Actions with Durations and Failures in BDI Languages[1]

Louise A. Dennis and **Michael Fisher**[2]

Abstract. BDI programming languages provide a well developed route to implementing intelligent agents. However, as such agents are increasingly being used in physical environments their treatment of external actions needs to be improved. In this paper we describe a mechanism for handling actions which have durations and failures.

1 Introduction

BDI (*Belief-Desire-Intention*; [3]) languages typically model interaction with the external environment either as an *action* or as a *capability*. It is generally implicitly assumed that these do not take long and usually execution of the program waits for them to complete before processing other intentions, goals and plans.

Agents are increasingly being used as the discrete components of *hybrid systems*. In these situations real actions may actually take considerable time to complete. Therefore, we do not want the agent program to suspend, but to continue operating, in order to perform error monitoring etc. Although *ad hoc* solutions to this problem exist, these frequently involve treating the action/capability as the *initiation* of an interaction with the environment. Perception is then used to judge when the interaction has concluded. Furthermore an action may need to be *aborted* while it is still executing.

Several life-cycles for goal processing in BDI agents have been proposed. It is agreed that goals need to transition through a number of states, including a Suspend state in which execution of any plans associated with the goal is halted and an Active state in which the goal is being processed. We adopt the semantics presented by Harland et al. [2] This semantics provides a comprehensive account of the goal life-cycle presented in the generic CAN (Conceptual Agent Notation) formal system [4] which we also adopt here.

Goals are represented as a tuple, $\langle I, G, Rules, State, P\rangle$ where I is a unique identifier for the goal, G is the goal type (`a` (achieve), `m` (maintain), `p` (perform) or `t` (test)), $Rules$ are a set of condition-goal action pairs. These govern how the goal moves between states. $State$ is the current goal state and P is the current plan body associated with the goal (if any). Further, ϵ indicates the absence of any plan body, nil a trivially successful plan and $fail$ a trivially unsuccessful plan. Harland et. al [2] assume that *means-end reasoning* is employed to select plan bodies. In our semantics for the addition of capabilities to this framework we will sometimes choose to specify the outcome of this means-end reasoning.

This paper is a shortened version of [1].

2 Adding Capabilities to the Goal life-cycle

We treat actions as capabilities and represent them as a tuple $\langle C, Pre, Post, \phi_s, \phi_f, \phi_a\rangle$, where C is an identifier for the capability, Pre and $Post$ are pre- and post-conditions, and ϕ_s, ϕ_f and ϕ_a are logical conditions for when the capability has "completed and succeeded", "completed and failed", or is "ongoing but in need of an abort". We will, for convenience, refer to a as C, and where relevant to its components as $Pre(C)$, $Post(C)$, etc. Where a capability has a non-trivial abort condition we will assume it also has a paired capability, $abort(C)$. We want condition-goal action rules that are active only while an interaction is being undertaken. Therefore we partition the set, $Rules$, into $\langle \mathcal{R}, \mathcal{R}_C\rangle$ where $\mathcal{R}_C$ is a dedicated set of condition-goal action rules associated with some capability C. We use the notation $a; P$ to indicate the sequential composition of some activity and a plan. Hence $C; P$ indicates that capability C is the next activity in some plan.

$$\frac{B \models Pre(C) \quad \mathbf{do}(C)}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, C; P\rangle\}\rangle \longrightarrow \\ \langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \{\langle \phi_s(C), \mathsf{reactivate}\rangle, \\ \langle \phi_f(C), \mathsf{reconsider}\rangle, \langle \phi_a(C), \mathsf{reactivate}\rangle\}\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle\end{array}} \tag{9}$$

Rule (9) adapts [2]'s semantics for the suspension of an active goal. We suspend the goal and add a set of rules which govern how the agent should react to *success*, *failure* or *abort* conditions. $\mathbf{do}(C)$ represents the activation of control systems to execute C.

Both *success* and *abort* states cause the goal to be reactivated. In [2] reactivation of a goal causes it to move to the Act state and replaces its plan body with ϵ (no plan). We specialise this rule.

$$\frac{\langle \phi_s(C), \mathsf{reactivate}\rangle \in \mathcal{R}_C \quad B \models \phi_s(C))}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \\ \langle B \cup Post(C), \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, P\rangle\}\rangle\end{array}} \tag{10}$$

Rule (10) controls the successful conclusion of a capability.

$$\frac{\langle \phi_a(C), \mathsf{reactivate}\rangle \in \mathcal{R}_C \quad B \models \phi_a(C)}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \\ \langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, abort(C); fail\rangle\}\rangle\end{array}} \tag{11}$$

Rule (11) handles aborts.

$$\frac{\langle \phi, \mathsf{reconsider}\rangle \in \mathcal{R}_C \quad B \models \phi}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \\ \langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Pnd}, \epsilon\rangle\}\rangle\end{array}} \tag{12}$$

Rule (12) covers the case where the capability completes but with failure. In the Pnd state goals are explicitly under consideration for either adopting and pursuing, or dropping.

3 Semantics in Action: A Simple Example

Consider a simple wheeled robot with two capabilities, $turn(\theta)$ (which turns through an angle, θ) and a $move(D)$ which moves forward a distance, D. Below is a simple $move(D)$ capability.

$$\begin{array}{l}\langle move(D), at(X,Y) \wedge angle(\theta), \top, \\ \quad \neg motors\_on \wedge at(X + Dsin(\theta), Y + Dcos(\theta)), \\ \quad \neg motors\_on \wedge \neg at(X + Dsin(\theta), Y + Dcos(\theta)), \bot\rangle\end{array}$$

[1] Work funded by EPSRC Project EP/J011770: "Reconfigurable Autonomy"

[2] University of Liverpool. email: `L.A.Dennis@liverpool.ac.uk`

Beliefs	**Goal**	
$\{at(0,1), angle(0)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, move(1) \rangle$	(1)
$\{at(0,1), angle(0), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \{\langle \neg motors\_on \wedge at(0,2), \mathsf{reactivate} \rangle,$ $\langle \neg motors\_on \wedge \neg at(0,2), \mathsf{reconsider} \rangle\}\rangle, \mathsf{Ssp}, move(1) \rangle$	(2)
$\{at(0,2), angle(0)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, nil \rangle$	(3)
$\{at(0.25, 0.3), angle(0)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{reconsider}, \epsilon \rangle$	(4)

Figure 1. Execution of a Wheeled Robot Agent with a simple Move capability

Beliefs	**Goal**	
$\{pp(0.5,1.5), cp(0.6,1.6), angle(90), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \{\langle \neg motors\_on \wedge at(0,2), \mathsf{reactivate} \rangle,$ $\langle \neg motors\_on \wedge \neg at(0,2), \mathsf{reconsider} \rangle,$ $\langle pp(X', Y') \wedge cp(X'', Y'') \wedge$ $\sqrt{X'^2 + (2-Y')^2} > \sqrt{X''^2 + (2-Y'')^2}, \mathsf{reactivate} \rangle\}\rangle,$ $\mathsf{Ssp}, move(1) \rangle$	(5)
$\{pp(0.5,1.5), cp(0.6,1.6), angle(90), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, abort(move(1)); fail \rangle$	(6)
$\{pp(0.5,1.5), cp(0.6,1.6), angle(90), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \{\langle \neg motors\_on, \mathsf{reactivate} \rangle\}\rangle, \mathsf{Ssp}, abort(move(1)); fail \rangle$	(7)
$\{pp(0.5,1.5), cp(0.6,1.6), angle(90)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, fail \rangle$	(8)

Figure 2. Execution of a Wheeled Robot Agent with a Move capability with an *abort* condition

This precondition determines the location of the robot and the direction it is facing. We assume perception informs the agent when its motors are engaged ($motors\_on$). It has no abort condition.

The agent's goal is to reach $(0, 2)$ identified as, $r02$. The agent's initial belief base is $\{at(0,1), angle(0)\}$. As a result of means-end reasoning it adopts the plan $move(1)$. The agent state at this point is shown in (1). For presentational reasons we have split the tuple $\langle B, \mathcal{G} \rangle$ and only show the single goal we are interested in, not the set of all goals. After (9) fires, the motors are engaged, and the goal is suspended. The new state is shown in (2). Beliefs continue to be updated via perception. Eventually the agent believes $at(0,2)$ and no longer believes $motors\_on$. (10) fires and the agent is in state (3).

If the robot at $(0.25, 0.3)$ when the motors stop, the agent transitions using (12) to (4).

We add two beliefs, $cp(X,Y)$ and $pp(X,Y)$ – the agent's current and previous position – and an abort condition and capability:

$$\langle move(D), at(X,Y) \wedge angle(\theta), \top,$$
$$\neg motors\_on \wedge at(X + Dsin(\theta), Y + Dcos(\theta)),$$
$$\neg motors\_on \wedge \neg at(X + Dsin(\theta), Y + Dcos(\theta)),$$
$$motors\_on \wedge pp(X', Y') \wedge cp(X'', Y'') \wedge$$
$$\sqrt{(X + Dsin(\theta) - X')^2 + (Y + Dsin(\theta) - Y')^2} >$$
$$\sqrt{(X + Dsin(\theta) - X'')^2 + (Y + Dsin(\theta) - Y'')^2} \rangle$$
$$\langle abort(move(D)), motors\_on, \top, \neg motors\_on, \bot, \bot \rangle \quad (13)$$

If the agent's state is that shown in (5), it transitions, via (11), to (6). The abort capability is invoked and (9) transitions the agent to (7). Assuming the motors stop this will then transition to (8).

4 Conclusion

We have argued that BDI representations of interactions with the environment need to account for actions taking time to complete and aborts. We have extended the semantics for the life-cycle of goals presented in [2] to show how the declarative representation of capabilities with durations, failures and aborts can be integrated with this semantics.

REFERENCES

[1] Louise A. Dennis and Michael Fisher, 'Actions with durations and failures in bdi languages', Technical Report SomeNumber, University of Liverpool, (2014).

[2] J. Harland, D. N. Morley, J. Thangarajah, and N. Yorke-Smith, 'An Operational Semantics for the Goal Life-Cycle in BDI Agents', *Auton. Ag. and M.-Ag. Sys.*, **28**(4), 682–719, (2014).

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

[4] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah, 'Declarative Procedural Goals in Intelligent Agent Systems', in *Proc. KR&R*, pp. 470–481, (2002).