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 to work in, and interact with, real 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. In particular we note that where an action has a duration it acquires two failure modes, one which requires an abort. We provide a formal semantics for this mechanism which is general enough to be used to add such actions to a wide range of BDI programming languages.

1 Introduction

The BDI model (*Belief-Desire-Intention*; [14]) is the predominant mechanism for describing deliberative, intelligent, agents. Consequently, a range of agent programming languages have been developed based on this approach, for example *Jason* [3], GWENDOLEN [6], 3APL [5] and GOAL [10]. BDI languages typically model interaction with the external environment either as an *action* or as a *capability*. It is generally implicitly assumed that an action or capability does not take long to finish and usually execution of the BDI program waits for the action to complete before processing other intentions, goals and plans.

The concept of an *agent* is increasingly being used in real-world applications, sometimes to control quite complex physical systems. Agent-based control can be found in cyber-physical and autonomous systems and, more generally, an agent is often seen within the discrete component of a *hybrid system*. Typical examples of the types of systems of relevance include ground-based mobile robots, spacecraft, unmanned aircraft and robot arms. (These are ones that we are using agent-based hybrid control for, and all utilise BDI agents at their heart; for example [7].) In these situations real actions may actually take considerable time to complete.

Consider the example of an autonomous spacecraft exploring an asteroid field which we implemented in GWENDOLEN and executed in a simulated environment created through a combination of MATLAB and jBullet (as described in [11]). In this scenario the spacecraft chooses an asteroid, calculates a path to intersect an orbit with the asteroid and then activates the control system which will follow and monitor the path. As the spacecraft moves, the high level agent control continues to monitor sensors and can react to the failure of a thruster by changing fuel lines and modifying the thruster control. In our implementation of this scenario, an action is used to initiate the "following of a path". The intention which contains the plan for moving to, and orbiting, an asteroid then suspends until the spacecraft is actually in position. This is achieved using a *wait until* ϕ construct available in the GWENDOLEN language. In the meantime, other monitoring procedures and intentions can continue to execute, allowing thruster failures to be handled in a timely fashion while the spacecraft moves. If the goal becomes unachievable (because, for instance, of catastrophic thruster failure) then separate plans had to be constructed to recognise the event, explicitly drop the goal and send 'abort' messages to the underlying control systems that were attempting the maneouvre. These plans were not directly linked with the action of moving into a particular orbit around some asteroid.

In such hybrid systems, we do not want the agent program to suspend during interaction, but to continue operating, in order to perform error monitoring etc. Although *ad hoc* solutions to this problem exist (not least in some of our own autonomous systems) these frequently involve treating the action/capability as the *initiation* of an interaction with the environment only. Perception is then used to judge when the interaction has concluded. Furthermore an action that takes significant time may need to be *aborted* while it is still executing (e.g., by sending the underlying hardware a "stop" signal).

A particular focus of interest at present is on if (and how) high-level agent control can assist with managing *graceful degradation*. Graceful degradation is a system's ability to adapt its behaviour in the event of the failure or damage to some of its (physical) sub-systems in such a way that it can continue to achieve some, or all of its goals. Managing graceful degradation draws on a range of techniques from across artificial intelligence and control engineering, such as learning techniques, and adaptive and reactive control. However, where a BDI-agent is involved at the highest (decision-making) level of the system we want such an agent to *recognise* that a component is no longer behaving as expected, and then *invoke* diagnosis subsystems and possibly *reconfigure* lower-level control processes. The first of these issues, *recognising* that a capability has not behaved as expected, is related to this paper's treatment of interactions with a significant duration within BDI programming languages.

In this paper we propose *principled* mechanisms and semantics for integrating actions and capabilities within BDI programming languages, which are presumed to take significant time and which may fail both during and at the end of their execution. We link our integration with work on goal life-cycles that have *active*, *suspend* and *abort* stages for goals and we show how these interact with actions/capabilities comprising both durations and potential failures.

2 Preliminaries and Terminology

We are seeking a generic description of interactions with the environment that is independent of any particular implementation of BDI programming. As such we need to consider some of the common representations of interactions in BDI languages.

Actions. Many BDI languages represent environmental interaction as an atomic *action*. When an action is invoked it executes some low level code, invisible at the BDI-level. While such an action is often

[1] Work funded by EPSRC Project EP/J011770: "Reconfigurable Autonomy"
[2] University of Liverpool. email: L.A.Dennis@liverpool.ac.uk

a *ground* atomic command, and its effect is judged via agent perception, it can return results either via the instantiation of variables in the action expression (using unification) or as an explicit return value (typically, 'success' or 'failure'). Languages which treat interaction in this way include *Jason*, GWENDOLEN and 3APL.

Capabilities. Interaction may also be modelled as *capabilities*. These have explicit *pre-* and *post*-conditions such that the interaction is executed only if the pre-conditions are true and, after the interaction has concluded, the post-conditions are asserted explicitly by the language. Other effects may be subsequently observed via perception mechanisms. It is possible that a capability executes no low-level code, particularly when an agent is executing in some simulated setting where it is considered sufficient to use just the post-conditions to represent the result of the interaction. Languages which treat interaction in this way include GOAL and 3APL.

In this paper treat interactions as capabilities, as we consider these to provide a more general representation. An action such as `move_to(X,Y)`[3] can be represented as a capability $\{True\}$ `move_to(X,Y)` $\{True\}$ (i.e., a capability with trivial pre- and post-conditions) and even where an action returns a result this can be captured via unification and post-conditions.

2.1 Goal Life-cycles in BDI Languages.

Several teams of researchers have proposed life-cycles for goal processing in BDI agents [13, 19, 9]. While there are differences in detail, there is also significant agreement. 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, either by the execution or the selection of a plan. In this paper we will adopt the semantics presented by Harland et al. [9] (the most recent elaboration of this work) as our starting point. This semantics provides a comprehensive account of the goal life-cycle, as shown in Figure 1. Goals can be in a number of states: Pending; Active; Monitoring; Suspended; and Aborting (abbreviated to Pnd, Act, Mn, Ssp and Ab in what follows). Specific *goal actions* (e.g., activate, reconsider, respond) move a goal between states in its life-cycle.

Four types of goals are in common use [19]: *achieve*; *perform*; *test*; and *maintain*. Work in [9] shows how *perform* goals (which motivate some activity but have no success criteria) and *test* goals (which seek to ascertain the truth of some fact or belief) can be represented as *achieve* goals (which motivate some activity in order to achieve some state of the world) and so we focus on achieve goals. Maintain goals motivate activity with a view to preserving some state of the world. We will discuss these separately in Section 3.3

Harland et. al [9] present an operational semantics for the goal life-cycle in the CAN (Conceptual Agent Notation) formal system [20]. CAN is sufficiently general that it can be used to specify the semantics of an operational BDI system in a generic way but one with sufficient detail that a mapping to an implementation is clear.

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 [9] assume that *means-end reasoning* is employed to select plan bodies, in particular, that such a reasoning mechanism is used whenever a goal is unsuspended to decide whether to continue with the existing plan body or a new plan. In our semantics for the addition of capabilities to this framework we will sometimes choose to specify the outcome of this means-end reasoning instead of delegating it to the underlying language.

A CAN agent is a tuple $\langle B, \mathcal{G}\rangle$ of a set of beliefs and a set of goals. This is an idealisation of the semantics of many BDI languages but works as a solid general framework. Rules are presented as

$$\frac{Condition}{\langle B, \mathcal{G}\rangle \longrightarrow \langle B', \mathcal{G}'\rangle}$$

Here $Condition$ is some property that must hold before the transition can take place, $\langle B, \mathcal{G}\rangle$ is the state of the agent before the transition, and $\langle B', \mathcal{G}'\rangle$ is the state of the agent after the transition.

We assume that, if an atomic plan execution step does not complete, then no further activity can take place on any goal until that step has finished execution. In Section 3.4, we briefly touch on how our semantics would change in the presence of parallel goal handling.

3 Adding Capabilities to the Goal life-cycle

We represent capabilities 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 expressions describing the state where the capability has "completed and succeeded", "completed and failed", or is "ongoing but in need of an abort". We will, for convenience, refer to the capability $\langle C, Pre, Post, \phi_s, \phi_f, \phi_a\rangle$ as capability, 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, $abort(C)$, which is a control for aborting capability C.

We aim to have condition-goal action rules that are active only while an interaction is being undertaken. Therefore we choose to 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.

3.1 Capabilities associated with Achieve Goals

We start with the semantics for a capability invoked as part of a plan for handling an achieve goal. 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.

We want activity on the goal to suspend until the capability has succeeded, failed or needs to be aborted.

$$\frac{B \models Pre(C) \quad \mathbf{do}(C)}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, C; P\rangle\}\rangle \longrightarrow \\ \langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \{\langle \phi_s(C), \mathsf{reactivate}\rangle, \\ \langle \phi_f(C), \mathsf{reconsider}\rangle, \langle \phi_a(C), \mathsf{reactivate}\rangle\}\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle\end{array}} \quad (1)$$

Rule (1) adapts [9]'s semantics for the suspension of an active goal. Where a capability's preconditions are implied by the current beliefs of the agent, we suspend the goal and add a set of rules which govern how the agent should react in the event that the capability's *success*, *failure* or *abort* conditions come to hold. $\mathbf{do}(C)$ represents the activation of control systems to execute C.

Both *success* and *abort* states cause the goal to be reactivated. In [9] reactivation of a goal causes it to move to the Act state and replaces its plan body with the non-existent plan, ϵ. The assumption is that means-end reasoning will typically reinstate the pre-existing plan body. We specialise this rule so that, in the event of successful completion of the interaction, the plan body continues processing as

[3] Capital letters represent variables that can be instantiated by unification.

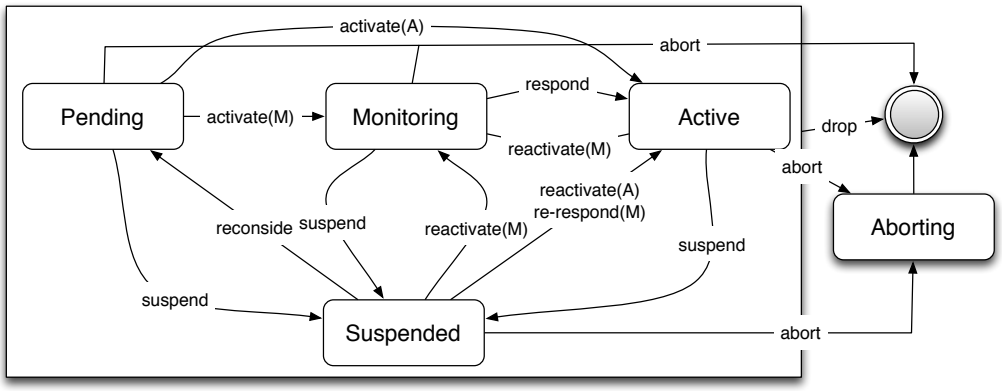

Figure 1. The Goal Life-Cycle from Harland et. al [9].

normal while, in the case where the interaction should be aborted, the associated abort capability is invoked.

$$\frac{\langle \phi, \mathsf{reactivate}\rangle \in \mathcal{R}_C \quad B \models \phi \quad \phi \equiv \phi_s(C)}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \\ \langle B \cup Post(C), \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, P\rangle\}\rangle\end{array} \quad (2)$$

Rule (2) controls the successful conclusion of a capability. If its success condition ϕ_s arises then processing of the current plan continues, the post-conditions are asserted and the goal becomes active once more. The remaining condition-goal action pairs associated with C (captured as $\mathcal{R}_C$) are then removed.

$$\frac{\langle \phi, \mathsf{reactivate}\rangle \in \mathcal{R}_C \quad B \models \phi \quad \phi \equiv \phi_a(C)}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \\ \langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, abort(C); fail\rangle\}\rangle\end{array} \quad (3)$$

Rule (3) also moves the goal back to the Act state. The previous plan is replaced by the plan $abort(C); fail$, indicating that the system needs to take some new action to cancel the ongoing action and then the existing plan will have failed (In [9], the *fail* plan will eventually lead to a new round of means-end reasoning to re-plan the goal). We could include, if appropriate, the condition-action pair $\langle \phi_a(C) \wedge done(abort(C)), \mathsf{abort}\rangle$ in $\mathcal{R}$, which would cause the whole goal to *abort* once the capability itself has aborted. In many cases however, although C has been aborted, we want to continue working towards achieving the goal.

$$\frac{\langle \phi, \mathsf{reconsider}\rangle \in \mathcal{R}_C \quad B \models \phi}{\begin{array}{c}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \\ \langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Pnd}, \epsilon\rangle\}\rangle\end{array} \quad (4)$$

Rule (4) covers the case where the capability completes but with failure. The postconditions are *not* asserted and the goal is moved into the Pnd state. In the Pnd state goals are explicitly under consideration for either adopting and pursuing, or dropping. This rule is almost identical to the equivalent version from [9] except that it clears away the temporary condition-goal action pairs from $\mathcal{R}$ (again captured as $\mathcal{R}_C$) that were associated with capability, C.

3.2 Incorporating Time Stamps

Recall that one of our primary motivations is to extend BDI languages to help with handling graceful degradation. This will require an agent to track a capability's successes and failures and, over time, analyse patterns of repeated failures. Let us assume that the system can provide us with time stamps for activities, such as capability and goal actions. We will assume a distinguished time stamp, **ct**, for the current time when some event takes place. We also assume the existence of a predicate $\mathbf{ct}(T)$ which will instantiate T to the current time when it is evaluated. We extend rules (2)–(4) above to give (5)–(7) in Figure 2 to record successes and failures.

Furthermore, the existence of time stamps allows us to set default durations for capabilities in the system by adapting rule (1) to provide rule (8) in Figure 2. and including

$$\langle suspended(I, C, T) \wedge \mathbf{ct} > T + timeout, \mathsf{reconsider}\rangle \quad (10)$$

in the goal's $Rules$. This considers any capability to have failed if it takes longer than $timeout$ for its success state to occur.

Of course, we may not want a global default timeout for capabilities (since some may be expected to take longer than others) in which case time out conditions could also be added to ϕ_s, ϕ_f or ϕ_a on a case-by-case basis as part of the *success*, *failure* or *abort* conditions.

3.3 Maintenance Goals and Monitoring

The Monitoring state is exclusively reserved for *maintenance* goals. These goals may have a predictive mechanisms which allow them to determine that the state they seek to maintain is about to be violated. When the state is (or is about to be) violated the goal moves into the Act state and creates a new achieve subgoal, G_s, which controls plans to preserve or recover the desired state. We assume, therefore, that capabilities are never invoked in the Monitoring state and we do not need to alter its semantics.

It is not unreasonable to suppose that a capability, particularly one prone to failure might want to set up a maintenance goal to monitor the progress of the interaction. We can frame rule (1) instead as rule (9) in Figure 2, where I' is a fresh goal identifier. Rule (9) sets up a new maintenance goal instead of adding the condition-goal action pair $\langle \phi_a(C), \mathsf{reactivate}\rangle$ to $\mathcal{R}_C$. I' provides condition-goal action pairs that will respond in the event that $\phi_a(C)$ holds (or is considered at risk of holding) and will drop the maintenance goal if C succeeds, fails or is aborted.

3.4 Parallel Goal Processing

In a system where processing is *not* suspended when a particular activity in a plan is yet to complete, we would not need to explicitly suspend a goal to manage actions with durations. However, we still need to handle situations where a capability must be aborted while it

$$\frac{\langle \phi, \mathsf{reactivate}\rangle \in \mathcal{R}_C \quad B \models \phi \quad \phi \equiv \phi_s(C)}{\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \langle B \cup \{success(C, \mathbf{ct})\} \cup Post(C), \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, P\rangle\}\rangle} \quad (5)$$

$$\frac{\langle \phi, \mathsf{reactivate}\rangle \in \mathcal{R}_C \quad B \models \phi \quad \phi \equiv \phi_a(C)}{\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \langle B \cup \{abort(C, \mathbf{ct})\}, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, abort(C); fail\rangle\}\rangle} \quad (6)$$

$$\frac{\langle \phi, \mathsf{reconsider}\rangle \in \mathcal{R}_C \quad B \models \phi}{\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \mathcal{R}_C\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle \longrightarrow \langle B \cup \{fail(C, \mathbf{ct})\}, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Pnd}, \epsilon\rangle\}\rangle} \quad (7)$$

$$\frac{B \models Pre(C) \quad \mathbf{do}(C)}{\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, C; P\rangle\}\rangle \longrightarrow \langle B \cup \{suspended(I, C, \mathbf{ct})\}, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \{\langle \phi_s, \mathsf{reactivate}\rangle, \langle \phi_f, \mathsf{reconsider}\rangle, \langle \phi_a, \mathsf{reactivate}\rangle\}\rangle, \mathsf{Ssp}, C; P\rangle\}\rangle} \quad (8)$$

$$\frac{B \models Pre(C) \quad \mathbf{do}(C)}{\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \emptyset\rangle, \mathsf{Act}, C; P\rangle\}\rangle \longrightarrow \begin{array}{r}\langle B, \mathcal{G} \cup \{\langle I, G, \langle \mathcal{R}, \{\langle \phi_s(C), \mathsf{reactivate}\rangle, \langle \phi_f(C), \mathsf{reconsider}\rangle\}\rangle, \mathsf{Ssp}, C; P\rangle\} \\ \{\langle I', \mathtt{m}, \{\langle \phi_a(C), \mathsf{respond}\rangle, \langle \phi_s(C) \vee \phi_f(C) \vee done(abort(C)), \mathsf{drop}\rangle\}, \mathsf{Mn}, \epsilon\rangle\}\rangle\end{array} \quad (9)$$

Figure 2. Semantic Rules Incorporating Time Stamps.

is executing, so a semantics for attaching abort conditions to capabilities remains a requirement (e.g., by adopting a maintenance goal to monitor the capability's execution).

4 Semantics in Action: A Simple Example

To see our semantics in action, 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 definition of the $move(D)$ capability.

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

This capability has a precondition which determines the location of the robot and the direction it is facing (as an angle relative to the x-axis). It has a trivial post-condition. We assume perception informs the agent when its motors are engaged so $\neg motors\_on$ is a part of both its success and failure conditions – the capability has completed when the motors switch off. It has no abort condition.

Let us assume the agent's goal is to reach the location $(0, 2)$ 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 (11). 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 (1) fires, the motors are engaged, and the goal is suspended. The new state is shown in (12).

The goal is suspended. However beliefs continue to be updated via perception. Assuming the move interaction is successful then eventually the agent believes $at(0, 2)$ and no longer believes $motors\_on$. Now (2) fires. The new state is shown in (13). Further reasoning will then remove the goal as it has successfully concluded.

If the robot is not at $(0, 2)$ when the activity completes, e.g, it is at $(0.25, 0.3)$, then the failure condition holds and the agent transitions using (4) to (14). The existing plan is discarded and the robot is invited to reconsider the goal which could involve dropping it altogether or constructing a new plan.

Let us expand our agent so it has two special unique beliefs, $cp(X, Y)$ and $pp(X, Y)$ which store its current position and its previously measured position respectively. Now the programmer can supply an abort condition (that the agent is moving away from it's target) and an abort move capability which will stop the motors:

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

$$\langle abort(move(D)), motors\_on, \top, \neg motors\_on, \bot, \bot\rangle \quad (15)$$

Assume the robot's motor capabilities are damaged and it travels on a curved path that moves it away from its target. e.g., at some point it is at position $(0.5, 1.5)$ ($\sqrt{0.5}$ away from $(0, 2)$) and the next time its position is checked it is at $(0.6, 1.6)$ ($\sqrt{0.52}$ away from $(0, 2)$).

At this point in time, the agent's state will be that shown in (16). This transitions, via (3), to (17). The abort capability is invoked and (1) transitions the agent to (18). Assuming the motors stop this will then transition to (19). The agent can now once again employ means-end reasoning in an attempt to achieve the goal.

We now rework this example using time stamps. Consider the $abort(move(D))$ capability. While we assume it takes some time for the abort to stop the motors, we can assume this is only a short duration, e.g. 0.5 seconds. Therefore the *abort* command has failed in the situation where $motors\_on$ still holds 0.5 seconds after the capability was invoked (determined as the current time when the precondition was evaluated). The reworked capability therefore becomes:

$$\begin{array}{l}\langle abort(move(D)), motors\_on \wedge \mathbf{ct}(T), \top, \\ \quad \neg motors\_on, \mathbf{ct} > T + 0.5 \wedge motors\_on, \bot\rangle\end{array}$$

Assume the agent chooses to abort its move at time 5.6. The agent state shown in (18) will instead be that shown in (20) (we omit the agent's beliefs about position and angle for space reasons). Rule (8) has added $suspended(r02, abort(move(1)), 5.6)$ and rule (6) has

Beliefs	**Goal**	
$\{at(0,1), angle(0)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, move(1) \rangle$	(11)
$\{at(0,1), angle(0), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \{\langle \neg motors\_on \wedge at(0,2), \mathsf{reactivate} \rangle,$ $\langle \neg motors\_on \wedge \neg at(0,2), \mathsf{reconsider} \rangle\}\rangle, \mathsf{Ssp}, move(1) \rangle$	(12)
$\{at(0,2), angle(0)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, nil \rangle$	(13)
$\{at(0.25, 0.3), angle(0)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{reconsider}, \epsilon \rangle$	(14)

Figure 3. Execution of a Wheeled Robot Agent with a simple Move capability

Beliefs	**Goal**	
$\{pp(0.5, 1.5), cp(0.6, 1.6), angle(90), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \{\langle \neg motors\_on \wedge at(0,2), \mathsf{reactivate} \rangle,$ $\langle \neg motors\_on \wedge \neg at(0,2), \mathsf{reconsider} \rangle,$ $\langle pp(X', Y') \wedge cp(X'', Y'') \wedge$ $\sqrt{X'^2 + (2-Y')^2} > \sqrt{X''^2 + (2-Y'')^2}, \mathsf{reactivate} \rangle\}\rangle,$ $\mathsf{Ssp}, move(1) \rangle$	(16)
$\{pp(0.5, 1.5), cp(0.6, 1.6), angle(90), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, abort(move(1)); fail \rangle$	(17)
$\{pp(0.5, 1.5), cp(0.6, 1.6), angle(90), motors\_on\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \{\langle \neg motors\_on, \mathsf{reactivate} \rangle\}\rangle, \mathsf{Ssp}, abort(move(1)); fail \rangle$	(18)
$\{pp(0.5, 1.5), cp(0.6, 1.6), angle(90)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset \rangle, \mathsf{Act}, fail \rangle$	(19)

Figure 4. Execution of a Wheeled Robot Agent with a Move capability with an *abort* condition

added $abort(move(1), 5.5)$ to the belief base. If (catastrophically) the motors fail to disengage after half a second (so at a time after 6.1), then rule (7) causes the state to transition to (21).

Alternatively we could make a global assumption that no interaction takes longer than 0.5 second. In this case we retain our original definition of $abort(move(D))$, (15), but, instead, by default all goals have rule (10) in which case (20) would become (22).

Adding time stamped beliefs about the success, failure, and aborts related to capabilities will be necessary for making judgements about degradation. However any naive implementation is likely to prove impractical in the efficient evaluation of an agent's beliefs.

5 Related Work

We have already considered the seam of research on goal life-cycles that informs the framework proposed here. In general BDI languages do not explicitly treat actions as having durations. A notable exception is the *Brahms* language [16] in which actions, called *activities*, explicitly involve durations. In its original presentation, *Brahms* had no formal semantics. However, one has recently been provided [17] though this focuses primarily on the effect of duration on simulation without any formal framework for activity failure or monitoring.

The field of AI Planning has invested considerable effort in the modelling of actions and capabilities with durations and stochastic outcomes, both theoretically as variants on Markov Decision Procedures [12, 21] and practically capturing such concepts in planners (e.g. [4]) and domain description languages such as the PDDL 2.1 extension of PDDL [8]. In planning the effect of the action duration is of most importance during the generation of the plan, rather than its execution. Executable plans are represented as sequences of actions and lack the manipulation of mental states that is the defining feature of BDI approaches.

The modelling of actions with durations has been considered in logics for agency. Troquard et. al [18] represent these using continuations within STIT logic. The logic does not explicitly link the issue of durations with aborts.

There has been a great deal of work on plan failure in BDI programming languages (e.g., [2, 15]). This has not distinguished goal failure from capability failure. This is understandable, when a capability fails its most important effect is on the goal which will need to be dropped or re-planned. As a result work has focused on goal dropping and re-planning mechanisms which are captured in the work on BDI goal life-cycles already discussed.

Beliefs	**Goal**	
$\{motors\_on, suspended(r02, abort(move(1)), 5.6),$ $abort(move(1), 5.5)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \{\langle \neg motors\_on, \mathsf{reactivate}\rangle,$ $\langle \mathbf{ct} > 6.1 \wedge motors\_on, \mathsf{reconsider}\rangle\}\rangle, \mathsf{Ssp}, abort(move(1)); fail\rangle$	(20)
$\{motors\_on, \ldots, fail(abort(move(1)), 6.2)\}$	$\langle r02, \mathtt{a}, \langle \emptyset, \emptyset\rangle, \mathsf{Pnd}, \epsilon\rangle$	(21)
$\{motors\_on, suspended(r02, abort(move(1)), 5.6)$ $abort(move(1), 5.5)\}$	$\langle r02, \mathtt{a}, \langle \{\langle suspended(I, C, T) \wedge \mathbf{ct} > T + 0.5, \mathsf{reconsider}\rangle\},$ $\{\langle \neg motors\_on, \mathsf{reactivate}\rangle\}\rangle, \mathsf{Ssp}, abort(move(1)); fail\rangle$	(22)

Figure 5. Aborting a Move using time outs

6 Conclusion and Further Work

We have argued that BDI representations of interactions with the environment need to account for actions taking time to complete and aborts. There also needs to be a declarative link between the success, failure and abort conditions for such interactions and that this should be treated separately from the success, failure and abort of a goal.

We have extended the semantics for the life-cycle of goals presented in [9] to show how the declarative representation of capabilities with durations, failures and aborts can be integrated with this semantics. While we have ignored many practical details of implementation we believe that describing a technique in a high level fashion based primarily on the manipulation of beliefs and goals provides a good guide to implementation in any particular system.

Our presentation does require a language to extend its representation of actions/capabilities to include their success, failure and abort conditions. Handling success and abort will involve pairing the execution of the action/capability with the language specific mechanisms for goal suspension, while handling failure will involve pairing execution of the action/capability with mechanisms for goal and plan failure. Most mature languages have such mechanisms.

Of immediate interest to us is implementing the semantics presented. We would look to implement them in an existing language with suitable support for the agent control of hybrid systems.

A critical element of our interest in BDI-based control of adaptability and reconfiguration will be a system's ability to create new plans within the BDI program when changes to capabilities have been detected and diagnosed. This will require integration with AI planning systems. At present we are targetting PDDL 2.1 planners that can reason about durative actions [8].

Our formalisation is a generic framework for handling interactions with the real world, as they tend to arise in cyber-physical systems. It aims to overcome practical problems that arise with the current treatment of actions but that remain in keeping with the declarative philosophy of BDI languages.

REFERENCES

[1] *Multi-Agent Programming: Languages, Platforms and Applications*, eds., R. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, Springer, 2005.

[2] R. H. Bordini and J. F. Hübner, 'Semantics for the *Jason* Variant of AgentSpeak (Plan Failure and some Internal Actions)', in *Proc. ECAI*, pp. 635–640. IOS Press, (2010).

[3] R. H. Bordini, J. F. Hübner, and R. Vieira, 'Jason and the Golden Fleece of Agent-Oriented Programming', In Bordini et al. [1], 3–37.

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

[5] M. Dastani, M. Birna van Riemsdijk, and John-Jules Ch. Meyer, 'Programming Multi-Agent Systems in 3APL', In Bordini et al. [1], chapter 2, 39–67.

[6] L. A. Dennis and B. Farwer, 'Gwendolen: A BDI Language for Verifiable Agents', in *Proc. AISB Workshop on Logic and the Simulation of Interaction and Reasoning*. AISB, (2008).

[7] L. A. Dennis, M. Fisher, A. Lisitsa, N. Lincoln, and S. M. Veres, 'Satellite Control Using Rational Agent Programming', *IEEE Intelligent Systems*, **25**(3), 92–97, (May/June 2010).

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

[9] 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).

[10] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J.Ch Meyer, 'Agent Programming with Declarative Goals', in *Intelligent Agents VII*, volume 1986 of *LNAI*, pp. 228–243. Springer, (2001).

[11] N. Lincoln, S. M. Veres, L. A. Dennis, M. Fisher, and A. Lisitsa, 'Autonomous Asteroid Exploration by Rational Agents', *IEEE Computational Intelligence Magazine*, **8**(4), 25–38, (2013).

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

[13] M. Morandini, L. Penserini, and A. Perini, 'Operational Semantics of Goal Models in Adaptive Agents', in *AAMAS 2009*, pp. 129–136. IFAAMAS, (2009).

[14] 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).

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

[16] M. Sierhuis, *Modeling and Simulating Work Practice. BRAHMS: a multiagent modeling and simulation language for work system analysis and design*, Ph.D. dissertation, SWI, University of Amsterdam, SIKS Dissertation Series No. 2001-10, 2001.

[17] R. Stocker, M. Sierhuis, L.A. Dennis, C. Dixon, and M. Fisher, 'A Formal Semantics for Brahms', in *Proc. 12th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA)*, volume 6814 of *LNCS*, pp. 259–274. Springer, (2011).

[18] N. Troquard and L. Vieu, 'Towards a logic of agency and actions with durations', in *Proc. ECAI*, pp. 775–776. IOS Press, (2006).

[19] M. Birna van Riemsdijk, Mehdi Dastani, and John-Jules Ch. Meyer, 'Goals in Conflict: Semantic Foundations of Goals in Agent Programming', *Auton. Ag. and M.-Ag. Sys.*, **18**(3), 471–500, (2009).

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

[21] H. L. A. Younes and R. G. Simmons, 'Solving Generalized Semi-Markov Decision Processes using Continuous Phase-type Distributions', in *Proc. AAAI*, p. 742, (2004).