Encapsulating Reactive Behaviour in Goal-Based Plans for Programming BDI Agents

Extended Abstract

Rafael H. Bordini
School of Technology, PUCRS
Porto Alegre, RS, Brazil
rafael.bordini@pucrs.br

Rem Collier
University College of Dublin
Dublin, Ireland
rem.collier@ucd.ie

Jomi F. Hübner
DAS, Federal University of Santa Catarina
Florianópolis, SC, Brasil
jomi.hubner@ufsc.br

Alessandro Ricci
DISI, University of Bologna
Cesena, Italy
a.ricci@unibo.it

ABSTRACT

Reactive behaviour in Belief Desire Intention (BDI)-based models and architectures adopted in agent programming is typically specified in terms of reactive plans not bound to any specific goal. In this paper, we present and discuss an extension of the plan model used in BDI programming languages in which goal-based plans encapsulate both proactive and reactive behaviour. This brings important benefits both to the practice of agent programming and in supporting agent reasoning at runtime. The approach is evaluated through concrete implementations based on two existing agent programming platforms, namely Jason and ASTRA.

CCS CONCEPTS

• **Computing methodologies** → **Intelligent agents**; • **Software and its engineering** → **Context specific languages**;

KEYWORDS

Agent-Oriented Programming; Agent Programming Languages; BDI; AgentSpeak(L); Jason; ASTRA; AgentSpeak(ER)

ACM Reference Format:
Rafael H. Bordini, Rem Collier, Jomi F. Hübner, and Alessandro Ricci. 2020. Encapsulating Reactive Behaviour in Goal-Based Plans for Programming BDI Agents. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 3 pages.

1 PLAN ENCAPSULATION IN BDI AGENT PROGRAMMING

Plans have a key role in programming BDI agents [10], regardless of the particular implementation or agent programming language/platform considered. Plans define the agent know-how, they express a course of action that can be used to bring about a state-of-affairs, particularly those that are desirable to the agent, hence plans have a strong relation to goal-orientation. A plan has either a goal to achieve or to maintain and so they form an essential part of agents' behaviour.

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

Listing 1 Jason implementation of CNP initiator

```
1  +!cnp(I,Task) <- announce_cfp(I,Task); !bids(I).
2
3  +!bids(I) <- .at("now +4 seconds", { +!contract(I) }).
4
5  +propose(I,_) : all_ans(I) <- !contract(I).
6  +refuse(I)    : all_ans(I) <- !contract(I).
7
8  +!announce_cfp(I,Task) <- ...
9  +!contract(I) : not .intend(contract(I)) <- ...
```

The plan model adopted in all concrete BDI-based computational platforms — from the very early days (e.g., PRS [7], dMARS [4], AgentSpeak(L) [9], JAM [6], JACK [13], and SPARK [8]) to more recent programming languages (e.g., CAN [11], Jason [1], ASTRA [2], and Gwendolen [3]) — have pitfalls that long-term experience has helped surface. This paper in particular focusses on issues that concern *plan encapsulation*. For plan encapsulation here we refer to how much a developer can include/encapsulate — when defining a plan — aspects that concerns both the state of affairs for which the plan has been devised and the strategy to bring about it. In that concern, the plan model that comes from the original dMARS specification [4] — adopted by all BDI-based computational platforms — suffers of two main weaknesses, described in the following.

The first is about the possibility to have plans *without explicit goals*. In the basic plan model, a plan can be triggered by a *trigger* or *invocation* condition [12] which specifies the circumstances under which the plan should be considered relevant. The invocation condition could be both events concerning new goals or belief changes related to percepts from the environment or data in general. The latter case makes it possible to define reactive behaviours and data/event-driven processing. Listing 1 shows an example in Jason of an agent playing the role of initiator in a Contract Net Protocol [5]. The agent program includes a plan used to handle a new goal to achieve (`+!cnp(I,Task)` in line 1), plans handling subgoals (`+!bids`, `+!contract`), and plans that define the reactive behaviour used to handle new proposals (`+propose(I,_)` and refuses ((`+refuse(I,_)`). At runtime, for example, when an agent perceives a new proposal (`+propose(I,_)`), a new intention is created to carry on the execution of the plan. The problem is that

this intention is goal-less, i.e., it does not have an explicit "state of affairs" associated to it. In plans triggered by invocation conditions that concern environment/data events, the state of affairs remains implicit, in the mind of the designer. However, in the practice of agent programming, even purely reactive behaviours (e.g., a robot reacting to a low-battery charge event) are always targeted to some task devised at the design level (e.g., a maintenance task to keep the battery level not lower than some threshold).

The second issue, which is related to the first one, is about the impossibility to encapsulate reactive behaviour in the body of a plan. In the basic plan model, the *body* of a plan defines a potentially quite complex course of actions which may consist of both goals (or subgoals) and primitive actions [12]. In many relevant cases in practice (such as the CNP example), such a recipe may include the capability to asynchronously react to events from the environment, mixing proactive and reactive behaviour, but in the context of the same intention. The model for plans (body) in BDI, i.e., a sequence of actions and (sub-)goals, does not make this straightforward. If we need to react to some event e in the context of a plan to achieve some goal G, then a different plan specifying e as the triggering condition must be used. The effect is a poor level of encapsulation about the plan strategy, which must be necessarily specified in terms of a set of unrelated plans. As in the previous case, the relation between the plans is in the mind of the designer, but is neither expressed explicitly in the source code nor is it captured by intentions at runtime. This characteristic limits the agent reasoning about its own intentions.

2 AN EXTENSION OF THE PLAN MODEL

To deal with these issues, we propose an extension of the plan model adopted in BDI agents which:

- enforces goal/task orientation, that is: every plan p has an explicit account for the task t to be either achieved or maintained. The unique invoking condition is always a goal/task to be achieved or maintained. This implies that every intention at runtime — as a plan in execution — is bound to an explicit goal.
- extends the plan specification to include both subplans and reactive behaviour, besides a plan body having sequences of actions and subgoals, so as to get full encapsulation of proactivity and reactivity in the definition of the strategy of a plan.

We defined an abstract formal language to capture the main aspects of the extension, including a new reasoning cycle extending the traditional one. The abstract language has been used then as a reference to implement and experiment with the model in concrete agent programming platforms. In particular, two extensions have been developed: Jason(ER) and ASTRA(ER), available as an open-source project.[1]

To have a taste of the extension, in the following we revisit the CNP example in Jason(ER) — other examples are available in the project distribution. Listing 2 shows the CNP program in Jason(ER). In this version, the plan to achieve the goal of running a CNP *encapsulates* a main body (line 5) and three sub-plans (lines 7–15). The plan to achieve goal `bids` (lines 7–12) encapsulates as well both proactive and reactive behaviours. The former as a sequence of actions (the body) and the latter as a set of (encapsulated) reactive rules. The reaction to answers is defined in the context of the goals `bids` and `cnp`. An agent knows therefore why (for which goal) it is executing those reactive rules.

Listing 2 Jason(ER) implementation of CNP initiator

```
1  all_ans(I) :- ... // true if all participants answered
2
3  // plan to achieve goal cnp, I identifies a CNP instance
4  +!cnp(I,Task) {
5      <- announce_cfp(I,Task); !bids(I);  !contract(I).
6
7      +!bids(I) {
8          <- .wait(4000); .done.
9          // reaction to event of new proposal / refusal
10         +propose(I,_) : all_ans(I) <- .done.
11         +refuse(I)    : all_ans(I) <- .done.
12     }
13
14     +!announce_cfp(I,Task) <- ...
15     +!contract(I) <- ...
16 }
```

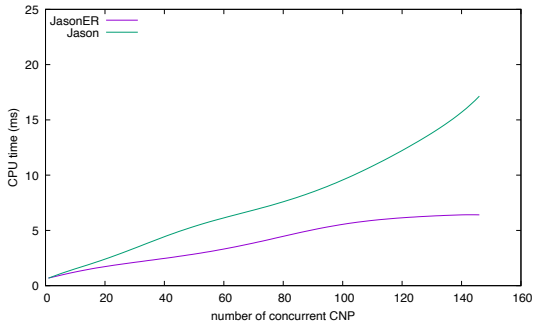

Figure 1: Jason(ER) performance evaluation.

First benchmarks show that the implementation of the extension does not negatively impact on the agent performance. [2] We evaluated how the approach scales considering a MAS that concurrently runs n CNPs. It is expected that the time required to finish n CNPs increases linearly on n. The MAS has one agent playing initiator and eleven playing participant. Only the initiator uses the new features of Jason(ER) as shown in Listing 2. The result of the experiment, shown in Figure 1, confirms that Jason(ER) scales linearly on the number of CNPs, getting even better performances than the original Jason version. Similar results are obtained with ASTRA(ER).

3 FUTURE WORK

Future work includes implementing the model in other languages and further evaluating its performance. More importantly, we aim to develop more complex systems with the extended languages in order to fully assess the practical impact of our approach on agent development.

[1] https://github.com/agentspeakers

[2] Benchmarks (source code) are available at https://github.com/agentspeakers

REFERENCES

[1] Rafael Bordini and Jomi Hübner. 2006. BDI Agent Programming in AgentSpeak Using Jason. In *CLIMA VI*, Francesca Toni and Paolo Torroni (Eds.). LNAI, Vol. 3900. Springer, 143–164.

[2] Rem W. Collier, Sean Edward Russell, and David Lillis. 2015. Reflecting on Agent Programming with AgentSpeak(L). In *PRIMA 2015: Principles and Practice of Multi-Agent Systems - 18th International Conference, Bertinoro, Italy, 2015, Proceedings (Lecture Notes in Computer Science)*, Qingliang Chen, Paolo Torroni, Serena Villata, Jane Yung-jen Hsu, and Andrea Omicini (Eds.), Vol. 9387. Springer, 351–366.

[3] Louise A. Dennis and Berndt Farwer. 2008. Gwendolen: A BDI Language for Verifiable Agents. In *Logic and the Simulation of Interaction and Reasoning*, Benedikt Löwe (Ed.). AISB, Aberdeen. AISB'08 Workshop.

[4] Mark D'Inverno, Michael Luck, Michael Georgeff, David Kinny, and Michael Wooldridge. 2004. The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System. *Autonomous Agents and Multi-Agent Systems* 9, 1-2 (July 2004), 5–53. https://doi.org/10.1023/B:AGNT.0000019688.11109.19

[5] Foundation for Intelligent Physical Agents. 2000. *FIPA Contract Net Interaction Protocol*. Geneva, Switzerland. http://www.fipa.org

[6] Marcus J. Huber. 1999. JAM: A BDI-theoretic Mobile Agent Architecture. In *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS '99)*. ACM, New York, NY, USA, 236–243. https://doi.org/10.1145/301136.301202

[7] Francois F. Ingrand, Michael P. Georgeff, and Anand S. Rao. 1992. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert: Intelligent Systems and Their Applications* 7, 6 (Dec. 1992), 34–44. https://doi.org/10.1109/64.180407

[8] David Morley and Karen Myers. 2004. The SPARK Agent Framework. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS '04)*. IEEE Computer Society, Washington, DC, USA, 714–721. http://dl.acm.org/citation.cfm?id=1018410.1018821

[9] Anand S. Rao. 1996. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings (Lecture Notes in Computer Science)*, Walter Van de Velde and John W. Perram (Eds.), Vol. 1038. Springer, 42–55.

[10] Anand S. Rao and Michael P. Georgeff. 1995. BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multiagent Systems, June 12-14, 1995, San Francisco, California, USA*, Victor R. Lesser and Les Gasser (Eds.). The MIT Press, 312–319.

[11] Sebastian Sardina and Lin Padgham. 2011. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems* 23, 1 (01 Jul 2011), 18–70.

[12] Sebastian Sardina and Lin Padgham. 2011. A BDI Agent Programming Language with Failure Handling, Declarative Goals, and Planning. *Autonomous Agents and Multi-Agent Systems* 23, 1 (July 2011), 18–70. https://doi.org/10.1007/s10458-010-9130-9

[13] Michael Winikoff. 2005. JACK^{TM} Intelligent Agents: An Industrial Strength Platform. In *Multi-Agent Programming: Languages, Platforms and Applications*, Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni (Eds.). Multiagent Systems, Artificial Societies, and Simulated Organizations, Vol. 15. Springer, 175–193.