Logan, Brian (2017) Future directions in agent programming. ALP Issue, 29 (4).

Future Directions in Agent Programming

Brian Logan

School of Computer Science
University of Nottingham
Nottingham, UK
bsl@cs.nott.ac.uk

Abstract. Agent programming is a subfield of Artificial Intelligence concerned with the development of intelligent autonomous systems that combine multiple capabilities, e.g., sensing, deliberation, problem-solving and action, in a single system. 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 currently confined to a small number of niche areas, and adoption of agent programming languages (APLs) in mainstream software development remains limited. In this paper, I argue that increased adoption of agent programming is contingent on being able to solve a larger class of AI problems with significantly less developer effort than is currently the case, and briefly sketch one possible approach to expanding the set of AI problems that can be addressed by APLs. Critically, the approach I propose requires minimal developer effort and expertise, and relies instead on expanding the basic capabilities of the language.

1 Introduction

Agent programming is a subfield of Artificial Intelligence concerned with the development of intelligent autonomous systems that combine multiple capabilities, e.g., sensing, deliberation, problem-solving and action, in a single system. It aims to provide the tools to realise and integrate these capabilities so as to achieve flexible intelligent behaviour in dynamic and unpredictable environments. Agent programming has a long history dating back to (at least) the mid 1980's, and has strong connections with logic programming; indeed many agent programming languages incorporate a sizeable subset of Prolog as a sublanguage. However the origins of agent programming lie more in work on reactive planning, e.g., [6], than logic programming per se.

Over the last 30 years, there has been considerable progress in both the theory and practice of agent programming, and a wide variety of agent programming languages (APL) and agent platforms have been developed. However, despite the increasing interest in the development of intelligent autonomous systems, e.g., driverless cars, UAVs, manufacturing, healthcare etc., the impact of current agent programming languages and agent programming generally 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 [5, 24, 12]. 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 [24] of applications appearing in the Autonomous Agents and Multiagent systems (AAMAS) Conference 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. The most recent survey by Müller & Fischer in 2014 [12] reports similar results: 82% of mature applications focus on the MAS level, while only 9% focus on 'intelligent agents'; and only 10% of mature applications clearly used a BDI-based platform.

In this paper I explore possible reasons for the lack of adoption of agent programming by the broader AI research community and developers of agent-based systems, and suggest some directions for future work to address the problem. The paper draws heavily on [11], which contains a more detailed analysis of the current state of the art in agent programming. However the present paper offers more concrete examples of the kinds of future research directions I am advocating.

2 Agent Programming

In this section, I define the problem agent programming is trying to solve and briefly review the state of the art in agent programming language research.

There are many different views of the aims and objectives of 'agent programming' considered as a field. At a high level, 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' [10]. The software engineering-oriented view on the other hand, focuses on synergies between software engineering and agent research.[1]

In what follows, I focus on the AI-oriented view. The AI-oriented view represents the original motivation for agent programming as a subfield, and arguably the most significant contributions of agent programming have emerged from this tradition, e.g., the 2007 International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS) *Influential Paper Award* for Rao & Georgeff's

[1] 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.

work on rational agents [14]. 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.

2.1 The BDI Model

The BDI approach to agent programming is based on early work on reactive planning, e.g., [6]. 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.

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* (goal directed) 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. In many BDI-based agent programming languages, beliefs and goals are maintained using some form of declarative knowledge representation, for example, an agent's beliefs are often represented by a set of Horn clauses. *Plans* are the means by which the agent can modify the environment in order to achieve its goals. Plans are often implemented by a set of rules. The rule conditions consist of queries to be evaluated against the agent's beliefs and goals, and the rule actions consist of a sequence of steps which are either basic actions that directly change the agent's environment or internal state, 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 pushed onto 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 (unifies with) 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 condition 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 selects one or more intentions and *executes* one (or more) steps of the plan which forms the top of the intention. This process of repeatedly choosing

and executing plans is referred to as 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.

2.2 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 [7], and a wide variety of agent languages and agent platforms have been developed which at least partially implement the BDI model, e.g., [23, 3, 2, 4, 8]. A number of these languages and platforms are now reasonably mature in terms of their feature set, 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 for the current goal)

All other aspects of implementing an autonomous agent is left to the programmer. More specifically, the things that must be coded 'from scratch' by a developer include:[2]

- 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, 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.

[2] There has been some preliminary work on how many of these capabilities could be implemented, see, for example, [1, 20, 15, 22, 16, 18, 21, 17, 13, 26, 19]. However, to date, this work has not been incorporated into the core feature set of popular BDI platforms.

3 Future Directions

The support currently offered by state of the art APLs is useful, particularly for some problems. However, as I have argued elsewhere [11], it is currently not useful enough for most developers to switch platforms. Widespread adoption of agent programming languages is contingent on being able to solve a larger class of AI problems with significantly less developer effort than is currently required using 'mainstream' languages and tools (or current APLs).

In this section, I briefly sketch one possible approach to expanding the set of AI problems that can be addressed by agent programming languages. Critically, the approach I propose requires minimal developer effort and expertise, and relies instead on expanding the basic capabilities of the APL.

In many BDI agent architectures, the plans comprising the agent's intentions are executed in parallel, e.g., by executing one step of an intention at each cycle in round robin fashion [23, 2]. Interactions between interleaved steps in plans in different intentions may result in conflicts, i.e., the execution of a step in one plan makes the execution of a step in another concurrently executing plan impossible. Such conflicts between intentions can sometimes be avoided by using *atomic* constructs available in languages such as Jason [2] and 2APL [4], that prevent the interleaving of actions in one plan with actions from other plans, or by limiting the agent to the execution of a single intention at a time (FIFO scheduling) [23]. However, it is difficult for the programmer to ensure that all potential interactions between plan steps are encapsulated within atomic constructs, and disallowing the interleaved execution of intentions may reduce the responsiveness of the agent to an unacceptable degree.

While current agent programming languages provide considerable syntactic support for steps one and two in the deliberation process (i.e., determining relevant applicable plans), support for the third and fourth steps (i.e., selection of which applicable plan to adopt for a particular goal and deciding which steps of which intentions to execute) is limited to some flags, and/or over-riding the default deliberation cycle behaviour by redefining 'selection functions' in the host language (the language in which the agent programming language is itself implemented). For example, the $S_{\mathcal{O}}$ and $S_{\mathcal{I}}$ selection functions of Jason [2] allow a developer to customise Jason's plan and intention selection for a particular application domain. While such customisation may be necessary or desirable for some problems, it requires specialist expertise on part of the developer to program at the level of the interpreter rather than writing BDI plans.

To address this problem, we have investigated an alternative approach, which involves extending the capabilities of an APL to incorporate reasoning about possible interactions between steps in an agent's intentions [26, 27, 25, 27]. Reasoning about interactions can be used to avoid conflicts between intentions [25], to recover from action failures without backtracking [28], and to schedule intentions so as to achieve goals by their deadlines [27]. Our approach, *SA*, involves stochastic sampling of possible future executions of the agent program. *SA* has been shown to out-perform current approaches to intention progression in a range

of application domains, and has modest computational cost (a few milliseconds per deliberation cycle).

SA effectively merges steps 3 and 4 of the standard BDI deliberation cycle into a single process, which is responsible for determining both which plan to adopt for a particular (sub)goal, and which step of which intention to execute at this cycle. Only the choice of which top-level goals to adopt is left to the developer: once a top-level goal is adopted, *SA* determines which plans to execute and how these plans should be interleaved so as to maximise a developer-specified measure of performance (e.g., number of goals achieved). If the intention selected for execution at the previous cycle posted a subgoal, *SA* explores through sampled pseudorandom simulation the implications of intending all relevant applicable plans for the subgoal (and the possible subplans of those plans) and their possible interleavings with all possible ways of achieving the agent's other intentions. As such it assumes responsibility for decisions that current APLs traditionally leave to the developer. Critically, it both reduces developer effort (the developer no longer has to anticipate and control possible interactions between intentions using atomic constructs, for example) and in many cases improves the performance of agent programs. The role of the developer changes from specifying low-level procedural knowledge (how a particular set of intentions should be interleaved) to specifying declarative knowledge (the pre- and postconditions of actions).

4 Discussion

The sketch of possible future directions for agent programming in the previous section is shaped by my own interests and research. However, as explained in Section 2.2, there are many other proposals for extending the capabilities of agent programming that may be fruitful directions for future research. Whichever direction(s) are followed, I believe that any future work must involve 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.

Clearly, developers 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.[3] 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 agent theories and languages that are much more powerful and easy to use, and encourage the adoption of agent programming in mainstream programming and AI generally.

[3] A similar point is made by Hindriks [9] when he advocates easy access to powerful AI techniques. However Hindriks sees this as merely desirable rather than a necessary condition for progress.

References

1. 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.
2. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. Wiley, 2007.
3. 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.
4. Mehdi Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
5. Virginia Dignum and Frank Dignum. Designing agent systems: state of the practice. *International Journal of Agent-Oriented Software Engineering*, 4(3):224–243, 2010.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.

14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. Michael Winikoff. JACK Intelligent Agents: An Industrial Strength Platform. In *Multi-Agent Programming*, pages 175–193. Springer, 2005.
24. Michael Winikoff. Challenges and directions for engineering multi-agent systems. *CoRR*, abs/1209.1428, 2012.
25. 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, IFAAMAS.
26. 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.

27. Yuan Yao, Brian Logan, and John Thangarajah. Intention selection with deadlines. In G. A. Kaminka, M. Fox, P. Bouquet, Hullermeijer E., Dignum F., Dignum V., and van Harmalen F., editors, *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI-2016)*, pages 1700–1701, The Hague, The Netherlands, August 2016. ECCAI, IOS Press.
28. Yuan Yao, Brian Logan, and John Thangarajah. Robust execution of BDI agent programs by exploiting synergies between intentions. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 2558–2564, Phoenix, USA, February 2016. AAAI, AAAI Press.