Formalising Proactive Maintenance Goals

Simon Duff and James Harland

School of Computer Science and Information Technology
RMIT University
Melbourne, Victoria, Australia 3000
{simon.duff,james.harland}@rmit.edu.au

Abstract. In intelligent agent systems, goals are an important concept and take a variety of forms. One such case are *maintenance goals*, where in contrast to *achievement goals*, define a state that an agent should continue to keep true, rather than a final state to bring about. However, existing agent systems have used maintenance goals as triggers, and are thus reactive.

Proactive maintenance goals, where the maintenance goals are incorporated into the deliberation cycle, have received less attention. The agent, by anticipating the effects of its actions, can avoid violating its maintenance conditions.

This work extends earlier work in proactive maintenance goals with the development of formal semantics of maintenance goals.

1 Introduction

Goals are a central concept to many types of intelligent agent systems. Many types of goals have been identified, each with differing properties. The most popular are *achievement* goals, which direct the agent towards an end state to bring about, after which the goal is dropped, such as a vacuuming robot cleaning a room. Once the room has been cleaned, there is no longer any point in keeping the goal to clean the room.

A contrasting notion of goal is that of a *maintenance* goal, which must be kept true indefinitely, such as the vacuuming robot maintaining a certain minimum level of battery power. If this level is breached, or the agent believes a breach will occur in the near future, specific action needs to be taken to restore this condition or prevent it becoming false (such as recharging the battery).

While there has been considerable research concerning achievement goals, research into maintenance goals has received less attention. Although maintenance goals have existed in agent systems for some time, their behaviour often closely mimicked achievement goals. Both goal types often wait for some condition to be met before directing towards executing a plan. The difference was that maintenance goals continued to exist after restoring their condition, whilst achievement goals were dropped once completed.

In [1], mechanisms for *proactive* maintenance goals were identified. Rather than waiting for a triggering condition to be met, if an agent was able to determine that due to its own actions, some maintenance condition would be violated, the agent should halt its current actions and proactively prevent the failure from occurring. This proactive behaviour causes rational behaviour by constraining the agents actions when appropriate.

Previous work on proactive maintenance goals [1] has used resource summaries [2], as a means of predicting maintenance goal violation. This is particularly useful for maintenance goals involving resources, as there is a simple way to define the preventative goal that should be achieved. A lookahead semantics for proactive maintenance goals was introduced in [3], which removed from consideration all future paths in which the maintenance goal was violated. Whilst this is an attractive idea, it effectively only deals with constraints, in that there are no specific goals declared for preventing failure. It therefore requires some other mechanism to prevent 'over-preventative behaviour', such as performing preventative actions repeatedly. In our case, we do not perform such lookahead at every step, but only when scheduling additional goals, and if a violation is anticipated, an appropriate response is scheduled.

This paper builds upon [1] by incorporating reactive and proactive maintenance goals into the Can agent framework [4]. We develop techniques for integrating maintenance goals into an existing framework for achievement goals. We also give an operational semantics for proactive and reactive maintenance goals, and we use these semantics to show that by utilising proactive maintenance goals, waste[1] can be completely eliminated in some circumstances.

The paper is organised as follows. In Section 2, we provide background material regarding our framework's support for reactive and proactive maintenance goals. Section 3 details the required representation, and Section 4 details the operational semantics. We close with a discussion and comparison of similar work in Section 5, and identify some areas of possible future work in Section 6.

2 Background

In this section we discuss the differences between reactive and proactive maintenance goals, and constraints. We also discuss how we can anticipate violation of maintenance goals, and give some background on Can , which will be used for our operational semantics.

2.1 Reactive maintenance goals

Some of the earliest examples of maintenance goals in agent systems, such as [5], are purely reactive; these maintenance goals act as triggers to adopt a new achievement goal when some condition no longer holds. In agent systems such as JACK[6] and Jadex[7], similar functionality can be had by assigning the negation of the maintenance condition as the context condition of an achievement goal. Here, the main difference between maintenance and achievement goal is that the achievement goal is dropped on completion, while the maintenance goal continues to exist.

[1] Superfluous actions that are of no use, refer to [1] for further details.

For example, consider an agent that controls a Mars Rover. The achievement goals of such an agent would be to visit a number of different locations on the planet. The rover consumes fuel as it moves, and there exists a refueling depot on the planet. Given that this agent has elements of autonomy, it must ensure that it never runs out of fuel. This can be modelled as a maintenance goal.

A *reactive maintenance goal* to ensure that the rover always had at least 20 units of fuel in its tank, refuelling when this level was reached, would take the form of *maintain fuel_level > 20.* In the case that this condition is no longer true, the rover should refuel, by pursuing an achievement goal of *refuel*, which fills the rover's fuel tank to 100 units.

This approach is intuitively appealing, but is often too simplistic, and may result in behaviour which is significantly less than optimal. Consider the case when a rover is located at the depot with 30 units of fuel. It has a goal of moving 15 units away. It begins to move, but upon moving 10 units, the maintenance condition is violated. The rover must return to the depot, refuel, and then move to the goal. This small scenario has caused the agent to consume 35 units of fuel (10 units initially, 10 units to return to base, and then 15 units to the goal).

2.2 Proactive maintenance goals

While it is important to respond quickly to sudden changes in the environment, a more active approach to maintenance goals would be to avoid performing actions that would lead to the maintenance goal becoming violated. If the agent is able to anticipate that its current actions will lead to causing a maintenance condition to be violated, it should no longer pursue the goal causing the violation. Before continuing this goal, the agent should instead take actions that prevent the maintenance condition from being violated, if possible to do so.

Returning to the Mars Rover example, a *proactive maintenance goal* has the same form as in the reactive case, i.e. *maintain fuel_level > 20.* The difference here is that if the agent anticipates that the fuel level will be less than 20 as a result of its actions, it should first take preventative action, which in this case is *refuel.*

Given that the rover is located at the fuel depot, has 30 units of fuel and is aiming to visit a location 15 units away, the agent must determine how much fuel will remain after achieving this goal – in this case, 15 units. As this is less than the required 20 units of fuel, the agent must first *refuel.* As it is already located at the depot, it can refuel to 100 units of fuel, and then move to the goal location. This time, the agent has consumed only 15 units of fuel to achieve its goal. In contrast, the reactive case above used 35 units, i.e. 20 more units (or more than double) the amount of fuel than the proactive case.

2.3 Constraints

Earlier work[1] has illustrated the benefits of proactive maintenance goals when compared to reactive maintenance goals, and also mentioned *passive maintenance goals* or *constraints*, which was later discussed further in [3]. A constraint is similar to a proactive maintenance goal, in that the agent will act to ensure that it is never violated. However, there is no appropriate action an agent may take to prevent such a failure from occurring.

Consider if the Mars Rover also had to carry soil samples or rocks it collected at various locations, but for safety reasons, was restricted from carrying over a given amount[2]. There is no action an agent can take to increase the number of samples it may carry, hence this is a constraint.

2.4 Methods of Anticipation

In order to support proactive behaviour, an agent requires some mechanism by which it can anticipate the state of certain attributes in the environment in the future. This anticipation mechanism is crucial to how well the agent will be at preventing maintenance goals from becoming violated.

Reactive maintenance goals are crucial in environments that are hard to anticipate, for example, highly dynamic environments. While proactive maintenance goals extend the deliberation capabilities of an agent and possibly improve efficiency, they are only effective when the anticipation technique is reliable. Therefore, both types of maintenance goals must be used together.

We provide the operational semantics that can include any anticipation method, including lookahead/planning, resource summaries, heuristic based, historic based, or another user defined strategy. These formal semantics are established in the well known CAN [4] framework, incorporating recent developments such as CANPLAN [8] and CANPLAN2 [9].

2.5 CAN

CAN [4] is a high-level agent language, in the same spirit to that of AgentSpeak(L) [10] and Kinny's Ψ[11], both of which attempt to extract the essence of a class of implemented BDI agent systems.

CAN is based around a series of rules about transitions between states of the system. Each state is of the form $\langle \mathcal{B}, \mathcal{G}, \mathsf{P} \rangle$ where $\mathcal{B}$ is the agent's current beliefs, $\mathcal{G}$ the current set of goals, and P the current program (i.e. concurrent set of plans) being executed. CAN contains a number of rules of the form

$$\frac{preconditions}{S \longrightarrow S'}$$

which define the transitions between states of the system. This says that the agent's state will change from S to S' if *preconditions* are satisfied.

2.6 CANPLAN and CANPLAN2

CANPLAN and CANPLAN2 extend the existing CAN rules by incorporating a 'lookahead' mechanism, allowing an agent to utilise a planning mechanism to

[2] This example is similar to the Carrier Agent example from [3]

augment existing BDI approaches (i.e. a plan library) with planning techniques, such as Hierarchical Task Networks. CanPlan2 introduces the Plan construct that allows an agent to plan for a given task, at a specific point in a program.

Section 4 will outline the additional Can rules that are required to support maintenance goals.

3 Representation

As in [1], we assume that the agents are standard Belief-Desire-Intention (BDI)[12] agents. In particular, we require that agents have the notions of beliefs, goals and plans.

3.1 Maintenance Goal

We consider that a maintenance goal is a quadruple tuple consisting of a *maintenance condition*, *failure condition*, *recovery goal* and a *preventative goal*. The *maintenance condition* is a logical expression over the agent's beliefs, and represent the condition that the agent is aiming to keep satisfied. The *failure condition* is also a logical expression, that represents when it is impossible for this maintenance condition to be maintained. For example, if the fuel tank ruptured, the rover should no longer reason about maintaining a desired fuel level. There are two achievement goals associated with a maintenance goal, and these are the *recovery* and *preventative* goals. The *recovery goal* is adopted when the maintenance condition is no longer satisfied. The purpose is to reattain the maintenance condition. In contrast to this, the *preventative goal*'s purpose is to keep the maintenance condition satisfied. We reiterate that the recovery and preventative goals are standard achievement goals, and thus utilise existing notions such as plan selection and priority mechanisms.

3.2 Phases of Operation

In many agent systems, an agent will deliberate over its options before acting. In this work, we make a distinction between these two phases, the *goal adoption* phase, and the *goal achievement phase*. The reason for this is that we view maintenance goals as a means of constraining the agent's behaviour. This means that we not only have to test whether a given set of achievement goals will violate the maintenance goals, but also to determine, if this test fails, a set of achievement goals which will not violate the maintenance goals. Naturally there could be many such methods of determining such an optimal set. In this paper we consider a very simple method, which is to add achievement goals to an "adopted" set of goals one by one, and stopping when a violation of the maintenance goals is predicted. It is possible that more sophisticated mechanism for goal selection can be incorporated here. We then attempt to achieve the set of adopted goals which has just been determined. Upon the achievement of this set of goals, we then return to the adoption phase, i.e. to incrementally expanding our set of adopted goals. In principle, the adoption and achievement phases will be interleaved in a more fine-grained manner; here we will keep to this scheme for simplicity.

Adoption phase In the adoption phase, the agent begins with an empty goal. Each of the existing maintenance goals are checked to ensure they are currently satisfied. The agent may then proceed to adopt a single goal each time, ensuring that the goal set does not violate any maintenance goals it currently has adopted, nor will any maintenance goal be violated in the pursuit of this goal. It is at this point that the agent performs a proactive check on its maintenance goals. The outcome of this phase is expected to be a maximal set of compatible goals which will not cause a conflict with the agent's existing maintenance goals.

Achievement phase Utilising the goal set established in the adoption phase, the agent pursues any of the goals that are available. Only the reactive behaviour of maintenance goals is utilised in this phase, as standard in many existing agent systems.

In normal situations, when reactive maintenance goals are not triggered, standard BDI execution occurs to achieve the goals. However, in situations when a reactive maintenance goal is triggered, the agent suspends attempting to pursue the goals in the current goal set. Instead, the agent begins to pursue the recovery goal for all violated maintenance goals. Only once these recovery goals are complete does the agent resume with the goals in the goal set. This is the only time the reactive maintenance goals are considered.

Once all goals have been completed (with either success or failure), the agent returns to the adoption phase to build a new set of goals that are compatible with the agent's maintenance goals.

3.3 Detailed example

To illustrate the separation of goal adoption and goal achievement phases, we return to the example of the Mars Rover, this time required to visit some locations, collect samples of soil and return them to the base for analysis[3]. Limitations imposed upon the rover are that it must carry less than 3 samples at any one time, and that it always has at least 20 units of fuel in its tank.

Figure 1 illustrates a possible scenario, identifying three locations that the agent must visit. There is a single path of length 10 units between depot D and location A, which is located directly between locations B and C, both of which are located 5 units from A.

Initially, the rover begins with 70 units of fuel in its tank, located at the depot, and has no achievement goals. It begins with two maintenance goals; the first is $fuel > 20$, and the second is $load < 3$.

[3] This example is similar to that in [3], treating the carrier agent as the rover and the packages as soil samples.

Fig. 1. Scenario overview

The rover is given three goals to achieve; collect soil samples from each location (A,B and C) and return them to the depot for analysis.

The agent begins with the goal adoption phase, constructing a set of goals that are compatible with each other and the agent's maintenance goals. In this case, the agent chooses to obtain a soil sample from location A. Achieving this task requires the agent to consume 20 units of fuel, and be carrying one sample. All maintenance conditions are satisfied relative to the current goal set.

The agent then selects a second goal and attempts to place this in the agents goal set, in this case, selecting goal B. Using the algorithms and techniques from [13] to detect positive goal interactions, the agent determines that the cost of achieving all goals in the goal set will be 40 units of fuel, and during this time, it will be laden with 2 samples. All the maintenance goals of the agent remain satisfied. After achieving these goals, the rover is left with 30 units of fuel.

Again, the agent attempts to add a goal to the adopted goal set, this time selecting goal C. The agent determines that the total amount of fuel required for this trip is 40 units of fuel, which continues to satisfy the maintenance goal of $fuel > 20$. However, in achieving this, the rover will be carrying 3 samples, which violates the maintenance condition of $load < 3$. There is no preventative action the agent can take to avoid the failure of this maintenance condition, as it is a constraint. The only option is to avoid adopting this goal at this time.

It is perhaps important to note that in many cases, a reactive maintenance goal would make the agent move to location C and attempt to collect the sample, only then determining that the maintenance condition was violated.

The agent moves to the goal achievement phase, and selects and performs relevant plans to achieve the adopted goals. The final state after successfully completing the goals in its goal set is that the rover is located at the depot with 30 units of fuel and has deposited the samples from A and B.

The goal adoption phases begins again, and the agent attempts to add goal C to the goal set. However, the agent determines that the cost of performing C requires 30 units of fuel, and while it has enough to achieve this goal, doing so will violate the maintenance goal of $fuel > 20$. This maintenance goal is proactive, however, and so the agent can first pursue the recovery goal, in this case *refuel*, prior to achieving goal C. The goal set then appears as *refuel ; goal C*.

There are no other goals to pursue, so the agent moves with the goal adoption phase. As it is already at the depot, it can refuel, bringing its fuel level to 100 units. It can then safely proceed with the goal of collecting a sample from location C.

4 Operational Semantics

In this section we give a formal definition of reactive and proactive maintenance goals in terms of CAN . We assume that for each maintenance goal we have a *maintenance condition* MC (which is the condition to be tested), as well as a *recovery goal* RG, which is to be pursued when a reactive violation is detected, and a *preventative goal* PG, which is to be pursued when a proactive violation is detected. Note that in principle all three of these may be different (although the recovery goal and preventative goal may often be the same). In particular, MC may differ markedly from the recovery goal, such as in the Mars Rover case (so that when the rover is low on fuel, it fills up the tank, rather than just restore the fuel supply to just over 20 units).

4.1 A Single Maintenance Goal

Let us consider the case of an agent with a single maintenance goal. The agent is given the maintenance task of ensuring MC remains true (for simplicity, we assume that MC is valid in the initial state). In the event of $\neg\mathsf{MC}$, the agent posts an event RG_e to pursue the reactive goal RG. When the agent can anticipate $\neg\mathsf{MC}$, the agent posts an event PG_e to pursue the proactive goal PG.

Intuitively, the recovery goal should be activated whenever the agent believes the maintenance condition has been violated. This leads us to the rule below.

$$\frac{\mathsf{B} \models \neg\mathsf{MC}}{\langle \mathsf{B}, \mathsf{G}, \mathsf{P} \rangle \longrightarrow \langle \mathsf{B}, \mathsf{G}, \mathsf{RG}_e; \mathsf{P} \rangle} \; RM$$

Note that the event RG_e is used here, in order to use the existing plan infrastructure of CAN to trigger the appropriate behaviour (i.e. the execution of the appropriate plans to achieve the recovery goal RG). Note also that the state of the current execution goes from P to $\mathsf{RG}_e; \mathsf{P}$, thus ensuring that the recovery goal has priority over whatever else is being done currently. In principle, it is possible to merely add the recovery goal to the existing set of achievement goals, and allow whatever prioritisation mechanism is used to determine an appropriate schedule (i.e. by using $\mathsf{RG}_e \parallel \mathsf{P}$ instead). Here we use this simpler method, which has the straightforward behaviour that the recovery goal has priority over all else.

When a maintenance goal detects that a violation will occur due to the adoption a new achievement goal, the rational course of action would be to not allow progress to be made towards the achievement of that goal – doing so renders the agent liable to encountering a circumstance where it is potentially lacking in resources. Therefore, that achievement goal should be postponed until the relevant preventative goal is executed, and then allowing the achievement goal to progress.

In the case of a constraint, there is no applicable preventative goal. In this case, the rational approach is to not adopt the achievement goal. The goal should be allowed to persist, as a pending goal, so that if the environment changes such that the violation would not occur, the goal could then be adopted.

The main issue is how to specify the predictive behaviour. We will address this question in more detail in Section 4.3 when discussing the $\mathsf{proactive\_check}$; for now, we note that the desired behaviour is of the following form.

$$\frac{\mathsf{B}, \mathsf{proactive\_test}(\mathsf{G} \cup \mathsf{G}_a) \models \neg \mathsf{MC}}{\langle \mathsf{B}, \mathsf{G}, \mathsf{P} \rangle \longrightarrow \langle \mathsf{B}, \mathsf{G}, \mathsf{PG}_e; \mathsf{P} \rangle} \; PM$$

where $\mathsf{proactive\_test}(\mathsf{G} \cup \mathsf{G}_a)$ specifies the updates to the beliefs B corresponding to the achievement of the goals $(\mathsf{G} \cup \mathsf{G}_a)$. Hence this rule states that if the agent believes that achieving G_a in addition to the current (as yet unachieved) goals G will violate the maintenance condition, then the preventative goal PG should be scheduled. Accordingly, the adoption phase should be ended, and an achievement phase commenced. For the second option (i.e. when neither the preventative goal nor the violating goal should be adopted), we simply end the adoption phase with the current set of goals.

As above, note that the preventative goal takes preference over all other goals. Again, one could use more general mechanisms if desired, but we use this for simplicity.

4.2 Multiple Maintenance Goals

Note that the above analysis assumes the existence of a single maintenance goal only. In principle, it is simple enough to extend the above rules to multiple maintenance goals, as we show here. We will assume that that these are mutually consistent and do not conflict.

In this case, the maintenance condition MC is a set of formulae such that $\mathsf{MC} = \mathsf{M}_1, \ldots, \mathsf{M}_n$. We denote by MT the set of all formulae M_i in MC such that $\mathsf{B} \models \mathsf{M}_i$. We denote by MF the set of all formulae M_i in MC such that $\mathsf{B} \models \neg \mathsf{M}_i$. Without loss of generality, let $\mathsf{MT} = \{\mathsf{M}_1, \ldots \mathsf{M}_j\}$ and $\mathsf{MF} = \{\mathsf{M}_{j+1}, \ldots \mathsf{M}_n\}$.

We can then write the above two rules as below.

$$\frac{\mathsf{B} \models \neg \mathsf{M}_1 \wedge \ldots \wedge \neg \mathsf{M}_j}{\langle \mathsf{B}, \mathsf{G}, \mathsf{P} \rangle \longrightarrow \langle \mathsf{B}, \mathsf{G}, (\mathsf{RG}_{ei} \parallel \ldots \parallel \mathsf{RG}_{ej}); \mathsf{P} \rangle} \; RMM$$

$$\frac{\mathsf{B}, \mathsf{proactive\_test}(\mathsf{G} \cup \mathsf{G}_a) \models \neg \mathsf{M}_1 \wedge \ldots \wedge \neg \mathsf{M}_j}{\langle \mathsf{B}, \mathsf{G}, \mathsf{P} \rangle \longrightarrow \langle \mathsf{B}, \mathsf{G}, (\mathsf{PG}_{ei} \parallel \ldots \parallel \mathsf{PG}_{ej}); \mathsf{P} \rangle} \; PMM$$

It is unlikely that many maintenance goals are violated at a single instance, but in cases where this does occur, the rational approach is to execute all relevant preventative (or recovery) goals. Therefore, the recovery goals are added in parallel – this allows the goals to be executed in any order, and interleaved as per normal CAN rules.

Whilst this is not a difficult extension, in the rest of the paper we will consider only a single maintenance goal, in order to simplify the presentation. Note that in this case we do not impose an order on the multiple recovery or preventative goals, but we do insist that all such goals (i.e. all recovery goals or all preventative goals) are achieved before any others.

4.3 The proactive_test construct

The proactive_test construct is crucial for an agent's proactive behaviour to maintenance goals. This construct must determine how the future may evolve, based on the current achievement goals and beliefs of an agent. The effectiveness of agent with regards to its maintenance goals is largely based on how accurate the prediction of future states is.

The objective is to construct a maximal subset of goals, which are compatible with each other and all maintenance goals, from the set of all goals the agent currently is considering. We refer to the goals that fall into this set as *adopted* goals. The agent can then pursue these goals freely.

In this section, we present several methods of writing the proactive_test construct, and provide some short discussion on the attributes of each. These approaches differ in their efficieny and practicallity.

proactive_test with CANPLAN2 The proactive_test construct may be constructed by utilising the planning mechanisms from CANPLAN2 . The idea is to use the planning functionality of CANPLAN2 to specify the predictive behaviour of the system. In particular, we use the $\xrightarrow{PLAN}$ relation to determine whether the plans which will be used to achieve the specified goals will always satisfy the maintenance condition or not. For example, if the Mars Rover wants to adopt goals A and B, we check all the plans to achieve these goals to ensure that every step of the plan does not violate the maintenance condition. In other words, we want to check that for every transition $\langle \mathsf{B}, \mathsf{P} \rangle \longrightarrow \langle \mathsf{B}', \mathsf{P}' \rangle$ in the achievement of these goals, we have that if $\mathsf{B} \models \mathsf{MC}$, then $\mathsf{B}' \models \mathsf{MC}$. Hence given a set of goals $\mathsf{G}_1, \ldots \mathsf{G}_n$, with a corresponding set of events $e_1 \ldots e_n$ we can write this condition as

$$\begin{gathered}\forall B' \text{ such that } \langle \mathsf{B}, !e_1 \parallel \ldots \parallel !e_n \rangle \xrightarrow{PLAN} \langle B', P' \rangle, \\ \text{we have } B' \models \mathsf{MC}\end{gathered}$$

To construct a safe goal set, the agent begins with an empty set, and adds a single goal each step, ensuring that the new goal set remains safe.

$$\langle \mathsf{B}, !e_1 \parallel \ldots \parallel !e_n \rangle \xrightarrow{PLAN} \langle \mathsf{B}'', P'' \rangle\ \mathsf{B}'' \models MC$$

given that

$$\langle \mathsf{B}, !e_1 \parallel \ldots \parallel !e_{n-1} \rangle \xrightarrow{PLAN} \langle \mathsf{B}', P' \rangle\ \mathsf{B}' \models MC$$

The event for each new goal is added to the goal set, and the agent may then plan for this new set, ensuring that the maintenance condition still holds.

As an optimisation, it may be possible for the agent to not require checking the entire goal set, only the additional actions that the new goal contributes.

$$\text{Let } \varDelta \text{ be } \langle \mathsf{B}, \mathsf{P} \rangle \in \overset{PLAN}{\longrightarrow} \langle \mathsf{B}, !e_1 \parallel \ldots \parallel !e_{n-1} \rangle$$

$$\text{Let } \varDelta' \text{ be } \langle \mathsf{B}, \mathsf{P} \rangle \in \overset{PLAN}{\longrightarrow} \langle \mathsf{B}, !e_1 \parallel \ldots \parallel !e_{n} \rangle$$

$$\text{Check } \varDelta' - \varDelta$$

Utilising the planning construct, the agent can determine the outcome of a set of goals. In an ideal environment, if utilising $\overset{PLAN}{\longrightarrow}$ has provided a set of goals that does not conflict with its maintenance goals, when the agent actually performs the actions (by using $\overset{CAN}{\longrightarrow}$ transitions) associated with this set, it can be certain that if the maintenance condition is currently held, it will continue to hold in the next transition.

$$\langle \mathsf{B}, \mathsf{P} \rangle \overset{CAN}{\longrightarrow} \langle \mathsf{B}', \mathsf{P}' \rangle, \mathsf{B} \models MC \text{ then } \mathsf{B}' \models MC$$

This mechanism is most similar to the method presented in [3], and shares the same flaws – to get accurate prediction, the agent must check all possible future states. This could be time consuming or even impossible in some circumstances.

proactive_test with resource summaries A second, more pragmatic approach to constructing the proactive_test construct is via resource summaries. From these summaries, which are generated once, the agent can determine the *necessary* and *possible* resources required for any given goal to succeed.

The necessary resources are based on the resources that are common to all possible means of achieving a goal – an agent unable to satisfy the necessary resource condition cannot achieve the goal by any means.

The possible resources are determined based on the resource requirements of all possible plans an agent has to achieve a goal – an agent that can satisfy the possible resources, can achieve that goal.

If the agent can satisfy the necessary resources, but cannot satisfy the possible, then the outcome is *uncertain.* The success of achieving the goal depends on which goals and plans are selected. In such a case, the agent may choose to proceed if it is a bold agent, but it may choose to pursue a preventative goal first if it is a cautious agent[4].

User defined strategies for proactive_test It is also possible to allow the agent developer to design their own proactive_test mechanism. In this way, the developer can incorporate appropriate technologies that can better estimate and predict future states based on the agent's beliefs and goals.

[4] The use of the terms 'bold' and 'cautious' for describing agent behaviour follows from [14] .

4.4 Phase rules

The rules providing the two distinct phases related to maintenance goals in CAN are given in Figure 2. It should be noted that in these rules, reactive maintenance goals are not considered – they are not part of the deliberation cycle, and are only employed during the actual CAN execution cycle. We expect that small changes to some existing CAN rules will be necessary, to ensure that these may only execute when the maintenance goals have not been violated. This is as simple as adding $\mathsf{B} \models \mathsf{M}$ as the precondition of the existing CAN steps.

$$\frac{\mathsf{proactive\_test}(B, G_A \bigcup \{G\}, P) \qquad G \in G_P}{adopt(B, G_A, G_P, P) \longrightarrow adopt(B, G_A \bigcup \{G\}, G_P \backslash \{G\}, P)} \; \textit{Proactive Check Succeeds}$$

$$\frac{\neg\mathsf{proactive\_test}(B, G_A \bigcup \{G\}, P) \qquad G \in G_P}{adopt(B, G_A, G_P, P) \longrightarrow achieve(B, G_A, G_P, P; PG_e)} \; \textit{Preventative Goal}$$

$$\frac{G \neq nil}{adopt(B, G, \phi, P) \longrightarrow achieve(B, G, \phi, P)} \; \textit{Begin Achieve Phase}$$

$$\frac{(B, P) \overset{CAN}{\longrightarrow} (B', P'), \; P' \neq nil}{achieve(B, G_A, G_P, P) \longrightarrow achieve(B', G_A, G_P, P')} \; \textit{Intermediate } \text{CAN} \; \textit{step}$$

$$\frac{(B, P) \overset{CAN}{\longrightarrow} (B', nil)}{achieve(B, G_A, G_P, P) \longrightarrow adopt(B', \phi, G_P, nil)} \; \textit{Final } \text{CAN} \; \textit{step}$$

Fig. 2. Adoption and Achievement rules in CAN

The initial state of the agent is $adopt(B, \phi, G_P, nil)$. The agent develops its adopted goal set (G_A) by selecting a single goal (G) from the pending goal set (G_P), and determining if the goal is compatible with other goals in its adopted goal set relative to its maintenance goals. The agent utilises the previously defined proactive_test construct, which determines if the goal set provided will continue to satisfy the agent's maintenance goals.

There are two possible outcomes of the proactive_test. If this construct indicates that the goal set is valid (*Proactive Check Succeeds*), that goal is added to the agent's adopted goal set ($G_A \bigcup \{G\}$), and the process repeats for the next goal.

The second outcome of the proactive_test is that the addition of the new goal would cause a maintenance goal to be violated. For proactive maintenance goals, the *Preventative Goal* is used. Rather than adopting the new (conflicting) achievement goal, the preventative goal (PG) for the violated maintenance goal is adopted instead. The plan set evolves to $P; PG_e$. This represents the agent pursuing the goals in the current goal set, then performing the preventative goal. Once this is complete, the agent enters the goal adoption phase again and may select to pursue this achievement goal if desired.

When the *Begin Achieve Phase* rule is applied, the agent moves from the goal adoption phase into the goal achievement phase, where it begins the actual execution of the goals that have been placed in the agent's adopted goal set, G_A. There is no point in proceeding to the achievement phase if there are no goals to pursue. While this could occur at any time, it is expected that this rule would occur in the event that the agent has adopted all the goals from its pending goal set.

Once the agent has entered the achievement phase, the agent follows Can behaviour, with the additional rules for reactive maintenance goals. No proactive checks are made during the achievement phase. The agent continues to perform Can steps until all the goals in the agent's adopted plan set are completed by their associated plans becoming completed. Once this occurs, the plan set becomes *nil* and the agent returns to the goal adoption phase.

4.5 Checking Perfect Prediction

Due to the predictive nature of proactive checking for violation of maintenance goals, it is impossible to guarantee success in typical agent environments. Despite this it is sensible to insist that in idealised conditions (i.e. in which there are no untoward environmental changes, and every planned action succeeds in exactly the way it is intended), proactive checking should be perfect. One way to measure this is to show that under such conditions, no reactive check for violation will ever succeed.

Intuitively, due to the way that proactive checking is defined in terms of $\xrightarrow{PLAN}$ in Section 4.3, given beliefs B, a set of adopted goals G and plan P, we know that $\forall \mathsf{B}'$ such that $\langle \mathsf{B}, \mathsf{P} \rangle \xrightarrow{PLAN} \langle \mathsf{B}', \mathsf{P}' \rangle$ we have $\mathsf{B}' \models \mathsf{MC}$. Hence by our assumption about idealised conditions, we know that every belief state B'' such that $achieve(\mathsf{B}, \mathsf{G}, \mathsf{G}_P, \mathsf{P}) \longrightarrow achieve(\mathsf{B}'', \mathsf{G}, \ \mathsf{G}_P, \mathsf{P}'')$ is such that $\mathsf{B}'' \models \mathsf{MC}$.

We sketch the formal statement of this property and its proof below. A more detailed account of the formalism is beyond the scope of this paper.

Proposition 1 *Let A be an agent in an idealised environment, and Let* $\langle \mathsf{B}, \mathsf{G}, \mathsf{P} \rangle$ *be an agent configuration of A such that* $adopt(\mathsf{B}, \emptyset, \mathsf{G}, nil) \xrightarrow{*} achieve(\mathsf{B}, \mathsf{P})$. *Then for every* B' *such that* $\langle \mathsf{B}, \mathsf{P} \rangle \xrightarrow{CAN} \langle \mathsf{B}', \mathsf{P}' \rangle$ *we have* $\mathsf{B}' \models \mathsf{MC}$.

Proof Sketch: Assume by contradiction that there is a configuration $\langle \mathsf{B}', \mathsf{P}' \rangle$ such that $\langle \mathsf{B}, \mathsf{P} \rangle \xrightarrow{CAN} \langle \mathsf{B}', \mathsf{P}' \rangle$ and $\mathsf{B}' \models \neg\mathsf{MC}$. By the definition of *adopt*, we have that $\forall \mathsf{B}''$ such that $\langle \mathsf{B}, \mathsf{P} \rangle \xrightarrow{PLAN} \langle \mathsf{B}'', \mathsf{P}'' \rangle$ we have $\mathsf{B}'' \models \mathsf{MC}$. By the idealised assumption, this is implies that $\forall \mathsf{B}''$ such that $\langle \mathsf{B}, \mathsf{P} \rangle \xrightarrow{CAN} \langle \mathsf{B}'', \mathsf{P}'' \rangle$ we have $\mathsf{B}'' \models \mathsf{MC}$, which is a contradiction. ∇

An immediate corollary of this is that under such conditions, the rule *RM* will never be used.

Naturally the idealised assumption is unrealistic, which makes it necessary to incorporate both the proactive and reactive mechanisms. It is reasonable to conclude, though, that we may judge the quality of the prediction used in the proactive mechanism by the number of times that the reactive trigger is fired. As the above result shows, in a perfect world, the reactive rule is never used. This corresponds to having zero waste, as defined by [1].

Note also that the use of *possible* and *necessary* resource summaries provides a pragmatic method of dealing with imprecision in the measurement of the resources used, as well as an appropriate mechanism for coping with less than ideal environments. In other words, when perfect prediction is an idealised environment is not possible (i.e. most of the time!), resource summaries are a useful heuristic.

It is also noteworthy that there may be weaker forms of the proactive check which may be appropriate. In particular, in the case when some plans will lead to violation of the maintenance goals and others will not, then it seems sensible to exclude the violating plans from further consideration. In this way the techniques of [3] could be combined with ours to produce a system in which the proactive check rests on weaker assumptions than in this paper.

5 Discussion

Current research in proactive maintenance goals is still limited. In [3], Hindriks and van Riemsdijk introduce a semantics for maintenance goals based in the GOAL language.

To determine if maintenance goal violation will occur, [3] uses a *lookahead* function with argument *lookahead range*, which allows the agent to determine the effects of its current actions for a possible future state, bounded by the size of the lookahead range. If the agent detects that its current action would lead to a violation of its maintenance goal in the future, it would instead execute some alternative, though unspecified, action.

In our approach, the use of CanPlan mechanisms allows behaviour akin to this lookahead beahviour. Additionally, we also promote the use resource summaries to determine maintenance goal violation, and use preventative goals to avoid future violation. Resource summaries are computationally efficient [15]. In addition, the use of preventative goals and recovery goals allow agents to work with declarative information, and to be able to trace actions to their corresponding goal of origin.

In [3], this lack of declarative information results in behaviour more akin to planning, or pruning the tree of possible futures to narrow the choice of possible next actions. However, the action that is executed by the agent may have no effect on the status of the violated maintenance goal.

One major difference between our original work and [3] is that ours checked for future violation only during the adoption phase of the agent, while in [3], this check could be performed at any stage. This can be useful in cases where the reactive maintenance goal is not triggered after an unexpected change in the environment. Take for example, a rover that suddenly springs a leak in its fuel tank and loses an amount of fuel, such that it would no longer be able to achieve its goal, but did not yet trigger its reactive maintenance goal. [3] may detect and respond to this change immediately, while the work presented in this paper would rely on the reactive maintenance goal to eventually be triggered.

An ideal solution would incorporate concepts from both approaches. As [3] can check for violation at every step, it is not a realistic solution in practice. While the work we have presented in this paper is practical, in some cases we would desire the agent to perform its proactive checks more often. A solution that allows for proactive checks to be made at any time, while not spending all its time doing so, appears as the most rational approach.

6 Conclusion and Future Work

This paper has presented a formal approach to proactive and reactive maintenance goals. The CAN framework has been extended to incorporate two distinct phases of operation, for goal adoption and goal achievement. Utilising these, the differences between reactive and proactive maintenance goals has been illustrated, and the mechanisms by which these goals can be implemented have been outlined in formal semantics. A proof showing the correctness of proactive maintenance goals in ideal environments has also been presented.

One interesting area of future research is allowing an agent to willfully allow a maintenance goal to be violated. If the agent can determine the degree to which a maintenance goal will be violated, and the effects of such violation, more rational behaviour could be expected. In the Mars Rover scenario, we may allow it to carry more than the safe limit only for a short distance.

In the examples we have presented, we have assumed that maintenance goals exist for the entirety of the agent's life cycle. It is also viable that maintenance goals could be adopted or dropped in a similar fashion to achievement goals. If the Mars Rover had a delicate item that it was transporting to the depot, a maintenance goal of $\textit{speed} < \textit{10km}$ may be an appropriate maintenance goal to adopt whilst performing the transport object goal. On completion of this transport goal, the maintenance goal is no longer required and can also be dropped.

A third area of research is in the area of maintenance goals with *urgency*. Consider a boiler whose temperature is controlled by an agent. It has a maintenance goal to keep the temperature less than 100^o. The agent is more likely to act as the temperature approaches 100^o. The agent may not pursue the preventative goal when the temperature is 60^o, but at 90^o, the preventative goal is likely to be a highly prioritised goal. It may also be important for the agent to consider aspects such as the rate of temperature increase or historical information in such a scenario.

The application of maintenance goals to the area of soft goals is also something that could be considered. A soft goal of minimising the amount of fuel used by an agent is directly related to the fuel-related maintenance goals presented in this paper. The agent aims to ensure that it is fuelled to a satisfactory level at all times, yet it does not want to refuel to often. We can model this behaviour by adding the soft goal of minimising fuel usage to the existing set of goals.

References

1. Duff, S., Harland, J., Thangarajah, J.: On Proactivity and Maintenance Goals. In: AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, USA, ACM Press (2006) 1033–1040
2. Thangarajah, J., Winikoff, M., Padgham, L., Fischer, K.: Avoiding resource conflicts in intelligent agents. In: Proceedings of the European Conference on Artificial Intelligence (ECAI'02). (2002)
3. Hindriks, K., van Riemsdijk, M.B.: Satisfying Maintenance Goals. In: Lecture Notes in Artificial Intelligence, Springer (2007)
4. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and Procedural Goals in Intelligent Agent Systems. In: Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR'02). (2002)
5. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: AAAI. (1987) 677–682
6. Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A.: Jack Intelligent Agents - Components for Intelligent Agents in Java (1999)
7. Braubach, L., Pokahr, A., Moldt, D., Lamersdorf, W.: Goal Representation for BDI Agent Systems. In: Proceedings of the AAMAS Workshop on Programming in Multi-Agent Systems (PROMAS'04). (2004) 44–65
8. Sardina, S., de Silva, L., Padgham, L.: Hierarchical planning in bdi agent programming languages: a formal approach. In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2006) 1001–1008
9. Sardina, S., Padgham, L.: Goals in the context of BDI plan failure and planning. In: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), Hawaii, USA, ACM Press (May 2007) 16–23 To appear.
10. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: MAAMAW '96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away, Secaucus, NJ, USA, Springer-Verlag New York, Inc. (1996) 42–55
11. Kinny, D.: The Psi Calculus: An Algebraic Agent Language. In: ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII, London, UK, Springer-Verlag (2002) 32–50
12. Rao, A.S., Georgeff, M.P.: BDI-agents: From Theory to Practice. In: Proceedings of the First International Conference on Multiagent Systems (ICMAS'95), San Francisco (1995)
13. Thangarajah, J., Padgham, L., Winikoff, M.: Detecting & exploiting positive goal interaction in intelligent agents. In: AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2003) 401–408
14. Pollack, M.E., Ringuette, M.: Introducing the tileworld: Experimentally evaluating agent architectures. In: AAAI. (1990) 183–189
15. Thangarajah, J., Padgham, L.: An empirical evaluation of reasoning about resource conflicts. In: AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society (2004) 1298–1299