A Future for Agent Programming*

Brian Logan

School of Computer Science
University of Nottingham
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 remedies have been proposed. In this paper, I will offer one more. Starting from the class of problems agent programming sets out to solve, I will argue that a combination of Moore's Law and advances elsewhere in AI, mean that key assumptions underlying the design of many BDI-based agent programming languages no longer hold. As a result, we are now in a position where we can rethink the foundations of BDI programming languages, and address some of the key challenges in agent development that have been largely ignored for the last twenty years. By doing so, I believe we can create theories and languages that are much more powerful and easy to use, and significantly broaden the impact of the work we do.

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* [29] with the potential to revolutionise our economy and society. For example, the UK Knowledge Transfer Network for Aerospace, Aviation & Defence report *Robotics and Autonomous Systems: Challenges and Opportunities for the UK* states: "the economic, cultural, environmental and social impacts and benefits [of autonomous systems] will be unprecedented" [17]. 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 academia and industry, one might expect

* This paper is a a revised and extended version of an invited talk given at EMAS 2015. I am grateful to the workshop organisers for the invitation, and the opportunity to contribute to the post-proceedings.

the agent programming community, which specialises in theories, architectures, languages and tools for the development of autonomous systems to be thriving, and the languages and tools they have developed to support the development of 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 [7, 31, 15]. More worrying, the most distinctive 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. A study by Winikoff [31] of applications appearing in the AAMAS Industry/Innovative Applications tracks in 2010 and 2012, reveals that the systems described do not require intelligent goal-based (BDI) agents, and the focus of many applications is at the multi-agent system (MAS) coordination level (e.g., game theory, MDPs). The most recent survey by Müller & Fischer in 2014 [15] reports 46 'mature' applications (out of 152 surveyed applications). They found that:

- 82% of mature 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
- only 10% of mature applications clearly used a BDI-based platform; of those that did, all used the JACK [30] platform

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 applications using a BDI platform may be higher than 10%. However even allowing for these factors, it seems hard to argue that BDI agents are having a significant impact in application development.

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 we 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 [4]. 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'

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

to refer to the developers of these languages and tools (exemplified by EMAS and its predecessor workshops) 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; however my proposals about what to do about it focus primarily on BDI-based approaches. The tenor of the paper is deliberately polemical, and some steps in the argument may be seen as contentious. However I believe the changes in the context of agent programming in artificial intelligence and computer science I point to, and the opportunities they present, are unassailable. So even if my analysis of the problem is flawed, the opportunities we have to do interesting work are very real.

2 Background

I am not the first to consider the low take-up of agent programming in mainstream software development, or how agent programming languages could or should develop to maximise their adoption. This paper follows in a tradition of talks and panel sessions at agent programming workshops, including the Dagstuhl Seminar on Engineering Multi-Agent Systems [8] and the EMAS 2013 & 2014 workshops. In this section I briefly review some of this work.

In [31] Winikoff identifies a number of challenges for engineering multi-agent systems and proposes directions for future work in engineering MAS. He focusses on the relevance of the AAMAS community to industry, and 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.[2] Drawing on data from [7] and the analysis of papers appearing in the AAMAS Industry/Innovative Applications tracks in 2010 and 2012 mentioned briefly above, he argues that AOPL and AOSE usage is quite limited, and that the applications described focus on coordination aspects and do not require goal-based agents. Based on these observations he recommends that the agent programming community should:

– stop designing AOPLs and AOSE methodologies ... and instead ...

[2] 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 multi-agent 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.

– 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 recommendation appears to ignore the possibility that the reason current applications focus on coordination of MAS rather than goal-based agents, is that the support provided by current agent programming languages for developing goal-based 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 design of agent programming languages.

In [13] Hindriks reviews the history of engineering multi-agent 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.[3]

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 we face. The key problem lies elsewhere, and has not previously been articulated. I turn to this in the next section.

[3] 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.

3 Why Are We Failing to Have an Impact?

I begin by elucidating the problem we are 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' [14]. The software engineering-oriented view on the other hand, focuses on synergies between software engineering and agent research.[4] Each tradition is associated with its own set of research questions and 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).

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 [18]. 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, and the distinctive set of ideas represented by the BDI paradigm) offers the best hope for agent programming to have an impact outside our community.

Perhaps the best characterisation of the AI-oriented view is given in the call for papers for the first ATAL workshop, held in 1994, which 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

[4] 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.

In this view, agents are a way of realising the broader aims of artificial intelligence. Agents are 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.

Given this characterisation of the goals of agent programming, the reason the agent programming research conducted by the agent programming community has failed to have an impact follows fairly immediately:

> *we can't solve a large enough class of AI problems well enough to be interesting to the wider AAMAS community or application developers*

In the remainder of this section, I will attempt to justify this claim

3.1 The BDI Model

The Belief-Desire-Intention (BDI) model and its underlying theoretical underpinnings 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 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. Plans are 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. 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.

3.2 Limitations of Current BDI-Based Languages

The BDI approach has been very successful, to the extent that it arguably the dominant paradigm in agent programming [11]. A wide variety of agent languages and agent platforms have been developed which at least partially implement the BDI model, e.g., [30, 5, 4, 6, 12]. 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). 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.

The features common to state of the art BDI languages, and which currently define this style of programming, are essentially limited to:

- selecting canned plans at run time based on the current context; and
- some support for handling plan failure (e.g., trying a different plan)

While these features are useful, and are key to implementing agents based on the BDI paradigm, everything else is left to the programmer.

What's left to the programmer is all the hard(er) parts of implementing an autonomous agent. More specifically, some of the things the current generation of agent programming languages can't do (in a *generic* way) includes:

- how to handle costs, preferences, time, resources, durative actions, etc.
- 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 or plan failure
- when to drop a goal or try a different approach
- and many others ...

While not all of these capabilities will be required in every agent application, a reasonable argument can be made that many are necessary in most, if not all, cases (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.

There has been some preliminary work on how to implement many of these capabilities, see, for example, [3, 26, 21, 28, 22, 24, 27, 23, 16, 32, 25]. 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, rests on the observation that, for any particular application, the detailed answers to these questions will differ. The argument that such issues should or must be left to the programmer is reminiscent of the 'New Jersey approach' [9]: 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 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.[5]

It is of course true that for specific applications, the detailed answers to these questions (or even whether a feature is needed at all) will vary. However do we as a community really want to claim that there are *no* general theories or approaches to these questions? If so, 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 will be limited.

In summary, the support currently offered by state of the art APLs is useful, particularly for some problems. However it is not useful enough for most developers to switch platforms, even if we polish our methodologies and tools. I believe we need to answer these questions, and I explain why in the next section.

4 The Broader Context

The analysis presented in the previous section actually underestimates the scale of the problem facing the agent programming community. In this section, I will argue that there is good reason to suppose that the already limited impact of agent programming on the wider AAMAS community 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 like simple forms of agent programming, potentially limiting the impact of APL technologies on mainstream development. I discuss each of these developments below.

4.1 Reactive Planning

The BDI approach to agent programming is based on early work on reactive planning, e.g., [10]. 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;
- 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.

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

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" [10].

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 an NP-hard 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) [19] which influenced the design of many current state of the art agent programming languages (see Figure 1).[6]

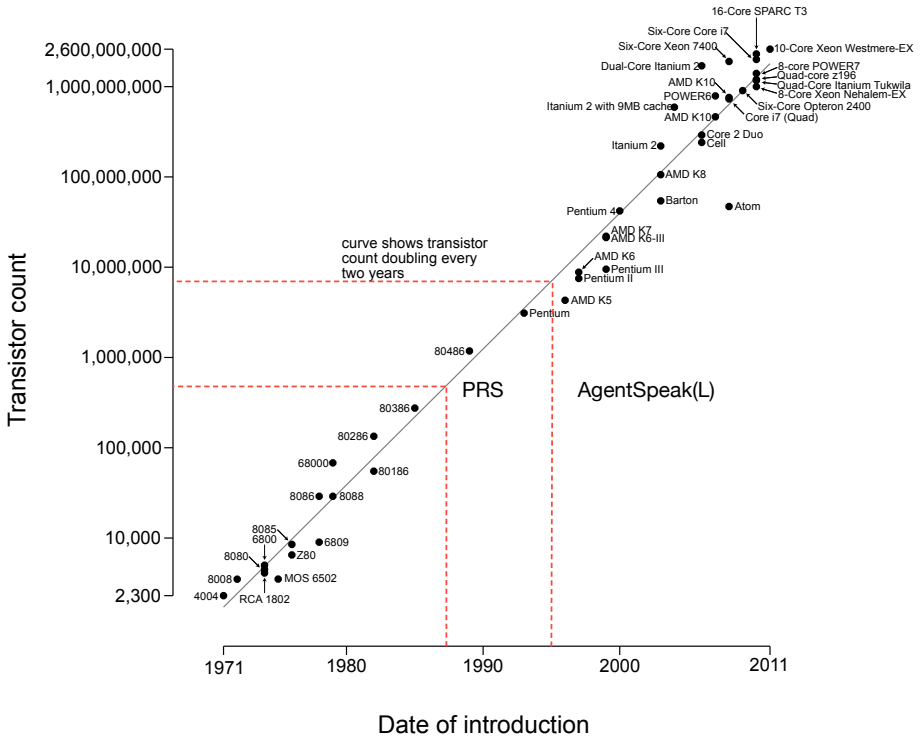

Fig. 1. Microprocessor transitor counts 1971-2011

[6] Creative Commons 'Transistor Count and Moore's Law' by Wgsimon used under CC-BY-SA-3.0. Transistor counts circa 1987 and 1996 highlighted.

It is perhaps now time to reconsider whether traditional planning techniques are unsuited to domains where replanning is frequently necessary. Many planning problems can now be solved in less than a second, and some generative planners are approaching the 100ms threshold necessary for agent planning in real time domains. Figure 2 shows the number of solved problem instances with time for all domains of the ICAPS International Planning Competition[7] and a range of generative planners.[8] As can be seen, the best planners (from the point of view of a real time agent) are capable of solving over half the problem instances in < 100ms. (Note that these results are from 2012.)

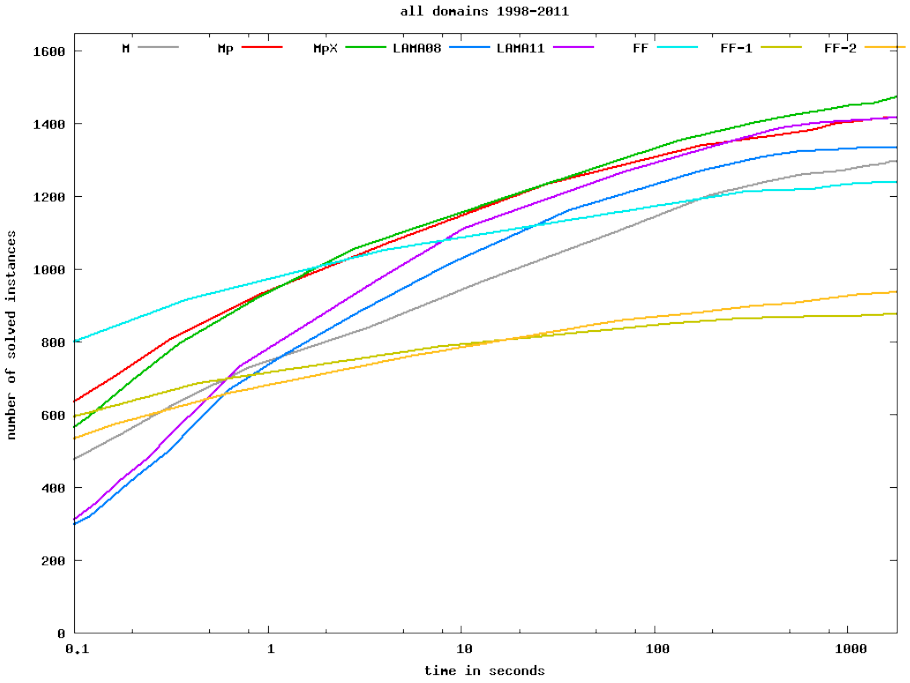

Fig. 2. Number of IPC problem instances solved by different planners with time

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

[7] www.icaps-conference.org

[8] The figure is from http://users.ics.aalto.fi/rintanen/jussi/satplan.html and is reproduced here with the permission of the author. See also [20].

with an increased interest in the planning community in 'real time' planning,[9] 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.

4.2 Reactive Programming

At the same time, work on event-driven and reactive programming[10] (e.g., in robotics) offers similar (or better) functionality to belief triggered plans in agent programming. Such approaches offer:

- a well defined model of streams (immutability, sampling, pull-based computation)
- very fast (microsecond) evaluation of simple SQL-like queries (e.g., LINQ,[11] cqengine[12]) that scale to very large 'belief bases' for evaluation of context conditions

Taken together, these technologies can provide a simple form of event-driven reactive agent behaviour (e.g., if subgoals are seen as a stream of events). These paradigms are now a part of 'mainstream' Computer Science. For example, 'Event Driven and Reactive Programming' is included in the 2013 ACM model curriculum for Computer Science [1].

The developments discussed above do not constitute a comprehensive list of the changes impacting or likely to impact the EMAS 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.

5 The Future

The advances in both hardware and related AI sub-disciplines highlighted in the previous section mean we are now in a position where we can rethink the foundations of agent programming languages. By engaging with cutting edge AI research, we can address some of the key challenges in agent development

[9] 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'.

[10] See, for example rx.codeplex.com.

[11] msdn.microsoft.com

[12] code.google.com/p/cqengine

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.

5.1 Some Ideas

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—there are many other ways things could go (for some alternative suggestions, see [13]). 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 (implications for plan selection)

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, norms etc; 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.

5.2 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 multi-agent systems, and artificial intelligence and computer science more generally. I therefore conclude this section with a set of 'progress metrics' which are rooted in the analysis presented in Section 3:

- 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

– evaluation of agent programs (and indirectly of APLs) requires richer benchmark problems (or less toy versions of current problems)
– 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. 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 entities.[13]

This list of performance metrics is preliminary and can (and should) be improved. However broad consensus around some set of metrics is essential for EMAS 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 multi-agent systems.

6 Conclusion

Agent programming 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 AAMAS community, including EMAS, 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 more interesting 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. 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.
4. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley, 2007.

[13] A similar point is made by Hindriks [13] when he advocates easy access to powerful AI techniques. However Hindriks sees this as a desirable rather than a necessary feature.

5. 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. Birkhuser Basel, 2005.
6. Mehdi Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
7. Virginia Dignum and Frank Dignum. Designing agent systems: state of the practice. *International Journal of Agent-Oriented Software Engineering*, 4(3):224–243, 2010.
8. Jürgen Dix, Koen V. Hindriks, Brian Logan, and Wayne Wobcke. Engineering Multi-Agent Systems (Dagstuhl Seminar 12342). *Dagstuhl Reports*, 2(8):74–98, 2012.
9. 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).
10. 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.
11. 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.
12. 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.
13. Koen V. Hindriks. The shaping of the agent-oriented mindset. In Fabiano Dalpiaz, Jrgen 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.
14. 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.
15. 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.
16. Lin Padgham and Dhirendra Singh. Situational preferences for BDI plans. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13*, pages 1013–1020. IFAAMAS, 2013.
17. Charles Patchett. Robotics and Autonomous Systems: Challenges and Opportunities for the UK, 2014.
18. 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.
19. 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.
20. Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 879–884. IOS Press, 2014.
26. 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.
27. 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.
28. 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.
29. David Willetts. Eight Great Technologies. Policy Exchange, 2013.
30. Michael Winikoff. JACK Intelligent Agents: An Industrial Strength Platform. In *Multi-Agent Programming*, pages 175–193. Springer, 2005.
31. Michael Winikoff. Challenges and directions for engineering multi-agent systems. *CoRR*, abs/1209.1428, 2012.
32. 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.