

On Proactivity and Maintenance Goals

Simon Duff
RMIT University
Melbourne, Australia
sduff@cs.rmit.edu.au

James Harland
RMIT University
Melbourne, Australia
jah@cs.rmit.edu.au

John Thangarajah
RMIT University
Melbourne, Australia
johthan@cs.rmit.edu.au

ABSTRACT

Goals are an important concept in intelligent agent systems, and can take a variety of forms. One such form is *maintenance goals*, which, unlike *achievement goals*, define states that must remain true, rather than a state that is to be achieved. Maintenance goals are generally restricted to acting as trigger conditions for goals or plans, and often take no part in any deliberation process. These goals are *reactive* and are only acted upon when the maintenance conditions are no longer true. In this paper, we study maintenance goals that are *proactive*, in that the agent system needs to not only react when the maintenance conditions fail, but also anticipate the failures of these conditions, and act in order to avoid them failing. This can be done by performing actions that prevent the condition from failing, or suspending goals that will cause the maintenance conditions to fail. We provide a representation for maintenance goals that captures both their reactive and proactive aspects, algorithms that identify in advance where maintenance conditions may not hold, and mechanisms for enabling preventative actions in such situations. We also provide some experimental results on an implementation of these ideas.

Categories and Subject Descriptors

D.3.1 [Programming Languages]: Formal Definitions and Theory; I.2.5 [Artificial Intelligence]: Programming Languages and Software; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods—*Representations (procedural and rule-based)*

Keywords

Agent programming languages, Formal models of agency, Task and resource allocation in agent systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...\$5.00.

1. INTRODUCTION

Goals are a crucial concept in agent programming languages. Agent properties such as proactiveness, reactivity, autonomy and social ability can be described in terms of the goals of the agent. A *proactive* agent is required to actively work towards achieving its goals, while a *reactive* agent is required to adapt to environmental changes in order to maximise its ability to achieve its goals. An *autonomous* agent is free to choose how to achieve its goals, how to resolve conflicts between its goals and whether to drop goals which appear to be unachievable. A *social* agent may choose to adopt a goal on a request from another agent, or to request another agent to adopt one of its goals. Accordingly, there are a number of agent systems in which goals are a central concept, such as 3APL[5], KAOS[4], and Tropos[7].

Goals can hence be seen as a link between the agent's reasoning and its actions; actions are only undertaken in order to achieve one or more of the agent's goals, and changes in the status of the agent's goals are reflected in changes in its actions.

For the most part, goals in agent systems are *achievement goals*, i.e. goals which are not currently true, and which the agent, by means of pursuing the appropriate actions, acts to make true. However, the agent has no obligation once the goal becomes true, in that there is no requirement to act to ensure that goal remains true.

There are some systems (such as Jadex[9] and JAM[8]) which include *maintenance goals* as well as achievement goals. In such systems, a maintenance goal is one which is initially true, and if it becomes false, the agent will take action to make it become true. Hence such goals act as a trigger, in that it is only when they become false that any action is taken. We refer to such goals as *reactive maintenance goals*, as no action is taken for such goals until the goal becomes false.

In this paper we extend the notion of maintenance goals to include *proactive* behaviour as well. In other words, an agent may take action not only to bring about an achievement goal or in response to a maintenance goal becoming false, but also to ensure that a maintenance goal will remain true. Hence the agent reasons about possible future events, and perform actions to avoid maintenance goals becoming false (as well as possibly taking actions if any such goals do become false).

For example, consider an agent controlling a Mars Rover, which has a number of target locations it is required to visit (the agent's achievement goals), while ensuring that certain conditions related to the safe operation of the rover are maintained (the agent's maintenance goals). A typical

example of a maintenance goal could be to ensure that the rover has sufficient battery charge to perform its activities, recharging when it has less than 20% of its battery remaining [1]. The rover knows several locations where it can recharge its battery. The process of moving consumes charge, and the electrical usage of the rover may be predictable in some cases (for example, where the rover has visited some location before). In such a case, the rover can determine how much charge it will use, and the expected battery level it will have upon reaching its destination. It would appear rational for a rover to anticipate that its maintenance condition may be unsatisfied if it attempts to move towards the location immediately – instead, it should take the opportunity to recharge its battery whilst it is near the charger, prior to moving to a distant location.

A second example can be found in the domain of *autonomic computing*. An intelligent operating system may be given the goal to maintain the availability of a cluster of machines by preventing all machines being shut down at once. There is no obvious mechanism that would allow a reactive maintenance goal to ensure that all machines are active, without first allowing all machines to be shut down, recognising that the maintenance condition is violated, and restarting a machine to re-establish the maintenance condition. On the other hand, a proactive maintenance goal would prevent the last machine from being shut down. We believe that the domain of autonomic computing is a rich one for proactive maintenance goals.

A final example consists of a boiler that produces some output power. An agent is able to control the output power by increasing or decreasing the temperature of the boiler. This example can be used to illustrate how maintenance goals can be made to interact. As a safety measure, the agent may have a maintenance goal to ensure that the boiler's temperature never exceeds some amount. There may also be a second maintenance goal that the output power always be greater than some minimum value. The agent may have several achievement goals, such as providing power for several devices, which require the agent to increase the temperature to produce enough power. A rational agent may limit the number of devices that can concurrently be powered, to ensure that the temperature required is not too elevated. The rational agent may also choose to ensure that there is always some device being powered by the boiler, to avoid wasted energy.

Hence, achievement goals drive the agent to perform actions to produce a desired state; maintenance goals either trigger corrective actions to be taken, or constrain the choice of actions so that the maintenance goals will not be violated.

This paper is organised as follows. In Section 2, we provide background material to a number of elements concerning the agent framework support for proactive maintenance goals. Next, in Section 3 we provide a detailed explanation of the requirements and mechanisms an agent system should incorporate to support our ideals of proactive maintenance goals. A discussion of mechanisms for implementing proactive behaviour follows in Section 4. We have done some preliminary experiments to evaluate our work which we present in Section 5. We conclude with possible future directions in Section 6.

2. MAINTENANCE GOALS

The earliest examples of maintenance goals (such as [6])

are purely reactive, i.e. they act as triggers to adopt an extra achievement goal, and only when the stated condition becomes false. In the Mars rover example, a reactive maintenance goal of maintaining a battery level of at least 20% will only cause the agent to act once the level drops below this amount. Hence the maintenance goal can be completely characterised as a trigger which adds an achievement goal such as “recharge the battery”. In a system such as JACK[2], this is not very different from adding the negation of the maintenance goal as a context condition for the plan which fills the tank.

The life cycle of such reactive maintenance goals is discussed in some detail in [1]. As discussed in that paper, “a maintenance goal is introduced to observe and maintain some world state as long as the goal exists”. Whilst the discussion of goal types and the way in which goals are managed during the execution cycle is quite comprehensive, their use of maintenance goals remains reactive.

A more active approach to maintenance goals includes not only triggering some recovery action once the goal is violated, but *taking actions in anticipation of the maintenance goal becoming false*. As mentioned above, this will presumably lead to more rational behaviour, such as the battery-charging activity of the robots discussed in [1] taking place shortly before the low power threshold is reached, rather than just after it. However, there is no point in recharging the robot too often, such as when the battery is 85% charged, just because the robot happens to be nearing a charging point. In other words, the agent has to have a reasonable expectation that the maintenance goal is likely to be violated in the near future; action is then taken to ensure that it is not violated.

In some sense, this approach assumes that it is important to prevent the maintenance goal from becoming false. For example, consider a maintenance goal to maintain the pressure of a gas cylinder at no more than a certain amount. Whilst emergency action is clearly needed if the pressure exceeds this value, rational management of this property would also include actively preventing the pressure from getting too close to the limit.

Hence the proactive approach takes action not only when the maintenance goal is violated, but also when the agent anticipates that it will be violated in the near future. In order to perform such reasoning, the agent needs to be aware of the effects of other goals on the maintenance condition. This can be of the form of altering the belief of an agent, or consuming or producing some resource involved in the maintenance condition. Therefore, we require the effects and required resources of goals to be explicitly represented in the structure of the agent goals, as well as having these structures utilised in the agent execution cycle.

It should be noted that maintenance goals are a means of improving the rationality of the agent by using a richer class of goals. Hence they can be seen as a way of more closely approximating perfectly rational behaviour whilst remaining intuitively simple. The main thrust of this paper is to show that the extra subtlety gained comes at a cost of comparatively little efficiency.

3. REPRESENTATION

Our agents are based on standard implementations of the Belief-Desire-Intention (BDI)[11]. In particular, our central notions are *beliefs*, *goals* and *plans*.

The agents we develop have a pre-defined set of plans that are utilised at run-time to satisfy goals. The agents also have belief-bases which contains facts about the world known to the agents. In this section we describe a representation for an agent's beliefs, plans, achievement goals and maintenance goals. We build on some of our earlier work to provide a representation for achievement goals and plans [12, 13], and provide a new structure for maintenance goals.

Beliefs can be represented in many different forms. For example, in AgentSpeak [10] they are represented in first order logic. In our framework, we present a restricted form of beliefs in order to better explain the reasoning mechanisms in later sections. We represent a belief as a tuple, $\langle Name, Type, Value \rangle$, where *Name* is an alphanumeric string which allows the agent's beliefs to be uniquely referenced, *Type* defines the type of object the Value attribute represents, and *Value* is the actual data representing the belief. Some examples of beliefs are:

```
(FuelLevel, Number, 60)
(GripperEmpty, Boolean, True)
(RoverID, Text, Rover1)
```

Plans are lists of instructions an agent can follow to potentially establish a desired state with respect to some goal. A plan is a tuple of the form $\langle Type, Pre-Condition, Effects, Plan Body \rangle$. *Type* is an alphanumeric string which identifies the type of the plan. The *Pre-Condition* is an expression over the agent's beliefs which must be satisfied before this plan can be executed. If it is not satisfied, this plan cannot be instantiated. In systems like JACK, pre-conditions are represented as *context conditions*. The *Effects* denote the changes to the agent's belief set that will occur if the plan is executed successfully.

The *Plan Body* consists of sub-goals and actions, which can be combined by either sequencing them (denoted by ;), or by executing them in parallel (denoted by ||). Sub-goals are treated like top-level goals. Actions can be any arbitrary code specified by the user.

An example of a plan is as follows:

```
Type RefuelRoverPlan
Pre-Condition FuelLevel ≥ 50
Plan Body {FindRefuelPoint ; MoveToRefuelPoint ;
           Refuel}
Effects {Update(FuelLevel,100)}
```

Achievement goals are states of the world that the agent works toward establishing. We follow the definition of [15] and represent them as follows.

$\langle Type, Success Condition, Failure Condition, Plans \rangle$
Type is an alphanumeric string that uniquely identifies a goal. *Success Conditions* are the desired outcomes of the goal and when true indicates the goal has been successfully achieved. The *Failure Condition* when true indicates the goal cannot be achieved by the agent by any means possible, and hence the agent should no longer pursue it.

The *Plans* attribute is a set of plan names which achieves the success condition. When an instance of an achievement goal is created, the agent selects a plan from this set and if its pre-condition is satisfied, begins executing it. If the plan fails, then another plan from the plan set is selected and tried. If no applicable plan exists, the goal is dropped. This mechanism is typical of BDI agent systems. An example of an achievement goal is:

```
Type: GatherSoilSample
Success Condition: ObtainedSoilSamples = true
Failure Condition: GripperBroken = true || RoverImmobile = true
Plans: {GatherSoilAtXPlan, GatherSoilAtYPlan}
```

This example describes a goal whose purpose is to obtain a soil sample, and it has two options in terms of the location which has soil (X and Y), hence two plans. If either plan succeeds, the rover will have obtained said sample, and the agent's belief of *ObtainedSoilSample == true* will be satisfied. The goal can (and should) be abandoned if the gripper, used to gather soil, is broken, or if the rover is incapable of moving.

As discussed earlier, our representation of maintenance goals needs to support both proactive reactive aspects.

Maintenance goals are tuples of the form $\langle Type, Maintenance Condition, Failure Condition, Recovery Plans, Preventative Goal \rangle$.

The *Type* and *Failure Condition* is similar to that of an achievement goal indicating the type of the goal and the condition which when true makes the goal no longer applicable.

The condition (belief) the goal requires the agent to maintain is given by *Maintenance Condition*. If this condition becomes false, the agent selects an applicable plan (i.e., one whose pre-condition is currently true) from *Recovery Plans*, in order to re-establish the maintenance condition.

In contrast to purely reactive maintenance goals, if the agent determines that the maintenance condition is currently true, but is anticipated to become false in the future, it will select and execute an applicable goal from *Preventative Goal*. These goals are designed to ensure that the maintenance condition remains true.

An example of a maintenance goal follows, which aims to ensure that the rover has an adequate battery level, recharging if the level drops below a certain threshold.

```
Maintenance Condition: BatteryLevel ≥ 20%
Failure Condition: ChargerBroken = True
Recovery Plans: {RechargeBatteryPlan} (Move a robot to a charging
                station and allow it to recharge its battery)
Preventative Goal: {RechargeBatteryGoal}
```

A second example of a maintenance goal ensures the speed of the rover does not exceed a maximum operating speed.

```
Maintenance Condition: Speed ≤ 100km/hr
Failure Condition: {}
Recovery Plans: {ReduceSpeedPlan} (Slow down vehicle)
Preventative Goal: {}
```

A third example of a maintenance goal originates from the domain of robot soccer. An agent controls the behaviour of a 'defender' robot player, whose role is currently to prevent an opposition 'forward' from obtaining the ball.

```
Maintenance Condition: OpponentHasBall == false
Failure Condition: {}
Recovery Plans: {TackleOpponentPlan}
Preventative Goal: {BlockOpponentGoal}
```

In such a case, the maintenance goal behaviour of the defender differs depending on who has control of the ball. If its opponent has the ball, the agent must 'recover' its maintenance goal of preventing this, and so must tackle the opponent to cause it to lose control of the ball. However, if the opponent does not have control of the ball, one approach is to block the opponent, by moving to a location between the opponent and the ball, ready to intercept a pass or prevent the attacker from moving to the ball. In this example, the behaviour of the agent is different depending on if the agent is recovering from or preventing maintenance goal failure.

In addition to the above attributes, at run-time plan and goal instances will further require an *instance name* which is a label that provides a unique handle to them.

4. REASONING ALGORITHMS

In the BDI theory [11], goals of an agent are required to be consistent. Thangarajah et al.[12, 13, 14] provide mechanisms for ensuring this property holds for achievement goals. We build on this work to incorporate maintenance goals into the set of goals an agent may pursue, and provide mechanisms that ensure that all goals (maintenance and achievement) that an agent pursues are consistent with each other.

We consider two aspects in which an agent’s maintenance goal may conflict; conflicts due to resource limitations, and conflicts due to the effects that goals produce. Our aim is to ensure that maintenance goals remain consistent with other goals with respect to these aspects.

In BDI-like systems, a goal can be decomposed into plans (that achieve the goal) and sub-goals. Often, there are several ways in which a goal can be realised, represented by a number of different plans and sub-goals. When a plan fails, a new plan is selected and tried. It is only when there are no applicable plans left that the agent declares a goal has failed. This leads to a natural tree structure which is termed a *goal-plan* tree [14], which indicates the possible ways in which the top level goal can be established. Every goal or sub goal in the tree has at least one plan as a child, and each plan may have a number of sub-goals. The plan nodes of a goal can be viewed as an “OR” since only one of the plans need to succeed for the goal to succeed, and the sub goal nodes can be viewed as “AND” nodes as they all must succeed for the plan to succeed.

Since there are a number of paths that an agent may take to satisfy a goal, the agent cannot be certain as to the resources consumed and the effects that are produced as a result of a goal. In [12, 13, 14], the notion of summary information is used to capture at an abstract level the resource requirements and the effects of executing a goal. They do this by requiring that each plan and goal in a goal-plan tree have information pertaining to the effects that the plan brings about, as well as the resources that each plan requires¹.

They use the goal-plan tree structure and the notion of summary information [3] to provide detailed algorithms that derive the *necessary* (but not sufficient) and *possible* resources for a goal [14], as well as the *definite* and *potential* effects of a goal [13].

Necessary resources are the resources that are common to all possible paths in achieving the top level goal. *Possible resources* are the maximum set of resources that may be required for an agent to achieve its goal, given any path through the goal-plan tree. The possible resource takes plan failure into consideration, and is often an exaggeration, but provides a clear upper bound. The necessary resources are a subset of the possible resources, which indicates that all necessary resources are possible resources as well. We refer to the combination of necessary and possible resources as *resource summaries*.

The *definite effects* of a goal are the effects that are certain to come about no matter which path through the goal-plan tree the agent follows in achieving a goal. Note that this

process ignores the possibility of a goal becoming satisfied by some external process. The *potential effects* of a goal are a set of all the effects that could potentially occur, given that any plan could be selected, and that plan may fail and another be selected and so on. We refer to the combination of definite and potential effects as *effect summaries*.

The summary information is initially derived at compile time and then later dynamically updated as the agent completes goals and plans (or if they fail). With respect to resources, they also distinguish between *consumable* and *re-usable* resources which are resources that once used are no longer available and those that once used can be re-used again respectively. Their algorithms consider these resource types as the behaviour varies between them.

In our work, we will use these notions of *resource summaries* and *effect summaries*, to determine if a maintenance goal is in conflict with an achievement goal. We rely on the methods developed in [13, 14] to obtain this summary information, and also on the conflict detection mechanisms between achievement goals. We do not reproduce these algorithms here, but show how they are incorporated into our own algorithms ahead.

If we consider an agent that supports proactive maintenance goals, several factors are required beyond that found in most agent systems. As described in Section 3, we require that each plan be tagged with additional information describing the effects the plans bring about, as well as the resources they use. The success condition of achievement goals defines the effects of the goal.

We assume that each agent has set of currently active goals \mathcal{G} which includes both the set of maintenance goals (\mathcal{G}_m) and the set of achievement goals (\mathcal{G}_a) (i.e. $\mathcal{G} = \mathcal{G}_m \cup \mathcal{G}_a$). The list of goals which are waiting for another goal/plan to complete before it can be pursued, are stored in a *Suspended Goals (SG)* set, where each entry is the suspended goal and the list of goals/plans that it is waiting on. Finally, the agent requires a resource table to determine the current resources available for its goals. We call this resource table \mathcal{R} , containing tuples which are name, value pairs describing the current availability of its resources.

Goals are added to the agent system one at a time². Of course once the goals have been added, when possible, the agent will execute goals in parallel. Section 4.1 looks at the mechanisms for ensuring that the goals are consistent when a new achievement goal is added to the system and section 4.2 investigates the addition of a new maintenance goal.

4.1 Adding Achievement Goals

Before an agent can pursue a new achievement goal, G_a , it must determine if it can be executed without causing any of the agent’s other goals to fail. Four elements must be checked to ensure that this is the case – an agent must determine if a conflict can occur over (a) the effects the goal can cause, and (b) the resources that a goal requires. In each of these cases, the agent must check if conflicts arise between the new goal and existing achievement goals as well as existing maintenance goals.

The agent’s first step is to determine the effects of G_a . This can be done using the algorithms developed in [13]. This algorithm results in the identification of two sets of effects – the *definite* effects (*DE*) of a goal, and the *potential*

¹Note that the resources and effects that are the result of child-nodes are not included at the parent level.

²A goal set maybe given to an agent but it processes them one after the other.

effects(PE) of a goal. Using the algorithms in [13] this information can be used to determine if a conflict can arise between G_a and any achievement goals in \mathcal{G}_a . We then provide an algorithm to check for conflict between G_a and the set of maintenance goals \mathcal{G}_m .

G_a must be checked for consistency with every maintenance goal in \mathcal{G}_m . This requires the identification of the definite and potential effects of G_a . The effects of G_a are combined with the effects of \mathcal{G}_a , producing the set of definite and potential effects for the agent's current goals, and the new achievement goal. This set is represented as $\langle DE_{A'}, PE_{A'} \rangle$. A maintenance goal is consistent if the definite and potential effects that the goal brings about, does not cause the maintenance condition of the maintenance goal to fail. A maintenance goal is inconsistent with the achievement goal if the definite effects alone cause the maintenance condition to fail. In situations where the definite effects do not cause the maintenance condition to fail, but some potential effects do cause failure, the agent is uncertain if the addition of G_a will cause the maintenance goal to become inconsistent.

```

function check-effect-consistency ( $\mathcal{G}_m, \langle DE_{A'}, PE_{A'} \rangle$ )
  number of maintenance goals := 0
  number of maintenance goals satisfied := 0
  number of maintenance goals unsatisfied := 0
  for each maintenance goal  $mg$  in  $\mathcal{G}_m$ 
     $mc := mg.maintenance\ condition$ 
    if  $mc$  is consistent with  $DE_{A'} + PE_{A'}$  then
      number of maintenance goals satisfied ++
    if  $mc$  is  $\neg$  consistent with  $DE_{A'}$  then
      number of maintenance goals unsatisfied ++
      number of maintenance goals ++
  if number-of-maintenance-goals-satisfied ==
    number-of-maintenance-goals then
    // all maintenance goals are definitely consistent
    return consistent
  else if number of maintenance goals unsatisfied > 0 then
    // some or all maintenance goals are not consistent
    return inconsistent
  else
    return uncertain

```

Goals can also conflict over the resources they require to execute successfully. The work in [12] provides a mechanism for determining the *necessary* (N) and *possible* (P) resource requirements of a goal. The necessary and possible resources of G_a (which is a tuple, $\langle N_a, P_a \rangle$), are combined with the necessary and possible resources of all achievement goals in \mathcal{G}_a ($\langle N, P \rangle$), producing $\langle N_{a'}, P_{a'} \rangle$ which describes the minimum and maximum resource needs if G_a is added to the goal set.

The agent must now determine if the maintenance goals in \mathcal{G}_m are consistent with $\langle N_{a'}, P_{a'} \rangle$ and the current set of available resources, \mathcal{R} . The achievement of all the goals in \mathcal{G}_a will lead the reduction of \mathcal{R} by *at least* $N_{a'}$, but *no more* than $P_{a'}$. These represent the upper and lower bounds to the resources consumed by the set of goals. It is rational for an agent to ensure that its maintenance goals will still hold after the achievement of its goals, i.e. after \mathcal{R} has been reduced by at least $N_{a'}$, but no more than $P_{a'}$, the maintenance conditions in \mathcal{G}_m still hold.

If a maintenance condition holds after \mathcal{R} is reduced by $P_{a'}$, this indicates that the maintenance condition will still hold no matter which method of achieving the goals is selected by the agent. If the maintenance condition does not hold after \mathcal{R} is reduced by $N_{a'}$, this indicates that the maintenance condition cannot be satisfied, no matter which method of achieving the goals are selected, i.e. the maintenance goal

is inconsistent with \mathcal{G}_a . A third alternative occurs when neither of the first two cases are met – this identifies a case where \mathcal{R} is insufficient to ensure that \mathcal{G}_m remains consistent after achieving all goals in \mathcal{G}_a , yet there are sufficient resources such that the goals may be achieved with the maintenance condition valid. All maintenance goals are required to be checked for consistency in this manner, and the results are aggregated in the following function. A set of maintenance goals that are inconsistent are also produced as a result of this function.

```

function check-resource-consistency ( $\mathcal{G}_m, \langle N_{A'}, P_{A'} \rangle, R$ )
  number of maintenance goals := 0
  number of maintenance goals satisfied := 0
  number of maintenance goals unsatisfied := 0
   $CFG := \langle \rangle$ 
  for each maintenance goal  $mg$  in  $\mathcal{G}_m$ 
     $mc := mg.maintenance\ condition$ 
    if  $mc$  is consistent with  $R - P_{A'}$ 
      number of maintenance goals satisfied ++
    if  $mc$  is  $\neg$  consistent with  $R - N_{A'}$ 
      number of maintenance goals unsatisfied ++
       $CFG := \langle CFG \cup mg \rangle$ 
      number of maintenance goals ++
  if number-of-maintenance-goals-satisfied ==
    number-of-maintenance-goals then
    // all maintenance goals are definitely consistent
    return consistent
  else if number of maintenance goals unsatisfied > 0 then
    // some or all maintenance goals are not consistent
    return  $\langle in-consistent, CFG \rangle$ 
  else
    return unknown

```

There are three possible outcomes of checking for consistency using the methods described above:

Case 1 *Consistent with all maintenance goals*

In the case that if after adding G_a

- the maximum possible resource requirements ($P_{a'}$) are consistent with the current maintenance goals; and
- the potential effects of all goals ($PE_{a'}$) are also consistent with the current maintenance goals,

then it is *safe* to add G_a as a goal to the system.

Case 2 *Not consistent with some or all maintenance goals*

In the case where after adding G_a if either

- the necessary resource requirements ($N_{a'}$); or
- the definite effects ($DE_{a'}$)

of the new goal set is not consistent with one or more maintenance goals then there will be a definite conflict if G_a is added to the system. To prevent this conflict, we can use the *preventative goal* associated with each maintenance goal as follows:

- For each maintenance goal G_m that is in conflict with G_a add the preventative goal of G_m to the system.
- Add the goal G_a to the suspended goals list with entry $(G_a, \{PG_1, \dots, PG_i\})$ where PG_1, \dots, PG_i are the preventative goals added in the previous step.

This method would then allow G_a to begin execution only after all the necessary preventative measures are taken.

In the case where there is no preventative goal prescribed for a conflicting maintenance goal then G_a has to wait until that maintenance goal is no longer in the system. We assume that maintenance goals take priority over achievement goals. Naturally, we could enforce any reasonable scheme for resolving such conflicts..

Case 3 *It is uncertain if there will be a conflict*

The *uncertain* case occurs when either:

- (a) the resource requirements to be maintained are greater than the necessary resource requirements and less than the possible resource requirements; or
- (b) the definite effects of the goal do not violate any maintenance condition but the potential effects do.

In this case we cannot determine for certain if there will be a conflict or not as it depends on the path chosen by the agent to achieve its goals.

In such situations it would appear rational for the agent to pursue G_a without performing additional actions, as there is a possibility that the appropriate path is chosen such that no conflicts occur. However, it may also be rational for an agent to execute some preventative goals that may make (more) certain that the agent's goals will be achieved successfully.

In [13], a process of *dynamic updates* was used to update the resource summaries of goal as the agent committed to certain plans (and sub-goals) at run-time. A more sensible approach for an agent would be to execute G_a when it is uncertain but monitor for situations, by dynamically updating the summary information and checking for inconsistencies, when the status changes to either definitely conflicting (case 2) or safe (case 1) and act accordingly.

There may be times, however, when the agent does not (or cannot) anticipate the future violation of its maintenance goals due to its pursuit of its achievement goals, and in these cases, the maintenance goals could be unsatisfied. The reactive nature of maintenance goals are still supported by our system. Hence, in such cases where the maintenance goal of the agent is spontaneously violated, the agent will trigger its recovery goal, which has the goal of re-establishing the maintenance condition of the goal.

We make a note here that preventative goals will most likely not be used in the case when an achievement goal conflicts with maintenance goals due to the effects that it brings about, rather than its resources. While our framework supports such a mechanism, we could not identify any cases where a preventative goal would be of use in avoiding conflicts over effects.

4.2 Adding Maintenance Goals

In the previous section, we have shown how achievement goals can be added whilst maintaining consistency between existing goals. We now look at the process of adding a maintenance goal to an agent system such that it is consistent with all other goals that the agent is pursuing. Maintenance goals present several challenges that separate them from the addition of achievement goals, as they do not achieve any effects per se but require conditions to be maintained.

The first step in adding a new maintenance goal G_m is to determine if it is consistent with the set of existing maintenance goals \mathcal{G}_m . This is done by simply checking the maintenance condition of each maintenance goal in \mathcal{G}_m against the maintenance condition of G_m . If G_m conflicts with any other maintenance goals, then it is not added to the system. If the application demands it, G_m may be placed in the suspended goals list, waiting on the conflicting goals.

If G_m is consistent with \mathcal{G}_m then the next step is to check if it is consistent with the current set of achievement goals

\mathcal{G}_a . This is done by combining G_m with \mathcal{G}_m and check that no achievements goal will violate the new set of maintenance goals, using the methods described in the previous section.

If there is no conflict then G_m is added to the system. In cases where adding G_m causes conflict, we prevent G_m from being added to the system, as it is difficult to determine the correct subset of achievement goals that are compatible with the set of maintenance goals. There may be a subset of achievement goals from the agent's current goal set that is consistent with the agent's maintenance goals (and current resources), but identifying such a subset can be difficult. It may be feasible, however, if there is a goal preferencing mechanism in place.

In these algorithms, we have assumed that the goal that is being added to the system is preferred less than the existing goals – that is, the agent does not drop goals that it is currently pursuing if a new conflicting goal is added. Further work is required in this area, and in the area of selecting between goals. itemize

5. EXPERIMENTAL EVALUATION

We have performed some preliminary experiments to quantify the benefits of proactive maintenance goals. X-JACK is an extension to JACK Intelligent Agents³ developed in [14, 13] for evaluating their work. We further extend X-JACK to incorporate reactive and proactive maintenance goals and the algorithms developed in this work.

Our experimental test bed is that of a Mars rover capable of moving about a 2 dimensional environment, consuming energy from its battery as it moves. For simplicity, we assume that each unit of distance drains the battery by 1 unit and that there is a refuelling depot located in the middle of this environment.

A *goal* within the system is a particular location that the agent must visit. To test our system we created a number of goals with randomly generated locations. The agent was then given goals varying from a set of 10 to 100 for simulating an agent's typical workload. Each set was run a 20 times and the results mentioned below are the average of these runs.

Our experiments were conducted in 3 cases as follows:

- No maintenance goal (NMG) - In the first case, we assume that the rover's movements did not consume battery power, therefore, the rover had no need to recharge. This allows us to determine a baseline in our experiments.
- Reactive maintenance goal (RMG) - The second case involves a rover with a maintenance goal that is *reactive* to ensure that there is always enough charge in the battery to be able to reach the recharging station. Therefore, as the rover moves about, it will periodically compare its current distance from the recharge station with the amount of charge remaining in its battery. If there is not enough charge to continue, the rover must first head to the recharge station and recharge, prior to pursuing any other goal.
- Proactive maintenance goal (PMG) - The third case is when the rover has the same maintenance goal as case 2, but it is *proactive* in this case. This requires the

³www.agent-software.com.au

Goals	NMG	RMG	PMG
10	539.6667	595	542.3333
20	941.3333	1103.3333	951.3333
30	1098	1287.6667	1150.6667
40	1376.6667	1566.3333	1443.3333
50	1450.3333	1762	1514.3333
60	1713	2056.6667	1886
70	1793.3333	2178.3333	1960.3333
80	2553	3110.3333	2726
90	2606	2849.3333	2745
100	2551.3333	3012	2575

Table 1: Total time (milliseconds) to achieve all goals

Goals	NMG	RMG	PMG
10	308	439	341
20	704	946	797
30	814	1196	1034
40	1284	1851	1587
50	1460	2067	1812
60	1689	2512	2112
70	1881	2619	2350
80	2392	3410	3086
90	2973	4330	3717
100	3124	4660	3925

Table 2: Total distance (units) travelled

agent, when commencing a new goal, to anticipate its resource usage in achieving that goal. If the resources consumed in achieving this goal violate the agent’s proactive maintenance goal, a preventative goal will be activated. The agent determines how much battery power will be left after achieving this goal, and if it is an insufficient amount, the recharge goal is executed first. In our experiment, to simulate the possible inaccuracies with estimating resource usage, the rover is able to estimate its resource usage to an accuracy of 10%.

The total time taken for the set of goals to complete in all 3 cases are shown in table 1. As expected, the case where the rover never needs to refuel (NMG) is the fastest. Note that there is little difference here between the reactive and proactive maintenance goals. In some instances the proactive maintenance is actually faster, due to less distance being travelled. As shown in Table 2, the total distance travelled by the rover in each of the cases is less for proactive maintenance than for reactive maintenance.

Tables 3 and 4 provides a breakdown of the rover’s movements for the reactive and proactive cases respectively. They show the distance the rover spent successfully pursuing a goal (*total distance*), the distance the rover spent to refuel (*backtrack*), and the distance the rover spent trying to accomplish a goal that failed due to insufficient fuel (*waste*). The *excess* column is the sum of the waste and backtrack distance. Where this value is small indicates that the rover did not spend many resources on movement that was not directly attributable to achieving one of its goals. Therefore, smaller values are better. The comparison of the excess col-

Goals	Total Distance	Waste	Backtrack	Excess
10	439	120	69	189
20	946	282	151	433
30	1196	237	218	455
40	1851	593	315	908
50	2067	521	352	873
60	2512	650	437	1087
70	2619	558	442	1000
80	3410	831	574	1405
90	4330	1151	781	1932
100	4660	1106	887	1993

Table 3: Distance breakdown for Reactive Maintenance Goals

Goals	Total Distance	Waste	Backtrack	Excess
10	341	0	91	91
20	797	0	259	259
30	1034	0	280	280
40	1587	43	524	567
50	1812	92	510	602
60	2112	127	546	673
70	2350	48	685	733
80	3086	158	887	1045
90	3717	93	1181	1274
100	3925	140	1075	1215

Table 4: Distance breakdown for Proactive Maintenance Goals

umn shows that the excess amount for the proactive case is always less than that for the reactive case. This is because the proactive maintenance goal avoids consuming resources for goals that it can anticipate will fail. The reactive maintenance goal can only determine that the goal will fail once it has already consumed some resources in trying to achieve it.

If we compare the waste and backtrack values for the reactive and proactive cases, it is observed that often the reactive case has less backtrack than the proactive. This is because in the reactive case, some movement that becomes waste actually brings the rover closer to the refuelling point – hence, reducing the backtrack distance.

We also note that the value of waste for the proactive case is often much less than that for the reactive, because the objective of the proactive maintenance goal is to eliminate waste, by anticipating the failure of its maintenance condition at an earlier point than the reactive goal. For the first goal sets (10,20,30) for the proactive case, this works perfectly. However, some waste still occurs as the number of goals increases. This is because the estimation of resource usage was not 100% accurate. Our results show that even with inaccurate estimation, proactive maintenance goals can offer a reduction in resource consumed.

When resource estimation is perfectly accurate, there would be no waste at all for proactive maintenance, as it will never attempt to achieve a goal unless it is certain that it will be successful. In other words, the only way that waste gets introduced in the proactive case is by the inaccurate estimate of resources.

6. DISCUSSION AND FUTURE WORK

We have seen how the use of proactive maintenance goals can improve the rational behaviour of BDI agent systems. In particular, the ability to anticipate the failure of maintenance goals (rather than merely reacting to their failure) can result in some significant improvements in the decision-making ability of the agent. We see this as the first step towards providing a richer variety of goal types for agent systems, resulting in greater rationality for the agents concerned.

By utilizing existing techniques for reasoning about achievement goals, we are able to smoothly integrate proactive versions of maintenance goals, together with reactive maintenance goals and achievement goals. This provides a richer programming environment which can be readily developed as a plug-in for existing systems such as Jadex.

A line of further work that we intend to pursue is that of passive maintenance goals. A reactive maintenance goal leads to action only when the goal is violated. As we have seen, proactive maintenance goals may also lead to action in anticipation of violation. A passive maintenance goal is one that does not lead to actions per se, but is one that is never allowed to be violated. Such goals act purely as constraints on other actions; for example, if the Mars rover is currently travelling at its maximum speed and there is a passive maintenance goal to this effect, then there is no point in considering any action which will increase the rover's speed.

An extension of this idea is to prioritise maintenance goals, possibly via their urgency. For example, a maintenance goal of keeping the temperature of a gas tank below 100° will become increasingly urgent as the temperature climbs above 90°. This kind of rational behaviour is particularly important for safety-critical systems.

The relationship between achievement goals and maintenance goals also raises some questions, such as under what circumstances an achievement goal can trigger a maintenance goal. For example, the Mars rover may have to arrive at a particular location with at least 50% battery power (to perform a particular experiment), and doing so will require it to travel at a lower maximum speed than normal.

A further direction is to explore the ways in which violation of maintenance goals can be predicted. When the goals are about resources, it is inevitable that they will be exhausted; the key issue is to determine the optimal point of replenishment and to see how this can be specified by proactive maintenance goals. Clearly the system's ability to utilize maintenance goals is a direct function of its predictive power.

Acknowledgments

The authors of this paper would like to thank Michael Winikoff for discussions related to this material. The support of the Australian Research Council and Agent Oriented Software through grant LP0453486 is gratefully acknowledged.

7. REFERENCES

- [1] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In *Proceedings of the AAMAS Workshop on Programming in Multi-Agent Systems (PROMAS'04)*, pages 44–65, 2004.
- [2] P. Busetta, R. Ronnquist, A. Hodgson, and A. Lucas. Jack Intelligent Agents - Components for Intelligent Agents in Java, 1999.
- [3] B. J. Clement and E. H. Durfee. Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information. In *AAAI/IAAI*, pages 495–502, 1999.
- [4] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Engineering. *Science of Computer Programming*, 20(1-2):3–50, April 1993.
- [5] M. Dastani, F. de Boer, F. Dignum, and J. Meyer. Programming Agent Deliberation: An Approach Illustrated Using the 3APL Language. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, pages 97–104, Melbourne, July 2003. ACM Press.
- [6] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *AAAI*, pages 677–682, 1987.
- [7] F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos Software Development Methodology: Processes, Models and Diagrams. In *Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, pages 35–42. ACM Press, July 2003.
- [8] M. J. Huber. JAM: A BDI-theoretic mobile agent architecture. In *Proceedings of The Third International Conference on Autonomous Agents*, pages 236–243, Seattle, WA, 1999.
- [9] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: Implementing a BDI-Infrastructure for JADE Agents. *EXP – in search of innovation*, 3(3):76–85, 2003.
- [10] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In R. van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
- [11] A. S. Rao and M. P. Georgeff. BDI-agents: From Theory to Practice. In *Proceedings of the First International Conference on Multiagent Systems (ICMAS'95)*, San Francisco, 1995.
- [12] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Avoiding Interference Between Goals in Intelligent Agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.
- [13] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Exploiting Positive Goal Interaction in Intelligent Agents. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, Melbourne, July 2003.
- [14] J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding Resource Conflicts in Intelligent Agents. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'02)*, 2002.
- [15] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative & Procedural Goals in Intelligent Agent Systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, Toulouse, April 2002.