402_12.
355. Wooldridge, M., & Ciancarini, P. (2001). Agent-oriented software engineering: The state of the art. In P. Ciancarini & M. J. Wooldridge (Eds.), *Agent-oriented software engineering, Lecture notes in computer science* (Vol. 1957, pp. 1–28). Berlin: Springer. https://doi.org/10.1007/3-540-44564-1_1.
356. Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, *10*(2), 115–152. https://doi.org/10.1017/S0269888900008122.
357. Wooldridge, M., Muller, J. P., & Tambe, M. (1996). Agent theories, architectures, and languages: A bibliography. In M. Wooldridge, J. P. Müller, & M. Tambe (Eds.), *Intelligent agents II agent theories, architectures, and languages, Lecture notes in computer science (Lecture notes in artificial intelligence)* (Vol. 1037, pp. 408–431). Berlin: Springer. https://doi.org/10.1007/3540608052_81.
358. Wooldridge, M., Fisher, M., Huget, M. P., & Parsons, S. (2002). Model checking multi-agent systems with MABLE. In *1st International joint conference on autonomous agents and multiagent systems (AAMAS 2002)* (pp. 952–959). Bologna: ACM. https://doi.org/10.1145/544862.544965.
359. Yen, J., Yin, J., Ioerger, T. R., Miller, M. S., Xu, D., & Volz, R. A. (2001). CAST: Collaborative agents for simulating teamwork. In B. Nebel (Ed.), *7th International joint conference on artificial intelligence* (pp. 1135–1144). Seattle, WA: Morgan Kaufmann. http://dl.acm.org/doi/10.5555/1642194.1642247.
360. Zatelli, M. R., Ricci, A., & Hübner, J. F. (2015). Evaluating different concurrency configurations for executing multi-agent systems. In M. Baldoni, L. Baresi, & M. Dastani (Eds.), *Engineering multi-agent systems, Lecture notes in computer science* (Vol. 9318, pp. 212–230). Berlin: Springer. https://doi.org/10.1007/978-3-319-26184-3_12.