Autonomous Agents and Multi-Agent Systems (2020) 34:37
https://doi.org/10.1007/s10458-020-09453-y

VIEWPOINT

Agent programming in the cognitive era

Rafael H. Bordini[1] · Amal El Fallah Seghrouchni[2] · Koen Hindriks[3] · Brian Logan[4] · Alessandro Ricci[5]

Published online: 12 May 2020

Abstract
It is claimed that, in the nascent 'Cognitive Era', intelligent systems will be trained using machine learning techniques rather than programmed by software developers. A contrary point of view argues that machine learning has limitations, and, taken in isolation, cannot form the basis of autonomous systems capable of intelligent behaviour in complex environments. In this paper, we explore the contributions that agent-oriented programming can make to the development of future intelligent systems. We briefly review the state of the art in agent programming, focussing particularly on BDI-based agent programming languages, and discuss previous work on integrating AI techniques (including machine learning) in agent-oriented programming. We argue that the unique strengths of BDI agent languages provide an ideal framework for integrating the wide range of AI capabilities necessary for progress towards the next-generation of intelligent systems. We identify a range of possible approaches to integrating AI into a BDI agent architecture. Some of these approaches, e.g., 'AI as a service', exploit immediate synergies between rapidly maturing AI techniques and agent programming, while others, e.g., 'AI embedded into agents' raise more fundamental research questions, and we sketch a programme of research directed towards identifying the most appropriate ways of integrating AI capabilities into agent programs.

Keywords Agent programming languages · Belief-desire-intention · Artificial intelligence · Machine learning

1 Introduction

In the quest to build autonomous intelligent systems[1] able to form hypotheses, make decisions, learn from data, and adapt their behaviour to changes in their environment, several large-scale organisations have begun to align around a concept known as the "Cognitive Era". Drawing on recent results in the AI sub-field of machine learning (ML), some authors, e.g., Kelly and Hamm [83], have argued that, in the era of cognitive computing, intelligent systems will be trained using ML techniques, rather than developed by human

[1] *Autonomous systems* here refers to systems that can effectively operate without human intervention.

✉ Brian Logan
brian.logan@nottingham.ac.uk

Extended author information available on the last page of the article

programmers. However, others, e.g., Marcus [99], argue that ML has limitations, and cannot form the sole basis of autonomous systems capable of intelligent behaviour in complex environments. Indeed they maintain that creating next-generation intelligent systems that sense, learn, reason, and interact with people in new ways will require pushing the boundaries of science and technology to integrate a wide range of AI capabilities.

In this viewpoint paper we explore the contributions that Agent-Oriented Programming (AOP), and particularly BDI-based approaches to AOP, can make to the development of future autonomous intelligent systems.[2] We argue that BDI-based agent programming languages offer significant advantages for developing autonomous systems, including more rapid development, context sensitive, robust behaviour, and, critically, greater intelligibility and verifiability compared to agents where programs are learned. Moreover, a BDI-based approach also facilitates the integration of a wide range of symbolic, stochastic and subsymbolic AI techniques. We survey previous work on integrating AI techniques into BDI-based agent programming languages, and sketch a roadmap of key research opportunities and challenges for future work in this area. We argue that the integration of AI techniques can raise the level of abstraction of agent programming, by increasing the basic competence of agent languages and platforms. As the competence of the agent increases, the role of the developer will change from programming exactly what the agent will do in all situations, to providing more strategic information, heuristics, advice, social knowledge (e.g., norms), etc. about what the agent should (or should not) do in general, leaving the details of the implementation of the strategy to the agent. This style of development has some similarities with the 'training' of systems advocated in the cognitive approach, and one can envision hybrid architectures in which some behaviours are learnt, while others, perhaps those with a supervisory or high-level decision making role, are 'programmed'. We believe future BDI-based agent programming approaches offer a fruitful framework for such controlled adaptation. Although we focus here on the unique strengths of the BDI approach, we believe our proposals are applicable, at least in part, to other approaches to AOP (discussed below in Sect. 2), particularly those languages in the broader cognitive agent tradition, e.g., GOAL [77], SOAR [87], that have similar logic-based or declarative "roots".

As such, the paper can be seen as presenting a vision for the future of Agent-Oriented Programming, particularly from the point of view of researchers with interests in Engineering Multi-Agent Systems (EMAS) in the Belief-Desire-Intention tradition. We hope it will be of interest to researchers and developers in both the broader agent-oriented programming community, and in the AI and Machine Learning communities interested in developing intelligent autonomous systems that exhibit flexible intelligent behaviour in dynamic and unpredictable environments. We wish to stress, however, that the paper is not intended to give a comprehensive survey of agent-oriented programming, nor does it enumerate all of the many important links between the work carried out by the EMAS community and other subareas of AAMAS. Rather we hope to contribute to the 20th anniversary special issue of JAAMAS by focusing on the work of the EMAS community and its future, which together with other papers may provide an interesting snapshot and roadmap of current and future research in Autonomous Agents and Multi-Agent Systems.

The remainder of the paper is structured as follows. Section 2 gives a brief historical overview of research on Agent-Oriented Programming. Section 3 focuses specifically on agent programming based on the belief-desire-intention (BDI) model, including the main components of a BDI agent and the BDI cycle. We briefly survey the historical

[2] The term 'Agent Oriented Programming' was coined by Shoham [129] who also proposed one of the first AOP languages, AGENT0 [130].

development of BDI-based programming languages, focusing on some of the best-known platforms in wide current use, and discuss the key contributions and limitations of the current BDI model. Section 4 reviews previous work on integrating AI techniques into each of the sense, plan and act phases of the BDI cycle. Section 5 discusses open research problems and possible future research directions. In particular, it discusses two ways AI techniques could be integrated into a BDI agent architecture, AI as a service and AI embedded into agents, outlines the challenges of engineering a BDI-based AI integration framework, and presents a roadmap of opportunities and open research challenges in this area. In Sect. 6, we conclude by arguing that although the BDI model will evolve, its integrating capability coupled with its cognitive foundations (the concepts of beliefs, goals, and intentions in the form of commitments to future behaviour) make it a good fit for designing stand-alone intelligent systems and for their broad acceptance.

2 Agent oriented programming

There has been considerable work on software systems exhibiting autonomy, rational behaviour, and social interaction that are capable of operation in dynamic and unpredictable environments. In the Autonomous Agents and Multi-Agent Systems community, the development of such systems typically makes use of a set of interrelated concepts including beliefs, actions, plans, intentions, norms, roles and groups, and the overall software development process including modelling, programming, testing, etc. based on such abstractions has been termed Agent-Oriented Software Engineering [153]. In an agent-oriented approach, agent development typically employs special purpose *agent programming languages* that provide programming abstractions that directly support concepts such as beliefs, goals, plans, intentions etc and *agent programming platforms* that provide features to support (multi-)agent system development, including, communication, distribution, IDEs etc. Agent programming platforms typically provide support for a particular agent programming language, but some, such as JADE[3] [11], SARL[4] [123] and JIAC[5] [97], target applications developed in 'mainstream' programming languages such as Java.

The predominant approach in agent programming is inspired by the Belief-Desire-Intention (BDI) model [23].[6] One of the earliest implementations of the BDI model, and perhaps the best known is the Procedural Reasoning System (PRS) which was based on reactive planning systems [64, 65]. Other prominent BDI-based languages of this period include AgentSpeak(L) [117] which was directly influenced by the theoretical work on BDI logics [120, 157], and 3APL [75], which was based on logic programming. However, the BDI approach is only one of a range of approaches to Agent-Oriented Programming (AOP) that were developed starting in the early 1990s. For example, MetateM [60] is based on linear temporal logic, Teleo-reactive programs [106] are based on guarded action rules, while Golog [92] and its successors, e.g., DTGolog [20] and IndiGolog [44] are based on

[3] https://jade.tilab.com.
[4] http://www.sarl.io.
[5] http://www.jiac.de.
[6] We discuss BDI-based languages in detail in the next section.

situation calculus. There are also languages based on functional programming [34], and process algebra such as CLAIM [59].

Building on the early work on PRS, and abstract languages such as AgentSpeak(L) and 3APL in the 1980s and 1990s, the 2000s saw a significant expansion of research in agent-oriented programming languages [16, 17, 19, 82, 101], particularly following a Dagstuhl Seminar [57] that helped set the research agenda for the area. In the last decade, there has been a marked move in research in the area from single to multi-agent aspects of the agent programming platforms including concepts such as roles, groups, and norms. Although the focus of this paper is mostly on the development of individual agents and the social level of a multi-agent system is thus out of scope, these are important abstractions for fully-fledged multi-agent oriented programming. Some notable work on organisational and normative aspects of multi-agent programming languages include OperA [3], Moise [79], 2OPL [140], JaCaMo [13] and SARL [123]. More recently, the focus has been less on new languages and more on important features that are required of mature programming languages, such as modularity, encapsulation, testing, debugging, as well as work on making the development frameworks robust enough to handle real-world programming.

The aim of this paper is not to provide a broad survey of work in AOP, and we refer the reader to the literature above for a detailed discussion of the wide range of agent-oriented languages that have been developed. In the next section we focus specifically on some of the current BDI programming languages and platforms, which form the foundation of our exploration of how AI techniques may be integrated into agent programming. As has been argued extensively in the literature and we elaborate below, such agent programming languages and platforms greatly facilitate the development of autonomous intelligent software systems whose decision making is explainable to end users.

3 Agent programming based on the BDI model

BDI models of agency approach autonomous behaviour through two related theories of the philosophical concept of intentionality: (i) the notion of an *Intentional System* as an entity with beliefs, desires and other propositional attitudes due to Dennett [49]; and (ii) the theory of *Practical Reasoning* proposed by Bratman [22] which is based on beliefs, desires and intentions in the form of partial plans. These two related notions of intentionality provide us with the tools to describe agents at an appropriate level of abstraction, i.e., in terms of belief, desires, and intentions (BDI), by adopting the *intentional stance*, and to design agents compatible with such intentional descriptions, i.e., as *practical reasoning systems*.

Many of the practical approaches to the development of multi-agent systems based on the BDI model are founded on the theoretical underpinnings of the BDI model [119] and inspired by ideas from IRMA [23] and PRS mentioned above. Such practical approaches to cognitive agents are arguably one of the main contribution of the agent programming community to the broader field of AI. The BDI approach can be seen as an attempt to characterise how flexible intelligent behaviour can be realised in dynamic and unpredictable environments, by specifying how an agent can balance reactive and proactive behaviour.

In this section, we briefly outline the key components of a BDI agent and the steps in the BDI cycle. We then briefly survey the historical development of BDI-based programming languages, focusing on some of the best-known platforms in current use. Finally, we conclude this section by highlighting the key contributions of BDI-based agent programming to the wider field of AI, and some of its limitations.

3.1 The components of a BDI agent

In BDI-based agent programming languages, the behaviour of an agent is specified in terms of beliefs, goals, intentions, and plans; see Fig. 1. *Beliefs* represent the agent's information about the environment (and itself). *Goals* represent either a desired course of action (procedural goals), a desired state of the environment the agent is trying to bring about (achievement goals), or a desired state of the environment the agent is trying to maintain (maintenance goals). The representation of beliefs and goals may include additional information, e.g., about the source of a belief or its certainty, the importance of the goal or the deadline by which it must be achieved, etc. *Plans* are the means by which the agent responds to events. In agent programming, plans are typically predefined by the agent developer and consist of a head and a body. The head consists of a trigger (an event template) and a context condition, which together specify the situations in which the plan should be considered as a means of achieving a goal, or responding to a change in the agent's beliefs. The body of a plan is composed of basic actions that directly change the agent's environment or its internal state, tests that check if a condition is true, sub-goals which are in turn achieved by other plans, and other operators for composing more basic plan bodies. Plans together with the agent's initial beliefs and goals, form the program of the agent.

At run-time, the *interpreter* updates the agent's beliefs and goals in response to messages and sensory information from the agent's environment (percepts), and manages the agent's intentions. An *intention* is a future course of action the agent is committed to carrying out. In practice, an intention is often implemented as a stack of partially instantiated plans, the execution of which are expected to achieve a (top-level) goal or respond to the change in the agent's beliefs. The interpreter is also responsible for choosing which intention to execute and for executing steps in the plan forming the top of the intention. The overall architecture of a BDI agent is shown in Fig. 1.

3.2 The BDI cycle

The interpreter repeatedly executes a '*sense-plan-act*' cycle [86] (sometimes called a *deliberation cycle* [40] or *agent reasoning cycle* [18]). The details of such cycles vary from language to language, but in all cases it includes the processing of events (*sense*), deciding how to update the agent's intentions in response to those events (*plan*), and selecting and executing a step in one or more intentions (*act*). In BDI-based agent programming

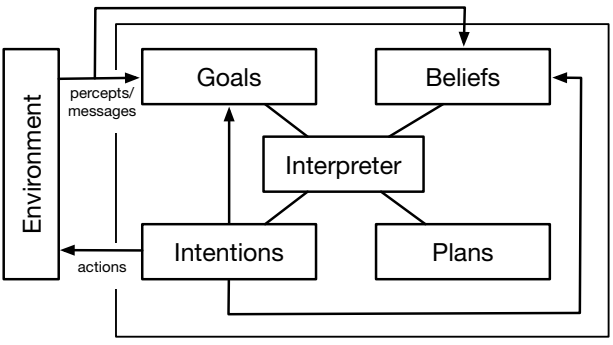

Fig. 1 The components of a BDI agent

languages, the terms *sense*, *plan* and *act* are used as shorthand for a variety of different kinds of processing, and each phase may have substeps as we outline briefly below. In Sect. 3.3, we describe how these phases are realised in some well-known BDI-based agent programming languages and platforms.

Sense phase In the sense phase, the agent's beliefs and goals are updated with messages from other agents (including the user or users), information from the agent's sensors, and updates from the execution of the agent's intentions at previous cycles. For example, a message from a user may request the agent to drop an existing goal, sensory information may give rise to a change in the agent's beliefs about the current state of its environment, and the execution of steps in a plan in an intention may result in the posting of a subgoal.[7] In most BDI-based languages, the agent's beliefs and goals are maintained using some form of declarative knowledge representation. Depending on the language, updating and querying this representation may involve sophisticated inference. For example, the incorporation of a new belief may involve some form of belief revision which takes into account the source of the new belief or its (un)certainty, while querying the agent's beliefs may involve SLD resolution as in Prolog, or reasoning over ontologies, constraints, etc.

Plan phase The sense phase gives rise to a set of *belief and goal change events*. The role of the plan phase is to determine how the agent should update its intentions to respond to these events. Belief and goal change events can give rise to changes in the agent's set of intentions, or to changes to individual intentions in the set. Changes in the set of intentions include: the creation of a new intention, e.g., when the agent adopts a new top-level goal or must respond to a change in its beliefs; the deletion of an intention, e.g., when the corresponding top-level goal is dropped or becomes unachievable; and the suspension (resp. resumption) of an intention, e.g., when the corresponding top-level goal becomes temporarily unachievable (resp. becomes achievable again).[8] Updates to existing intentions are primarily of two kinds: the extension of an intention that posts a sub-goal with a plan to achieve the goal; and the modification of an intention when the plan selected to achieve the current sub-goal fails, which typically involves replacing the current plan for the sub-goal (or a parent goal of the sub-goal) with a new plan. As with belief and goal updates, updating the set of intentions may involve sophisticated reasoning or deliberation. For example, in many BDI-based agent programming languages, extending an intention with a plan to achieve a sub-goal involves several steps. First the set of relevant plans is determined. A plan is *relevant* if its triggering condition matches a goal to be achieved or a change in the agent's beliefs the agent should respond to. Second, the set of applicable plans are determined. A plan is *applicable* if its context condition evaluates to true, given the agent's current beliefs. Third, a relevant, applicable plan is selected and pushed onto the intention stack that generated the triggering event. Updating an intention when the plan selected to achieve the currently active subgoal fails may be even more complex, potentially involving the execution of 'cleanup' actions (e.g., to release resources) before determining the appropriate higher-level motivating (sub)goal in the intention to backtrack to, and finally the selection of a new plan for that (sub)goal [71].

Act phase The result of the plan phase is a set of updated, potentially executable intentions. The role of the act phase is to select one or more intentions for execution and execute one (or more) steps of the plan forming the top of each selected intention. As in the

[7] Depending on the APL, the sense phase may involve other types of events, e.g., events signalling the failure of external actions. In the interests of simplicity, we treat such events as a special kind of belief update.

[8] The suspension and resumption of an intention can also be seen as updating the state of the intention, rather than temporarily moving the intention out of or into the set of intentions.

preceding phases, the selection of which intention(s) to execute may involve sophisticated reasoning, e.g., to avoid unwanted interactions between the effects of actions in different intentions, or to determine when an intention must be executed in order to achieve a goal by its deadline. Execution of steps in an intention may change the agent's beliefs and goals, either directly, e.g., by adding or deleting beliefs or (sub)goals, or indirectly, e.g., by changing the agent's environment leading to changes the agent's beliefs and goals in the next sense phase.

The cycle then repeats, allowing the agent to sense, and respond to, the effects of its actions and other changes in its environment.

3.3 Historical development of BDI-based agent programming languages

In this section we briefly survey the historical development of some of the best-known BDI-based programming languages and platforms.

As noted in Sect. 2, arguably the first implementation of the BDI model was PRS [64, 65]. PRS incorporates many of the ideas found in later BDI-based languages, including beliefs, goals and, critically, intentions. PRS is a relatively complex language, incorporating temporally extended goals, graph-based representations of plans, and meta-level programming constructs which allowed the agent's deliberation to be customised to a particular application. Perhaps because of this, it lacked a formal semantics until recently, when de Silva et al. [46] provided an operational semantics for a significant fragment of PRS.

Together with subsequent developments in the theoretical underpinnings of BDI-based agent programming [36, 118, 119], PRS and its successor dMARS [55, 56] led to the development of other, more abstract (and therefore simpler to reason about) agent programming languages. Perhaps the best known of these languages is AgentSpeak(L) [117]. AgentSpeak(L) omitted many features of PRS, including maintenance goals, graph-based plans and a meta-level. However, it added several new features, including Prolog-style inference over the agent's beliefs (an idea that has persisted in many subsequent BDI-based languages). AgentSpeak(L) was essentially an abstract programming language with a precise operational semantics [14, 117], though an implementation in Prolog was later developed by Winikoff [151]. Although it was an abstract language (initially) lacking an implementation, AgentSpeak(L) was influential in the design of many other languages and platforms, notably Jason [18] and ASTRA [37], as well as further work on abstract BDI programming, in particular CAN [155].

Another influential agent programming language proposed around the same time as AgentSpeak(L) was 3APL [78]. Similarly to AgentSpeak(L), 3APL is a rule-based agent programming language that builds on the notions of beliefs, goals, and plans. It combined ideas from both imperative and logic programming, inheriting the full range of regular programming constructs from imperative programming and the proof as computation model for querying the belief base of an agent from logic programming. The original version of the 3APL language included some of the advanced features of PRS, such as rules for the runtime modification of the agent's intentions, but these were dropped in later versions of the language. Later work building on 3APL led to the language 2APL [43].

Many other agent programming languages and platforms have been developed that at least partially implement the BDI model, e.g., JACK [27], Jadex [24], and GOAL [77]. As with the languages mentioned above, they encompass each of the components of the BDI model in some form, and often have a solid theoretical foundation in the form of a precise

operational semantics specifying what beliefs, desires and intentions mean, and how they should be implemented.

3.4 State of the art in BDI-based agent programming languages

In this section, we briefly survey some of the best-known BDI-based languages that are currently widely-used in academia and research.[9] In particular, we include only languages that are influenced by the BDI tradition and that have been under active development for at least the last five years. As we stated above, our aim is not to provide an exhaustive survey of AOP or BDI agent programming languages. Rather, the purpose of the survey is to illustrate how widely-used BDI platforms have interpreted the BDI model, in order to ground our critique of current BDI platforms in the next section. At the same time we wish to highlight that there are mature, well-established platforms that can serve as a basis for the integration of AI techniques as proposed in Sect. 5.

Perhaps surprisingly, the first language to mention is PRS.[10] Despite, or perhaps because of, its maturity, it is still actively used [67], particularly in robotics, e.g., [2, 62, 81, 90, 105]. For example, PRS-based systems secured first and second places in recent RoboCup and ICAPS logistics competitions.

Another widely-used language is Jason[11] [18]. Jason is a multi-agent system development platform based on an interpreter for an extended version of AgentSpeak (for the formal semantics of those extensions see [141]), aimed at producing a language that is useful in practical programming. However, not all practical features of Jason have formal semantics. One example of such a feature is the atomic execution of plans to avoid race conditions, which was later formalised and implemented in 2APL[12] [43]. Other recent extensions of Jason that have not been formalised but have proved useful in practical programming include meta events, modules and namespaces, concurrent agent architectures, concurrency operators in plans, and plan conflict specifications. Much of the recent development effort in Jason has been driven by its use as part of JaCaMo[13] [13], a fully-fledged multi-agent programming platform that uses Jason for programming the agent dimension, CArtAgO[14] [121] for the environment dimension, and Moise[15] [79] for the organisation dimension.

Another platform based on AgentSpeak is ASTRA[16] [37]. It builds on the AgentFactory platform[17] [124], which has been developed for many years and used in a number of applications before the move to an AgentSpeak-based programming language. ASTRA was influenced by AgentSpeak(TR), a language introduced in Clark and Robinson [33] combining AgentSpeak and teleo-reactive rules. Teleo-reactive programming was introduced by Nilsson [106], as a formalism for computing and organising the actions of autonomous agents in dynamic environments. A teleo-reactive program is a sequence of guarded action rules that directs an agent towards a goal (hence *teleo*) and takes into account the dynamics

[9] For a survey of the use of BDI-based languages in industry, see, e.g., [54, 104].
[10] https://git.openrobots.org/projects/openprs.
[11] http://jason.sourceforge.net.
[12] http://apapl.sourceforge.net.
[13] http://jacamo.sourceforge.net/wp/.
[14] http://cartago.sourceforge.net.
[15] http://moise.sourceforge.net.
[16] http://www.astralanguage.com/wordpress/.
[17] https://sourceforge.net/projects/agentfactory/.

of a changing environment (hence *reactive*). The goals of teleo-reactive agent programs are implicitly encoded by the order of the rules in the program. The guards of the action rules query the belief base of the agent. Nilsson [107] proposed a "triple-tower" architecture which adds: (1) a module (or 'tower') for processing percepts; (2) a truth-maintenance system for maintaining a model of the environment; and (3) a decision-making tower that uses teleo-reactive programs for selecting the actions to perform next. Building on this work, Clark and Robinson [33] introduced TeleoR, an extension of Nilsson's teleo-reactive agent programming language. TeleoR extends the teleo-reactive language with various features including extra forms of action rules that temporarily inhibit other rules, wait/repeat restart of failed actions, and message-sending actions.

GOAL[18] [76] is a language for programming *cognitive agents*. Its cognitive model includes concepts such as beliefs and (declarative) goals but deviates from the classic BDI model in that it does not provide explicit support for intentions. It could be argued that GOAL modules can be seen as a form of intention, as done in Hindriks [72], but their semantics is different from the classic intention-as-stacks-of-plans model. GOAL is a rule-based language and its rules are similar to plans in the BDI model. The main motivation for not including intentions as first-class citizens in the language was to simplify the reasoning cycle (compared to the BDI cycle, cf. Sect. 3.2). The basic cycle of a GOAL agent still follows a sense-plan-act scheme but both the planning and the acting step in this cycle are much simpler as planning simply means applying a rule to select a (possibly composed) action and acting means (completely) performing the selected action. This simplification avoids the many subtle choices that are available in intention-based architectures.

Before concluding this section, it is important to mention that there are other widely-used BDI platforms with a less formal basis, such as Jadex[19] [116] and JACK[20] [27, 150], that rely on extending Java rather than defining a new programming language as such. In addition to being one of the most popular BDI platforms, Jadex was among the first BDI platforms to provide an implementation of various types of goals including achieve, query, maintain and perform, with a well-defined complex goal lifecycle [25]. JACK on the other hand is the platform that has been most widely used in commercial applications, and the only platform described here that is not open source.

3.5 Key contributions and limitations of the current BDI model

The BDI approach has been very successful, and arguably is the dominant paradigm in agent programming languages [94]. In this section, we briefly review some of the key contributions of the BDI-based agent programming languages to agent development and to AI more generally. We also highlight some of the limitations of current BDI languages and platforms.

From a programming point of view, BDI-based languages offer a number of advantages compared to developing agents in 'mainstream' programming languages such as C++ or Java. End users find programs couched in terms of mentalistic terms such as 'beliefs' and 'goals' easier to understand [109], and the use of predefined plans encoding standard responses to situations gives predictable behaviour that can be more easily validated by end-users and other stakeholders. For example, Wei and Hindriks [149] argue that an agent

[18] https://goalapl.atlassian.net/wiki/spaces/GOAL/.
[19] https://www.activecomponents.org.
[20] http://aosgrp.com/products/jack/.

programming approach provides a clear separation of concerns relating to symbolic reasoning (symbolic knowledge representation, modular, high-level action selection, and support for multiple declarative goals) and sub-symbolic processing in the control of robots. In addition, the high-level of BDI-based APLs allows more rapid development with fewer programmer errors compared to mainstream languages [12].

In terms of execution, BDI-based languages result in behaviour that is both sensitive to the current context and robust in the face of failures. The BDI cycle, in which the selection of plans is deferred until the corresponding (sub)goal must be achieved, allows an agent to respond flexibly to changes in the environment, by adapting the means used to achieve a goal to the current situation. Such context sensitivity is more awkward to code in mainstream languages and the resulting programs are more difficult to maintain. This interleaving of plan selection and action execution also results in behaviour that is robust to changes in the agent's environment. In contrast to classical AI planning [66], which assumes that the set of (achievement) goals and the environment are static, the paradigm of BDI agents is one of interleaved plan selection and execution in a dynamic environment. This dynamism arises because of exogenous 'natural' events (such as weather) and the actions of other agents, and also because of limitations in the agent's perception, its actions (e.g., action failure), or both. When a plan fails or becomes unexecutable, a BDI agent is often able to recover gracefully, by backtracking to the most recently adopted (sub)goal that has a relevant, applicable plan [155]. Taken together, the declarative style of BDI programs combined with the BDI cycle give a significantly expanded behaviour space [156] for lower programmer effort compared to development in mainstream languages.

Perhaps the key contribution of the BDI-based approach, at least compared to agents where programs are learned, is the intelligibility of the resulting agent behaviour by end users and other stakeholders, see, e.g., [26, 88, 108, 128]. This has been termed Explainable AI (XAI) [69], i.e., the development of autonomous systems capable of explaining their decisions and actions to human users. As noted by Bratman [22], when attempting to achieve a goal, humans commit to a particular course of action and use this commitment to filter their adoption of further goals and the plans selected to achieve them. Moreover, they typically only reconsider their commitments when a significant change occurs in either their goals or the environment. Bratman terms this commitment to a course of action an *intention*, and the programmatic expression of intentions in the BDI-based approach is critical to the intelligibility of the behaviour of BDI agents. Intentions make it easier for end users to understand why an agent is doing what it's doing, and to predict what it is likely to do next. Bratman argued that this is because intentions were developed to support human interaction, and the interpretation of behaviour in terms of intentions (sometimes called "folk psychology") therefore comes naturally. In addition, the structuring of the agent program around beliefs, goals and plans, and the resulting intention-driven behaviour, makes it relatively straightforward to answer "why questions", e.g., which goal(s) the agent is working towards, and why a particular course of action has been chosen rather than some other way of achieving a goal [70, 73, 152, 154]. The relative transparency of BDI agent programs also facilitates Ethics by Design [53], by making it easier for autonomous agents to reason about the ethical aspects of their decisions.

The very high level of abstraction (compared to conventional programming) also makes it easier to formally verify complex decision making, making it possible to give guarantees that an agent will behave correctly in all circumstances [61]. This is key to the engineering of trustworthy AI systems [115], and has been extensively explored in the agent-oriented programming community, e.g., [50, 51]. In contrast, action selection based on learned policies is often opaque, and formal guarantees of correctness are unavailable, see, e.g.,

[99]. In application domains where agents interact with humans, and/or where the agent's actions are safety critical (e.g., disaster response, medical care, etc.), this lack of transparency and predictability is likely to be a significant impediment to the adoption of agents whose decision making is based on learning.

Current implementations of the BDI model, however, also suffer from a number of limitations. The features common to state-of-the-art BDI-based agent languages, and that currently define this style of programming, are essentially limited to:

- selecting pre-defined plans at run time based on the triggering event and the agent's current beliefs; and
- some support for handling plan failure, e.g., aborting the plan in which the failure occurred and trying another applicable plan to achieve the current subgoal.

While these features are useful, and are key to implementing agents based on the BDI paradigm, much of the sophisticated reasoning necessary to use these features intelligently (e.g., which plan to select in the current context, which alternative applicable plan to try on plan failure) must be implemented by the developer for each application. Indeed, significant programmer effort is often required to adapt the default BDI cycle (e.g., with-domain specific plan/intention selection functions) for advanced applications, and to ensure key non-functional requirements (e.g., the timeliness of the agent's response). Moreover, there are many things for which state-of-the-art BDI languages and platforms provide no support at all, e.g., generating new (not developer-provided) plans at runtime. In the next section we consider how AI can contribute both to the sophisticated reasoning necessary to use BDI features intelligently and to extending the core capabilities of BDI platforms, and briefly review some of the previous research that has attempted to address these challenges.

4 AI in BDI agent programming

At each cycle, an agent programmed in a BDI-based agent programming language must make decisions about 'what to do next': what means (i.e., plan) to achieve a given (sub) goal; which of the currently adopted plan(s) (i.e., intentions) to progress at the current moment; and how these plan(s) should be progressed. Many of the issues involved in making these decisions can be addressed using AI techniques such as machine learning, and there has been some work exploring how such techniques can be integrated into BDI-based agent programming languages and the BDI architecture and cycle generally. In this section, we briefly survey some of this work. The presentation broadly follows the BDI cycle described in Sect. 3.2; that is, we first discuss how AI techniques have been applied in the sense phase, before considering the plan and act phases. However the correspondence is not exact, as some AI techniques imply changes to more than one phase; for example, the incorporation of richer knowledge representation frameworks such as description logic [6] has implications both for how beliefs are represented and updated in the sense phase, and how queries used to determine the applicability of plans are evaluated in the plan phase.

4.1 AI in the sense phase

Work on integrating AI techniques into the sense phase broadly falls into two main areas: enriching the underlying declarative knowledge representation, e.g., to handle

complex ontologies or uncertain information; and managing the update of the agent's beliefs and goals.

Enriching the knowledge representation There have been several strands of work that explore how the underlying knowledge representation technology used by a BDI-based APL can be extended.

One strand focusses on ontologies. For example, Moreira et al. [103] formally define a variant of AgentSpeak(L) based on description logic rather than Prolog. They argue that description logic allows complex belief base queries to be expressed more concisely, consistency checking of belief updates, more flexible retrieval of plans based on the subsumption relation between concepts, and the sharing of knowledge expressed in ontology languages such as OWL. This work was later extended by Mascardi et al. [100]. A similar approach was taken by Klapiscak and Bordini [85]. They introduce an extension of the Jason agent platform called JASDL that makes use of an existing OWL-API to provide features such as plan trigger generalisation based on ontological knowledge and the use of such knowledge in querying the agent's belief base.

Another strand of work seeks to avoid a commitment to any particular knowledge representation technology, and instead aims to make the underlying KR technology a "pluggable" component. For example, Dastani et al. [42] present two different approaches to integrating and combining multiple knowledge representation techniques into a BDI-based agent programming language: a meaning-preserving translation approach that maps one representation to another, and an approach based on 'bridge rules' that increases the inferential power of an agent system with multiple knowledge representation technologies. An alternative approach to the integration of KR languages based on a generic KR interface is proposed by Bagosi et al. [7]. They illustrate how the interface can be used to integrate both Prolog and OWL with rules.

Finally, Silva and Gluz [131] consider the problem of how to integrate probabilistic knowledge, beliefs and goals into a BDI-based agent programming language. They present a variant of AgentSpeak(L), AgentSpeak(PL), that supports probabilistic beliefs through the use of Bayesian Networks.

Belief and goal update The other main area of work relating to the sense phase focusses on how the agent's underlying knowledge representation is updated.

One strand of work focusses on how an agent's beliefs are updated. For example, Alechina et al. [5] have investigated how AI belief revision techniques can be used to ensure consistency of a BDI agent's belief base, and Alechina et al. [4] present an approach to caching the results of context queries to improve the run-time performance of BDI-based agents. Another strand of work investigates how argumentation theory can be used to reason about contradictory information, e.g., in multi-agent interactions. For example, Panisson et al. [113] present techniques to allow Jason/JaCaMo agents to choose between conflicting conclusions, or to choose the most promising arguments in a dialogue by considering information sources of varying degrees of trustworthiness.

Yet another strand of work considers the problems inherent in processing large volumes of (often sub-symbolic) sensor data in applications such as in gaming and robotics. The problem of detecting discrepancies between the agent's environment and the agent's representation of it has been recognised since the early days of agent programming, e.g., [41, 45, 91, 93]. More recently, Cranefield and Ranathunga [39] have argued that percept processing may give rise to a bottleneck in the agent's reasoning cycle. They propose an approach based on policies for handling high-frequency belief updates in BDI-based agent languages, and illustrate their approach using an Apache Camel-based implementation for handling updates in Jason. Ziafati et al. [163] have proposed several extensions

to BDI-based agent programming languages to allow the connection of agent programs to robots, including subscribing to events of interest at run-time, asynchronous reception of events, maintaining histories of events and run-time querying of the histories. Pantoja et al. [114] adopt a similar approach in the ARGO architecture for programming embedded robotic agents using Jason. Their approach builds on work by Stabile and Sichman [135], and uses perception filters to ensure the real-time processing necessary for robot control, e.g., to prevent collisions effectively. Finally, Wei and Hindriks [149] discuss the integration of the OpenCV library [21] for computer vision and processing of raw camera images into the GOAL architecture, and how the resulting percepts can be used in agent programs.

There has been somewhat less work on updating goals.[21] Sardiña and Padgham [126] present an extension to the CAN agent programming language [155] named CANPLAN2, which incorporates mechanisms for the proactive adoption of new goals, ensuring that goals that are deemed impossible are immediately dropped, and preventing an agent from blindly persisting with a blocked subgoal when alternative plans for achieving a higher-level motivating goal exist. Another strand of work has focussed on analysis of the potential interactions between an agent's plans to detect possible conflicts between top-level goals, and hence which subsets of the agent's goals can be pursued at the same time. For example, Thangarajah et al. [138] and Thangarajah and Padgham [139] have proposed an approach based on *summary information* which involves reasoning about necessary and possible pre- and post-conditions of different ways of achieving a goal. They present mechanisms to determine whether a newly adopted goal will definitely be safe to execute without conflicts, or will definitely result in conflicts (i.e., there is no way of achieving a goal that does not make the (concurrent) achievement of another goal impossible), or may result in conflicts. If the goal cannot be executed safely, the goal may be suspended [71, 136].

4.2 AI in the plan phase

Work on integrating AI techniques into the plan phase can also be broken down into two main areas. Work in the first area seeks to improve the selection of the agent's (developer provided) plans, e.g., to ensure that the plans most likely to succeed in the current context are selected, or to favour plans with lower cost or higher utility. In contrast, work in the second area focusses on the generation of new plans at run time, e.g., to provide a plan for a situation that was not anticipated by the agent developer.

Plan selection Plan selection in many BDI agent programming languages is heavily reliant on software developers, both to provide appropriate context conditions for plans, and to specify which applicable plan is selected, e.g., through the ordering of plans or by customising the selection of applicable plans for a particular application. It is therefore not surprising that there are several strands of work that explore how AI techniques can be applied to make plan selection in BDI-based APLs more intelligent.

One approach involves the use of machine learning techniques to guide the choice between relevant applicable plans by exploiting the contextual information previously encountered when executing a plan. For example, Singh et al. [133, 134] show how the belief contexts that resulted in the success and failure of a plan can be used to evaluate the relevance of executing the plan in the current context. In a similar way, Singh and Hindriks

[21] It should be noted that there has been more work in this area in the cognitive architectures community, see, e.g., [1].

[132] use reinforcement learning to optimise action selection in rule-based agent programming languages, and Guerra-Hernández et al. [68] use induction of logical decision trees to learn when plans are successfully executable.

Another approach focuses on using preferences or utilities to guide the choice of plan. For example, Padgham and Singh [112] define a declarative language for specifying preferences over plans as a function of the current context and a function for aggregation of multiple preferences. A similar approach is taken by Visser et al. [144]. Nunes and Luck [110] evaluate the contribution of each plan in terms of utility and preference, seen as parameters for optimisation.

Yet another approach adopts techniques based on HTN planning and search [58]. For example, Sardiña et al. [127] present an AgentSpeak-like language, CanPlan, which integrates HTN-planning into a BDI architecture to find hierarchical decompositions of (agent) plans that avoid incorrect decisions at choice points, and so are less likely to fail during execution. Similarly, Yao and Logan [158] and Yao et al. [160, 161] show how techniques based on Monte-Carlo Tree Search (MCTS) can be used to select plans that minimise conflicts between intentions, achieve goals by their deadlines and recover from failures.

In more practical terms, HTN planning has been made available to Jason and JaCaMo agents [28, 29, 98]. This can be used for example to achieve organisational goals, find a specific course of action that will allow the agent to achieve a number of goals given its plan library, or to provide alternatives in case JaCaMo social plans fail through a combination of task allocation, individual planning, and runtime coordination through JaCaMo artifacts. To support goal allocation before HTN planning is carried out, Cardoso and Bordini [29] make use of plan-library metrics as heuristics, while Baségio and Bordini [8] propose a task allocation mechanism for JaCaMo systems.

Finally, the approach of Chaouche et al. [32] combines both plan composition and learning. They present AgLOTOS, an algebraic language that offers two levels of plans: 'elementary plans' are first composed to produce intention plans, which are in turn composed to build an 'agent plan' satisfying several intentions. This approach helps the agent to select an optimal plan that satisfies several of its intentions while preserving their consistency. The selection is based on a formal construction called the contextual planning system (CPS), which provides potential paths (a path in the CPS is a feasible 'agent plan') with their associated contexts while removing inconsistent options. Past experience is used to reinforce the information available to the CPS. To accommodate possible unexpected changes of context, the learning approach in Chaouche et al. [31] is not based on stochastic processes, as in Nunes and Luck [110], but on an online acquisition of contextual past-experiences concerning the executions of actions.

Plan generation The second main area of work focusses on the application of first-principles AI planning techniques to generate new plans at run-time.

The integration of a planner into the agent language Jadex for providing dynamic plans at runtime was proposed by Walczak et al. [146]. Hindriks and Roberti [74] exploit the fact that agent programs are similar in many respects to languages used for representing planning problems. They show that GOAL agent programs can be used as a planning formalism. By translating agent programs to a PDDL specification, classical planners can be used to return plans to achieve an agent's goals. By adding control features in the agent language and by adapting the agent's execution cycle, classical planners can be used to execute these plans while monitoring for failure, as part of an agent program. de Silva et al. [47] take a similar approach of using classical first-principles planning to find plans that have not been provided by the developer as part of the agent's plan library. However, they focus on

producing abstract, so-called hybrid plans. Hybrid-plans may include abstract operators, which can be mapped back to BDI goals, allowing the agent to execute a generated plan using its (developer-provided) BDI plan library. For a survey of integrating planning algorithms and agent reasoning, see Meneguzzi and de Silva [102].

4.3 AI in the act phase

Finally, we consider work on integrating AI capabilities into the act phase. Work in this area has focussed principally on supporting or automating deliberation about how a set of intentions can be progressed, e.g., to avoid conflicts between intentions, to ensure goals are achieved by their deadlines, or to recover from plan failures.

Intention progression One strand of work is closely related to the analysis of potential interactions between an agent's plans to detect possible conflicts between top-level goals discussed in Sect. 4.1 and the HTN and MCTS-based approaches to plan selection discussed in Sect. 4.2. However in this case, the issue is the order in which the steps of plans to achieve different goals should be executed to avoid conflicts or ensure that goals are achieved by their deadlines, rather than detecting conflicts between goals or to choosing a plan for a subgoal that doesn't conflict with the agent's existing intentions.

The approach to detecting possible conflicts between intentions based on *summary information* proposed by Thangarajah et al. [138] can also be used to determine whether a newly adopted subgoal will definitely be safe to execute without conflicts, or will definitely result in conflicts (e.g., e.g., where the execution of a plan makes the execution of another concurrently executing plan impossible), or may result in conflicts. If the subgoal cannot be executed safely, the subgoal may be suspended and execution of the corresponding intention deferred. This approach is extended in Thangarajah et al. [136] to incorporate other reasons for suspending and resuming (sub)goals. Closely related, Thangarajah et al. [137] have proposed various measures of goal completeness, which can be used by an agent developer, e.g., to prioritise intentions whose top-level goal is close to completion in situations where there is a conflict between intentions.

Another strand of work has investigated how the execution of a set of intentions can be automatically scheduled so as to avoid conflicts, or to achieve some other desirable property, e.g., recover from plan failures, maximise the likelihood of achieving goals in a dynamic environment, or achieve goals by specified deadlines etc. For example, the TÆMS (Task Analysis, Environment Modelling, and Simulation) framework [48] together with Design-To-Criteria (DTC) scheduling [145] have been used in agent architectures such the Soft Real-Time Agent Architecture [143] and AgentSpeak(XL) [15] to select the optimal intention to progress at the current cycle. Yao et al. [159, 161] and Yao and Logan [158] present a stochastic approach to scheduling intentions to avoid conflicts, recover from plan failures (by reestablishing a precondition), and maximise fairness in the progression of the intentions for each goal. Waters et al. [147, 148] present a *coverage-based* approach to intention selection, in which the intention with the lowest coverage, i.e., the highest probability of becoming non-executable due to changes in the environment, is selected for execution. Vikhorev et al. [142] propose an approach that schedules intentions so as to achieve a priority-maximal set of intentions by their deadlines with a specified level of confidence, and Yao et al. [160] extend the stochastic scheduling approach of Yao and Logan [158] for task environments where goals may have both a preferred achievement time and a deadline. Their approach schedules intentions so as to maximise the number of goals achieved and

minimise tardiness (i.e, the difference between the time a goal is achieved and its preferred achievement time).

Much of the work discussed above assumes that actions are instantaneous. A closely related strand of work considers intention progression in the setting of durative actions, that is, where multiple actions may execute concurrently. Durative actions are characteristic of robotics applications, and much of the work in this area has taken place at the intersection of agent programming and robotics. For example, the work of Ziafati et al. [163] discussed in Sect. 4.1 also considers synchronisation issues related to the representation of complex plans and the parallel execution of plans. Coffey and Clark [35] propose an hybrid architecture for robot control, using a plan library in which plans are comprised of hierarchical, suspendable and recoverable teleo-reactive programs programmed in Nilsson's Teleo-Reactive rule-based agent programming language [107]. Clark and Robinson [33] present an extension of teleo-reactive programming, TeleoR, which provides explicit support for the high level programming of concurrent behaviours that require access to shared robotic resources.

The approaches to integrating AI in the act phase differ in their degree of encapsulation, giving rise to a rich space of possible designs for enhancing this phase of the BDI cycle. For example, approaches based on summary information, such as [138, 147], can be used to determine when progression of an intention must be suspended but leave decisions about how the remaining intentions should be progressed to other deliberation, while in Yao and Logan [158] all scheduling decisions are encapsulated in a single process.

4.4 Summary: State of the art in AI in BDI agent programming

Although not exhaustive, the survey above illustrates the broad range of approaches that have been explored to integrating AI techniques into BDI-based agent programming. Much of this research has yet to be incorporated into the 'production' versions of BDI-based languages and platforms, and it is not year clear which approaches will be most fruitful. Moreover, research has proceeded in a somewhat ad-hoc manner, with different aspects of the agent's behaviour considered largely in isolation. It would be beneficial to have a 'roadmap' that clarifies the relationships between existing work, and to guide future research. In the next section we consider possible future directions for research on integrating AI into BDI agents, and how such integration may be engineered.

5 Discussion and research directions

From the research surveyed in the previous section, it is clear that the notion of an agent, and in particular the BDI model of agency based on high-level intentional concepts such as beliefs, goals and plans, forms an ideal framework for the integration of a wide range of symbolic, stochastic and sub-symbolic AI techniques. Such a marriage combines the advantages of the BDI approach highlighted in Sect. 3, specifically the intelligibility and verifiability resulting from the programmatic expression of intentions, with enhanced context sensitivity and robustness, e.g., through the application of machine learning, context awareness and AI planning techniques. The adoption of BDI as a formal model of agent reasoning and decision making, empowered with AI techniques in specific stages of the

reasoning cycle, provides a conceptual and design space to deal with key issues in the engineering of *robust* and *beneficial* AI systems [125].

In this section, we take a step back, and attempt to take a broader view of the possible future relationship between AI/ML and agent programming. In particular, we consider the ways in which AI techniques considered in the abstract, may be integrated into a BDI agent architecture, and sketch a programme of research directed towards identifying the most appropriate ways of integrating AI capabilities into agents. The research programme encompasses both short term opportunities ('low hanging fruit' in the form of immediate synergies between rapidly maturing AI techniques and agent programming), and longer term research questions.

5.1 Architectural strategies for integrating AI in BDI agents

We can identify two main architectural strategies to integrate AI with BDI-based agent programming languages and agent programming more generally: (*i*) AI as a service (i.e., exogenous case); and (*ii*) AI embedded into agents (i.e., endogenous case). In reality, these two strategies form the ends of a spectrum of possibilities, giving rise to a range of 'hybrid' approaches, in which some AI components are exogenous while others are endogenous. However, in the interest of brevity, we focus on purely exogenous and purely endogenous strategies below.

5.1.1 AI as a service

In the exogenous approach, the AI is exploited by the agent *as a service*, that is, it is packaged as a separate, independent component, either running within the same (agent) system, or in a distributed fashion accessed through the network. This latter includes the *cognition-as-a-service* case, in which the AI component is provided as a service in the cloud [52].

From an abstraction point of view, in this case the AI service can be modelled as part of the agents' *application environment* [122]. Examples include external image/speech recognition systems, text-to-speech (TTS) services, document analysis capabilities, etc., now becoming available from major vendors such as Amazon, Google, and IBM. An agent program accesses and exploits the AI capability by means of (external or internal) actions and percepts, as in Fig. 1. From an agent development perspective, this approach is conceptually similar to the way in which an agent program interacts with the underlying agent hardware/platform and the agent's environment. In particular, the agent developer is responsible for integrating the AI (service) into the agent program: invoking the appropriate service(s) at the appropriate point in the execution of the agent, and writing code to exploit the expanded set of percepts (represented as beliefs) made available by the AI service.

The integration of AI as a service with agent programming languages could be supported by means of environment-oriented frameworks such as CArtAgO [121], which implements the A&A conceptual model [111]. In this case the AI service may be modelled as an *artifact* or tool that agents can share and use. Another example is the Environment Interface Standard (EIS) [9], which may be used to define a uniform interface for agents to access and exploit AI services. Yet another approach is to integrate agents with external resources and services using existing middleware standards, e.g., the use of the Apache Camel enterprise integration framework by Cranefield and Ranathunga [39].

Such an approach has the advantage of being easy to integrate into existing agent platforms and development methodologies, while at the same time facilitating the development

of sophisticated applications. For example, a 'personal assistant' agent involving text or speech interaction that previously required specialist development expertise [162] will increasingly fall within the scope of an average agent developer. Further research is required to determine whether frameworks such as CArtAgO and EIS provide the support needed from a software engineering perspective for the effective integration of a broad and diverse range of AI services, for example those requiring large amounts of sub-symbolic information, or those services that return essentially "control information" (which plan to adopt, which intention to progress) that cannot easily be represented by object-level beliefs. Moreover, the need to explicitly invoke a service by an action in a plan in this approach may make it difficult to integrate techniques such as machine learning, except in the form of pre-trained components. As a result, to fully exploit the potential of AI as a service may require at least some changes to the BDI architecture and/or cycle as discussed below. However, while the use of AI as a service has the potential to significantly extend the capabilities and sophistication of BDI-based agents, it is difficult to fully address many of the problems identified in Sect. 3.5 within this approach.

5.1.2 AI embedded into BDI agents

In the endogenous approach, the use of AI techniques is embedded into the agent architecture, that is, AI components or techniques are used to augment or replace elements of the standard BDI architecture and/or cycle defined in Sects. 3.1 and 3.2 . For example, AI components may be used to induce appropriate context conditions for plans, to learn which applicable plan is most appropriate in a given situation, or to manage potential conflicts between intentions.

As evidenced by the wide range of approaches surveyed in Sect. 4, where, how and even which AI techniques should be integrated into the BDI architecture or cycle is still an active research area, and a clear consensus has yet to emerge. Each of the components of a BDI agent shown in Figure 1 can be extended with AI techniques in a variety of ways, giving rise to a very large space of possible extended BDI architectures. (We outline some questions that we believe may be useful in guiding future research in Sect. 5.3 below.)

The problem of where AI should fit in the BDI architecture could be seen as one for the agent language/platform developer rather than the developer of agent applications. However, it is arguable that the most appropriate integration of AI into the BDI architecture is at least partly application specific. If so, there may be no single "best" way of embedding AI into an agent, suggesting that integration of AI should take the form of "pluggable" components that can be selected as needed by the application developer. This could be seen as an extension of the approach proposed for knowledge-representation functionality in Bagosi et al. [7] and discussed in Sect. 4.1 (for other work in this direction, see, e.g., [38]). Another possibility is to exploit a meta-level programming approach to allow the BDI architecture and cycle to be configured by developers to meet the needs of a particular application. Several BDI languages, including PRS [65], 3APL [40, 78], JACK [27, 150] and meta-APL [89], support the use of meta-level plans or rules to modify the 'default' BDI cycle, and such facilities could be extended to encompass the integration of AI components and techniques. While further work is required to identify the appropriate meta-level abstractions to support such 'programmable' AI capabilities, the cognitive concepts (i.e., mental attitudes) that form the core of BDI-based agent languages can be seen as an ideal foundation for the integration of capabilities such as reasoning, planning and scheduling.

However implemented, the aim of this approach is to raise the level of abstraction of agent programming, by increasing the basic 'competence' of the agent language or platform. The aim is not to build a 'richer and more powerful' language, but rather a simpler language (or languages) in which the programmer has to do less, and so worry less about the execution model because the agent is more capable: it "does the right thing" without it having to be explicitly programmed. The agent developer no longer has to encode reasoning such as plan selection, plan failure recovery, etc. in the agent's plans and/or by configuring the agent platform, because those are now built-in features of the APL. There is of course a risk that if the integration of AI techniques is done badly, the agent will behave in ways that are difficult for the developer (and end users) to understand. However we would see this as a failure (e.g., inappropriate choice of AI technique or poor integration of the technique) rather than an inevitable consequence of a more complex language requiring mitigation. (Where learning is involved, the situation is a little more complex, as there is always the risk of the wrong behaviour being learned, but this is a generic problem with learning that should be addressed within a learning component rather than surfaced, and mitigated, at the agent programming level.)

As the competence of the agent increases, the role of the agent developer will change from programming exactly what the agent will do in all situations, to providing more strategic information, heuristics, advice, social knowledge (e.g., norms), etc. about what the agent should (or should not) do in a given situation, leaving the details of the implementation of the strategy to the agent. This style of development has some similarities with the 'training' of systems advocated in the Cognitive approach, and one can envision hybrid architectures in which some behaviours are learnt, while others, perhaps those with a supervisory or high-level decision making role, are 'programmed'. For this reason, and also for the reasons set out at the end of Sect. 3.5, we believe that it is likely that there will continue to be a role for agent programming, at least for the foreseeable future.

For many applications, particularly those involving interaction with humans, there are standard (often codified) ways of approaching a task that must be followed for safety, regulatory, quality control or other reasons. While machine learning (or other AI techniques such as planning) can be used to adapt the behaviour of the agent, e.g., to a particular user or to generate a novel implementation of a high-level action, the critical aspects of the agent's behaviour are required to fall within a particular envelope or follow a particular pattern. This tension is of course reflected in some work in machine learning, and to a lesser extent in planning. We believe BDI-based agent programming approaches enhanced with AI techniques offer a fruitful framework for such 'controlled adaptation', as the structuring of BDI agent programs in terms of goals and plans allows different (sub)goals to be associated with differing degrees of adaptation in a natural way. For example, the means used to achieve some goals may be precisely specified by developer-supplied plans, while the means used to achieve other goals may be learned, or synthesised at run-time using first-principles AI planning techniques.

5.2 Engineering a BDI-based AI integration framework

A fully-fledged BDI-based framework integrating AI techniques should ideally allow the use of different approaches to develop different capabilities of an agent in flexible and modular way. From a developer's perspective, the approaches used to develop capabilities can be classified into one of three main types [63]: the *programming-based* approach, where the agent program is given by the programmer; the *learning-based* approach, where

the program is induced from experience via a learning algorithm; and the *model-based* approach, where the agent program is derived from a model of the problem, e.g., planning. As observed by Geffner [63] and we have argued here, the three approaches are not orthogonal, and exhibit different advantages and limitations. Programming agents by hand puts all the burden on the programmer, who must anticipate all possible contingencies, and often results in systems that are brittle. Learning methods provide greater flexibility, but tend to be limited in scope, given their computational complexity. Model-based methods (which include planning), require a model of the actions, sensors, and goals, and face the computational problem of solving the model.

In this classification, the basic BDI model corresponds to the programming-based approach; however, it provides an appropriate level of abstraction to understand and integrate both the *learning-based* and *model-based* approaches. For a goal to be achieved, the corresponding plans to be used by the agent could be either entirely programmed by hand by a developer, or learnt—either partially or totally, either offline or online—or devised dynamically by planning. For the same goal, we could even have multiple plans developed using different approaches to be used in different contexts (depending on the contextual conditions), with deliberation used to choose the best plan among the various options. For example, the plans selected for each sub-goal in a higher-level plan could be developed using different approaches.

Currently the tools provided by agent programming platforms for programming and running agents mostly provide support for a pure programming-based approach (i.e., an editor, a runtime, and a debugger). Enabling and facilitating the selection of any of these three approaches, switching between programming-, learning-, and model-based approaches, calls for rethinking and extending the development environments and tools used to program and run agent systems. For example, first-class support for a learning-based approach may require the integration of some form of simulator into the tool chain to support the training stage, and the inclusion of explicit periods of training in each stage of the agent development cycle.

5.3 Opportunities and open issues

In the short term, the AI as a service approach offers the opportunity to develop significantly more capable agent applications within the standard BDI model. Sensing and behaviour that would previously have required specialist programming can be easily integrated into existing agent programming platforms and exploited by application developers. Moreover, exploiting such capabilities does not require any modifications to standard agent development methodologies. However, while the opportunities offered by such an approach are significant, it does not fully address the key agent programming issues identified in Sect. 3.5.

At the same time, we would expect to see a continued exploration of where and how AI components can be integrated into the BDI architecture and cycle. Despite a steady stream of research over the last 15 years, work to date has only scratched the surface of this complex question. However, the problem of how to integrate techniques from different subfields of AI is central to engineering more flexible and intelligent agents, and progress in this direction is likely to result in greater gains in the medium to longer term.

As noted in Sect. 5.1, there is a very large space of possible extended BDI architectures. One way to structure future research in this area is to explore where and how AI techniques can be embedded to enable implementation of the BDI cycle *within the BDI model itself*.

It seems plausible that the BDI approach of determining which plan to adopt based on the agent's current beliefs should apply equally to the problem of selecting an appropriate deliberation strategy given the agent's current state. Such an approach has the potential to transfer the intelligibility of the intention-driven BDI model highlighted in Sect. 3.5 to the embedding of AI.

At each BDI cycle, an agent with multiple goals must make decisions about 'what to do next': what means (i.e., plan) to use to achieve a given (sub)goal; which of the currently adopted plan(s) (i.e., intentions) to progress at the current moment; and how these plan(s) should be progressed. Making such decisions involves (at least):

- which plan to adopt if several are applicable;
- which intention to execute next;
- how to handle interactions between intentions;
- how to estimate progress of an intention;
- how to handle lack of progress of a plan or intention; and
- when to drop a goal or try a different approach.

While not all of these capabilities will be required in every agent application, a reasonable argument can be made that many are necessary for most applications (e.g., which plan to adopt, which intention to execute next, how to handle plan failure), and each feature is required for a significant class of applications. Support for making such decisions in current BDI platforms is limited at best. Moreover, current BDI languages typically lack the basic underlying representations for costs, preferences, time, resources, durative actions, expectations, norms etc. necessary to implement such capabilities. For example, it is difficult to implement deliberation about whether to adopt a goal, as the agent's intentions are not first class objects in most BDI-based languages, i.e., the agent's current intentions do not form part of its belief state [96].

Key to this approach to embedding AI into BDI agent programs, is what features of intentions and the context of their execution are necessary to implement the BDI cycle for a particular application. This includes issues already touched on in Sect. 4, e.g., when a goal should be adopted or dropped, which plan to use to achieve a goal, when the plan should next be executed, etc., but also many other issues relating to the social context of the agent, e.g., the expectations of humans and other agents, the prevalent norms, ethics and values and how these should determine the behaviour of an agent. The fact that, in the BDI model, these issues must always be addressed in the context of other intentions with potentially differing characteristics is a unique strength of the BDI approach, in that it necessitates the adoption of a holistic view of the problem of developing autonomous intelligent systems.

Alongside the purely architectural questions of how the overall problem of intelligent behaviour should be broken down (e.g., what are the most appropriate components/APIs), work in this direction gives rise to a range of new research problems centred around the notion of *bounded adaptation*. How should the split between programmer-determined fixed or canonical behaviours and agent-determined adaptations of these behaviours (e.g., refinements, or implementations of (very) high-level actions, etc.) be characterised? What development methodologies and verification approaches can be used to specify and certify the behaviour of agents that integrate significant AI capabilities into their decision making? This can be seen as establishing a new strand of research exploring *hybrids* of the the programming-based, learning-based, and model-based approaches to developing AI capabilities identified by Geffner [63]. We believe that the agent programming paradigm forms an ideal framework in which to explore the resulting complex mix of scientific and engineering questions.

One way of incentivising research on these questions is through the identification of 'challenge problems'. Such challenges, often formalised as competitions or contests, have been used successfully in many areas of AI as a means to foster research. They range from 'grand challenges' (e.g., those from DARPA) to commercial contests (e.g., Netflix Challenge[22]), to more academic contests (e.g., IPC[23], TAC [84]) and competitions with long-term objectives (e.g., RoboCup[24] and its derivatives). Other approaches involve the use of standardised data and problem sets such as the NIST and CIFAR-10 image recognition datasets, and the planning domains and problems used by the International Planning Competition. Yet others focus on standardised environments, often based on games, such as the Arcade Learning Environment [10] and Starcraft [30].

The EMAS community has also developed challenge problems, such as the Multi-agent Programming Contest (MAPC)[25] and the Intention Progression Competition (IPC)[26][95]. However to the extent they consider AI, these tend to focus more on the benefits of incorporating AI techniques or components (e.g., in the case of the Intention Progression Competition) rather than specifically on how such components should be integrated into the architecture of an agent. There is clearly value in the EMAS community engaging more with competitions such as the MAPC and IPC, and with other 'mainstream' AI competitions, e.g., the NASA Space Robotics Challenge,[27] where the advantages of an agent-oriented programming approach can be showcased to researchers in other areas of AI. However, to be maximally effective in advancing research into the integration of AI into agents, a challenge problem should include both multi-agent elements (e.g., autonomous decision making, coordination, etc.), and AI elements (e.g., learning, planning, scheduling, reasoning etc.) where BDI-based agent programming approaches have traditionally offered limited support. At the same time, it should be (just) complex/generic enough to benefit from explicit consideration of where and how AI should be integrated into the agent's architecture. We believe that defining such problems will be a key activity for the EMAS community over the short to medium term.

6 Conclusions

It seems clear that recent developments in artificial intelligence, and particularly in machine learning, have the potential to have significant impact on the development and run-time behaviour of agent systems. Some authors have gone so far as to claim that this will lead to a fundamental paradigm shift away from a "programming based" model of development to a "training based" model, in which intelligent systems are trained to exhibit correct behaviour through machine learning.

In this paper, we have argued that such claims overstate the likely impacts of AI and ML on agent development, and while the incorporation of AI/ML techniques will lead to a significant evolution of the BDI model, the agent programming paradigm will still have a key role to play in the development of intelligent autonomous agents and systems.

[22] www.netflixprize.com.
[23] www.icaps-conference.org/index.php/Main/Competitions.
[24] www.robocup.org.
[25] www.multiagentcontest.org.
[26] www.intentionprogression.org.
[27] http://www.nasa.gov/spacebot.

We briefly surveyed the development and maturation of the BDI-based agent programming model over the last twenty years. In its current form, it offers significant advantages of intelligibility and verifiability that are difficult to replicate using only machine learning techniques. Moreover these advantages are key to the widespread acceptance and adoption of autonomous systems [80]. In addition, as our survey of previous work on how AI can be integrated into the BDI model demonstrates, the agent programming community already has considerable experience in exploiting AI techniques. However, much of this work has yet to be incorporated into mature BDI-based agent platforms, and significant open research questions remain.

We also identified a number of possible directions for future research. In the short term, the 'AI as a service' model is likely to offer significant opportunities to increase the capabilities of agents within the standard BDI model. However, in the medium to longer term, embedding AI into the BDI architecture and cycle is likely to offer greater gains in the flexibility, robustness, and intelligence of agent behaviour. While the BDI model will undoubtedly evolve, we believe that the BDI-based agent development paradigm and the concepts of beliefs, goals, and intentions (in the form of commitments to future behaviour) on which it is based, will continue to form a fruitful framework for research for at least the next twenty years.

Acknowledgements At the time of writing, Rafael Bordini was on a sabbatical leave funded by CAPES at the University of Genoa.

References

1. Aha, D. W., Klenk, M., Munoz-Avila, H., Ram, A., & Shapiro, D. (Eds) (2010). *Goal-directed autonomy: Notes from the AAAI 2010 workshop (W4)*. AAAI.
2. Alami, R., Chatila, R., Fleury, S., Ghallab, M., & Ingrand, F. (1998). An architecture for autonomy. *The International Journal of Robotics Research*, *17*(4), 315–337.
3. Aldewereld, H., & Dignum, V. (2011) . OperettA: Organization-oriented development environment. In *Languages, methodologies, and development tools for multi-agent systems* (pp. 1–18). Springer.
4. Alechina, N., Behrens, T., Dastani, M., Hindriks, K., Hubner, J., Logan, B., et al. (2013). Multi-cycle query caching in agent programming. In *Proceedings of the twenty-seventh AAAI conference on artificial intelligence (AAAI 2013)* (pp. 32–38). AAAI Press.
5. Alechina, N., Bordini, R., Hubner, J., Jago, M., & Logan, B. (2006). Automating belief revision for AgentSpeak. In *Proceedings of the fourth international workshop on declarative agent languages and technologies (DALT 2006)*.
6. Baader, F., & Zarrieß, B. (2013). Verification of Golog programs over description logic actions. In *Proceedings of 9th international symposium frontiers of combining systems—FroCoS 2013. Lecture notes in computer science* (Vol. 8152, pp. 181–196), Nancy, France, September 18–20, 2013. Springer.
7. Bagosi, T., de Greeff, J., Hindriks, K. V., & Neerincx, M. A. (2015). Designing a knowledge representation interface for cognitive agents. In *Engineering multi-agent systems* (pp. 33–50). Springer.

8. Baségio, T. L., & Bordini, R. H. (2019). Allocating structured tasks in heterogeneous agent teams. *Computational Intelligence*, *35*(1), 124–155.
9. Behrens, T. M., Hindriks, K. V., & Dix, J. (2011). Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, *61*(4), 261–295.
10. Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.
11. Bellifemine, F., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with JADE*. Hoboken: Wiley.
12. Benfield, S. S., Hendrickson, J., & Galanti, D. (2006). Making a strong business case for multiagent technology. In *Proceedings of the 5th international joint conference on autonomous agents and multiagent systems (AAMAS 2006)* (pp. 10–15), ACM.
13. Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., & Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, *78*(6), 747–761.
14. Bordini, R. H., & Moreira, Á. F. (2002). Proving the asymmetry thesis principles for a BDI agent-oriented programming language. In *Pre-proceedings of 3rd international workshop on computational logic in multi-agent systems, CLIMA'02* (Vol. 93, pp. 94–108), Copenhagen, Denmark, August 1, 2002, Roskilde University, Datalogiske Skrifter.
15. Bordini, R., Bazzan, A. L. C., de O Jannone, R., Basso, D. M., Vicari, R. M., & Lesser, V. R . (2002). AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In *Proceedings of the first international conference on autonomous agents and multiagent systems (AAMAS'02)* (pp. 1294–1302). ACM Press.
16. Bordini, R. H., Dastani, M., Dix, J., & Fallah-Seghrouchni, A. E. (Eds.). (2005). *Multi-agent programming: Languages, platforms and applications*. Berlin: Springer.
17. 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*, *30*(1), 33–44.
18. Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. Hoboken: Wiley.
19. Bordini, R. H., Dastani, M., Dix, J., & Fallah-Seghrouchni, A. E. (Eds.). (2009). *Multi-agent programming, languages, tools and applications*. Berlin: Springer.
20. Boutilier, C., Reiter, R., Soutchanski, M., & Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the seventeenth national conference on artificial intelligence and twelfth conference on on innovative applications of artificial intelligence* (pp. 355–362), July 30–August 3, 2000, Austin, Texas, USA. AAAI Press/The MIT Press.
21. Bradski, G., & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. Sebastopol: O'Reilly Media.
22. Bratman, M. E. (1987). *Intention, plans, and practical reason*. Cambridge: Harvard University Press.
23. Bratman, M. E., Israel, D. J., & Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, *4*(4), 349–355.
24. Braubach, L., Pokahr, A., & Lamersdorf, W. (2005). Jadex: A BDI-agent system combining middleware and reasoning. In *Software agent-based applications, platforms and development kits* (pp. 143–168). Whitestein Series in Software Agent Technologies, Birkhäuser Basel.
25. Braubach, L., Pokahr, A., Moldt, D., & Lamersdorf, W .(2005). Goal representation for BDI agent systems. In *Second international workshop on programming multi-agent systems, ProMAS 2004. 2004 Selected revised and invited papers. Lecture notes in computer science* (Vol. 3346, pp. 44–65), New York, NY, USA, July 20. Springer.
26. Broekens, J., Harbers, M., Hindriks, K. V., van den Bosch, K., Jonker, C. M., & Meyer, J. C. (2010) . Do you get it? User-evaluated explainable BDI agents. In *Proceedings of multiagent system technologies, 8th German conference, MATES 2010. Lecture notes in computer science* (Vol. 6251, pp. 28–39), Leipzig, Germany, September 27–29, 2010, Springer.
27. Busetta, P., Ronnquist, R., Hodgson, A., & Lucas, A. (1999). JACK intelligent agents—Components for intelligent agents in Java. AgentLink News 2.
28. Cardoso, R. C., & Bordini, R. H. (2019). Decentralised planning for multi-agent programming platforms. In *Proceedings of the 18th international conference on autonomous agents and multiagent systems, AAMAS '19* (pp. 799–818), Montreal, QC, Canada, May 13–17, 2019. International Foundation for Autonomous Agents and Multiagent Systems.
29. Cardoso, R. C., & Bordini, R. H . (2016). Allocating social goals using the contract net protocol in online multi-agent planning. In *5th Brazilian conference on intelligent systems, BRACIS 2016* (pp. 199–204), Recife, Brazil, October 9–12, 2016. IEEE.

30. Certický, M., Churchill, D., Kim, K., Certický, M., & Kelly, R. (2019). Starcraft AI competitions, bots, and tournament manager software. *IEEE Transactions on Games*, *11*(3), 227–237.
31. Chaouche, A., Fallah-Seghrouchni, A. E., Ilié, J., & Saïdouni, D. (2015). Improving the contextual selection of BDI plans by incorporating situated experiments. In *Proceedings of 11th IFIP WG 12.5 international conference on artificial intelligence applications and innovations—AIAI 2015. IFIP advances in information and communication technology* (Vol. 458, pp. 266–281), Bayonne, France, September 14–17, 2015. Springer.
32. Chaouche, A., Fallah-Seghrouchni, A. E., Ilié, J., & Saïdouni, D. (2016). Learning from situated experiences for a contextual planning guidance. *Journal of Ambient Intelligence and Humanized Computing*, *7*(4), 555–566.
33. Clark, K. L., & Robinson, P. J. (2015). Robotic agent programming in TeleoR. In *2015 IEEE international conference on robotics and automation (ICRA 2015)* (pp. 5040–5047).
34. 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*(2–4), 171–206.
35. Coffey, S., & Clark, K.(2006). A hybrid, teleo-reactive architecture for robot control. In *Proceedings of the second international workshop on multi-agent robotic systems*.
36. Cohen, P. R., & Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, *42*(2–3), 213–261.
37. Collier, R. W., Russell, S. E., & Lillis, D. (2015). Reflecting on agent programming with AgentSpeak(L). In *Proceedings of 18th international conference on PRIMA 2015: principles and practice of multi-agent systems. Lecture Notes in Computer Science* (Vol. 9387, pp. 351–366), Bertinoro, Italy, October 26–30, 2015. Springer.
38. Costantini, S. (2015). ACE: a flexible environment for complex event processing in logical agents. In *Third international workshop on engineering multi-agent systems, EMAS 2015. Lecture notes in computer science* (Vol. 9318, pp. 70–91), Istanbul, Turkey, May 5, 2015, Revised, Selected, and Invited Papers. Springer.
39. Cranefield, S., & Ranathunga, S. (2013). Embedding agents in business processes using enterprise integration patterns. In *First international workshop on engineering multi-agent systems, EMAS 2013. Lecture notes in computer science* (Vol. 8245, pp. 97–116), St. Paul, MN, USA, May 6–7, 2013. Revised Selected Papers, Springer.
40. Dastani, M., de Boer, F., Dignum, F., & Meyer, J. C. (2003) . Programming agent deliberation: An approach illustrated using the 3APL language. In *Proceedings of the 2nd international joint conference on autonomous agents and multiagent systems (AAMAS 2003)*. ACM (pp. 97–104).
41. Dastani, M., de Boer, F., Dignum, F., Van Der Hoek, W., Kroese, M., & Meyer, J. J., et al. (2002). Programming the deliberation cycle of cognitive robots. In *Proceedings of the 3rd international cognitive robotics workshop*.
42. Dastani, M. M., Hindriks, K. V., Novák, P., & Tinnemeier, N. A. M. (2009) . Combining multiple knowledge representation technologies into agent programming languages. In *Declarative agent languages and technologies VI*, Springer, (pp. 60–74).
43. Dastani, M. (2008). 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, *16*(3), 214–248.
44. De Giacomo, G., Lespérance, Y., Levesque, H. J., & Sardiña, S. (2009). IndiGolog: A high-level programming language for embedded reasoning agents. In *Multi-agent programming, languages, tools and applications* (pp. 31–72). Berlin: Springer.
45. De Giacomo, G., Reiter, R., & Soutchanski, M. (1998). Execution monitoring of high-level robot programs. In *Principles of knowledge representation and reasoning-international conference* (pp. 453–465). Morgan Kaufmann Publishers.
46. de Silva, L., Meneguzzi, F., & Logan, B. (2018). An operational semantics for a fragment of PRS. In *Proceedings of the 27th international joint conference on artificial intelligence (IJCAI 2018)* (pp 195–202). IJCAI.
47. de Silva, L., Sardiña, S., & Padgham, L. (2009). First principles planning in BDI systems. In *8th international joint conference on autonomous agents and multiagent systems (AAMAS 2009)* (vol. 2, pp. 1105–1112), Budapest, Hungary, May 10–15, 2009, IFAAMAS.
48. Decker, K. S., & Lesser, V. R. (1993). Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management*, *2*, 215–234.
49. Dennett, D. C. (1987). *The intentional stance*. Cambridge: The MIT Press.
50. Dennis, L. A., Fisher, M., Webster, M. P., & Bordini, R. H. (2012). Model checking agent programming languages. *Automated Software Engineering*, *19*(1), 5–63.

51. Dennis, L. A., Fisher, M., Lincoln, N. K., Lisitsa, A., & Veres, S. M. (2016). Practical verification of decision-making in agent-based autonomous systems. *Automated Software Engineering*, *23*(3), 305–359.
52. Dignum, V .(2015). Mind as a service: Building socially intelligent agents. In *International workshops on coordination, organizations, institutions, and normes in agent systems XI—COIN 2015 , COIN@AAMAS. Lecture Notes in Computer Science* (Vol. 9628, pp. 119–133), Istanbul, Turkey, May 4, 2015, COIN@IJCAI, Buenos Aires, Argentina, July 26, 2015, Revised Selected Papers, Springer.
53. Dignum, V., Baldoni, M., Baroglio, C., Caon, M., Chatila, R., Dennis, L. A., et al.(2018). Ethics by design: Necessity or curse? In *Proceedings of the 2018 AAAI/ACM conference on AI, ethics, and society, AIES 2018* (pp. 60–66), New Orleans, LA, USA, February 02–03, 2018.
54. Dignum, V., & Dignum, F. (2010). Designing agent systems: State of the practice. *International Journal of Agent-Oriented Software Engineering*, *4*(3), 224–243.
55. d'Inverno, M., Kinny, D., Luck, M., & Wooldridge, M. J. (1998). A formal specification of dMARS. In: *Proceedings of 4th international workshop intelligent agents IV, agent theories, architectures, and languages, ATAL '97. Lecture notes in computer science* (Vol. 1365, pp. 155–176), Providence, Rhode Island, USA, July 24–26, 1997. Springer.
56. d'Inverno, M., Luck, M., Georgeff, M. P., Kinny, D., & Wooldridge, M. J. (2004). The dMARS architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, *9*(1–2), 5–53.
57. Dix, J., & Fisher, M .(2002). Programming multi agent systems based on logic. Technical report, Dagstuhl Seminar 02481, IBFI GmbH, Schloß Dagstuhl
58. Erol, K., Hendler, J., & Nau, D. S. (1994). HTN planning: Complexity and expressivity. In *Proceedings of the national conference on artificial intelligence (AAAI 1994)* (pp. 1123–1128).
59. Fallah-Seghrouchni, A. E., & Suna, A. (2003). CLAIM: A computational language for autonomous, intelligent and mobile agents. In *First international workshop on programming multi-agent systems, PROMAS 2003. Lecture notes in computer science* (Vol. 3067, pp. 90–110), Melbourne, Australia, July 15, 2003, Selected Revised and Invited Papers, Springer.
60. Fisher, M. (1993). Towards a semantics for concurrent METATEM. In *Proceedings of executable modal and temporal logics, IJCAI '93, workshop. Lecture Notes in Computer Science* (Vol. 897, pp. 86–102), Chambéry, France, August 28, 1993, Springer.
61. Fisher, M., Dennis, L., & Webster, M. (2013). Verifying autonomous systems. *Communications of the ACM*, *56*(9), 84–93.
62. Foughali, M., Berthomieu, B., Dal Zilio, S., Ingrand, F., & Mallet, A. (2016). Model checking real-time properties on the functional layer of autonomous robots. In *Proceedings of the international conference on formal engineering methods (ICFEM)* (pp. 383–399).
63. Geffner, H. (2010). The model-based approach to autonomous behavior: A personal view. In *Proceedings of the AAAI conference on artificial intelligence (AAAI-2010)* (pp. 1709–1712), AAAI Press.
64. Georgeff, M. P., & Ingrand, F. F. (1989). Decision-making in an embedded reasoning system. In *Proceedings of the 11th international joint conference on artificial intelligence* (pp. 972–978), Detroit, MI, USA, August 1989. Morgan Kaufmann.
65. Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the sixth national conference on artificial intelligence (AAAI-87)* (pp. 677–682). Morgan Kaufmann.
66. Ghallab, M., Nau, D. S., & Traverso, P. (2004). *Automated planning—Theory and practice*. Amsterdam: Elsevier.
67. Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated planning and acting*. Cambridge: Cambridge University Press.
68. Guerra-Hernández, A., El Fallah Seghrouchni, A., & Soldano, H. (2004). Learning in BDI multi-agent systems. In *CLIMA 2004—4th international workshop on computational logic in multi-agent systems. Lecture Notes in Computer Science* (Vol. 3259, pp. 218–233). Springer.
69. Gunning, D., & Aha, D. (2019). DARPA's explainable artificial intelligence (XAI) program. *AI Magazine*, *40*(2), 44–59.
70. Harbers, M., van den Bosch, K., & Meyer, J. C. (2010). Design and evaluation of explainable BDI agents. In *Proceedings of the 2010 IEEE/WIC/ACM international conference on intelligent agent technology, IAT 2010* (pp. 125–132), Toronto, Canada, August 31–September 3, 2010. IEEE Computer Society Press.
71. Harland, J., Morley, D. N., Thangarajah, J., & Yorke-Smith, N. (2017). Aborting, suspending, and resuming goals and plans in BDI agents. *Autonomous Agents and Multi-Agent Systems*, *31*(2), 288–331.

72. Hindriks, K .(2008). Modules as policy-based intentions: Modular agent programming in GOAL. In *Programming multi-agent systems* (pp. 156–171). Springer.
73. Hindriks, K. V. (2012). Debugging is explaining. In *PRIMA 2012: Principles and practice of multi-agent systems* (pp. 31–45). Springer.
74. Hindriks, K. V., & Roberti, T. (2009). GOAL as a planning formalism. In *Multiagent system technologies* (pp. 29–40). Springer.
75. Hindriks, K. V., de Boer, F. S., van der Hoek, W., & Meyer, J. J. C. (1998). Formal semantics for an abstract agent programming language. In *Intelligent agents IV agent theories, architectures, and languages* (pp. 215–229). Springer.
76. Hindriks, K. V., de Boer, F. S., van der Hoek, W., & Meyer, J. C. (2000). Agent programming with declarative goals. In *Proceedings of 7th international workshop on intelligent agents VII. Agent theories architectures and languages, ATAL 2000. Lecture notes in computer science* (Vol. 1986, pp. 228–243), Boston, MA, USA, July 7–9, 2000. Springer.
77. Hindriks, K. V. (2009). Programming rational agents in GOAL. *Multi-agent programming: Languages, tools and applications* (pp. 119–157). Berlin: Springer.
78. Hindriks, K. V., de Boer, F. S., van der Hoek, W., & Meyer, J. J. C. (1999). Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, *2*(4), 357–401.
79. Hübner, J. F., Sichman, J. S., & Boissier, O. (2007). Developing organised multiagent systems using the MOISE. *IJAOSE*, *1*(3/4), 370–395.
80. IEEE. (2019). Ethically aligned design: A vision for prioritizing human well-being with autonomous and intelligent systems, first edition. The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems.
81. Ingrand, F. F., Chatila, R., Alami, R., & Robert, F. (1996). PRS: A high level supervision and control language for autonomous mobile robots. In *Proceedings of the international conference on robotics and automation (ICRA 1996)* (pp. 43–49).
82. Jordan, H. R., Botterweck, G., Noll, J., Butterfield, A., & Collier, R. W. (2015). A feature model of actor, agent, functional, object, and procedural programming languages. *Science of Computer Programming*, *98*, 120–139.
83. Kelly, J. E., & Hamm, S. (2013). *Smart machines: IBM's Watson and the era of cognitive computing*. New York: Columbia University Press.
84. Ketter, W., & Symeonidis, A. L. (2012). Competitive benchmarking: Lessons learned from the trading agent competition. *AI Magazine*, *33*(2), 103–107.
85. Klapiscak, T., & Bordini, R. H. (2009). JASDL: A practical programming approach combining agent and semantic web technologies. In *Declarative agent languages and technologies VI* (pp. 91–110). Springer.
86. Kowalski, R., & Sadri, F. (1999). From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, *25*(3), 391–419.
87. Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.
88. Larsen, J. B. (2018). Agent programming languages and logics in agent-based simulation. In: *10th Asian Conference on modern approaches for intelligent information and database systems, ACIIDS 2018. Studies in computational intelligence* (Vol. 769, pp. 517–526), Dong Hoi City, Vietnam, March 19–21, 2018, Extended Posters. Springer.
89. Leask, S., & Logan, B. (2015). Programming deliberation strategies in meta-APL. In *Proceedings of the 18th conference on principles and practice of multi-agent systems (PRIMA 2015). Lecture notes in computer science* (Vol. 9387, pp. 433–448). Springer.
90. Lemaignan, S., Warnier, M., Sisbot, E. A., Clodic, A., & Alami, R. (2017). Artificial cognition for social human–robot interaction: An implementation. *Artificial Intelligence*, *247*, 45–69.
91. Lespérance, Y., Levesque, H. J., Lin, F., Marcu, D., Reiter, R., & Scherl, R. B. (1994). A logical approach to high-level robot programming—a progress report. In *Control of the physical world by intelligent systems. Papers from the 1994 AAAI fall symposium* (pp. 79–85).
92. Lespérance, Y., Levesque, H. J., Lin, F., Marcu, D., Reiter, R., & Scherl, R. B.(1995). Foundations of a logical approach to agent programming. In *Proceedings of intelligent agents II, agent theories, architectures, and languages, IJCAI '95, workshop (ATAL). Lecture notes in computer science* (Vol. 1037, pp. 331–346), Montreal, Canada, August 19–20, 1995. Springer.
93. Levesque, H., & Reiter, R. (1998). High-level robotic control: Beyond planning—A position paper. In *AIII 1998 spring symposium: Integrating robotics research: taking the next big leap* (Vol. 37).
94. Logan, B. (2015). A future for agent programming. In *Engineering multi-agent systems: Third international workshop, EMAS 2015* (pp. 3–17), Istanbul, Turkey, May 5, 2015, Revised, Selected, and Invited Papers, Springer.

95. Logan, B., Thangarajah, J., & Yorke-Smith, N. (2017). Progressing intention progresson: A call for a goal-plan tree contest. In *Proceedings of the 16th international conference on autonomous agents and multiagent systems (AAMAS 2017)* (pp. 768–772), IFAAMAS, IFAAMAS, Sao Paulo, Brazil.
96. Logan, B. (2018). An agent programming manifesto. *International Journal of Agent-Oriented Software Engineering*, *6*(2), 187–210.
97. Lützenberger, M., Konnerth, T., & Küster, T .(2015). Programming of multiagent applications with JIAC. In *Industrial agents* (pp. 381–398). Morgan Kaufmann.
98. Maia, A., & Sichman, J. S. (2018). Explicit representation of planning autonomy in MOISE organizational model. In *2018 7th Brazilian conference on intelligent systems (BRACIS)* (pp. 384–389).
99. Marcus, G. (2018). Deep learning: A critical appraisal. *CoRR* http://arxiv.org/abs/1801.00631.
100. Mascardi, V., Ancona, D., Bordini, R. H., & Ricci, A. (2011). CooL-AgentSpeak: Enhancing AgentSpeak-DL agents with plan exchange and ontology services. In *Proceedings of the 2011 IEEE/WIC/ACM international conference on intelligent agent technology, IAT 2011* (pp. 109–116), Campus Scientifique de la Doua, Lyon, France, August 22–27, 2011, IEEE Computer Society.
101. Mascardi, V., Demergasso, D., & Ancona, D. (2005). Languages for programming bdi-style agents: An overview. In *WOA 2005: Dagli Oggetti agli Agenti. 6th AI*IA, TABOO joint workshop "From objects to agents": Simulation and formal analysis of complex systems* (pp. 9–15), 14–16, November 2005. *Camerino*. Italy, Pitagora Editrice Bologna: MC.
102. Meneguzzi, F., & De Silva, L. (2015). Planning in BDI agents: A survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review*, *30*(1), 1–44.
103. Moreira, Á. F., Vieira, R., Bordini, R. H., & Hübner, J. F. (2006). Agent-oriented programming with underlying ontological reasoning. In *Declarative agent languages and technologies III* (pp. 155–170). Springer.
104. Müller, J. P., & Fischer, K. (2014). Application impact of multi-agent systems and technologies: A survey. In *Agent-oriented software engineering* (pp. 27–53). Springer.
105. Niemueller, T., Zwilling, F., Lakemeyer, G., Löbach, M., Reuter, S., Jeschke, S., & Ferrein, A. (2017). Cyber-physical system intelligence. In *Industrial internet of things: Cybermanufacturing systems* (pp. 447–472). Springer.
106. Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*(1), 139–158.
107. Nilsson, N. J. (2001). Teleo-reactive programs and the triple-tower architecture. *Electronic Transactions on Artificial Intelligence*, *5*, 99–110.
108. Norling, E. (2004). Folk psychology for human modelling: Extending the BDI paradigm. In *Proceedings of the third international joint conference on autonomous agents and multiagent systems. AAMAS '04* (pp. 202–209). IEEE Computer Society.
109. Norling, E., & Ritter, F. E. (2004). Towards supporting psychologically plausible variability in agent-based human modelling. In *3rd international joint conference on autonomous agents and multiagent systems (AAMAS 2004)*, 19–23 August 2004 (pp. 758–765). IEEE Computer Society, New York, NY, USA.
110. Nunes, I., & Luck, M. (2014). Softgoal-based plan selection in model-driven BDI agents. In *Proceedings of the 13th international conference on autonomous agents and multiagent systems (AAMAS 2014)* (pp. 749–756). IFAAMAS.
111. 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.
112. Padgham, L., & Singh, D. (2013). Situational preferences for BDI plans. In *Proceedings of the 12th international conference on autonomous agents and multi-agent systems (AAMAS 2013)* (pp. 1013–1020). IFAAMAS.
113. Panisson, A. R., Parsons, S., McBurney, P., & Bordini, R. H. (2018). Choosing appropriate arguments from trustworthy sources. In *Proceedings of COMMA 2018—Computational models of argument. Frontiers in Artificial Intelligence and Applications* (Vol. 305, pp. 345–352), Warsaw, Poland, 12–14 September 2018. IOS Press.
114. 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 *Engineering multi-agent systems* (pp. 136–155). Springer.
115. Parnas, D. L. (2017). The real risks of artificial intelligence. *Communications of the ACM*, *60*(10), 27–31.
116. Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine. In: *Multi-agent programming: Languages, platforms and applications* (pp. 149–174). Springer.
117. Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of agents breaking away, 7th European workshop on modelling autonomous agents in a*

multi-agent world. Lecture notes in computer science (Vol. 1038, pp. 42–55), Eindhoven, The Netherlands, January 22–25, 1996. Springer.
118. Rao, A. S., & Georgeff, M. P. (1991). Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics. In *Proceedings of the 12th international joint conference on artificial intelligence (IJCAI'91)* (pp. 498–505). Morgan Kaufmann.
119. Rao, A. S., & Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In *Proceedings of the second international conference on principles of knowledge representation and reasoning (KR'91)* (pp. 473–484).
120. Rao, A. S., & Georgeff, M. P. (1998). Decision procedures for BDI logics. *Journal of Logic and Computation*, *8*(3), 293–342.
121. Ricci, A., Piunti, M., Viroli, M., & Omicini, A. (2009). Environment programming in CArtAgO. In *Bordini et al. (2009)* (pp. 259–288).
122. 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.
123. Rodriguez, S., Gaud, N., & Galland, S. (2014). SARL: A general-purpose agent-oriented programming language. In *The 2014 IEEE/WIC/ACM international conference on intelligent agent technology*. IEEE Computer Society Press.
124. Ross, R. J., Collier, R. W., O'Hare, G. M. P. (2004). AF-APL—Bridging principles and practice in agent oriented languages. In *Second international workshop on programming multi-agent systems, ProMAS 2004. Lecture notes in computer science* (Vol. 3346, pp. 66–88), New York, NY, USA, July 20, 2004. Selected Revised and Invited Papers, Springer.
125. Russell, S., Dewey, D., & Tegmark, M. (2015). Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, *36*(4), 105–114.
126. Sardiña, S., & Padgham, L. (2007). Goals in the context of BDI plan failure and planning. In *Proceedings of the Sixth international joint conference on autonomous agents and multiagent systems (AAMAS 2007)* (pp. 1–8). ACM.
127. Sardiña, S., de Silva, L., & Padgham, L. (2006). Hierarchical planning in BDI agent programming languages: A formal approach. In *5th international joint conference on autonomous agents and multiagent systems* (pp. 1001–1008), ACM.
128. Scerri, D., Hickmott, S. L., & Padgham, L . (2012). User understanding of cognitive processes in simulation: A tool for exploring and modifying. In *Winter simulation conference, WSC '12* (pp. 240:1–240:12), Berlin, Germany, December 9–12, 2012, WSC.
129. Shoham, Y. (1990). Agent-oriented programming. Technical Report STAN-CS-90-1335, Stanford University
130. Shoham, Y. (1991). AGENT0: A simple agent language and its interpreter. In *Proceedings of the 9th national conference on artificial intelligence (AAAI-91)* (pp. 704–709). MIT Press
131. Silva, D. G., & Gluz, J. C. (2011). AgentSpeak(PL): A new programming language for BDI agents with integrated bayesian network model. In *2011 international conference on information science and applications* (pp. 1–7).
132. Singh, D., & Hindriks, K. V. (2013). Learning to improve agent behaviours in GOAL. In *Programming multi-agent systems. Lecture notes in computer science* (Vol. 7837, pp. 158–173). Springer.
133. Singh, D., Sardiña, S., Padgham, L., & James, G. (2011). Integrating learning into a BDI agent for environments with changing dynamics. In *IJCAI 2011, Proceedings of the 22nd international joint conference on artificial intelligence* (pp. 2525–2530), Barcelona, Catalonia, Spain, July 16–22, 2011, IJCAI/AAAI.
134. Singh, D., Sardiña, S., & Padgham, L. (2010). Extending BDI plan selection to incorporate learning from experience. *Robotics and Autonomous Systems*, *58*(9), 1067–1075.
135. Stabile, M. F., & Sichman, J. S. (2015). Evaluating perception filters in BDI Jason agents. In *2015 Brazilian conference on intelligent systems (BRACIS 2015)* (pp. 116–121).
136. Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2008). Suspending and resuming tasks in BDI agents. In *Proceedings of the seventh international conference on autonomous agents and multi agent systems (AAMAS'08)* (pp. 405–412).
137. Thangarajah, J., Harland, J., Morley, D. N., & Yorke-Smith, N. (2014). Quantifying the completeness of goals in BDI agent systems. In *Proceedings of the 21st European conference on artificial intelligence (ECAI-2014)*(pp. 879–884), IOS Press.
138. Thangarajah, J., Padgham, L., & Winikoff, M. (2003). Detecting & avoiding interference between goals in intelligent agents. In *Proceedings of the Eighteenth international joint conference on artificial intelligence (IJCAI-03)* (pp. 721–726), Morgan Kaufmann.
139. Thangarajah, J., & Padgham, L. (2011). Computationally effective reasoning about goal interactions. *Journal of Automated Reasoning*, *47*(1), 17–56.

140. Tinnemeier, N. A. M., Dastani, M., Meyer, J. C., & van der Torre, L. W. N. (2009). Programming normative artifacts with declarative obligations and prohibitions. In *Proceedings of the 2009 IEEE/WIC/ACM international conference on intelligent agent technology, IAT 2009* (pp. 145–152), Milan, Italy, 15–18 September 2009. IEEE Computer Society.
141. Vieira, R., Moreira, Á. F., 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*, 221–267.
142. Vikhorev, K., Alechina, N., & Logan, B. (2011) . Agent programming with priorities and deadlines. In *Proceedings of the tenth international conference on autonomous agents and multiagent systems (AAMAS 2011)* (pp. 397–404).
143. Vincent, R., Horling, B., Lesser, V., & Wagner, T. (2001). Implementing soft real-time agent control. In *Proceedings of the fifth international conference on autonomous agents (AGENTS'01)* (pp. 355–362).
144. Visser, S., Thangarajah, J., Harland, J., & Dignum, F. (2016). Preference-based reasoning in BDI agent systems. *Autonomous Agents and Multi-Agent Systems*, *30*(2), 291–330.
145. Wagner, T., Garvey, A., & Lesser, V. (1998). Criteria-directed heuristic task scheduling. *International Journal of Approximate Reasoning*, *19*, 91–118.
146. Walczak, A., Braubach, L., Pokahr, A., & Lamersdorf, W. (2007) . Augmenting BDI agents with deliberative planning techniques. In *Proceedings of the 4th international conference on programming multi-agent systems (ProMAS'06)* (pp. 113–127). Springer.
147. Waters, M., Padgham, L., & Sardina, S. (2014). Evaluating coverage based intention selection. In *Proceedings of the 13th international conference on autonomous agents and multi-agent systems (AAMAS 2014), IFAAMAS* (pp. 957–964).
148. Waters, M., Padgham, L., & Sardiña, S. (2015). Improving domain-independent intention selection in BDI systems. *Autonomous Agents and Multi-Agent Systems*, *29*(4), 683–717.
149. Wei, C., & Hindriks, K. V .(2013) . An agent-based cognitive robot architecture. In *Programming multi-agent systems* (pp. 54–71). Springer.
150. Winikoff, M. (2005). JACK^{TM} intelligent agents: An industrial strength platform. In *Bordini et al. (2005)* (pp. 175–193).
151. Winikoff, M. (2006). An AgentSpeak meta-interpreter and its applications. In *Programming multi-agent systems: third international workshop, ProMAS 2005* (pp. 123–138), Utrecht, The Netherlands, July 26, 2005. Revised and Invited Papers, Springer.
152. Winikoff, M. (2017). Debugging agent programs with "Why?" questions. In *Proceedings of the 16th conference on autonomous agents and multiagent systems, AAMAS 2017* (pp. 251–259), São Paulo, Brazil, May 8–12, 2017. ACM.
153. Winikoff, M., & Padgham, L .(2013) . Agent oriented software engineering. In *Multiagent systems*. MIT Press.
154. Winikoff, M., Dignum, V., & Dignum, F. (2018). Why bad coffee? explaining agent plans with valuings. In *Proceedings of computer safety, reliability, and security—SAFECOMP 2018 workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Västerås. Lecture Notes in Computer Science* (Vol. 11094, pp. 521–534), Sweden, September 18, 2018. Springer.
155. Winikoff, M., Padgham, L., Harland, J., & Thangarajah, J. (2002). Declarative & procedural goals in intelligent agent systems. In *Proceedings of the eighth international conference on principles and knowledge representation and reasoning (KR-02)* (pp. 470–481), Toulouse, France, April 22–25, 2002. Morgan Kaufmann.
156. Winikoff, M. (2017a). BDI agent testability revisited. *Journal of Autonomous Agents and Multi-Agent Systems*, *31*(5), 1094–1132.
157. Wooldridge, M. (2000). *Reasoning about rational agents*. Cambridge: MIT Press.
158. Yao, Y., & Logan, B. (2016) . Action-level intention selection for BDI agents. In *Proceedings of the 15th international conference on autonomous agents and multiagent systems (AAMAS 2016), IFAAMAS* (pp. 1227–1235).
159. Yao, Y., Logan, B., & Thangarajah, J. (2014). SP-MCTS-based intention scheduling for BDI agents. In *Proceedings of the 21st European conference on artificial intelligence (ECAI-2014), ECCAI* (pp. 1133–1134). IOS Press.
160. Yao, Y., Logan, B., & Thangarajah, J. (2016). Intention selection with deadlines. In *Proceedings of the 22nd European conference on artificial intelligence (ECAI-2016), ECCAI* (pp. 1700–1701). IOS Press.
161. Yao, Y., Logan, B., & Thangarajah, J. (2016). Robust execution of BDI agent programs by exploiting synergies between intentions. In *Proceedings of the thirtieth AAAI conference on artificial intelligence (AAAI-16)* (pp. 2558–2564). AAAI Press.

162. Yorke-Smith, N., Saadati, S., Myers, K. L., & Morley, D. N. (2012). The design of a proactive personal agent for task management. *International Journal on Artificial Intelligence Tools*, *21*(1), 1250004.
163. Ziafati, P., Dastani, M., Meyer, J. J., & van der Torre, L. (2013). Agent programming languages requirements for programming autonomous robots. In *Programming multi-agent systems* (pp. 35–53). Springer.

Affiliations

Rafael H. Bordini[1] · Amal El Fallah Seghrouchni[2] · Koen Hindriks[3] · Brian Logan[4] · Alessandro Ricci[5]

Rafael H. Bordini
rafael.bordini@pucrs.br

Amal El Fallah Seghrouchni
Amal.Elfallah@lip6.fr

Koen Hindriks
k.v.hindriks@vu.nl

Alessandro Ricci
a.ricci@unibo.it

1 School of Technology, PUCRS, Porto Alegre, RS, Brazil

2 LIP6 UMR 7606 CNRS, Sorbonne Université, Paris, France

3 Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

4 School of Computer Science, University of Nottingham, Nottingham, UK

5 Department of Computer Science and Engineering (DISI), University of Bologna, Cesena, Italy