Operational Behaviour for Executing, Suspending, and Aborting Goals in BDI Agent Systems

John Thangarajah[1], James Harland[1], David Morley[2], and Neil Yorke-Smith[2,3]

[1] RMIT University, Melbourne, Australia. {johnt, james.harland}@rmit.edu.au
[2] SRI International, Menlo Park, USA. morley@AI.SRI.COM
[3] American University of Beirut, Lebanon. nysmith@aub.edu.lb

Abstract. Deliberation over and management of goals is a key aspect of an agent's architecture. We consider the various types of goals studied in the literature, including performance, achievement, and maintenance goals. Focusing on BDI agents, we develop a detailed description of goal states (such as whether goals have been suspended or not) and a comprehensive suite of operations that may be applied to goals (including dropping, aborting, and suspending and resuming them). We show how to specify an operational semantics corresponding to this detailed description in an abstract agent language (CAN). The three contributions of our generic framework for goal states and transitions are (1) to encompass both goals of accomplishment and rich goals of monitoring, (2) to provide the first specification of abort and suspend for all the common goal types, and (3) to account for plan execution as well as the dynamics of sub-goaling. A prototype implementation provides proof of concept.

1 Introduction

Deliberation over what courses of action to pursue is fundamental to agent systems. Agents designed to work in dynamic environments, such as a rescue robot or an online travel agent, must be able to reason about what actions they should take, incorporating deliberation into their execution cycle so that decisions can be reviewed and corrective action taken with an appropriate focus and frequency.

In systems based on the well-known *Belief-Desire-Intention* (BDI) framework [14], most often a set of *goals* is ascribed to the agent, which is equipped with various techniques to deliberate over and manage this set. The centrality of reasoning over goals is seen in the techniques investigated in the literature, which include subgoaling and plan selection, detection and resolution of conflicts [20, 17] or opportunities for cooperation [21], checking goal properties to specification [22, 12], failure recovery and planning [15, 16, 7], and dropping, suspending and resuming [19], or aborting goals [18]. A variety of goals are described in the literature, including goals of *performance* of a task, *achievement* of a state, *querying* truth of a statement, *testing* veracity of beliefs, and *maintenance* of a condition [3, 25].

An agent must manage a variety of goals, while incorporating pertinent sources of information into its decisions over them, such as (user) preferences, quality goals, motivational goals, and advice [22]. The complexity of agent goal management—which

stems from this combination of the variety of goals and the breadth of deliberation considerations—is furthered because each goal can be dropped, aborted, suspended, or resumed at arbitrary times. Note that while goals themselves are static (i.e., they are specified at design time, and do not change during execution), their behaviour is dynamic: a goal may undergo a variety of changes of state during its execution cycle [12]. This evolution may include its initial adoption by the agent, being actively pursued, being suspended and then later resumed, and eventually succeeding (or failing). (Maintenance goals have a subtle life-cycle: the goal is retained even when the desired property is true; it is possible that such goals are never dropped.)

This paper analyzes the behaviour of the above types of goals. We consider the complete life-cycle of goals, from their initial adoption by the agent until they are no longer of interest, and all stages in between, including being suspended and resumed.

Scenario. As a running example, consider a team of three robots—Alpha, Bravo and Charlie—that are searching for the survivors of an air crash. Each has a battery life of four hours, and has to return to its base to recharge within this time. The three robots search individually for survivors, but when one is found, each may call on the others for assistance to bring the survivor to the base.

Initially Alpha is told to search a particular area. After 30 minutes, Alpha finds a survivor with a broken leg. Alpha calls for help from Bravo, as it will require at least two robots to carry the survivor. Once Bravo arrives, both robots carry the survivor back to the base, and then both resume searching. A little later, Alpha receives a call for help from Bravo, who has found another survivor. It takes longer than expected for Alpha to get to the location. Before Alpha arrives, another message from Bravo is received, stating that the survivor has been transported back to the base and so Alpha's assistance is no longer required. Alpha resumes its search. Later it receives a call for help from Charlie, who has found a survivor. Once Charlie's survivor is safely back at the base, Alpha considers resuming its search, but as it has only 30 minutes of battery life left, and it predicts that it will take at least 15 minutes of travel time to get to where it needs to be, Alpha decides to recharge. Once this is done, Alpha resumes its search. Eventually it completes searching its given area, finding no more survivors, and returns to the base.

This example illustrates some of the complexity and richness of goal deliberation and management and the need for a comprehensive and principled approach. In the scenario, Alpha initially adopts the performance goal of searching its assigned area; this goal is suspended when a survivor is found, and later resumed. Note that each robot is designated an area to search, and that its task is complete once it has searched this area. In the interim times, Alpha adopts achievement goals (getting survivors to the base), which it may have to abort (when Alpha is too late to help Bravo). Alpha also has the important maintenance goal to monitor its power usage and recharge when appropriate.

Contribution. Our work extends previous efforts in three main directions. Our first area of innovation is to develop a rich and detailed specification of the appropriate operational behaviour when a goal is pursued, succeeded or failed, aborted, suspended, or resumed. We **(1)** include sophisticated maintenance goals, along the lines of Duff et al. [8], that encompass proactive behaviour (i.e., anticipating failure of a given condition) as well as reactive behaviour (i.e., waiting until the condition becomes false), and al-

low for different responses in each case. This contrasts over most work on maintenance goals, in which only the reactive behaviour is developed [25, 12]. We **(2)** develop an appropriate set of states for goals (which generalizes the two states of suspended and active of van Riemsdijk et al. [25]), and a set of operations to move goals between these states. These operations are richer than previous works, by including suspending and resuming for all the common goal types, and the corresponding state transitions can be non-trivial. We provide a detailed specification and formal semantics.

Our second area of innovation is to address execution of plans to achieve goals within our semantics. The spirit of our work is shared by Morandini et al. [12], who build on van Riemsdijk et al. [25] by providing operational semantics for non-leaf goals, i.e., semantics for subgoaling and goal achievement conditions. We **(3)** encompass the same dynamic execution behaviour, but further consider plans as well as goals. Thus we consider the execution cycle, not only the design phase like Morandini et al.

2 Goal Types and Their Abstract States

Let us follow the syntax of goals given by Winikoff et al. [26], using the above robot rescue scenario as a running example. Goals have a specification with both declarative and procedural aspects. We take a goal G to have a *context* (or *pre-condition*) that is a necessary condition before the goal may be adopted, a *success condition* S that denotes when the goal may be considered to have succeeded, and a *failure condition* F that denotes when it may be considered to have failed. Any of these conditions may be empty. We take a plan P to have declarative success and failure conditions, and procedural success and failure methods that are invoked upon its success and failure respectively. A plan may have other dedicated methods attached, such as an abort clean-up method [18], and suspend and resume methods [19]. By *task* we mean an abstract action rather than a specific goal or plan.

Braubach et al. [3] are among those who survey the types of goals found in agent systems. The consensus in the literature agrees that perform, achieve, query, test, and maintain cover the widespread uses of goals [26, 3, 5, 25]. We note that querying and testing goals can be reduced to achievement and performance goals, respectively [25].

perform(τ, S, F)*: accomplish a task* τ These goals, sometimes called *goals-to-do*, demand that a set of plans be identified to perform a task; they do not require a particular state of the world be achieved. A perform goal succeeds if one or more of its plans complete execution; it fails otherwise, such as if no plan is applicable or all applicable plans fail to execute. Hence, the success condition S will express that "one of the plans in the given set succeeds" to accomplish τ [26, 25]. The perform goal also has a failure condition, F. If F is true at any point during execution, the goal terminates with failure, and execution of all plans is terminated. The association between the task τ, which is not more than an identifier, and the plans, is akin to the association between event type and plans in the agent programming language JACK [4].

Example: Search a particular area for survivors.

achieve(S, F)*: reach a state* S These goals, sometimes called *goals-to-be*, generate plans to achieve a state, S, and should not be dropped until the state is achieved or is

found to be unachievable, signified by the condition F. An achieve goal differs from a perform goal in that it checks its success condition during plan execution and after a plan completes. If the success condition S is true (at any point during execution), the goal terminates successfully; if the failure condition F is true (at any point during execution), the goal terminates with failure. Otherwise, the goal returns to plan generation, even if the previous plan completed successfully.

An important difference between perform and achieve goals is their behaviour on multiple instances of the same goal. An agent that is given three identical instances of a perform goal will execute the goal three times (unless there is an unexpected plan failure). An agent that is given three identical instances of an achieve goal may achieve this goal between one and three times, depending on environmental conditions.

Example: Ensure a survivor gets to the base. Note that this is an achieve goal rather than a perform goal as it can only succeed when the survivor is at the base.

The goals we have considered so far are *goals of accomplishment*: they all directly result in activity. Maintenance goals, by contrast, are *goals of monitoring*, in that they may give rise to other goals when particular triggering conditions are met, but they do not themselves directly cause activity.

maintain(C, π, R, P, S, F)*: keep a condition* C *true* Maintenance goals monitor a *maintenance condition*, C, initiating a *recovery goal* (either R or P; see below) to restore the condition to true when it becomes false. Note that a recovery goal is initiated, not a plan. More precisely, as introduced by Duff et al. [8], we allow a maintain goal to be *reactive*, waiting until the maintenance condition is found to be false, $B \models \neg C$ (where B is the knowledge base of beliefs of the agent), and then acting to restore it by adopting a reactive recovery goal R; or to be *proactive*, waiting until the condition is *predicted* to become false, $B \models \pi(\neg C)$ (where π is some prediction mechanism, say using lookahead reasoning, e.g., [21, 10]) and then acting to prevent it by adopting a proactive preventative goal P. Although not specified in prior work, we insist that R and P be achieve goals. The maintenance goal continues until either the success condition S or failure condition F become true.

Note that we take the closed world assumption for maintenance conditions C. The agent could have a pessimistic attitude (i.e., $B \not\models \pi(C)$ implies violation) or an optimistic attitude, as above (i.e., C is only deemed violated when $B \models \pi(\neg C)$).

Example: Ensure that Alpha is always adequately charged.

2.1 Abstract Goal States and Transitions

We now move towards a formal characterization of goal states and the transitions a goal undergoes between these states. Our focus is the life-cycle of each particular goal that the agent has. Hence, our perspective is that of an individual goal, rather than the overall agent per se. This means that we will not be concerned with issues such as the agent's overall deliberation, generation of goals (from Desires), or prioritization of goals. These relevant topics are outside the scope of this paper.

Our objective is to specify the life-cycle of goals and the mechanisms of the agent. The life-cycle we capture as four states, Pending, Waiting, Active, and Suspended, shown in Fig. 1, together with the initial state (left) and the terminal state (right). The transition

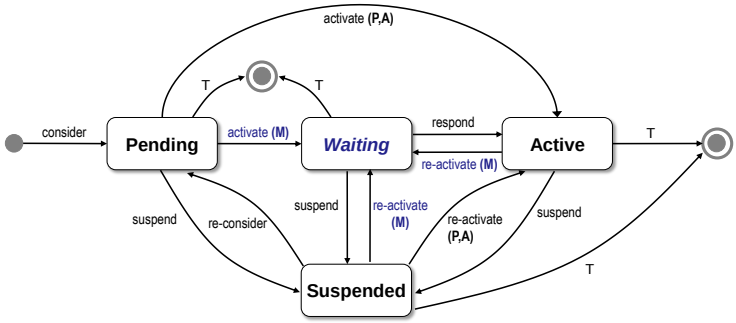

Fig. 1. Goal life-cycle composed of abstract states. P – perform goal, A – achieve goal, M – maintain goal, $\top$ – drop/abort/succeed/fail.

from each state to the terminal state is shown. We combine the drop, abort, succeed, and fail transitions into a single transition, T, as shown.

The states can be arranged into a precedence order: Pending $\prec$ Waiting $\prec$ Active $\prec$ Suspended. Observe that, if a goal transitions from a state s to Suspended, it may not then next transition from Suspended to a state higher than s in the order. Some transitions are essentially controlled by conditions, while others depend on an explicit agent decision (or a combination of conditions and a decision), as will be made precise.

A new candidate goal may arise from a source external or internal to the agent's control cycle [13]. External to the control cycle, it may arise from obligations or commitments concerning other agents, or from the agent's own motivations. Internal to the control cycle, it may arise from subgoaling within an executing plan. Either way, a new candidate goal begins life in the Pending state if the agent has decided to consider the goal. In the next section we describe the goal control cycle in detail, including the mechanisms to perform the goal operations of interest.

3 Transitions Between Goal States

The heart of our work is the effects that different operations an agent may apply to its goals of different types, in each of the four states introduced. We now describe in detail the life-cycle of a goal in each of the states. We call a *top-level command* a decision by the agent's deliberation to impose an operation upon a goal.

First, to any goal in any state, the *drop* operation implies that the goal and any goal-related actions are halted; the goal is discarded with no further action. The agent may choose to drop a goal if, for example, it believes the goal is accomplished, is no longer required, impossible, or if it inhibits a higher priority goal. Note that there are three essential cases here: the goal is dropped because it has succeeded, dropped because it has failed, or dropped because the agent has decided to drop it.

Pending State Goals in the Pending state are inactive, awaiting the agent to deliberate over them and execute a particular operation. The *activate* operation on a perform or

achieve goal transitions the goal to the Active state where the goal is pursued. By contrast, the *activate* operation on a maintain goal transitions the goal to the Waiting state.

The *suspend* operation takes a goal to the Suspended state. The *abort* operation simply drops the goal; no clean-up is required since no plans for the goal are in execution. If the success or failure condition become true in the Pending state, the goal is dropped. Note that although perform goals do not contain an explicit success condition (see Sect. 2), we make the distinction here for simplicity.

Waiting State The Waiting state is shown with italics in Fig. 1 to emphasize that it exclusively applies to goals of monitoring: maintain goals that (actively) check for a triggering condition to be known. In this state, as in Pending, no plans are being executed. Goals transition into this state when they are (1) activated from Pending, (2) re-activated from Suspended, or (3) re-activated from Active when the subgoal succeeds, as described earlier. Should the maintenance condition be violated—or, in the proactive case, should it be predicted to be violated—then the goal transitions to the Active state with the *respond* operation. The *suspend* operation moves the goal to the Suspended state, whilst *abort* simply drops it since no plans are in execution. The goal may also be dropped if the success or failure condition becomes true.

Active State Active goals are actively pursuing tasks: they may therefore have plan(s) associated. We must define how the agent manages the plan(s) in accordance with the operations it applies to the goal. Fig. 2 provides the internal details of the abstract Active state. Transitions with bold label denote top-level commands and other transitions occur when some condition is met. Sub-states of the active state that are shaded (e.g., aborting) are uninterruptable states where top-level commands cannot be applied.

Maintain goals enter the Active state from the Waiting state when the triggering condition is true, and move to a post subgoal sub-state. A maintain goal posts a recovery goal R if the maintenance condition was violated or a preventative goal P if the maintenance condition is predicted to be violated. Recovery and preventative goals are always achieve goals, and commence in the Pending state.[4]

If the subgoal succeeds, then the parent maintain goal g transitions back to the Waiting state. If the subgoal fails, g is dropped. Should g be aborted or should its success or failure condition become true, then it transitions to the abort subgoal sub-state where the subgoal is aborted and then g is dropped. Should the goal g be suspended, the subgoal is aborted in the abort subgoal$^{\mathsf{S}}$ sub-state and then g moves to the suspended waiting sub-state of Suspended. Any generated plans are handled according to the mechanisms described in the literature [19].

Perform and achieve goals enter the Active state from the Pending state, or from the Suspended state when the re-activation condition becomes true. These goals are first examined in the check goal sub-state to determine if the success or failure condition is true; if either is true, the goal is dropped. Otherwise, a plan is generated to achieve the goal in the plan generation state. If no plan is found, the goal is dropped: this reflects the

[4] An argument can be made for commencing these goals in the Active state. However, commencing in the Pending state allows more flexibility, in that a trivial activation condition will see these goals immediately transition to the Active state, if that is desired.

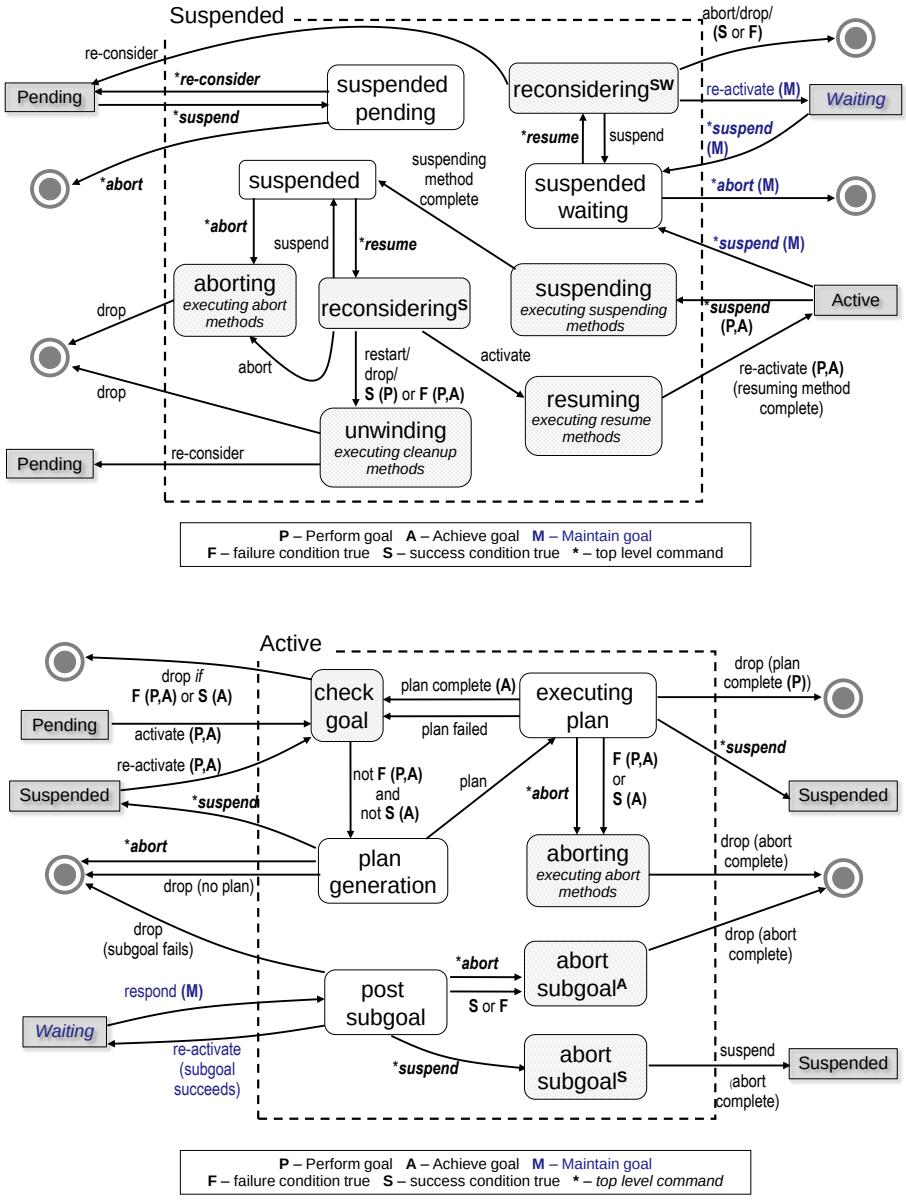

Fig. 2. Active and Suspended states in detail.

most common behaviour in BDI systems. A goal is also dropped if it is aborted in this sub-state since no plan is in execution. If a plan is found, then the goal transitions to the executing plan sub-state.

In the executing plan sub-state, if the plan fails then the goal moves back to the check goal state to retry the process of generating a new plan to achieve the goal.[4] If the plan completes for a perform goal, the goal succeeds and hence is dropped. If the plan completes for an achieve goal, however, the goal is checked for its success condition in the check goal state. If the success condition is not true then a new plan needs to be generated and executed to achieve it. While executing a plan in the executing plan sub-state, if the goal is aborted, or the success or failure condition become true, the goal transitions to the aborting sub-state where abort methods are executed [18]; the goal is dropped when they complete. If a goal is suspended in the executing plan sub-state it transitions to the Suspended state.

Suspended State This state contains a goal of any type that is suspended, monitoring its reconsideration condition [19], awaiting possible resumption.

Goals of accomplishment may have one or more plans associated. We again must define how the agent manages the plan(s) in accordance with the operations it applies to the goal. Fig. 2 provides the internal details of the abstract Suspended state. Goals transition to this state when the *suspend* operation is applied to them. Goals arriving from the Pending state (top left) are held in a suspended pending sub-state and, when resumed, move back to the Pending state.

Maintain goals suspended from the Active state or the Waiting state are held in a suspended waiting sub-state. From this sub-state, a goal may be aborted, in which case it is simply dropped, or resumed. If resumed, a goal moves to a reconsidering sub-state where the agent deliberates over it and may either (re-)suspend the goal (back to suspended waiting sub-state), reconsider the goal (back to Pending state), re-activate it (back to Waiting), or simply abort or drop it. When a maintain is suspended, our semantics specifies that its subgoals be aborted.

Perform or achieve goals suspended from the Active state first require any suspend methods to be executed. This occurs in the suspending sub-state; then the goal moves to the suspended state. A goal may be aborted from this state, causing its abort method to be performed [18] in the aborting sub-state before it is dropped. If not aborted prior to resumption, a goal may be resumed when its *reconsideration condition* becomes true, or when the agent decides to resume it.[5] Upon resumption of the goal, the agent deliberates over it in the reconsidering state. The agent may opt to (1) abort the goal (move to aborting sub-state), (2) (re)-suspend it (move to suspended sub-state), (3) re-activate the goal by performing resume methods [19] in the resuming sub-state before transitioning to the Active state, (4) *restart* the goal, or (5) drop the goal. To restart is to halt any suspended plans and re-consider the goal. Therefore, prior to restarting, any existing plans need to be terminated in the unwinding sub-state, before the goal transitions to the Pending state to be re-considered. Goals to be dropped follow a similar transition. Note that suspend and resume methods, like abort methods, are assumed not to fail [19].

[5] That is, *resume* is a top-level command. The reconsideration condition is the agent's guide to its deliberation over the suspended goal: a sufficient but not necessary condition for when it should next look at the goal.

4 Designing Rules

We will now consider how to apply the transitions from Fig. 1 to the example. We will then discuss how we may design a system of transitions which will implement these transitions. For reasons of space we will limit our discussion of transitions to those in Fig. 1; the issues for the transitions of Fig. 2 are similar.

4.1 Design Issues

In order to use Fig. 1 as a specification of a goal deliberation process, we need to determine what information is required for each goal, and how this information is used to make decisions about when the transitions of Fig. 1 should be applied. Ultimately, we wish to provide formal definitions of the transitions for each goal in the CAN [26, 15, 16] language, utilising the generic approach initiated by van Riemsdijk et al. [25], with some variations. Two of the key differences in our work, which enable us to support the full variety of goal types and operations upon goals, is that we have four basic goal states (Pending, Waiting, Active and Suspended) rather than two, and that not all transitions are possible (for example, goals of accomplishment can never be in the Waiting state). Further, unlike Morandini et al. [12], our semantics deals with plans as well as goals.

Using CAN as a basis means that the formal transitions are between agent configurations of the form $\langle B, \mathcal{G} \rangle$, where B is the agent's beliefs, and $\mathcal{G}$ is a set of goals that the agent is pursuing concurrently. This approach follows the same methodology as in previous works [25, 12]. Each element of $\mathcal{G}$ will contain more than just the goal itself; each $G \in \mathcal{G}$ is the *goal context* tuple $\langle \text{Id}, Goal, Rules, State, Plan \rangle$, where

- Id is a unique identifier for each goal
- $Goal$ is the goal content (as given in Section 2),
- $Rules$ is a set of *condition-action* pairs of the form $\langle C, A \rangle$, where C is a condition and A is one of $\{$ activate, reactivate, reconsider, respond, suspend, drop, abort $\}$
- $State$ is one of $\{$ Pending, Waiting, Active, Suspended $\}$
- $Plan$ is the current plan (possibly none) being executed for this goal

The existence of unique identifiers ensures that goals can refer to each other. Recall that goals are fixed at design time and do not change during execution; hence $Goal$ is fixed throughout execution. $Rules$, $State$, and $Plan$ are dynamic and may change during execution. Note that our notation for G from this point on differs from the informal notational convenience used in Sect. 2.

Our deliberation process will be specified by transitions between tuples of the form $\langle B, \mathcal{G} \rangle$. Before discussing the process in more detail, we clarify assumptions.

- *All goals are known at compile time, and are given unique identifiers.* This ensures that goals can explicitly refer to other goals, allowing the agent designer to specify transitions such as one goal being suspended when another specific goal is activated. Note that we do **not** assume that *goal contexts* are known at compile time, so that we do not necessarily know in advance the plans that may be executed for any particular goal. For example, we may know in advance that Alpha's goals include achieve(at_base, $\perp$), but not what plans will be used to achieve it.

– *Any change in any goal's state has preference over any executing plans.* This means that execution can only take place when the set of goal contexts is stable, i.e., none of the transitions in Fig. 1 can currently occur. While somewhat conservative, this allows the agent designer freedom to specify whatever interaction between goals is desired (such as suspending certain goals when others become active), knowing that any change in any goal's status will result in the status of all goals being reconsidered. This, in turn, may result in a corresponding change in what is being executed.
– *Plans are not necessarily known in advance, but may be generated online.* This means that we do not assume that the agent necessarily has a plan library (although this is a perfectly valid option if desired), and so we cannot rely on plans to be of a particular form. This also means that we have to explicitly allow for plan generation in our formal definition; we do so using the techniques of van Riemsdijk et al. [25].
– *No restriction is made on the number of goals that may be active at once.* It may be desirable to allow that there should be at most one active goal at any time, or perhaps that there should be at most one goal active when any maintenance goal is active (but allow any number of concurrent achievement goals to be active otherwise). Hence we need to be able to provide the agent designer with mechanisms to enforce restrictions like these if desired, but not to build them into the CAN rules. Accordingly we will have a standard pattern for goal transition rules, which can be tailored by the designer to suit the particular application.

4.2 Worked Example

A detailed description of CAN and the appropriate rules to match the requirements discussed above is beyond the scope of this paper. Instead, we give a specification of the desired sequence of goal transitions for the robot rescue scenario, and discuss the issues involved in designing an appropriate set of CAN rules. The sequence of goal transitions in the robot rescue scenario are given in Table 1.

Alpha's initial goals include the perform goal $\mathsf{perform}(search, S, F)$ where $search$ is a search plan for a region 10 units square, which Alpha searches one square at a time. We will assume that $search$ consists of the eleven steps $search_1; \ldots search_{10}; return$ where $search_i$ searches column i of the grid and $return$ makes Alpha return to the base. The success condition S is that each column has been searched and Alpha is at the base.

Alpha's initial goals also include the maintenance goal that it should always retain sufficient charge to return to the base. Alpha must estimate how long it will take to return to the base from its current position, and if its remaining charge falls to this level, it should immediately suspend whatever it is doing and return to the base to recharge. Hence Alpha's initial goals include $\mathsf{maintain}(C, \pi, R, P, \bot, \bot)$ where C is the condition that $current\_charge > return\_time$, π is an appropriate prediction mechanism (such as estimating the time that will be taken by each of the currently adopted plans), R and P are both the achievement goal of returning to the base, i.e., $\mathsf{achieve}(\text{at\_base}, \bot)$.

Alpha will adopt appropriate achievement goals when assisting a survivor back to the base, and when responding to calls for help. Respectively, these we denote by $\mathsf{achieve}(at(\text{Survivor}, \text{base}), \bot)$ and $\mathsf{achieve}(satisfied(\text{Other}), \bot)$. The success condition for the former is when the survivor is safely back at the base. The success condition for the latter is determined by the other agent; hence the goal only succeeds when Alpha

Stage	Performance goals	Achievement goals	Maintenance goals
1	$\langle search, Ser, R_1, \mathsf{Pending}, search\rangle$	-	$\langle recharge, Main, R_2, \mathsf{Pending}, e\rangle$
2	$\langle search, Ser, R_1, \mathsf{Active}, search\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
3	$\langle search, Ser, R_1, \mathsf{Active}, s2\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
4	$\langle search, Ser, R_5, \mathsf{Suspended}, s2\rangle$	$\langle save, Sav, R_4, \mathsf{Pending}, e\rangle$	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
5	$\langle search, Ser, R_5, \mathsf{Suspended}, s2\rangle$	$\langle save, Sav, R_4, \mathsf{Active}, assist\rangle$	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
6	$\langle search, Ser, R_5, \mathsf{Suspended}, s2\rangle$	(dropped after success)	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
7	$\langle search, Ser, R_1, \mathsf{Pending}, s2\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
8	$\langle search, Ser, R_1, \mathsf{Active}, r2; s2\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
9	$\langle search, Ser, R_1, \mathsf{Active}, s5\rangle$	$\langle help, Help, R_6, \mathsf{Pending}, e\rangle$	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
10	$\langle search, Ser, R_5, \mathsf{Suspended}, s5\rangle$	$\langle help, Help, R_6, \mathsf{Active}, assist\rangle$	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
11	$\langle search, Ser, R_5, \mathsf{Suspended}, s5\rangle$	(aborted)	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
12	$\langle search, Ser, R_1, \mathsf{Active}, r5; s5\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
13	$\langle search, Ser, R_1, \mathsf{Active}, s8\rangle$	$\langle help, Help, R_6, \mathsf{Pending}, e\rangle$	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
14	$\langle search, Ser, R_5, \mathsf{Suspended}, s8\rangle$	$\langle help, Help, R_6, \mathsf{Pending}, e\rangle$	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
15	$\langle search, Ser, R_9, \mathsf{Suspended}, s8\rangle$	$\langle help, Help, R_6, \mathsf{Active}, assist\rangle$	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
16	$\langle search, Ser, R_5, \mathsf{Suspended}, s8\rangle$	(dropped after success)	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
17	$\langle search, Ser, R_5, \mathsf{Pending}, r8; s8\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
18	$\langle search, Ser, R_5, \mathsf{Active}, r8; s8\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
19	$\langle search, Ser, R_5, \mathsf{Suspended}, r8; s8\rangle$	$\langle charge, Charge, R_7, \mathsf{Pending}, e\rangle$	$\langle recharge, Main, R_3, \mathsf{Active}, e\rangle$
20	$\langle search, Ser, R_5, \mathsf{Suspended}, r8; s8\rangle$	$\langle charge, Charge, R_7, \mathsf{Active}, charge\rangle$	$\langle recharge, Main, R_3, \mathsf{Active}, e\rangle$
21	$\langle search, Ser, R_5, \mathsf{Suspended}, r8; s8\rangle$	(dropped after success)	$\langle recharge, Main, R_3, \mathsf{Active}, e\rangle$
22	$\langle search, Ser, R_5, \mathsf{Suspended}, r8; s8\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
23	$\langle search, Ser, R_1, \mathsf{Active}, r8; s8\rangle$	-	$\langle recharge, Main, R_3, \mathsf{Waiting}, e\rangle$
24	(dropped after success)	-	$\langle recharge, Main, R_2, \mathsf{Waiting}, e\rangle$

Table 1. Alpha's sequence of goal states. Ser is the performance goal $\mathsf{perform}(search, S, F)$, $Main$ is the maintenance goal $\mathsf{maintain}(MC, \pi, Charge, Charge, \bot, \bot)$, MC is the condition $current\_charge > return\_time$, $Charge$ is the achievement goal $\mathsf{achieve}(recharged, \bot)$, si is $search_i; \ldots search_{10}; return$, ri is a plan which returns the robot to sector i, R_1 is $standard(search, \{S, F\}, \top)$[7], R_2 is $standard(recharge, \{\mathsf{drop}(charge), \mathsf{abort}(charge)\}, \top) \cup \{\langle recharged, \mathsf{reactivate}\rangle\}$, R_3 is $R_2 \cup \{\langle\{\neg MC, MC \wedge \pi(\neg MC)\}, respond\rangle\}$, R_4 is $standard(save, \{at(survivor, base)\}\top)$, R_5 is $R_1 \cup \{at(base, survivor), reconsider\rangle\}$, R_6 is $standard(help, satisfied(Other), \top)$, R_7 is $standard(charge, recharged, \neg C \vee (C \wedge \pi(\neg C))) \cup \{\langle\{\mathsf{drop}(recharge), \mathsf{suspend}(recharge), \mathsf{abort}(recharge)\}, \mathsf{abort}\rangle\}$

believes the other agent is satisfied, i.e., when it receives a message notifying it that the goal has been achieved. The plans to achieve this goal will also be generated by the other agent. Activation of either of these goals will suspend the search goal.

Each of these achievement goals will be triggered by a rule in Alpha's plan library, so that we assume that Alpha contains in its library the following two rules:[8]

$survivor\_found : \top \rightarrow \mathsf{achieve}(at(\text{Survivor, base}), \bot)$

$request\_received : \top \rightarrow \mathsf{achieve}(satisfied(\text{Other}), \bot)$

Alpha's sequence of goal states is given in Table 1. Its initial goals are the perform goal Ser to search and the maintain goal $Main$ concerned with its battery power. Alpha's first decisions are to activate both goals, so that the perform goal moves into the Active state, and the maintain goal moves into the Waiting state (stage 2). Alpha thus starts to execute its search pattern. Alpha successfully executes $search_1$ and is in the midst of sub-plan $search_2$ when the survivor is found (stage 3). The event *survivor_found* is raised, and the rule in Alpha's plan library is fired, resulting the goal Sav being

[8] Another possibility is to have these two goals initially in the Pending state and to use the $survivor\_found$ and $request\_received$ events as part of the activation condition for them; pursuing this possibility is part of our future work.

added to Alpha's goals (stage 4). This triggers the suspension of the search goal. The reconsideration condition is when the survivor is safely at the base.

Alpha now activates the Sav goal. It plans to achieve $at(\text{Survivor}, \text{base})$ by calling Bravo for help, asking the survivor about any others nearby, waiting until Bravo arrives and together carrying the survivor to the base (stage 5). This plan is executed successfully; thus Sav is achieved, the goal is dropped, and Alpha resumes searching (stages 6–8). It is in sector 5 when Bravo's call for help is received (stage 9). Again, this event fires the appropriate rule in Alpha's plan library, and a $Help$ goal is added to Alpha's goal state. Alpha then suspends the search plan and adopts the goal of assisting Bravo (stage 10). The reconsideration condition is when the survivor is safely at the base.

While Alpha is still executing the action find(bravo), a message from Bravo arrives saying that the survivor is now safely at the base, and so Alpha aborts the plan to find Bravo and the $Help$ goal is dropped (stage 11). Alpha resumes its search, and then gets the call from Charlie when it is in sector 8. As before, it suspends searching (stages 13–15), and adopts a $Help$ goal. The reconsideration condition is when the survivor is safely at the base. Alpha finds Charlie, the survivor is brought to the base, and so the $Help$ goal is dropped (stage 16).

At this point, Alpha reconsiders Ser, and activates the searching goal, only to discover that resuming its search will soon violate the maintenance goal, as it has only 30 minutes of charge remaining. Hence the searching goal is re-suspended while Alpha recharges (stages 17–20). Once charging is finished, $Charge$ is dropped (stage 21), and $recharge$ goes back to the Waiting state (stage 22), which means that searching can be resumed (stage 23). As the perform goal Ser has now succeeded, it is dropped, and Alpha is now idle (stage 24).

4.3 Overview of Rules and Approach to the Semantics

The technical key to our operational semantics is the appropriate definition of $Rules$ for each goal. These definitions, which codify the specification set out in Sect. 3, follow the same general principles, but can be tailored for individual goals. For example, all maintenance goals have the same general structure and transitions that are enabled; achievement goals can have their rules designed so that they are suspended when any maintenance goal is active. It is also helpful to use CAN's expressiveness to alter $Rules$ dynamically, such as adding reconsideration conditions to suspended goals.

The intuition is that for $Rules = \{\langle C_1, A_1\rangle, \ldots \langle C_n, A_n\rangle\}$, if for some $c \in C_i$ we have $B \models c$, then action A_i is performed (where C_i is a set of first-order terms, and $\models$ is an appropriate consequence relation; in this paper, classical logic will suffice). For this to be coherent, we require $Rules$ to have the property that there is at most one action that can be performed, i.e., that there is at most one i such that $B \models c$ for some $c \in C_i$. Note that the A_i need not be distinct; hence strictly we need only consider pairs of the form $\langle c, A_i\rangle$ where c is a single condition rather than a set of conditions. However, it is often more convenient to use the form $\langle C, A_n\rangle$ when specifying rules; we will often use this form.

As mentioned above, the details of the CAN rules are beyond our scope; in the rest of this section we concentrate on the rules that need to be defined for each goal type. In some cases, the agent wants a condition C to be evaluated 'autonomously',

i.e., without any further deliberation. In other cases, the agent wants an explicit condition. Thus, we require that all conditions for a given action to contain a formula of the form $reconsider(\text{Id})$, so that a reconsideration condition $Cond$ is specified as $Cond \wedge reconsider(\text{Id})$. This means that for the goal to change state, not only must $Cond$ hold, we must also have that the agent has explicitly decided to resume the goal by adding $reconsider(\text{Id})$ to its beliefs. This mechanism also allows us to provide for the possibility that the agent may decide to drop, abort, or suspend any goal at any time: it can do so by adding $drop(\text{Id})$ (resp. $abort(\text{Id})$, $suspend(\text{Id})$) to its beliefs.

As mentioned at the beginning of Sect.2, it is also common to include an activation condition of the form $\langle\{Cond \wedge \text{activate(Id)}\}, \textsf{activate}\rangle\}$, so that the goal can only be activated when both $Cond$ is true and the agent has decided to activate the goal. As a result, we will denote as $standard(\text{Id}, Succ, Cond)$ the set of rules:

$$\{\langle\{Succ, \text{drop(Id)}\}, \textsf{drop}\rangle, \langle\{\text{abort(Id)}\}, \textsf{abort}\rangle,$$
$$\langle\{\text{suspend(Id)}\}, \textsf{suspend}\rangle, \langle\{Cond \wedge \text{activate(Id)}\}, \textsf{activate}\rangle\}$$

perform(τ, S, F)**:** We commence with $Rules$ as $standard(\text{Id}, \{S, F\}, Cond)$, for a goal with identifier Id. We do not initially include any rules for the actions reconsider or reactivate; these are added to $Rules$ when the goal is suspended.

For the suspend action, we need to add reconsideration conditions to $Rules$. When suspending a goal in the Pending state, the first of the following rules is added by the transition; when suspending a goal in the Active state, both are added.

$$\{\langle\{RC \wedge reconsider(\text{Id})\}, \textsf{reconsider}\rangle\}$$
$$\{\langle\{RC \wedge reactivate(\text{Id})\}, \textsf{reactivate}\rangle\}$$

Here, RC is the reconsideration condition, which is determined by the agent. Note that, since the reactivate action is not possible if the goal was in the Pending state when suspended, in this case we only include the rule for reconsider. In a similar manner, the reconsider and reactivate actions remove the condition-action pairs for both of themselves when either of these actions is performed. This allows different reconsideration conditions to be attached each time a suspension occurs.

achieve(S, F)**:** The high-level rules for this goal type are the same as for a perform goal, i.e., $standard(\text{Id}, \{S, F\}, Cond)$. This reflects the fact that the transitions in Fig. 1 are the same for these goal types. Note that the detailed transitions, and so the corresponding rules, differ in the two sides of Fig. 2.

maintain$(C, \pi, Recover, Prevent, S, F)$**:** The two pertinent differences between monitoring versus accomplishment goals are that (1) there is now the extra state Waiting, in which the maintenance condition is being monitored, but no action is being taken yet, and (2) when the maintenance goal becomes active, it triggers the adoption of an extra achievement goal, with the intention that when this new goal is achieved, the violation of the maintenance condition (either actual or predicted) will be overcome. Hence, the respond action, which is only available to maintenance goals, will result in not only the maintenance goal becoming active, but also the adoption of a new achievement goal.

The initial set of rules is the same as for perform goals above. The transitions for drop and abort are as above. The transition for activate now puts the goal into the Waiting state

rather than the Active state, and adds the rule: $\langle\{\neg C, C \wedge \pi(\neg C)\}, \mathsf{respond}\rangle$. Hence the respond rule is only present when the goal is in the Waiting state.

The only significant difference to perform goals is the transition from Waiting to Active states, which is as follows:

$$\frac{\langle C, \mathsf{respond}\rangle \in R_1 \quad c \in C \quad B \models c}{\langle B, \{\langle \mathrm{Id}, MG, R_1, \mathsf{Waiting}, e\rangle\} \cup \mathcal{G}\rangle \rightarrow \langle B, \{\langle \mathrm{Id}, MG, R_2, \mathsf{Active}, e\rangle, AG\} \cup \mathcal{G}\rangle}$$

MG is maintain$(C, \pi, Recover, Prevent, S, F)$; $Recover$ is achieve(S_R, F_R); $Prevent$ is achieve(S_P, F_P); AG is $\langle$NewId, $SG, R_3, \mathsf{Pending}, e\rangle$; SG is achieve(S_A, F_A); S_A is $S_R \wedge F_A$ is F_R if $\neg C$ is true and $S_P \wedge F_P$ otherwise; R_2 is *standard*(Id, $\{F_A, S, F\}$, $\top) \cup \{\langle\{drop(\mathrm{NewId}), abort(\mathrm{NewId})\}, \mathsf{drop}\rangle, \langle\{S\}, \mathsf{reactivate}\rangle\}$; R_3 is *standard*(NewId, $\{S_A, F_A\}, \neg C \vee (C \wedge \pi(\neg C))) \cup \{\langle\{drop(\mathrm{Id}), suspend(\mathrm{Id}), abort(\mathrm{Id})\}, \mathsf{abort}\rangle\}$.

Observe that the achievement goal SG has been added (initially in the Pending state) to attempt re-establishment of the maintenance condition. If SG succeeds, we reactivate MG (i.e., MG returns to the Waiting state), due to the success condition S_A of SG being the only condition for the reactivate rule in R_2. MG is dropped if SG fails or is dropped or aborted, as reflected in R_2. MG is also dropped if either its success condition S or failure condition F becomes true.

The rules R_3 for SG specify it will be activated immediately (due to the activation condition incorporating the maintenance condition), and that it should be aborted if the agent decides to drop, abort or suspend MG (as reflected in the rules for drop in R_3). Note also that if SG is suspended, the maintenance goal remains in the Active state.

As in the above cases, the suspend transition will need to attach a reconsideration condition. The only difference here is that we need to add the rule for reactivate when in the Waiting state as well as in the Active state. Correspondingly, and as before, these rules are deleted when either of the reconsider or reactivate actions are performed. Note, though, that the reactivate action in this case puts the goal into the Waiting state rather than the Active state.

We have developed a prototype implementation of CAN together with the appropriate actions for the above scenario. This implementation, denoted *Orpheus*, consists of around 700 lines of Prolog. It has been tested under Ciao and SWI-Prolog, and is available from the authors at: `http://www.cs.rmit.edu.au/~jah/orpheus`. Using Prolog greatly simplifies the implementation of the CAN rules, and the rules for goal state transitions.

5 Related Work

Goals play a central role in *cognitive* agent frameworks [25]: "mental attitudes representing preferred progressions of a particular (multi)agent system that the agent has chosen to put effort into bringing about." Winikoff et al. [26] argue for the importance of both declarative and procedural representations, and present the specification of goals with context, in-conditions, and effects.

A goal type has been defined as "a specific agent attitude towards goals" [5]. The different types of goals found in the literature and in implemented agent systems are surveyed by Braubach et al. [3]. While there is broad agreement about perform and achieve

goals, less attention has been directed towards maintain goals. The reactive and proactive semantics for maintenance goals is explored by Duff et al. [8]. However, they do not consider aborting or suspending goals, and do not give formal rules for the behaviour of maintenance goals. Mechanisms for adopting and dropping goals, and generating plans for them, have been variously explored at both the semantic theoretical and implemented system levels; we do not cite here the extensive body of work, including Hübner et al. [11]. Thangarajah et al. formalized the mechanisms for the operations of aborting, suspending, and resuming goals [18, 19]. However, those authors considered only achieve goals. We find the literature lacks a state and transition specification for all classes of goals that accounts for the current mechanisms for aborting and suspending. Beyond our scope are recent examples of exploring goal failure and re-planning [16, 7].

The Jason system [1] implements an extended form of the AgentSpeak language. Hübner et al. [11] specify various forms of declarative goals as *plan patterns*, based on internal Jason actions. The internal actions allow, for instance, suspending, resuming, and dropping achieve goals [1]. Our focus here is not the mechanisms for these operations, but to develop a goal semantics that relies on the existence of such operations.

Braubach et al. [3] build the Jadex agent system [13] on an explicit state-based manipulation of goals. Goals begin in a New state. When adopted, they move to the Option state (akin to our Pending), and from there to Active (akin to our own Active). A goal moves to the Suspended state if its in-condition ("context" [3]) becomes false: this is a different concept from our deliberation-directed suspension and resumption. The aim of Braubach et al. is to define a principled yet pragmatic foundation for the Jadex system; no attempt is made for a generic formalization with a uniform set of operations on goals at an abstract representational level. Braubach et al. [2] discuss *long-term* goals, which may be considered as an input for determining when a goal should be dropped, aborted or suspended; here we are concerned with the consequences of such decisions, rather than the reason that they are made.

van Riemsdijk et al. [23, 24] provide semantics based on default logic, emphasizing that, while the set of an agent's goals need not be consistent, its set of intentions must be. This and similar work is complementary to ours, in that we do not consider the process by which the agent decides whether to adopt a goal and whether to adopt an intention (plan) from it. The authors [5, 6] expand their analysis of declarative goals to perform, achieve goals, and maintain goals, providing a logic-based operational semantics.

van Riemsdijk et al. [25] present a generic, abstract, type-neutral goal model consisting of suspend and active states. Their two states can be thought of as "not currently executing a plan" and "currently executing a plan", respectively. Their work, which like ours encompasses achieve, perform, query, and maintain goals, has overly simple accounting for maintenance goals and for aborting and suspending. Further, we argue that the states of non-execution and suspension should be distinguished, and that goals should be created into the Pending not Suspend state.

Morandini et al. [12] use the generic goal model of van Riemsdijk et al. to reduce the semantic gap between design-time goal models and run-time agent implementations. Their operational semantics is focused on providing an account of the relationship between a goal and its subgoals, including success conditions which are not necessarily the same as those of the subgoals. Our work likewise encompasses dynamic achievement of a goal according to logical conditions, enabled by a subgoaling mechanism.

Crucially, since we are concerned with execution, our semantics accounts for plans as well as goals. This means that our goal states contain finer distinctions, and in particular the sub-division of the Active and Suspended states. Our work is further distinguished by a richer range of operations that may be applied to a goal (e.g., a richer semantics for suspending a goal and its children; aborting as well as failing), and by the inclusion of proactive maintenance goals.

6 Conclusion and Further Work

Management of goals is central to intelligent agents in the BDI tradition. This paper provides mechanisms for goal management across the common goal types in the literature, including goals of maintenance. The three contributions of our generic framework for goal states and transitions are (1) to encompass both goals of accomplishment and rich goals of monitoring, (2) to provide the first specification of abort and suspend for all the common goal types, and (3) to account for plan execution as well as the dynamics of sub-goaling. To the best of our knowledge, no existing framework for goal operation accounts all of these points.

By developing the formal operational semantics for our generic framework in the agent language CAN [16], and implementing the proof of concept in Prolog, we have not been tied to any particular agent implementation. However, besides disseminating the formal semantics, a first priority is to assess the value and practicality our framework may have for the benefit of the agent programmer.

This paper accounts for the life-cycle of each goal. We have not sought to address overall agent deliberation, plan deliberation, resource management, or plan scheduling. Thus far we have examined the same questions as Braubach et al. [3]; future work is to address the other questions they pose. Likewise, we have not considered failure handling and exceptions. Our work is complementary to works that consider generic or application-specific reasoning about goal interactions, such as [21, 17], and works that consider goal and plan selection, such as Hindriks et al. [9].

Acknowledgements. We thank Lin Padgham and the anonymous reviewers for their extensive and thoughtful comments. The first author acknowledges the support of the Australian Research Council and Agent Oriented Software under grant LP0453486. The work of the two authors at SRI International was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, or the Air Force Research Laboratory.

References

1. R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, New York, 2007.
2. L. Braubach and A. Pokahr. Representing long-term and interest BDI goals. In *Proc. of ProMAS'09*, 2009.
3. L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal representation for BDI Agent systems. In *Proc. of ProMAS'04*, 2004.

4. P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK intelligent agents — components for intelligent agents in Java. AgentLink News, Issue 2, 1999.
5. M. Dastani, M. B. van Riemsdijk, and J.-J. C. Meyer. Goal types in agent programming. In *Proc. of AAMAS'06*, 2006.
6. M. Dastani, M. B. van Riemsdijk, and J.-J. C. Meyer. Goal types in agent programming. In *Proc. of ECAI-06*, 2006.
7. L. de Silva, S. Sardina, and L. Padgham. First principles planning in BDI systems. In *Proc. of AAMAS'09*, 2009.
8. S. Duff, J. Harland, and J. Thangarajah. On proactivity and maintenance goals. In *Proc. of AAMAS'06*, 2006.
9. K. V. Hindriks, W. van der Hoek, and M. B. van Riemsdij. Agent programming with temporally extended goals. In *Proc. of AAMAS'09*, 2009.
10. K. V. Hindriks and M. B. van Riemsdijk. Using temporal logic to integrate goals and qualitative preferences into agent programming. In *Proc. of DALT'08*, 2008.
11. J. F. Hübner, R. H. Bordini, and M. Wooldridge. Programming declarative goals using plan patterns. In *Proc. of DALT'06*, 2006.
12. M. Morandini, L. Penserini, and A. Perini. Operational semantics of goal models in adaptive agents. In *Proc. of AAMAS'09*, 2009.
13. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In *Multi-Agent Programming*, Berlin, 2005. Springer.
14. A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In *Proc. of KR'92*.
15. S. Sardiña, L. de Silva, and L. Padgham. Hierarchical planning in BDI agent programming languages: a formal approach. In *Proc. of AAMAS'06*, 2006.
16. S. Sardiña and L. Padgham. Goals in the context of BDI plan failure and planning. In *Proc. of AAMAS'07*, 2007.
17. P. H. Shaw, B. Farwer, and R. H. Bordini. Theoretical and experimental results on the goal-plan tree problem. In *Proc. of AAMAS'08*, 2008.
18. J. Thangarajah, J. Harland, D. Morley, and N. Yorke-Smith. Aborting tasks in BDI agents. In *Proc. of AAMAS'07*, 2007.
19. J. Thangarajah, J. Harland, D. Morley, and N. Yorke-Smith. Suspending and resuming tasks in BDI agents. In *Proc. of AAMAS'08*, 2008.
20. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proc. of IJCAI'03*, 2003.
21. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and exploiting positive goal interaction in intelligent agents. In *Proc. of AAMAS'03*, 2003.
22. A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. of Intl. Joint Conf. on Requirements Engineering (RE'01)*, 2001.
23. M. B. van Riemsdijk, M. Dastani, and J.-J. C. Meyer. Semantics of declarative goals in agent programming. In *Proc. of AAMAS'05*, 2005.
24. M. B. van Riemsdijk, M. Dastani, and J.-J. C. Meyer. Goals in conflict: Semantic foundations of goals in agent programming. *J. Autonomous Agents and Multi-Agent Systems*, 18(3), 2009.
25. M. B. van Riemsdijk, M. Dastani, and M. Winikoff. Goals in agent systems: A unifying framework. In *Proc. of AAMAS'08*, 2008.
26. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proc. of KR'02*, 2002.