Updating Action Descriptions and Plans for Cognitive Agents

Extended Abstract

Peter Stringer
University of Manchester
Manchester, United Kingdom
peter.stringer@manchester.ac.uk

Rafael C. Cardoso
University of Aberdeen
Aberdeen, United Kingdom
rafael.cardoso@abdn.co.uk

Clare Dixon
University of Manchester
Manchester, United Kingdom
clare.dixon@manchester.ac.uk

Michael Fisher
University of Manchester
Manchester, United Kingdom
michael.fisher@manchester.ac.uk

Louise A. Dennis
University of Manchester
Manchester, United Kingdom
louise.dennis@manchester.ac.uk

ABSTRACT

In this paper, we present an extension of Belief-Desire-Intention agents which can adapt their performance in response to changes in their environment. Our main contributions are the underlying theoretical mechanisms for data collection about action performance, the synthesis of new action descriptions from this data, the integration with plan reconfiguration, and a practical implementation to validate the semantics.

KEYWORDS

Beliefs-Desires-Intentions; Action Descriptions; AI Planning

ACM Reference Format:
Peter Stringer, Rafael C. Cardoso, Clare Dixon, Michael Fisher, and Louise A. Dennis. 2023. Updating Action Descriptions and Plans for Cognitive Agents: Extended Abstract. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 3 pages.

1 INTRODUCTION AND BACKGROUND

Long-term autonomy requires autonomous systems to adapt once their capabilities no longer perform as expected. To achieve this, a system must first be capable of detecting such changes and then adapting its internal reasoning processes to accommodate these. For example, deploying an autonomous robot into a dynamic environment can result in actions becoming unreliable over time, as the environment changes, producing unexpected outcomes that were unforeseeable before runtime. The autonomous agent must be capable of observing these changes and adapting accordingly.

Our work focuses on cognitive agents [3, 17, 21] programmed in a Belief-Desire-Intention (BDI) [16, 17] programming language providing high-level decision-making in an autonomous system, as outlined in [8]. Programs written in these languages use *plans* created in advance by a programmer to select *actions* to execute in the environment. These plans make implicit assumptions about the behaviour of the actions they execute. Therefore, in this context, the challenge becomes to make these assumptions explicit, detect when they no longer hold, and then modify the plans accordingly.

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

Some Belief-Desire-Intention (BDI) languages use *action descriptions* (sometimes referred to as *capabilities* in the literature), which consist of explicit pre- and post-conditions for all known actions. These have their roots in AI planning and STRIPS operators [10]. Mechanisms and semantics used for such functionality are discussed in [7, 14, 19]. A version of Gwendolen (which we have used for our implementation) also exists that contains an implementation of action descriptions [19].

Definition 1.1 (Action Description). We assume a language $\mathcal{L}$ of first-order terms constructed in the usual way. *Action descriptions* are a triple $\{Pre\}A\{Post\}$ where A is a term in $\mathcal{L}$ representing an action, $\{Pre\}$ is a set of terms representing the action's preconditions and $\{Post\}$ is a set of expressions of the form $+t$ or $-t$ (where t is a term in $\mathcal{L}$). Note: $+t$ means that the term t should be added to the agent's belief base after execution of the action and $-t$ that it should be removed.

Gwendolen tracks the performance of actions over time in an *action log*. An *action log* keeps a record of action outcomes in an array of fixed, application-specific size, where the oldest entry is removed before adding a new one, once the log reaches its size limit. The action log therefore enables Gwendolen to reason about the probability of action success and opens the possibility of implementing an *action lifecycle* [20], inspired by the concept of goal life-cycles for BDI languages [13].

The automated planning research community has invested considerable effort in the modelling of actions with stochastic outcomes. both theoretically [15, 22], and practically (e.g. [5, 11]). This community deploys action descriptions to flexibly plan on-the-fly for each new goal, which avoids the problem faced in BDI languages that an action whose behaviour has changed may result in failing, and therefore useless, plans. Plan failure has been extensively researched in BDI programming languages (e.g., [2, 9, 18]), however, it has not been linked with action descriptions perhaps because most languages do not use action descriptions as a mechanism to detect action failure. The closest work to our own is in [13] with a proposal for BDI goal life-cycles.

A key component of our approach is synthesizing or learning a new action description when an action ceases to perform as expected. Using algorithms to discover the effects of actions has been explored in the AI Planning domain [1]. We have based our approach on ideas from [6] and [12] where new action descriptions are learned from traces of action behaviour with a weighting

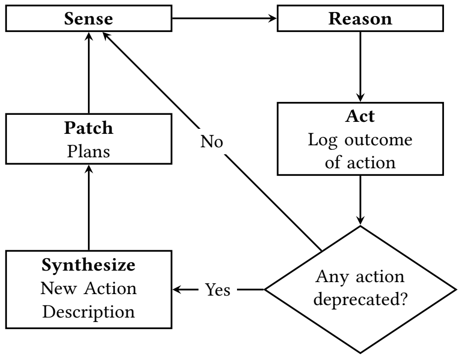

Figure 1: Extended Sense-Reason-Act cycle.

mechanism guiding the choice of additions and deletions to the constructed action post-condition.

A mechanism for patching BDI plans by combining BDI agents and automated planning was presented in [4], but it does not account for how failure is detected. We leverage this work in ours. If an action is deprecated by the action lifecycle, then any plans involving that action are patched using this mechanism.

To the best of our knowledge, there is no end-to-end framework in cognitive agents for updating action descriptions and patching the associated plans, as presented here. Our contribution is a methodology to detect faulty actions, modify their descriptions and reconfigure BDI plans based on these new descriptions, enabling long-term autonomy. Our work applies to BDI programming languages that allow action descriptions.

2 FRAMEWORK

Our starting point is the system architecture from [8] in which a cognitive agent performs high-level mission reasoning, such as deciding in which order some set of waypoints are to be visited. Cognitive agents employ a *reasoning cycle* which governs a sense-reason-act process. The action log integrates with the *act* phase and compares the outcomes of executed actions to the post-conditions in the action's description. If the post-conditions are successful, then a success is logged, and if they are not, a failure is logged. In all situations, the action log also records the changes in beliefs from the moment when the action was executed to the moment when it succeeded or failed. These changes are stored as a list of expressions of the form $+t$ or $-t$ where t is a term — that is, in the same format as post-conditions in action descriptions. Our framework extends action descriptions to include a *failure threshold*.

If the number of failures for the action in the action log exceeds the failure threshold, then the action becomes deprecated. Note that the action log should be of fixed length, so that an action can not become deprecated as the result of a slow build up of occasional failure over time.

We extend the act phase of the reasoning so that after the execution of an action, the action log is consulted. If the most recent action has not become deprecated the cycle continues as before. If it has become deprecated, then a new action description is synthesized from the information in the log and relevant plans are patched before the agent continues to the sense phase. This reasoning cycle is shown in Figure 1.

We synthesize a new action description by extracting, from the action log, all the failed instances of the deprecated action. We then have a list of new candidate post-conditions for the action in the form of the change in beliefs as the action executed. Each item in this list is assigned a weight score based on how recent the item is. The weights for identical items are then summed, and the highest-scoring item is selected as the new post-condition for the action. Pseudo-code for this process is shown in Algorithm 1. Line 2 instantiates the initial weight score (n) to 1, and in Line 3 it sets *post_scores* to an empty map. Lines 4–7 will loop through every entry in the action log to find entries that match with the deprecated action (same action) and where the outcome of the entry was reported as a failure. When this happens, the post-conditions of the action are added to the *post_scores* map along with the weight score, which is then incremented by one for the future iterations of the action log. In line 8 we initialise *best* with 0. Lines 9–11 iterate over the keys in the *post_scores* map to select the candidate post-condition with the highest weight score.

Algorithm 1: Algorithm for synthesizing post-conditions.

```
1  if action is deprecated then
2      n ← 1;
3      post_scores ← {} // map data-structure
4      for entry ∈ action log do
5          if entry[0] = action & entry[2] = Failure then
6              post_scores[entry[1]] ←
                 post_scores[entry[1]] + n;
7              n ← n + 1
8      best ← 0;
9      for post ∈ keys(post_scores) do
10         if post_scores[post] > best then
11             best ← post
```

Once we have a new action description, we use the plan patching mechanism from [4] to patch any plans containing the action.

3 EVALUATION AND CONCLUSION

We evaluated our approach with a navigation example [1]. Our environment consisted of five waypoints and our agent had a plan for a patrol mission to visit each waypoint in turn. Each move action had a description of the form: $\{at(X)\}$`move(X, Y)`$\{-at(X), +at(Y)\}$ We allowed one move action to change its behaviour so that the agent arrived at a different waypoint to the one anticipated (e.g., because of obstacle avoidance behaviour). The system would then observe this changed behaviour and update the action description. It then attempts to patch its plans – typically by finding a different route to the desired waypoint that avoided whatever was blocking the altered move action.

We have presented here the over-arching template of a framework for adapting BDI agent plans in the face of changed action behaviour. While there is a great deal of scope for extending the framework we believe the basic architecture and concept provides a sound foundation for this further work.

[1]All code can be found at https://github.com/mcapl/mcapl/tree/reconfig_peter

ACKNOWLEDGMENTS

This work has been supported by The University of Manchester's Department of Computer Science and the EPSRC "Robotics and AI for Nuclear" (EP/R026084/1) and "Future AI and Robotics for Space" (EP/R026092/1) Hubs.

REFERENCES

[1] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Etivier, and Sylvie Pesty. 2018. A review of learning planning action models. *Knowledge Engineering Review* 33 (2018).

[2] Rafael H Bordini and Jomi Fred Hübner. 2010. Semantics for the Jason Variant of AgentSpeak (Plan Failure and some Internal Actions).. In *ECAI*. IOS Press, 635–640. https://doi.org/10.3233/978-1-60750-606-5-635

[3] M. E. Bratman. 1987. *Intentions, Plans, and Practical Reason.* Harvard University Press.

[4] Rafael C. Cardoso, Louise A. Dennis, and Michael Fisher. 2019. Plan Library Reconfigurability in BDI Agents. In *Proc. of the 7th International Workshop on Engineering Multi-Agent Systems (EMAS)*. Springer, 195–212.

[5] M. Cirillo, L. Karlsson, and A. Saffiotti. 2010. Human-Aware Task-Planning: An Application to Mobile Robots. *ACM Trans. Intelligent Systems Technology* 1, 2 (2010), 15.

[6] Paul R Cohen and Edward A Feigenbaum. 2014. *The handbook of artificial intelligence: Volume 3*. Vol. 3. Butterworth-Heinemann.

[7] Mehdi Dastani, M. van Birna Riemsdijk, and John-Jules Ch. Meyer. 2005. Programming Multi-Agent Systems in 3APL. In *Multi-Agent Programming: Languages, Platforms and Applications*, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni (Eds.). Springer US, Boston, MA, 39–67. https://doi.org/10.1007/0-387-26350-0_2

[8] Louise A Dennis, Michael Fisher, Nicholas K Lincoln, Alexei Lisitsa, and Sandor M Veres. 2016. Practical verification of decision-making in agent-based autonomous systems. *Automated Software Engineering* 23, 3 (2016), 305–359.

[9] Angelo Ferrando and Rafael C. Cardoso. 2022. Safety Shields, an Automated Failure Handling Mechanism for BDI Agents. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems* (Virtual Event, New Zealand) *(AAMAS '22)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1589–1591. https://www.ifaamas.org/Proceedings/aamas2022/pdfs/p1589.pdf

[10] Richard E. Fikes and Nils J. Nilsson. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2, 3 (1971), 189–208. https://doi.org/10.1016/0004-3702(71)90010-5

[11] M. Fox and D. Long. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *JAIR* 20 (2003), 61–124.

[12] Alejandro Guerra-Hernández, Amal El Fallah-Seghrouchni, and Henry Soldano. 2004. Learning in BDI multi-agent systems. In *International Workshop on Computational Logic in Multi-Agent Systems*. Springer, 218–233.

[13] James Harland, David N Morley, John Thangarajah, and Neil Yorke-Smith. 2014. An operational semantics for the goal life-cycle in BDI agents. *Autonomous agents and multi-agent systems* 28, 4 (2014), 682–719. https://doi.org/10.1007/s10458-013-9238-9

[14] Koen V. Hindriks. 2009. Programming Rational Agents in GOAL. In *Multi-Agent Programming: Languages, Tools and Applications*, Amal El Fallah Seghrouchni, Jürgen Dix, Mehdi Dastani, and Rafael H. Bordini (Eds.). Springer US, Boston, MA, 119–157. https://doi.org/10.1007/978-0-387-89299-3_4

[15] Mausam and Daniel S. Weld. 2008. Planning with Durative Actions in Stochastic Domains. *JAIR* 31 (2008), 33–82.

[16] A. S. Rao and M. P. Georgeff. 1991. Modeling Agents within a BDI-Architecture. In *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR&R)* (mdfbook). Morgan Kaufmann, 473–484.

[17] A. S. Rao and M. P. Georgeff. 1992. An Abstract Architecture for Rational Agents. In *Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR&R)*. Morgan Kaufmann, 439–449.

[18] S. Sardina and L. Padgham. 2011. A BDI Agent Programming Language with Failure Handling, Declarative Goals, and Planning. *Autonomous Agents and Multi-Agent Systems* 23, 1 (2011), 18–70.

[19] Peter Stringer, Rafael C. Cardoso, Clare Dixon, and Louise A. Dennis. 2022. Implementing Durative Actions with Failure Detection in Gwendolen. In *Engineering Multi-Agent Systems*, Natasha Alechina, Matteo Baldoni, and Brian Logan (Eds.). Springer International Publishing, Cham, 332–351.

[20] Peter Stringer, Rafael C. Cardoso, Xiaowei Huang, and Louise A. Dennis. 2020. Adaptable and Verifiable BDI Reasoning. In Proceedings of the First Workshop on *Agents and Robots for reliable Engineered Autonomy,* Virtual event, 4th September 2020 *(Electronic Proceedings in Theoretical Computer Science, Vol. 319)*, Rafael C. Cardoso, Angelo Ferrando, Daniela Briola, Claudio Menghi, and Tobias Ahlbrecht (Eds.). Open Publishing Association, 117–125. https://doi.org/10.4204/EPTCS.319.9

[21] M. Wooldridge and A. Rao (Eds.). 1999. *Foundations of Rational Agency.* Kluwer Academic Publishers.

[22] H. L. A. Younes and R. G. Simmons. 2004. Solving Generalized Semi-Markov Decision Processes using Continuous Phase-type Distributions. In *Proc. AAAI*. AAAI Press, 742–747.