An Agent Programming Manifesto

Brian Logan

School of Computer Science
University of Nottingham, UK
E-mail: bsl@cs.nott.ac.uk

Abstract: There has been considerable progress in both the theory and practice of agent programming since Georgeff & Rao's seminal work on the Belief-Desire-Intention paradigm. However, despite increasing interest in the development of autonomous systems, applications of agent programming are confined to a small number of niche areas, and adoption of agent programming languages in mainstream software development remains limited. This state of affairs is widely acknowledged within the community, and a number of reasons and remedies have been proposed. In this paper, I present an analysis of why agent programming has failed to make an impact that is rooted in the class of programming problems agent programming sets out to solve, namely the realisation of flexible intelligent behaviour in dynamic and unpredictable environments. Based on this analysis, I outline some suggestions for the future direction of agent programming, and some principles that I believe any successful future direction must follow.

1 Introduction

There is increasing interest in the application of autonomous intelligent systems technology in areas such as driverless cars, UAVs, manufacturing, healthcare, personal assistants, etc. Robotics and autonomous systems have been identified as one of the *Eight Great Technologies* [49] with the potential to revolutionise our economy and society. There is also an increasing focus on autonomous systems in artificial intelligence research, with, for example, special tracks on Cognitive Systems and Integrated Systems/Integrated AI Capabilities at AAAI 2015 & 2016. Given the level of interest in both industry and academia, one might expect the agent programming community, which specialises in theories, architectures, languages and tools for the development of intelligent autonomous systems to be thriving, and the languages and tools they have developed to support the development of intelligent autonomous agents to be in widespread use in AI research and perhaps in industry.

However the impact of agent programming in both mainstream AI and in applications is minimal. Surveys suggest that the adoption of Agent-Oriented Programming Languages (AOPL) and Agent-Oriented Software Engineering (AOSE) in both research and industry is limited [14, 52, 27]. Moreover, the most distinctive and mature outputs of the agent programming community, the Belief-Desire-Intention (BDI)-based approaches which specifically target the development of intelligent or cognitive agents and which would appear to be best suited to the development of autonomous systems, are least used. One could

argue that the current relatively low take-up of agent programming is not necessarily a cause for concern; there is often a lag in the adoption of new programming paradigms and languages. For example, it took nearly twenty years for object-oriented languages to be adopted in mainstream development. However I believe such a view of the adoption of agent programming languages and methodologies is too sanguine. Adoption of object-oriented languages was driven by the emergence of new kinds of applications that were difficult to program in languages such as C (e.g., GUI-based applications). The corresponding 'forcing' applications for agent programming (e.g., control of simple autonomous vehicles, high-level control in cognitive robotics), are already being developed, and not in agent programming languages.

In this paper I explore the reasons for the apparent lack of interest in agent programming in the broader AI research community and developers of agent-based systems, and make some proposals about what the agent programming community can (and should) do about it. By 'agent programming' I mean the whole spectrum of agent programming techniques developed to support the implementation of autonomous agents, from more 'object oriented' approaches such as JADE [2] to BDI-based approaches such as Jason [5]. I focus on models, languages and platforms for programming individual agents, as these are most relevant to the implementation of autonomous systems.[1] I will use the term 'agent programming community' to refer to the developers of these languages and tools (exemplified by the Engineering Multi-Agent Systems (EMAS) workshop and its predecessors) rather than their intended users. Except where the distinction is relevant, in the interests of brevity, I often do not distinguish between agent programming languages and the theories on which a language is based or the platform implementing the language specification, and use 'agent programming language' (APL) as a general term to denote the outputs of the agent programming community. My analysis of why agent programming is failing to have an impact applies to all forms of agent programming, as the capabilities necessary to realise intelligent autonomous behaviour (enumerated in Section 4 below) are independent of the approach used. However my proposals focus primarily on BDI-based approaches.

The analysis and proposals together can be seen as a 'manifesto' (in the sense of a declaration of aims and policy) for agent programming. Like all manifestos, it embodies a set of assumptions and values which may not necessarily be shared by all in the agent programming community. However the fact that agent programming is failing to have an impact seems unarguable, and, if nothing else, I hope this paper will contribute to the ongoing discussion of the future direction of agent programming as an area of research.

This paper elaborates on themes sketched in [26]. However it offers a deeper analysis of why extending the feature set of current BDI-based APLs is necessary for wider adoption of agent technology, and the proposals for how these features can be incorporated within an extended BDI architecture are more fully fleshed out (though still not complete). The remainder of the paper is organised as follows. In Section 2, I briefly review previous work on the impact of APLs on AI generally and the development of applications, and discuss previous proposals to increase the adoption of APLs in Section 3. In Section 4, I define the problem the problem that agent programming is trying to solve. In Section 5, I summarise the current state of the art in BDI-based agent programming languages, and highlight some of the key limitations of the current 'BDI core feature set'. In Section 6, I briefly sketch some of the developments in AI in CS generally that will impact the adoption and future development of agent programming languages. In Section 7, I present some proposals for future research in agent programming, before concluding in Section 8.

2 Current Adoption of APLs

In [14], Dignum & Dignum report the results of an on-line survey of the use of agent languages and frameworks in the implementation of industrial agent-based systems. The survey was conducted in 2008, and respondents were asked to provide information about the type of agent-based application, the key characteristics of the application domain, and whether an agent programming language or methodology had been used in developing the system. They received 25 responses. The majority of responses related to academic projects with industrial prototypes or deployments (84%); only 8% of responses were from industry.[2] An analysis of the data revealed that only 8% of projects had resulted in large-scale deployments, with most (56%) resulting in a prototype or lab demonstrator. 40% of projects made no use of an agent programming language; in the remaining 60%, the most commonly used APL was Jade/Jadex (20%), and one project used JACK. Other BDI-based languages such as Jason or 2APL were not used. Even if all the Jade/Jadex projects used Jadex, this suggests that at most 24% of projects used some kind of BDI-based APL. The most common application domains were logistics (24% of projects) and crisis management (12%). In an attempt to determine the extent to which the results of their survey reflected the state of practice, Dignum & Dignum also analysed the papers appearing in the industrial applications track of the main agents conference, the International Conference on Autonomous Agents and Multiagent Systems (AAMAS) in 2007 and 2008. Of a total of 29 papers, 24% clearly described real-world applications that went further than proof of concept or a prototype, and in 24% MAS implementation platforms were used. While this analysis suggests that the proportion of projects reaching industrial deployment may be higher than indicated by the survey results, the number of projects using agent programming languages is broadly consistent with the survey.

In [52], Winikoff presents an analysis of applications appearing in the AAMAS Industry/Innovative Applications tracks in 2010 and 2012. The results are broadly similar to those reported in [14], in that most of the systems described do not require intelligent goal-based (BDI) agents, and the focus of many applications is at the multiagent system (MAS) coordination level (e.g., game theory, MDPs). Moreover, the use of agent programming languages and agent-oriented software engineering methodologies in the papers surveyed was limited. Winikoff also considers industrial involvement in the AAMAS conference over the period 2007–2012. If industrial involvement were high, it is perhaps more plausible that the low adoption of APLs in industry may be attributed to deficiencies in the languages and platforms themselves. However the data suggest that industrial involvement in AAMAS declined over the period. Low adoption may therefore be simply a consequence of agent-based solutions being perceived as irrelevant to industrial problems. Given the increased interest in autonomous systems, this seems surprising; however it is consistent with the analysis I present in Section 4.

The most recent survey is that by Müller & Fischer in 2014 [27]. They identified 46 ‘mature’ applications (out of 152 surveyed applications), 55 ‘industry validated research prototypes’ and 46 ‘research prototypes’. They found that:

- 82% of all surveyed applications focus on the MAS level, while only 9% focus on ‘intelligent agents’;
- the majority of mature applications are concentrated in a few industrial sectors: logistics & manufacturing, telecoms, aerospace and e-commerce; and

- only 10% of mature applications clearly used a BDI-based platform; of those that did, all used the JACK [50] platform.[3]

They conclude that while there are a considerable number of deployed agent-based systems (some of considerable longevity), the adoption of multiagent systems and technologies is mostly limited to niche markets. Müller and Fisher list a number of caveats concerning their study data. In particular, "the large number of applications in the multi-agent systems category certainly reflects the focus towards multi-agent topics in the call for participation rather than a lack of intelligent agent[s]". In addition, they note that some applications used more than one platform, and for some applications the information was not available, so the number of mature applications using a BDI platform may be higher than 10%. However, even allowing for these factors, from their results it seems hard to argue that BDI agents are having a significant impact in application development.

The survey by Müller & Fischer is useful in establishing that agent and multiagent systems and technologies are being applied in real applications (though even allowing for the time required to translate research ideas into practice [28], one might question whether 46 is a "considerable number"). Otherwise the broad picture is similar to that reported by Dignum & Dignum and Winikoff, namely that relatively few deployed systems involve 'intelligent agents'. An obvious question then arises: why do so few fielded applications involve the use of intelligent agents? Before returning to this question in Section 5, in the next section I briefly review previous proposals for how the adoption of APLs can be increased.

3 Previous Proposals to Increase the Adoption of APLs

The state of affairs reported by Müller & Fischer has been recognised for some time within the agent programming community, and there have been a number of proposals as to how to increase the take-up of agent programming in mainstream software development and how agent programming languages should evolve in order to maximise their adoption. These proposals reflect and draw on a larger discussion within the community, in the form of talks and panel sessions at agent programming workshops, including the Dagstuhl Seminar on Engineering Multi-Agent Systems [15] and the EMAS 2013 & 2014 workshops. In this section I briefly review some of the proposals that have emerged from this discussion.

In [52] Winikoff draws on his analysis of the adoption of APLs (discussed in the previous section) to identify a number of challenges for engineering multiagent systems and propose directions for future work in engineering MAS. He focusses on the relevance of the AAMAS community to industry, the relevance of the 'agent engineering' sub-community to the rest of AAMAS, and, in particular, the extent to which the methodologies, languages and tools developed by this sub-community are used both in the wider AAMAS community and in industry. I focus here on the latter question, as being more closely related to the topic of the current paper.[4] He makes five recommendations, of which two are particularly relevant here:

- stop designing AOPLs and AOSE methodologies . . . and instead . . .
- move to the "macro" level: develop techniques for designing and implementing interaction, integrate micro (single cognitive agent) and macro (MAS) design and implementation

However this proposed change of focus appears to ignore the possibility that the reason current applications focus on coordination of MAS rather than intelligent agents, is that the support provided by current agent programming languages for developing intelligent agents is inadequate. As such, it risks becoming a self-fulfilling prophecy. I will return to this point in the next section. Winikoff also identifies the lack of techniques for the assurance of MAS as a barrier to adoption of agent technology. This is a valid concern, but falls outside the scope of the current paper, which is more narrowly focussed on the role and design of agent programming languages.

In [21] Hindriks reviews the history of engineering multiagent systems, including agent programming, and presents a vision of how 'cognitive agent technology' can form a basis for the development of next-generation autonomous decision-making systems. Like Winikoff, he makes a number recommendations and identifies a number of directions for future research, including:

- pay more attention to the kind of support (specifically tools) required for engineering MAS applications
- focus more on issues related to ease of use, scalability and performance, and testing
- facilitate the integration of sophisticated AI techniques into agents
- show that agent-orientation can solve key concurrency and distributed computing issues
- put more effort into integrating agent-based methodologies and programming languages

He concludes that to stimulate the adoption of cognitive agent technology and MAS, the agent programming community must provide "methods and tools that jointly support the agent-oriented mindset". However Hindriks's analysis does not directly address the causes of the low take-up of agent programming in mainstream software development. Rather his proposals can be read as possible or desirable extensions to current APLs rather than features necessary for wider adoption.[5]

Many of the features identified by Winikoff and Hindriks are clearly important for the wider adoption of agent programming languages. However I believe their analyses fundamentally mistake the nature of the problem faced by the agent programming community. The key problem lies elsewhere, and has not previously been articulated. I turn to this in the next section.

4 The Problem Agent Programming is Trying to Solve

To understand what it might mean for agent programming to have an impact, we must first define the problem that agent programming is trying to solve.

There are many different views of the aims and objectives of 'agent programming' considered as a field. As a first approximation, these differing perspectives can be broadly characterised as being either 'AI-oriented' or 'software engineering-oriented'. The AI-oriented view focuses on connections with the broader field of artificial intelligence, and sees agents as 'an overarching framework for bringing together the component AI sub-disciplines that are necessary to design and build intelligent entities' [24]. The software engineering-oriented view on the other hand, focuses on synergies between software engineering and agent research.[6] Each tradition is associated with its own set of research questions and academic workshops. For example the AI-oriented view is represented by workshops such

as Agent Architectures Theories and Languages (ATAL), while the software engineering-oriented view is represented by workshops such as Agent-Oriented Software Engineering (AOSE). The call for papers for the first ATAL workshop, held in 1994, states:

> Artificial Intelligence is concerned with building, modeling and understanding systems that exhibit some aspect of intelligent behaviour. Yet it is only comparatively recently — since about the mid 1980s — that issues surrounding the synthesis of intelligent autonomous agents have entered the mainstream of AI. . . . The aim of this workshop . . . is to provide an arena in which researchers working in all areas related to the theoretical and practical aspects of both hardware and software agent synthesis can further extend their understanding and expertise by meeting and exchanging ideas, techniques and results with researchers working in related areas.
>
> — ATAL 1994 CfP

In contrast, the call for papers for the first AOSE workshop, held in 2000, focuses on synergies with software engineering research, and states:

> 'Since the 1980s, software agents and multi-agent systems have grown into what is now one of the most active areas of research and development activity in computing generally. There are many reasons for the current intensity of interest, but certainly one of the most important is that the concept of an agent as an autonomous system, capable of interacting with other agents in order to satisfy its design objectives, is a natural one for software designers. Just as we can understand many systems as being composed of essentially passive objects, which have state, and upon which we can perform operations, so we can understand many others as being made up of interacting, semi-autonomous agents.
>
> This recognition has led to the growth of interest in agents as a new paradigm for software engineering. In this workshop . . . we will seek to examine the credentials of agent-based approaches as a software engineering paradigm, and to gain an insight into what agent-oriented software engineering will look like.'
>
> — AOSE 2000 CfP

The differences in emphasis between the two views of agent programming is clearly reflected in the research focus of the two workshops. In the AI-oriented view, the focus is on building systems that exhibit intelligent autonomous behaviour, while in the software engineering-oriented view, (semi)autonomous agents are seen as a way of conceptualising complex software systems containing many dynamically interacting components, each with their own thread of control and engaging in complex coordination protocols.

In what follows, I focus on the AI-oriented view. There are several reasons for this choice. The AI-oriented view represents the original motivation for agent programming as a subfield, and I would argue that the most significant contributions of agent programming to the broader AAMAS community have emerged from this tradition, e.g., the 2007 IFAAMAS Influential Paper Award for Rao & Georgeff's work on rational agents [31]. In addition, the agent programming languages and tools developed in this tradition are arguably the most mature software products of the agent programming community, representing approximately thirty years of cumulative development. Lastly, the combination of these two factors (a clear need in the AI community to develop intelligent autonomous systems, and a distinctive set of ideas about how to implement intelligent autonomous agents in the form of the BDI paradigm) offers the best hope for agent programming to have a broader impact.

In the AI-oriented view, agents are a way of realising the broader aims of artificial intelligence. Agents are intelligent autonomous systems which combine multiple capabilities, e.g., sensing, problem-solving and action, in a single system. Agent programming is seen as a means of realising and integrating these capabilities to achieve flexible intelligent behaviour in dynamic and unpredictable environments. This view rests

on the belief that there is commonality, at least at the architecture level, between agents in widely differing domains, such as autonomous vehicles, healthcare, training simulations, entertainment, etc. The question then is what might this commonality consist in?

To answer this question, we must first define what it means to 'achieve flexible intelligent behaviour in dynamic and unpredictable environments'. There are many definitions of intelligent behaviour; however, key to most is consideration of both *ends* (goals) and *means* (plans) when choosing which course(s) of action to pursue. As a working definition, we will say that *an agent is intelligent to the extent that the set of means to which it commits at any given time, and the ways those means are progressed, approximates an optimal progression of an optimal set of means given the goals of the agent and the plans it has to achieve them*.[7] What counts as 'optimal' is clearly dependent on the particular domain or application of interest. Different ends will have differing characteristics (e.g., the utility resulting from achieving the end), and different means will likewise have different characteristics (e.g., time, cost and resources required by the means). The particular tradeoff between the set of means adopted (and how they are progressed) and the ends achieved depends on the domain; for example, whether the use of higher cost means is justified in achieving an additional end. However such domain specific details should not obscure the key interaction between means and ends in intelligent behaviour.

This view of intelligence can be traced back at least as far as Bratman [7, 6] and the early work on the Procedural Reasoning System (PRS) [18], and underpins much of the subsequent work in areas such as rational action selection, e.g., [34], intention reconsideration, e.g., [54], goal lifecycles, e.g., [39], etc. It also forms the basis of much of the work in the BDI paradigm (reviewed in the next section); however the core problems are independent of their characterisation in terms of beliefs, desires and intentions. Rather it implies a set of key capabilities of an intelligent agent (however realised), and hence any agent programming language that aims to support the development of such agents. These capabilities include being able to take into account the possible ends to which an agent might commit, the ends to which it is currently committed, the intended means of achieving these ends, and its available means, when deciding what courses of action to follow [6]. In BDI terminology, this amounts to the ability to deliberate over desires (in the sense of potential goals), goals, intentions and plans, and reason about the interactions between them. Note that this is a stronger requirement than simply selecting plans rationally (unless the agent's beliefs extend to beliefs about its available means, the courses of action to which it is currently committed, etc.). An agent may select a course of action to achieve an end that conflicts with its currently intended means and still be rational in the sense of [31], even if it has alternative means of achieving the end that do not conflict with its current intentions.

For example, consider an agent that is committed to a set of ends and a corresponding set of intended means, and which must decide whether to commit to a new end, and if so, how to achieve it. Making an intelligent decision in such a situation involves consideration of the means available to achieve the end, and whether they conflict with the means to which the agent is currently committed. Whether a set of intentions conflict may in turn depend on the way in which they are progressed; some execution orders may give rise to conflicts, while others may not. If a conflict is inevitable, the agent must decide whether the conflict can be resolved by suspending progression of one or more intended means, whether to achieve an end to which it is committed in another way, or whether to abandon one or more of the ends to which it is currently committed (or the new end). In some circumstances, deciding whether to commit to a new end may also depend on anticipated or possible ends that may arise in future. For example, a taxi agent may commit to one fare (in the centre of town) and

not another (in the suburbs), in the expectation that there will be more fares in the centre of town in the future.

In the next section, I examine the extent to which state of the art agent programming languages, and in particular BDI-based APLs, support these key capabilities necessary to realise intelligent behaviour.

5 The State of the Art and its Limitations

The Belief-Desire-Intention (BDI) model [6] and its underlying theoretical underpinnings [31] are arguably 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 BDI-based agent programming languages, the behaviour of an agent is specified in terms of beliefs, goals, and plans. *Beliefs* represent the agent's information about the environment (and itself). *Goals* represent desired states of the environment the agent is trying to bring about. *Plans* are the means by which the agent can modify the environment in order to achieve its goals. A plan consists 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 steps which are either basic actions that directly change the agent's environment or subgoals which are in turn achieved by other plans. Plans are pre-defined by the agent developer, and, together with the agent's initial beliefs and goals, form the program of the agent.

For each event (belief change or top-level goal), the agent selects a plan which forms the root of an *intention* and commences executing the steps in the plan. If the next step in an intention is a subgoal, a (sub)plan is selected to achieve the subgoal and added to the intention. In most BDI-based agent programming languages, plan selection follows four 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 belief context evaluates to true, given the agent's current beliefs. Third, the agent commits to (*intends*) one or more of its relevant, applicable plans. Finally, from this updated set of intentions, the agent then selects one or more intentions, and *executes* one (or more) steps of the plan for that intention. This process of repeatedly choosing and executing plans is referred to the agent's *deliberation cycle* [9] or *agent reasoning cycle* [5]. Deferring the selection plans until the corresponding goal must be achieved allows BDI agents to respond flexibly to changes in the environment, by adapting the means used to achieve a goal to the current circumstances.

The BDI approach has been very successful, to the extent that it is arguably the dominant paradigm in agent programming [19], and a wide variety of agent languages and platforms have been developed that at least partially implement the BDI model. A number of these languages and platforms are now reasonably mature in terms of their feature set (if not always in terms of their software engineering) e.g., [50, 8, 5, 10, 20]. They encompass each of the components of the BDI model in at least rudimentary 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. It is therefore appropriate to consider what the *scientific contribution* of this work consists of.

5.1 The BDI Core Feature Set

The features common to state of the art BDI 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 (perhaps after performing some 'cleanup' actions), and trying another applicable plan to achieve the current subgoal; if there are no applicable plans which have not previously been tried for the current subgoal, failure is propagated to higher-level motivating goals.

These features constitute the 'core feature set' of BDI-based agent programming, in the sense that they are found in nearly all BDI agent platforms, and there is a fair degree of agreement about their semantics. Other aspects either differ from platform to platform, e.g., the order in which intentions are executed (though the operational semantics typically assumes execution is nondeterministic), or must be implemented by the developer using low-level platform-specific features, often in the host language.

As might be expected of mature software platforms, the key theoretical ideas underpinning these core features were first proposed some time ago. The approach to plan selection based on triggering events and context conditions dates back to PRS [18] (where they are called cues and preconditions), and, in the form in which they appear in most current BDI-based APLs, to AgentSpeak(L) [32]. The backtracking approach to plan failure, in which the plan in which the failure occurred is aborted and another plan selected for the current subgoal, was introduced in [53].

While these features are useful, and are key to implementing agents based on the BDI paradigm, the programs that can be written using (only) the core feature set is restricted to those that realise behaviours where: the selection of applicable means is determined purely by the agent's beliefs about the current state of the environment (and is otherwise non-deterministic); the means available to achieve the possible ends don't conflict (since the execution of intentions may be arbitrarily interleaved, actions in different plans must not conflict); the agent always commits to new ends (since means are assumed not to conflict, the only reason an agent does not adopt a new intention in response to an event is a lack of an applicable plan); and where goals are never dropped until all available plans have been tried (i.e., the agent is blindly committed to its goals).

As a result, the set of behaviours that can be programmed 'natively' using the core feature set is a only small subset of those identified in Section 5 as required for intelligent autonomous behaviour. Moreover, such relatively simple behaviours can often be programmed without using a BDI-based agent programming language. While a BDI-based platform may facilitate development, any saving in development time may be more than offset by the time required to learn how to program in the 'BDI style'.[8] In contrast, it can be argued that the MAS-level coordination techniques that form the basis of most applications discussed in [14, 52, 27], e.g., auctions, economic approaches, game theory etc., offer more significant advantages (often supported by theoretical guarantees) compared

to conventional distributed systems approaches. In addition, the algorithms involved can typically be programmed in any language, allowing easier integration into mainstream software development practices.

5.2 Limitations of Current BDI Languages and Platforms

In the remainder of this section, I briefly highlight some of the key capabilities of an intelligent autonomous agent that can't easily be programmed in the current generation of agent programming languages, and review the progress that has been made towards implementing such capabilities in agent platforms and in the literature. Critically, the features identified below are much harder to program from scratch in an imperative style or using the alternative, mainstream programming techniques discussed in Section 6.2. Core support for such capabilities would therefore give BDI languages and platforms a clear competitive edge in the development of intelligent autonomous agents.

At each deliberation cycle, an intelligent autonomous 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 and often non-existent. Moreover, current BDI languages typically lack the basic underlying representations for costs, preferences, time, resources, durative actions, etc. necessary to implement such capabilities. Of course, some aspects of these capabilbites can be programmed in current BDI-based agent languages. Most platforms provide 'hooks', and/or primitive constructs and operations that allow a developer to control which plan is adopted, which intention is scheduled for execution at the current cycle, etc. For example, the $\mathcal{S}_{\mathcal{E}}$, $\mathcal{S}_{\mathcal{O}}$ and $\mathcal{S}_{\mathcal{I}}$ selection functions of Jason [5] allow a developer to customise Jason's event, plan and intention selection for a particular application domain; conflicts between intentions can be avoided by using *atomic* constructs available in languages such as Jason and 2APL [10] that prevent the interleaving of actions in one plan with actions from other plans; and both Jason and 2APL provide primitive actions to drop goals (and in 2APL, to adopt them). However use of these non-core features requires either programming in another language (e.g., the $\mathcal{S}_{\mathcal{E}}$, $\mathcal{S}_{\mathcal{O}}$ and $\mathcal{S}_{\mathcal{I}}$ functions must be programmed in Java), or requires complex, low-level bookkeeping by the developer (e.g., neither Jason nor 2APL provide primitives

to return the current intentions of an agent, or the plans selected for each (sub)goal).[9] As a result, the most difficult parts of developing an intelligent autonomous agent must be programmed using conventional programming techniques. It is therefore perhaps not surprising that languages and platforms which provide little or no support for the hardest (and arguably the most central or characteristic) aspects of agent programming have not been widely adopted.

There has been preliminary work on how support for some of the features listed above can be integrated into BDI-based APLs, see, for example, [4, 42, 35, 46, 36, 38, 43, 37, 29, 56, 40]. (I discuss some of this work in more detail in Section 7.) However, to date, this work has not been incorporated into the core feature set of popular BDI platforms.

One possible explanation for the current state of the art is that the ways in which decisions regarding which plan to adopt, which intention to execute next etc., are too variable to be easily expressed within the BDI framework, and are best left to the developer. The argument that such issues should or must be left to the programmer is reminiscent of the 'New Jersey approach' [17]: do the basic cases well, and leave the programmer to do the hard bits. While this approach may explain the relative popularity of C vs Lisp (the focus of Gabriel's paper), the assumption that the current 'BDI core feature set' is a good tradeoff in terms of the kinds of behaviours that can be easily programmed while at the same time being easy for programmers to learn doesn't seem to hold.[10] It is of course possible that, while the general problem of selecting which means to adopt for a particular end given the agent's beliefs can be reduced to a single programmatic approach (encompassed in the current core feature set), decisions regarding which ends to adopt and/or how an agent's intentions should be progressed cannot. However, if there are *no* general theories or approaches to these questions, developing agents capable of flexible intelligent behaviour in dynamic and unpredictable environments is going to be very hard (and hence very expensive), and the amount that agent programming can contribute to realising intelligent behaviour will be limited.

The argument that these problems cannot or should not be addressed within (some suitable extension) of the BDI architecture, is not, I believe, a tenable position. While the support currently offered by state of the art APLs is useful, particularly for some problems, the set of behaviours that can be programmed is too small relative to the requirements set out in Section 4 for most mainstream developers to switch to BDI-based platforms. It therefore follows that for agent programming to have a broader impact on mainstream development and AI research generally, these problems must be addressed.

6 The Broader Context

The analysis presented in the previous section actually underestimates the scale of the challenge facing the agent programming community. In this section, I argue that there is good reason to suppose that the already limited impact of agent programming on commercial/industrial agent development and AI generally is likely to decline in the foreseeable future due to changes in the broader AI and CS context. Key assumptions on which the BDI agent programming model are based are not as true as they once were, allowing other AI subfields to colonise the APL space. In addition, some mainstream computing paradigms are starting to look somewhat like simple forms of agent programming, potentially further limiting the impact of APL technologies on mainstream development. I discuss each of these developments below.

6.1 Reactive Planning

The BDI approach to agent programming is based on early work on reactive planning, e.g., [18]. The underlying rationale for reactive planning rests on a number of key assumptions, including:

- the environment is dynamic, so it's not worth planning too far ahead as the environment will change; and
- the choice of plans should be deferred for as long as possible — plans should be selected based on the context in which the plan will be executed.

In their 1987 paper Georgeff & Lansky emphasise the difference between reactive planning and traditional (first principles) planning: "[traditional] planning techniques [are] not suited to domains where replanning is frequently necessary" [18].

A key implicit assumption underlying this claim is the time required by a traditional planner to find a plan for a given goal. While generative planning remains a PSPACE problem, advances in classical planning and increases in processing power have increased the size of problems that can be solved by a traditional planner in a given amount of time. Since Georgeff & Lansky's paper, available computational power has increased by a factor of approximately 10^5, and by a factor of 10^4 since Rao's classic paper on AgentSpeak(L) [32] which influenced the design of many current state of the art agent programming languages. It is therefore perhaps now time to reconsider whether traditional planning techniques are unsuited to domains where replanning is frequently necessary.

Generative planners can now solve a significant number of problem instances from the ICAPS (International Conference on Automated Planning and Scheduling) International Planning Competition[11] in less than a second. The best planners are capable of solving over half the problem instances in less than 100ms [33], which could be taken as the threshold necessary for agent planning in (soft) real time domains. (Note that these results are from 2012.)

While there is some dispute about the extent to which Moore's law continues to hold, it seems safe to assume that available computational power will continue to increase, at least in terms of transistor count, for the foreseeable future. It also seems safe to assume that advances in classical planning will continue. Coupled with an increased interest in the planning community in 'real time' planning,[12] it seems likely that the range of problems amenable to first principles planning will increase in the future. This does not mean the end of reactive planning, but hybrid approaches will become increasingly feasible.

6.2 Reactive Programming

At the same time, work on event-driven and reactive programming[13] (e.g., in robotics) offers similar (or better) functionality to belief triggered plans in agent programming. In reactive programming, programs are defined in terms of queries over streams of events and their composition. Compared to the event queue/belief update and triggering of relevant applicable plans found in most current BDI-based agent programming languages and platforms, reactive programming-based approaches have a number of advantages including:

- a well defined model of streams (immutability, sampling, pull-based computation);
- very fast (microsecond) evaluation of simple SQL-like queries (e.g., LINQ,[14] cqengine[15]) that scale to very large 'belief bases' for evaluation of context conditions.

If belief updates and (sub)goals are seen as a stream of events in the reactive programming sense, such programming techniques can be used to implement key aspects of the core BDI model outlined in Section 5, e.g., the selection of pre-defined plans at run time based on a triggering event and the agent's current beliefs.[16] Moreover, these paradigms are increasingly a part of 'mainstream' Computer Science, (e.g., 'Event Driven and Reactive Programming' is included in the 2013 ACM model curriculum for Computer Science [1]) and are being deployed in large scale applications.

6.3 Implications for Agent Programming

The developments discussed above do not constitute a comprehensive list of the changes impacting or likely to impact the agent programming community. It is possible to point to similar advances in other subfields of both AI (such as reasoning and scheduling) and CS relevant to agent programming. Together their effect is to erode the niche currently occupied by the current generation of agent programming languages. It follows that agent programming as a discipline will only remain relevant if it is possible to increase the size of the niche it occupies. To do so, agent programming languages must become capable of addressing a wider range of problems in a generic way. The good news is that the same developments which pose a threat to the future of agent programming can enable this transition, as I explain in the next section.

7 The Future

In Section 4, I argued that implementing intelligent autonomous behaviour requires deliberation over desires (in the sense of potential goals), goals, intentions, plans, and the interactions between them. In Section 5 we saw that current BDI-based languages support only a relatively small subset of these capabilities. For example, most BDI-based languages provide little or no support for reasoning about whether a potential goal should be adopted. When updating the agent's currently intended means and determining how those means should be progressed, the standard BDI deliberation cycle considers only the relevant applicable means for the current event, given the agent's current beliefs. Similarly, when determining which intention to progress at the current cycle, possible interactions between intentions are ignored, and most current BDI-based languages by default adopt a simple 'round robin' approach to scheduling intentions.

In this section, I explore how these missing capabilities might be incorporated within the BDI model. I first consider adding support for deliberating over desires and intentions within an extended BDI architecture, i.e., where beliefs are first order formulas and plans are predefined by a human developer. I then consider more speculative proposals that relax the assumptions regarding declarative and procedural knowledge in the standard BDI model.

7.1 Extending the BDI Architecture

Deliberation over desires and the interactions between them cannot be implemented in terms of the current BDI core feature set in the way that, e.g., deliberative goals can be implemented in terms of procedural goals (goals to do) through the use of design patterns [23]. 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.[17] Even if it were possible to gain access to the agent's current intentions, there is no easy way to write BDI code to choose between different plans for a goal based on its current intentions. For example, assume we were to extend the syntax of plans to include a "intention context" that specifies whether a relevant, applicable plan should be added to the agent's intentions, given the agent's current intentions. The agent developer could define a predicate, e.g., `intend-p()`, that evaluates to true if the agent should commit to the current plan given its other intentions. While this would have the effect of moving part of the definition of the deliberation cycle "into" the APL, such an approach violates the principle of modularity of agent plans. The plan trigger specifies the kinds of events a plan can be used to respond to. The context specifies minimal conditions that must hold for it to be applicable. For any given plan, this information can be determined without reference to other plans. This simplifies development, allowing the definition of plans to achieve particular goals or respond to particular events without reference to the other plans of the agent. However adding a new plan to the agent program now entails modifying the definition of `intend-p()` to specify the sets of circumstances in which instances of the new plan type should be intended; for example, the cases in which it should be preferred to other plans for achieving the same goal. Similarly, there is no way to write BDI code to decide which of the agent's current intentions to execute at the current cycle.

Implementing the capabilities necessary to realise flexible intelligent behaviour in dynamic and unpredictable environments thus involves *extending* the standard BDI architecture underlying most state of the art agent programming platforms with facilities to represent and reason about desires, plans, and intentions and their progression. This is not a new idea. Starting with PRS, a range of 'extended' BDI architectures have been proposed. (It is slightly misleading to call these 'extended BDI architectures'; as noted in Section 5, the original accounts of the Belief–Desire–Intention approach included many of the capabilities proposed here.) In the remainder of this subsection, I briefly review this work, and identify a set of requirements for such an extended architecture.

PRS [18] incorporated a meta-level, and supported reflection [13] of some aspects of the state of the execution of the agent, such as the set of applicable plans, allowing an agent developer to program deliberation about the choice of plans in the same language used to represent object-level plans. In [9] Dastani et al. proposed extending 3APL with a set of programming constructs which allow the agent's deliberation cycle to be programmed in terms of a small set of primitives. The language is imperative and extends that proposed in [22], mainly by adding a primitive that invokes a planner to generate a new plan. Plans can be compared on their cost and the gain resulting from achieving the corresponding goal, and a plan for which the cost is less than gain is selected. Later work on 3APL took a different approach, in allowing the developer to modify the 3APL deliberation cycle by reprogramming the 3APL interpreter in the interpreter host language. In [11] Dastani et al. define a collection of Java classes for each mental attitude, where each class has a collection of methods representing operations for manipulating the attitude. In order to implement a particular deliberation cycle, the programmer must essentially modify the interpreter to call the methods of these Java classes in a particular order. In [30] Pokahr et al present an 'Easy Deliberation' strategy, which aims to provide an intuitive interface for programming deliberation at the agent-programming level. The strategy allows an agent to reason about the conflicts and dependencies between their goals, and so select an appropriate set of active goals for the current situation. An extended Jadex interpreter is given, augmented with meta-actions for (de)activating goals. However consideration of conflicts is restricted to the goal

level, and does not take into account the plans used to achieve the goals. In [51] Winikoff presents a meta-interpreter for AgentSpeak programs written in AgentSpeak. He shows how the meta-interpreter can be used to define alternative approaches to failure handling, and to provide control over plan selection. In [25], Leask et al. show how procedural reflection in the agent programming language meta-APL [16] can be used by an agent programmer to customise the deliberation cycle of a BDI agent to control when to deliberate, which relevant applicable plan(s) to intend, and which intention(s) to execute. Like Winikoff, and in contrast to Dastani et al., their approach allows object-level plans and agent's deliberation cycle to expressed in the same language. However their use of procedural reflection allows a broader range of capabilities to be programmed, including deliberation over desires, plans, and intentions, and the scheduling of intention execution to avoid conflicts between actions.

There has also been a considerable amount of work on basic capabilities to support deliberation about which plan to adopt for a goal, and whether a set of intentions can be progressed without conflicts, etc. For example, Thangarajah et al. [42, 41] have proposed an approach to detecting and avoiding possible conflicts between intentions 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 (sub)goal will definitely be safe to execute without conflicts, or will definitely result in conflicts, or may result in conflicts. If the goal cannot be executed safely, execution of the intention is deferred. In [47, 48], Waters et al. 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. In [35] Sardiña et al. show how an HTN planner can be integrated into a BDI architecture to find hierarchical decompositions of plans that avoid incorrect decisions at choice points, and so are less likely to fail during execution. Other work has investigated how the execution of a set of intentions can be scheduled so as to avoid conflicts. For example, the TÆMS (Task Analysis, Environment Modelling, and Simulation) framework [12] together with Design-To-Criteria (DTC) scheduling [45] have been used in agent architectures such the Soft Real-Time Agent Architecture [44] and AgentSpeak(XL) [4] to schedule intentions, and in [56, 55] Yao et al. present a stochastic approach to scheduling intentions that avoids both conflicts and maximises fairness in the progression of the intentions for each goal.

The basic capabilities implemented in these approaches differ in their degree of encapsulation, giving rise to a rich space of possible designs for an extended BDI architecture. For example, the approaches based on summary information can be used to determine when progression of an intention must be suspended, leaving decisions about how the remaining intentions should be progressed to other deliberation, while in [55], all scheduling decisions are encapsulated in a single process.

I believe the key requirements for such an extended architecture include a clear operational semantics, as in e.g., [51, 25] (so preserving the key link between theory and practice found in previous BDI-based approaches), and a uniform approach to programming all aspects of agent behaviour, as in [25]. 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, and moreover facilitates a modular, incremental approach to the development of deliberation strategies. It is also desirable that the implementation of such an extended architecture should take advantage of recent work in scalable query evaluation described in Section 6.2. However, the focus must be on extending the capabilities of agents to allow them to

deliberate over both means and ends, rather than efficiency issues per se. There seems little benefit in, e.g., reimplementing a standard BDI architecture using reactive programming techniques. The key problem with existing agent platforms is not that they run too slowly (though in some cases they may do so), but that the behaviours that can be easily programmed are too limited to make them worth adopting by developers outside the agent programming community.

7.2 The Longer Term

The advances in both hardware and related AI sub-disciplines highlighted in Section 6 mean we are now in a position where we can rethink the foundations of agent programming languages. In doing so we can address some of the key challenges in agent development that have been largely ignored for the last twenty years. Note that this is not just 'more of the same'—the rest of AI has moved on significantly since the early work on the BDI model, creating significant new opportunities that agent programming can exploit.

Below, I briefly sketch one possible path such developments could take. The ideas are shaped by my own interests and are not intended to be exhaustive or prescriptive (for some alternative suggestions, see [21]). However I believe that all feasible futures for agent programming entail a fundamental shift in emphasis: agent programming must become more about describing the problem rather than 'hacking code', with the agent programming language/platform doing (more of) the hard bits currently left to the agent developer.

Beliefs: how and when beliefs are updated (active sensing, lazy update); handling uncertain and inconsistent beliefs.

Goals: goals with priorities and deadlines; maintenance and other repeating goals; when to adopt, suspend and drop goals (cf work on goal life cycles); how to tell if a goal is achieved (e.g., if beliefs are uncertain).

Plans: plans with durative and nondeterministic actions; plans with partially ordered steps; when (and how) to synthesise new plans.

Intentions: how to estimate the time required to execute an intention; which intentions to progress next; how to schedule intentions to avoid interference; how to handle plan failure.

MAS level: how to decide when to join an open system/coalition/team; deliberation about roles and norms; multiagent deliberation about team plans and commitments; strategic reasoning about other agents.

A key feature common to all these possible research directions, is that they involve the APL rather than the agent developer solving a problem.

7.3 What Counts as Progress

Identifying possible research directions is, on its own, insufficient. To count as progress, future research in agent programming must meet a number of criteria that characterise the unique contribution of agent programming (and agent architectures) as a field, distinct from other subfields of multiagent systems, and artificial intelligence and computer science generally. I therefore conclude this section with a set of 'progress metrics' which are rooted in the analysis presented in Section 4:

- extensions need to be integrated, e.g., uncertain and inconsistent beliefs have implications for plan selection and determining when a goal is achieved, plans with nondeterministic actions may determine when sensing is required, etc.;
- ideally, the agent language/platform needs to be modular, so that an agent developer only needs to master the features necessary for their application; and
- the key evaluation criterion should be *whether* a developer has to explicitly program something rather than *how long* it takes them to program it or *how many errors* they make.

The last point is critical. Wherever possible, any extension must aim to make development easier (or at least no more difficult than at present), while increasing the range of agent behaviours. Clearly, the developer will have to write code specific to their particular application. The aim is to raise the level of abstraction offered by the agent programming language, and by doing so address the challenge of integrating the AI sub-disciplines necessary to design and build intelligent autonomous entities.[18] Evaluating agent programs (and indirectly APLs) on this criterion will require richer benchmark problems (or less toy versions of current problems). However such benchmarks are required anyway if we are to demonstrate that APLs can solve problems that can't easily be solved using mainstream programming techniques.

This list of performance metrics is preliminary and can (and should) be improved. However broad consensus around some set of metrics is essential for subfield of agent programming to be coherent as a community, and I would argue that a list something like the above is necessary for our research to have impact in the wider field of multiagent systems.

8 Conclusion

Agent programming has come a long way since since Georgeff & Rao's seminal work on the Belief-Desire-Intention paradigm. BDI has developed into the dominant approach to agent development, and a wide variety of agent languages have been developed based on the BDI model. Many of these languages have a solid theoretical foundation in the form of a precise operational semantics, and several have mature platforms with at least basic tool support.

However in all disciplines, there is a need to pause periodically to take stock, and consider the possible shape of the future. I believe that in agent programming the need to reflect on the future is more than usually pressing. The increasing focus on autonomous systems such as driverless cars, UAVs, manufacturing, healthcare, personal assistants, etc., presents unprecedented opportunities for agent programming, but these opportunities will only be realised if there is a significant change in direction (or at least emphasis) in agent programming research.

This paper represents an attempt to take stock of where we are, what has been achieved, and what key open problems remain. It presents an analysis of the current state of the art in agent programming (and in particular BDI-based agent programming), and its limitations. Based on this analysis, I have outlined some suggestions for the future direction of agent programming, and some principles that I believe any successful future direction must follow.

My analysis of why agent programming is failing to have an impact differs from that in, e.g., [21]. It is probably the case that the lack of more polished methodologies and tools

is a contributory factor in the limited adoption of APLs; however it seems unlikely to be the main one. There are many examples of the adoption of a programming paradigm or language preceding (often by a significant margin) the development of high quality tools. For example, early C++ development environments were significantly less useable than their C equivalents; C++ was adopted because it was easier to develop certain kinds of applications which were difficult to program in C. Rather, I believe the key reason is that there is little incentive for developers to switch to current agent programming languages, as the behaviours that can be easily programmed are sufficiently simple to be implementable in mainstream languages with only a small overhead in coding time (and, as pointed out in Section 6.2, this overhead is about to get significantly smaller). When the costs of transitioning to APLs is also factored in, the barrier to adoption is even higher. As a result, apart from a small number of niche applications, e.g., development of military simulations, the costs outweigh the benefits.

Agent programming therefore isn't (and can't be for a long while) *primarily* about software engineering. Software engineering is important, but only as a means to an end. The agent programming community is primarily a scientific community. It's products are new knowledge about how to achieve flexible intelligent behaviour in dynamic and unpredictable environments, rather than software artefacts or tools. To make progress, we need to focus on solving challenging AI problems in an integrated, general and tractable way. By doing so, I believe we can create theories and languages that are much more powerful and easy to use, and secure a future for agent programming as a discipline.

References

[1] Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science. ACM/IEEE, December 2013.

[2] F.L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[3] Steve S. Benfield, Jim Hendrickson, and Daniel Galanti. Making a strong business case for multiagent technology. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 10–15, Hakodate, Japan, May 2006. ACM.

[4] Rafael Bordini, Ana L. C. Bazzan, Rafael de O. Jannone, Daniel M. Basso, Rosa M. Vicari, and Victor R. Lesser. 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)*, pages 1294–1302, New York, NY, USA, 2002. ACM Press.

[5] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley, 2007.

[6] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[7] Michael E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[8] Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Jadex: A BDI-agent system combining middleware and reasoning. In Rainer Unland, Monique Calisti, and Matthias Klusch, editors, *Software Agent-Based Applications, Platforms and Development Kits*, Whitestein Series in Software Agent Technologies, pages 143–168. BirkhÃ¤user Basel, 2005.

[9] M. Dastani, F. de Boer, F. Dignum, and J.J. Ch. Meyer. 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)*, pages 97–104. ACM, 2003.

[10] Mehdi Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[11] Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules Ch. Meyer. Programming multi-agent systems in 3APL. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 39–67. Springer, 2005.

[12] K. S. Decker and V. R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 2:215–234, 1993.

[13] Jim des Rivières and Brian Cantwell Smith. The implementation of procedurally reflective languages. In *Proceedings of the 1984 ACM Symposium on LISP and functional programming*, pages 331–347, New York, NY, USA, 1984. ACM.

[14] Virginia Dignum and Frank Dignum. Designing agent systems: state of the practice. *International Journal of Agent-Oriented Software Engineering*, 4(3):224–243, 2010.

[15] Jürgen Dix, Koen V. Hindriks, Brian Logan, and Wayne Wobcke. Engineering Multi-Agent Systems (Dagstuhl Seminar 12342). *Dagstuhl Reports*, 2(8):74–98, 2012.

[16] Thu Trang Doan, Yuan Yao, Natasha Alechina, and Brian Logan. Verifying heterogeneous multi-agent programs. In Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns, editors, *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*, pages 149–156, Paris, France, May 2014. IFAAMAS.

[17] Richard P. Gabriel. Lisp: Good news, bad news, how to win big. In *European Conference on the Practical Applications of Lisp*, 1990. (Reprinted in the April 1991 issue of AI Expert magazine).

[18] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, 1987.

[19] Michael P. Georgeff, Barney Pell, Martha E. Pollack, Milind Tambe, and Michael Wooldridge. The belief-desire-intention model of agency. In Jörg P. Müller, Munindar P. Singh, and Anand S. Rao, editors, *Intelligent Agents V, Agent Theories, Architectures, and Languages, 5th International Workshop, (ATAL'98), Paris, France, July 4-7, 1998, Proceedings*, volume 1555 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 1999.

[20] Koen V. Hindriks. Programming rational agents in GOAL. In Amal El Fallah Seghrouchni, Jürgen Dix, Mehdi Dastani, and Rafael H. Bordini, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 119–157. Springer US, 2009.

[21] Koen V. Hindriks. The shaping of the agent-oriented mindset. In Fabiano Dalpiaz, Jürgen Dix, and M. Birna van Riemsdijk, editors, *Engineering Multi-Agent Systems*, volume 8758 of *Lecture Notes in Computer Science*, pages 1–14. Springer International Publishing, 2014.

[22] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[23] Jomi Fred Hübner, Rafael H. Bordini, and Michael Wooldridge. Programming declarative goals using plan patterns. In Matteo Baldoni and Ulle Endriss, editors, *Declarative Agent Languages and Technologies IV, 4th International Workshop, DALT 2006, Hakodate, Japan, May 8, 2006, Selected, Revised and Invited Papers*, volume 4327 of *Lecture Notes in Computer Science*, pages 123–140. Springer, 2006.

[24] Nicholas R. Jennings. Agent-oriented software engineering. In Ibrahim Imam, Yves Kodratoff, Ayman El-Dessouki, and Moonis Ali, editors, *Multiple Approaches to Intelligent Systems*, volume 1611 of *Lecture Notes in Computer Science*, pages 4–10. Springer Berlin Heidelberg, 1999.

[25] Sam Leask and Brian Logan. Programming deliberation strategies in meta-APL. In Qingliang Chen, Paolo Torroni, Serena Villata, Jane Hsu, and Andrea Omicini, editors, *Proceedings of the 18th Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2015)*, volume 9387 of *Lecture Notes in Computer Science*, pages 433–448, Bertinoro, Italy, October 2015. Springer International Publishing.

[26] Brian Logan. A future for agent programming. In Matteo Baldoni, Luciano Baresi, and Mehdi Dastani, editors, *Engineering Multi-Agent Systems: Third International Workshop, EMAS 2015, Istanbul, Turkey, May 5, 2015, Revised, Selected, and Invited Papers*, pages 3–17. Springer International Publishing, 2015.

[27] Jörg P. Müller and Klaus Fischer. Application impact of multi-agent systems and technologies: A survey. In Onn Shehory and Arnon Sturm, editors, *Agent-Oriented Software Engineering*, pages 27–53. Springer Berlin Heidelberg, 2014.

[28] L. J. Osterweil, C. Ghezzi, J. Kramer, and A. Wolf. Determining the impact of software engineering research on practice. *Computer*, 41(3):39–49, 2008.

[29] Lin Padgham and Dhirendra Singh. Situational preferences for BDI plans. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *Proceedings of*

the 12th International conference on Autonomous Agents and Multi-Agent Systems, (AAMAS 2013), pages 1013–1020, Saint Paul, USA, May 2013. IFAAMAS.

[30] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. A goal deliberation strategy for BDI agent systems. In Torsten Eymann, Franziska Klügl, Winfried Lamersdorf, Matthias Klusch, and Michael N. Huhns, editors, *Multiagent System Technologies, Third German Conference, MATES 2005, Koblenz, Germany, September 11-13, 2005, Proceedings*, volume 3550 of *Lecture Notes in Computer Science*, pages 82–93. Springer, 2005.

[31] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, 1991.

[32] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *MAAMAW'96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world: agents breaking away*, pages 42–55, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[33] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[34] Stuart Russell and Eric Wefald. *Do the Right Thing*. MIT Press, 1991.

[35] Sebastian Sardiña, Lavindra de Silva, and Lin Padgham. Hierarchical planning in BDI agent programming languages: a formal approach. In Hideyuki Nakashima, Michael P. Wellman, erhard Weiss, and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1001–1008, Hakodate, Japan, May 2006. ACM.

[36] Sebastian Sardiña and Lin Padgham. Goals in the context of BDI plan failure and planning. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, pages 1–8. ACM, 2007.

[37] Dhirendra Singh and KoenV. Hindriks. Learning to improve agent behaviours in GOAL. In Mehdi Dastani, Jomi F. Hübner, and Brian Logan, editors, *Programming Multi-Agent Systems*, volume 7837 of *Lecture Notes in Computer Science*, pages 158–173. Springer Berlin Heidelberg, 2013.

[38] J. Thangarajah, J. Harland, D. Morley, and N. Yorke-Smith. Suspending and resuming tasks in BDI agents. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multi Agent Systems (AAMAS'08)*, pages 405–412, Estoril, Portugal, May 2008.

[39] John Thangarajah, James Harland, David N. Morley, and Neil Yorke-Smith. On the life-cycle of BDI agent goals. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 1031–1032, Lisbon, Portugal, August 2010. IOS Press.

[40] John Thangarajah, James Harland, David N. Morley, and Neil Yorke-Smith. Quantifying the completeness of goals in BDI agent systems. In Torsten Schaub,

Gerhard Friedrich, and Barry O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI-2014)*, pages 879–884, Prague, Czech Republic, August 2014. IOS Press.

[41] John Thangarajah and Lin Padgham. Computationally effective reasoning about goal interactions. *Journal of Automated Reasoning*, 47(1):17–56, 2011.

[42] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting & avoiding interference between goals in intelligent agents. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 721–726. Morgan Kaufmann, August 2003.

[43] Konstantin Vikhorev, Natasha Alechina, and Brian Logan. Agent programming with priorities and deadlines. In Kagan Turner, Pinar Yolum, Liz Sonenberg, and Peter Stone, editors, *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 397–404, Taipei, Taiwan, May 2011.

[44] Régis Vincent, Bryan Horling, Victor Lesser, and Thomas Wagner. Implementing soft real-time agent control. In *Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS'01)*, pages 355–362, 2001.

[45] T. Wagner, A. Garvey, and V. Lesser. Criteria-directed heuristic task scheduling. *International Journal of Approximate Reasoning*, 19:91–118, 1998.

[46] Andrzej Walczak, Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Augmenting BDI agents with deliberative planning techniques. In *Proceedings of the 4th International Conference on Programming Multi-agent Systems (ProMAS'06)*, pages 113–127, Berlin, Heidelberg, 2007. Springer-Verlag.

[47] Max Waters, Lin Padgham, and Sebastian Sardina. Evaluating coverage based intention selection. In Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns, editors, *Proceedings of the 13th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2014)*, pages 957–964. IFAAMAS, 2014.

[48] Max Waters, Lin Padgham, and Sebastian Sardiña. Improving domain-independent intention selection in BDI systems. *Autonomous Agents and Multi-Agent Systems*, 29(4):683–717, 2015.

[49] David Willetts. Eight Great Technologies. Policy Exchange, 2013.

[50] Michael Winikoff. JACK Intelligent Agents: An Industrial Strength Platform. In *Multi-Agent Programming*, pages 175–193. Springer, 2005.

[51] Michael Winikoff. An agentspeak meta-interpreter and its applications. In Rafael H. Bordini, Mehdi M. Dastani, Jürgen Dix, and Amal Fallah Seghrouchni, editors, *Programming Multi-Agent Systems: Third International Workshop, ProMAS 2005, Utrecht, The Netherlands, July 26, 2005, Revised and Invited Papers*, pages 123–138, Berlin, Heidelberg, 2006. Springer.

[52] Michael Winikoff. Challenges and directions for engineering multi-agent systems. *CoRR*, abs/1209.1428, 2012.

[53] Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative & procedural goals in intelligent agent systems. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness, and Mary-Anne Williams, editors, *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR-02)*, pages 470–481, Toulouse, France, April 2002. Morgan Kaufmann.

[54] Michael Wooldridge and Simon Parsons. Intention reconsideration reconsidered. In *Proceedings of the 5th International Workshop on Intelligent Agents V, Agent Theories, Architectures, and Languages (ATAL'98)*, pages 63–79, London, UK, 1999. Springer-Verlag.

[55] Yuan Yao and Brian Logan. Action-level intention selection for BDI agents. In J. Thangarajah, K. Tuyls, C. Jonker, and S. Marsella, editors, *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, pages 1227–1235, Singapore, May 2016. IFAAMAS.

[56] Yuan Yao, Brian Logan, and John Thangarajah. SP-MCTS-based intention scheduling for BDI agents. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI-2014)*, pages 1133–1134, Prague, Czech Republic, August 2014. ECCAI, IOS Press.

Note

[1]Programming frameworks for the development of MAS are an important output of the agent programming community, but in many cases are not essential for the implementation of individual autonomous systems.

[2]As the authors note, this may be a result of the academic mailing list used to advertise the survey.

[3]The number of applications using a BDI-based platform is somewhat higher if all applications are considered (16%).

[4]The engagement of industry with the AAMAS conference as a whole also does not seem a particularly relevant metric when considering future directions for engineering multiagent systems. AAMAS is a large conference, and there are typically only a relatively small number of papers on agent programming; even if these papers were very relevant to industry, industrial engagement with the conference as a whole could still be low.

[5]The alternative interpretation, that they are all necessary for the wider adoption of APLs, implies that agent programming as a field *must* progress on a very broad front, and is even more daunting than my analysis below.

[6]There are, of course, overlaps between the two views. In particular, there is a strand of work in what I am characterising as the AI-oriented view, that focuses on the engineering of intelligent autonomous systems. However the focus of work in the software engineering-oriented tradition is much less on AI and more on distributed systems.

[7]Some, e.g., [34], go further, and argue that a system's ability to reason about its own deliberation is key to intelligence. While taking resource-boundedness into account is clearly necessary when designing practical agents, the focus of this paper is on the functional requirements of an intelligent autonomous agent.

[8]There is some evidence, e.g., [3], which suggests that BDI-based languages can significantly increase programmer productivity relative to programming in, e.g., Java. Even if this is the case, for current BDI-based languages to be widely adopted, there needs to be a significant class of applications that can be mostly programmed using only the current BDI core feature set, and there is little evidence to support this.

[9]The Jason `.intend` internal action checks if a particular triggering event occurs in any plan within an intention. However it provides no information about the plans that have been selected for these events, how far execution of these plans have progressed etc. (this information is available for the *current* intention via the `.current_intention` internal action, but not for other intentions).

[10]The argument that an APL should support only basic features for plan selection and failure recovery seems spurious for another reason—most widely used programming languages provide support for many more features than will be used in any particular application.

[11]`www.icaps-conference.org`

[12]The 2014 edition of the International Planning Competition included a *Sequential Agile* track for the first time. The objective of the Agile track is to 'minimize the CPU time needed for finding a plan'.

[13]See, for example `rx.codeplex.com`.

[14]`msdn.microsoft.com`

[15]`code.google.com/p/cqengine`

[16]Implementing support for handling plan failure in a reactive programming model is slightly more complex, but possible.

[17]In 2APL it is possible to write a planning goal rule to drop a goal based on the agent's other goals, but it is not possible to decide whether to adopt a goal based on the agent's current goals. One can write a procedure call rule to, e.g., convert an event into a goal, but the condition of the rule is restricted to being a belief query. In Jason, even getting access to the agent's current goals is difficult.

[18]A similar point is made by Hindriks [21] when he advocates easy access to powerful AI techniques. However Hindriks sees this as a desirable rather than a necessary feature.