Avoiding Resource Conflicts in Intelligent Agents

John Thangarajah and **Michael Winikoff** and **Lin Padgham** and **Klaus Fischer**

Abstract. An intelligent agent should be *rational*, in particular it should at least avoid pursuing goals which are definitely conflicting. In this paper we focus on *resource conflict* in agents that use a plan library organised around goals. We characterise different types of resources and define resource requirements summaries. We give algorithms for deriving resource requirements, using resource requirements to detect conflict, and maintaining dynamic updates of resource requirements. We also discuss ways of resolving resource conflict. Our approach does not represent time, rather it keeps resource summaries current. This enables an agent's decisions to be made on the basis of up-to-date information and allows us to develop efficient runtime (online) algorithms.

1 Introduction

An intelligent agent should be *rational*. Part of being rational is to avoid actions that interfere with each other. A rational agent should not simultaneously pursue a goal G_1 and a goal G_2 if G_1 prevents the achievement of G_2. Typically an intelligent complex agent will have multiple goals which are active simultaneously. In adopting goals it should be aware of conflicts that make it clearly irrational to pursue certain goals simultaneously [11].

One important type of conflict is *resource* conflict. In traditional planning resource issues are managed as part of a complete planning process. However, in agent systems the plan to be used is chosen at runtime, based on the current context. This approach is important because in general agents operate in dynamic environments where things outside the agent's control are likely to change the environment - and thus the possible ways to achieve a goal that the agent is pursuing.

Different ways of accomplishing a goal may use different resources. Consequently we cannot always say in advance precisely what resources will be needed to accomplish a given goal. However we could represent and reason about *possible* resource needs for achieving a goal. If particular resources are needed in every way that the agent knows how to achieve a given goal, then we can conclude that these are *necessary* resources.[1]

Resources may be consumable, i.e. they are no longer available following use, or reusable, i.e. following usage they are again available. For example energy is a typical consumable resource, whereas a communication channel is a typical reusable resource.

We present in this paper mechanisms that allow agents to be aware of necessary and possible resource needs when adopting a goal, and how these interact with resource needs of goals already being pursued. We define ways in which rational agents should act with respect to goal adoption and resource needs, and also define additional reasoning which intelligent agents may do to more successfully manage potential conflicts over resources. For example consider an autonomous robot with 10 units of energy, and two goals, one of which requires 8 energy units, the other 5. It would not be rational if the robot pursued both of these goals, as it is obvious that one must fail. If this same robot had two goals, each requiring a communications channel, (and only one communications channel was available) an intelligent robot may conclude that it could pursue both, but that there may be scheduling issues around this resource.

The approach we use involves a notion of resource summaries similar to that used by Clement et al [2, 4, 3], though with some differences[2]. The algorithms they present in [2] allow for expansion and scheduling of appropriate plans prior to execution, whereas our approach allows for online assessment and if necessary monitoring and scheduling. Their approach is not suitable for the kind of agent systems which we work with, which operate in dynamic domains where in general it is not possible to decide which plans to use in advance as they depend on (changing) environmental conditions.

2 Plan & Goal Representation in Agents

For the purposes of this paper we assume that agents have a library of programmer-provided plans. Each plan consists of (i) an indication of the goal for which it is relevant, (ii) a context condition which describes the situations in which the plan is applicable, and (iii) a plan body which specifies what the plan does. We assume that plan bodies can contain subgoals and actions. These are combined by either sequencing them (e.g. "achieve goal G_1 and then perform action A" written as "$G_1; A$") or by performing them in parallel (e.g. "achieve goals G_1 and G_2" written as "$G_1 \| G_2$").

Formally, we have a mapping from a goal G to the set of relevant plans $P(G)$ for that goal. Each plan P has a context condition *context*(P) that is a logical formula over the agent's beliefs; and a plan body *body*(P) where *body*$(P) ::= A \mid G \mid P_1 \parallel P_2 \mid P_1; P_2$. A plan which is relevant to goal G, has context C and body P is written as $G : C \leftarrow P$. We shall sometimes write $\pi_i = G : C \leftarrow P$ to indicate that π_i is an abbreviation for the plan in subsequent discussion. An empty context (C) is equivalent to true.

The execution cycle of an agent consists of the steps:

1. Match a goal instance against plans in the plan library obtaining a set of *relevant* plan types.
2. For each relevant plan type, evaluate its context condition giving a plan instance for each context condition which evaluates to true.

[1] Note that having all necessary resources does not mean that sufficient resources are available to achieve the goal. For example, if plan A for achieving G requires resources X and Y, and plan B requires resources Y and Z, then we can conclude that Y is a necessary resource. However sufficient resources for successful achievement of G would be (Y and (X or Z)).

[2] Clement et al. attach resource summaries to actions and propagate upwards to the level of abstract plans, while we attach resources to goals and plans, with no distinction between abstract and concrete plans.

3. Remove any plan instances which are equivalent to previously failed plan instances for this goal instance. The remaining set of plans are the *applicable plans*.
4. Select an applicable plan and execute it.

If a plan fails, the goal instance remains active and new applicable plans are calculated and tried. If there are no applicable plans left, the goal fails.

The assumptions made correspond to a class of agent systems. One particular type of agent system that fits in with these assumptions is those based on the Belief-Desire-Intention (BDI) model [9]. For example, PRS [7], JAM [6], dMARS [5], and JACK [1].

2.1 Example

Consider a rover robot which is deployed on the Martian surface. The robot obtains energy from solar panels and is given requests to perform various experiments (Exp) on various substances (A,B). The following are some resources arising naturally in this domain:

- **Energy** is a consumable resource (which can be renewed by basking in the sun at suitable hours of the day).
- **Communication channels** for communicating with other rovers and transmitting results back to the control station. They are a reusable resource.

The rover may have the following among its set of plans:

π_1 = $Exp(A\&B)$: $\leftarrow$ *Exp(A) ; Exp(B) ; Merge&StoreData*

π_2 = $Exp(A\&B) : FreeRover(x) \leftarrow$ *(Exp(A) $\|$ Delegate(x, Exp(B))) ; Merge&StoreData*

π_3 = $Exp(A)$: $\leftarrow$ *Collect(A) ; Analyse(A)*

π_4 = $Exp(B)$: $\leftarrow$ *Collect(B) ; Analyse(B)*

π_5 = $Delegate(x, Exp(B))$: $\leftarrow$ *TransmitRequest(x, Exp(B)) ; WaitResponse(x, Exp(B))*

We will use this example to illustrate the kinds of reasoning our agent can do regarding resource conflicts and pursuit of goals.

3 Characterisation of Resources

In order to reason about resource needs and how these affect the pursuit of goals, we must develop a representation of resources which supports the desired reasoning.

We assume a set of resource types $\mathcal{T} = \{t_1, \ldots, t_n\}$. For example $\mathcal{T}_{\text{rover}} = \{\textit{energy}, ComCh\}$.

A *resource requirement* is a pair of a resource type and a number, (t, n), where $n \geq 0$. For example $(\textit{energy}, 200)$. The number represents an amount of the resource.

We define a *resource set* R as a set of resource requirements. In our formal definitions we shall assume that resource sets are *normalised* so that each resource type appears *exactly* once. For example, given the resource types above the resource set $\{(\textit{energy}, 20), (\textit{energy}, 30)\}$ is normalised to $\{(\textit{energy}, 50), (ComCh, 0)\}$. In future we assume the presence of the resource types that have a value of 0 in the normalised resource set and do not explicitly include them for clarity.

We denote the *reusable resources* of resource set R by R^r and the *consumable resources* by R^c ($R = R^c \cup R^r$, $R^c \cap R^r = \emptyset$).

Resource set R_1 is smaller than resource set R_2 (written $R_1 \sqsubseteq R_2$) if any resource type that appears in R_1 also appears in R_2 and if whenever a resource type t appears in R_1, the amount required is less than the amount required in R_2. Formally (recall that we assume normalised resource sets) $R_1 \sqsubseteq R_2$ iff $\forall t. R_1(t) \leq R_2(t)$ where $R(t)$ denotes the amount of resource type t in R. Note that $\sqsubseteq$ is a partial order (reflexive, transitive, antisymmetric). We use $R_1 \sqsubset R_2$ to denote $R_1 \sqsubseteq R_2 \wedge R_1 \neq R_2$.

We use two types of resource sets: *necessary* (N) and *possible* (P). A *resource summary* $\mathcal{S}$ consists of these two sets of resource requirements: Formally $\mathcal{S} = \langle N, P \rangle$. Note that P is always greater than N: any resource which is necessary for successful execution is also possibly required (formally $N \sqsubseteq P$). Therefore N gives a lower bound of the resources required while P gives an upper bound.

3.1 Deriving Resource Requirements

We now describe how we can determine the resource summary $\mathcal{S}$ for each plan and each goal within our system. This resource summary can then be used to determine various levels of potential conflicts, or when it is safe to pursue given goals in parallel.

We derive the resource requirements for a goal by combining the resource requirements of all the relevant plans for that goal. The resource requirements of a plan are calculated by combining the resource requirements of the subgoals and actions within the plan body.

We do not attach resources to each individual action but rather allow the programmer to specify for a plan the resource summary ($P^{\mathcal{R}}$) which captures the necessary resource requirements for the actions in that plan[3]. For example the Martian rover agent's plans might be annotated to indicate that π_4 uses {*(energy,100)*} and π_5 uses {*(energy,10), (ComCh,1)*}.

We now define a number of operators over resource sets and resource summaries to facilitate the computation of resource summaries of goals and plans at all levels. We define $\sqcup$ which computes the upper bound (maximum) of two resource sets. If $R_1 = \{(a, 10), (b, 15)\}$ and $R_2 = \{(a, 5), (b, 17)\}$ then (assume a and b are reusable) $R_1 \sqcup R_2 = \{(a, 10), (b, 17)\}$. Dually, $\sqcap$ computes the lower bound (minimum) and is used when merging necessary conditions: if the relevant plans for G are $\{P_1, P_2\}$ and the necessary resource for P_1 and P_2 are respectively $\{(a, 12)\}$ and $\{(a, 7), (b, 2)\}$ then the *necessary* resources for G are $\{(a, 7)\}$.[4] When determining the resource needs of a plan we need to distinguish between sub-goals within a plan that are achieved in sequence and in parallel. If G_1 and G_2 are done in parallel then the resources can be simply added ($\oplus$) together. If they are done in sequence, addition would suffice for resources that are consumable, but for resources that are reusable we need to merge the resources together using the upper bound ($\sqcup$) since after G_1 completes it would release the resources which can then be reused by G_2. For example, if G_1 requires $\{(ComCh, 1), (\textit{energy}, 100)\}$ and G_2 requires $\{(ComCh, 1), (energy, 50)\}$ then $G_1 \| G_2$ requires $\{(ComCh, 2), (\textit{energy}, 150)\}$ whereas $G_1; G_2$ requires $\{(ComCh, 1), (\textit{energy}, 150)\}$.

$$
\begin{aligned}
R_1 \sqcup R_2 &= \{(t, \max(R_1(t), R_2(t))) \mid t \in \mathcal{T}\} \\
R_1 \sqcap R_2 &= \{(t, \min(R_1(t), R_2(t))) \mid t \in \mathcal{T}\} \\
R_1 \oplus R_2 &= \{(t, R_1(t) + R_2(t)) \mid t \in \mathcal{T}\} \\
R_1 \otimes R_2 &= (R_1^c \oplus R_2^c) \cup (R_1^r \sqcup R_2^r)
\end{aligned}
$$

[3] The reason for this is that it gives a better mapping to implementations, where a plan may consist of arbitrary code plus subgoals, rather than the simpler formalisation given here. It also relieves the programmer from the need to specify resource requirements at the level of each action.

[4] Note that $\{(a, 7)\}$ is *necessary*, however, it is not sufficient.

In computing the (maximum) resource requirements for a goal we must consider the fact that more than one plan may be executed in order to achieve a goal. Consumable resources that have been used in this process cannot be recovered. Consequently the possible resource needs of the goal must combine the resource needs of *all* relevant plans, as if each were to be executed sequentially. However the treatment of necessary resources requires that only resources that are necessary in *all* relevant plans are necessary for the goal. We define the operator $\uplus$ which combines the resources as follows:

$$\langle N_1, P_1\rangle \uplus \langle N_2, P_2\rangle = \langle N_1 \sqcap N_2, P_1 \otimes P_2\rangle$$

For example, if $R_1 = \langle\{(\textit{energy}, 30)\}, \{(ComCh, 1), (\textit{energy}, 50)\}\rangle$ and $R_2 = \langle\{(\textit{energy}, 20)\}, \{(\textit{energy}, 20), (ComCh, 1)\}\ \rangle$ then $R_1 \uplus R_2 = \langle\ \{(\textit{energy}, 20)\}, \{(\textit{energy}, 70), (ComCh, 1)\}\rangle$. In the above case the necessary energy is 20 because that is the minimum energy required to achieve the goal (i.e. by choosing the plan requiring resources R_2). The possible energy required is 70 because if the plan requiring resources R_2 is chosen and fails, followed by execution of the plan requiring resources R_1, then the total possible energy required is 70 units. Since the resource $ComCh$ is a reusable resource, even though it is required by both R_1 and R_2 it is required only once overall.

We can now formally define a function $\mathcal{S}$ which takes a goal or program and computes the resource summary by combining the resource summaries of its components. We define the lifting of the operators $\oplus, \otimes, \sqcap$, and $\sqcup$ to operate on pairs of resources in the obvious way, for example $\langle N_1, P_1\rangle \oplus \langle N_2, P_2\rangle = \langle N_1 \oplus N_2, P_1 \oplus P_2\rangle$. We denote the resource requirements of a goal G and a plan P by $\mathcal{S}(G)$ and $\mathcal{S}(P)$ respectively. Note that they are each a two tuple $\langle N, P\rangle$. We denote the set of relevant plans for a goal G by $P(G)$.

$$\begin{aligned}
\mathcal{S}(P) &= \langle P^{\mathcal{R}}, P^{\mathcal{R}}\rangle \oplus \mathcal{S}(body(P)) \\
\mathcal{S}(A) &= \langle \emptyset, \emptyset\rangle \\
\mathcal{S}(G) &= \biguplus_{p \in P(G)} \mathcal{S}(p) \\
\mathcal{S}(P_1 \| P_2) &= \mathcal{S}(P_1) \oplus \mathcal{S}(P_2) \\
\mathcal{S}(P_1; P_2) &= \mathcal{S}(P_1) \otimes \mathcal{S}(P_2))
\end{aligned}$$

Let us now consider our example. The Mars Rover has a goal $exp(A\&B)$ to perform experiments on substances A and B. Recall that the rover has two plans for this. The first, π_1, performs experiments on A, followed by experiments on B and then stores the results. The second plan, π_2, performs experiments on A while simultaneously delegating the experiments on B to another rover that is free. Let us assume the following resource summaries for π_3, π_4, and π_5, and assume the action *Merge&StoreData* does not require any resources:

$$\begin{aligned}
\mathcal{S}(\pi_3) &= \langle\{(\textit{energy}, 100)\}, \{(\textit{energy}, 100)\}\rangle \\
\mathcal{S}(\pi_4) &= \langle\{(\textit{energy}, 100)\}, \{(\textit{energy}, 100)\}\rangle \\
\mathcal{S}(\pi_5) &= \langle\{(\textit{energy}, 10), (ComCh, 1)\}, \\
&\quad \{(\textit{energy}, 10), (ComCh, 1)\}\rangle
\end{aligned}$$

we then compute the resource summaries of π_1 and π_2 as follows:

$$\begin{aligned}
\mathcal{S}(\pi_1) &= \mathcal{S}(\pi_3) \otimes \mathcal{S}(\pi_4) = \langle\{(\textit{energy}, 200)\}, \{(\textit{energy}, 200)\}\rangle \\
\mathcal{S}(\pi_2) &= \mathcal{S}(\pi_3) \oplus \mathcal{S}(\pi_5) = \langle\{(\textit{energy}, 110), (ComCh, 1)\}, \\
&\quad \{(\textit{energy}, 110), (ComCh, 1)\}\rangle
\end{aligned}$$

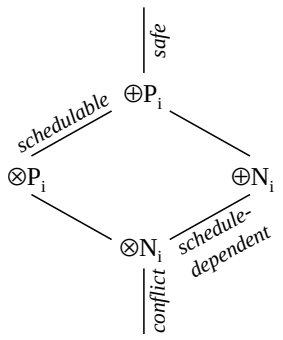
We compute the resource summary for $exp(A\&B)$ as follows:

$$\begin{aligned}
\mathcal{S}(exp(A\&B)) &= \mathcal{S}(\pi_1) \uplus \mathcal{S}(\pi_2) \\
&= \langle\{(\textit{energy}, 110)\}, \\
&\quad \{(\textit{energy}, 310), (ComCh, 1)\}\rangle
\end{aligned}$$

4 Resource Conflicts

We can use the resource summary information to detect if a set of goals can be executed concurrently with no resource conflicts. Given a set of goals $\{G_1, \ldots, G_n\}$ with associated resource summaries $\langle N_i, P_i\rangle$ we define the *sequential upper bound* $\bigotimes P_i$, the *parallel upper bound* $\bigoplus P_i$, the *sequential lower bound* $\bigotimes N_i$, and the *parallel lower bound* $\bigoplus N_i$. The sequential cases correspond to the resource requirements with maximal reuse (of reusable resources) whereas the parallel cases assume no reuse. These quantities are related as follows[5] (see also the diagram below) $\bigotimes N_i \sqsubseteq \bigoplus N_i \sqsubseteq \bigoplus P_i$ and $\bigotimes N_i \sqsubseteq \bigotimes P_i \sqsubseteq \bigoplus P_i$, however $\bigotimes P_i$ and $\bigoplus N_i$ are not, in general, comparable. Thus there are three primary cases: the available resources, $\mathcal{R}$, can be either less than the sequential lower bound, greater than or equal to the parallel upper bound, or in between the two. The first case, $\mathcal{R} \sqsubset \bigotimes N_i$, indicates that it is impossible to achieve all of the goals given the available resources. The second case, $\bigoplus P_i \sqsubseteq \mathcal{R}$, indicates that even if no resource reuse takes place, there are sufficient resources. That is, the goals can be simultaneously executed in any way.

The third case ($\bigotimes N_i \sqsubseteq \mathcal{R} \sqsubset \bigoplus P_i$) indicates uncertainty. However, there are two interesting sub-cases. The first is where $\bigotimes P_i \sqsubseteq \mathcal{R} \sqsubset \bigoplus P_i$. In this case there are enough resources to achieve the goals, but resource reuse is required. That is, the goals are *schedulable*: by scheduling appropriately we can *guarantee* that there will be enough resources. The second interesting sub-case is where $\bigotimes N_i \sqsubseteq \mathcal{R} \sqsubset \bigoplus N_i$. In this case we know that if we do not reuse resources appropriately, then there will not be enough resources. We term this case *schedule-dependent*.



These cases define how a set of goals[6] can be compared to available resources to determine whether there are enough resources and whether scheduling is necessary and/or possible. Note that it is possible for a set of goals to be both schedulable and schedule-dependent if $\bigotimes P_i \sqsubset \bigoplus N_i$, in this case we know that reuse is both necessary and possible, i.e. that we succeed if and only if resources are suitably reused. We denote the status[7] (according to the cases above) of a set of goals G with respect to the available resources $\mathcal{R}$ as *status*$(G, \mathcal{R})$.

Let us return to our example. Suppose that the agent has two goals G_1 and G_2 with the following resource requirements: $\mathcal{S}(G_1) = \langle\{(\textit{energy}, 110), (ComCh, 1)\}, \{(\textit{energy}, 310), (ComCh, 1)\}\rangle$ and $\mathcal{S}(G_2) = \langle\{(\textit{energy}, 50), (ComCh, 1)\}, \{(\textit{energy}, 90), (ComCh, 1)\}\rangle$. Then *status*$(\{G_1, G_2\}, \mathcal{R})$ is:

[5] Observe that $A \otimes B \sqsubseteq A \oplus B$ for any A and B, that $\otimes$ and $\oplus$ are both monotonic, and that $N \sqsubseteq P$.

[6] A special case is where the "set" of goals consists of a single goal, in this case "conflicting" means that there are not enough available resources for the goal to succeed.

[7] We use "schedulable&schedule-dependent" to denote the case where both sub-cases apply.

- *Safe*[8] if $\bigoplus P_i = \{(\textit{energy}, 400), (ComCh, 2)\} \sqsubseteq \mathcal{R}$
- *Schedulable* if $\bigotimes P_i = \{(\textit{energy}, 400), (ComCh, 1)\} \sqsubseteq \mathcal{R} \sqsubset \bigoplus P_i$
- *Schedule-dependent* if $\bigotimes N_i = \{(energy, 160), (ComCh, 1)\} \sqsubseteq \mathcal{R} \sqsubset \bigoplus N_i = \{(energy, 160), (ComCh, 2)\}$
- *Conflicting* if $\mathcal{R} \sqsubset \{(\textit{energy}, 160), (ComCh, 1)\} = \bigotimes N_i$
- *Uncertain* otherwise.

However, instead of assessing the safety of a set of goals, what we wish to do most often is to determine the risks associated with adding a new goal or goal set G to an existing goal set E. However, we cannot simply apply the definitions above. The problem is that if *status*$(E, \mathcal{R})$ is not safe then *status*$(E \cup G, \mathcal{R})$ will not be safe even if G uses no resources and so we cannot determine whether or not G requires scheduling. The solution is to determine the risk of adopting G by only considering the resource types which are used by G. Formally, we define a notion of restriction – given resource sets R_1 and R_2 the restriction of R_1 by R_2 (written $R_1 \downarrow R_2$) is those resources in R_1 that are also used in R_2:

$$R_1 \downarrow R_2 = \{(t, \text{if } R_2(t) > 0 \text{ then } R_1(t) \text{ else } 0) \mid t \in \mathcal{T}\}$$

We then use the definitions given above but restrict resource sets before comparing them. In comparisons involving P_i we restrict by the possible set of the goals G. In comparisons involving N_i we restrict by the necessary set of the goals. So, for example, if we have existing goals $E = \{G_1, G_2\}$ as above and the new goal being considered is G_{new} where $\mathcal{S}(G_{new}) = \langle N_{G_{new}}, P_{G_{new}} \rangle$, then the test for safety is whether $(P_{G_{new}} \oplus \bigoplus P_E) \downarrow P_{G_{new}} \sqsubseteq \mathcal{R}$ and the check for conflict is $\mathcal{R} \sqsubset (N_{G_{new}} \otimes \bigotimes N_E) \downarrow N_{G_{new}}$.

4.1 Algorithm

The algorithm below computes the status of adding a new set of goals G to an existing set of goals E where the resources $\mathcal{R}$ are available. The algorithm considers each resource type separately. After all resource types are computed the results are joined yielding a status for the goal set G. We define *status*$_t(G, \mathcal{R})$ to be the same as *status*$(G, \mathcal{R})$ except that all resource types other than t are ignored.

function *status*$(G,E,\mathcal{R})$
$\langle N_G, P_G \rangle := \mathcal{S}(G)$
for each resource type $t \in \mathcal{T}$
  if $N_G(t) = 0 \wedge P_G(t) = 0$ **then** statusG[t] := safe
  else if $N_G(t) = 0$ **then**
    Ignore comparisons with necessary resources.
    if *status*$_t(E \cup G, \mathcal{R})$=conflict **then** statusG[t]:=uncertain
    else if *status*$_t(E \cup G, \mathcal{R})$=schedulable&schedule-dependent **then** statusG[t]:=schedulable
    else if *status*$_t(E \cup G, \mathcal{R})$=schedule-dependent **then** statusG[t]:=uncertain
    else statusG[t] := *status*$_t(E \cup G, \mathcal{R})$
  else statusG[t] := *status*$_t(E \cup G, \mathcal{R})$
endfor
if $\exists t$: statusG[t]=conflict **then return** conflict
if $\forall t$: statusG[t]=safe **then return** safe
if $\forall t$: statusG[t] $\in$ {safe,schedulable&schedule-dependent, schedulable} **then**
  if $\exists t$: statusG[t]=schedulable&schedule-dependent **then return** schedulable&schedule-dependent
  else return schedulable
if $\exists t$: statusG[t] = schedule-dependent **then return** schedule-dependent
else return uncertain

The algorithm presented allows the agent to determine how safe it is to adopt a goal with respect to its existing set of active goals. This information allows the agent to be rational in its adoption of goal by not adopting conflicting goal sets[9].

4.2 Dealing with conflicts

If the agent determines that a new goal set G is conflicting with regard to its currently active goals E (and adoption will therefore inevitably cause the failure of some goal), then it is clearly not rational to simply adopt the new goal set. The only rational behaviour is to not adopt G, or to drop some existing goal(s) giving a revised set of existing goals $E^{'}$ such that $\bigotimes N_i \sqsubseteq \mathcal{R}$ for the set $G \cup E^{'}$.

In the case that G is neither conflicting nor safe with regard to E the agent has a number of choices. It may be cautious and always defer or reject a non-safe new goal set; or it may be optimistic and always go ahead as long as the goals aren't definitely conflicting. Alternatively, it may do further reasoning about the remaining resource needs of partially completed goals, and/or choose to monitor the execution of the non-safe goals. These choices may depend on whether the new goals were determined to be schedulable and/or schedule-dependent, or uncertain with respect to the existing goals.

In pursuing a goal E_i, once a sub-goal of E_i has completed, all possible and necessary resources associated with that sub-goal are no longer relevant. Reasoning about the remaining resource requirements of partially achieved goals can enable an agent to make more up-to-date choices regarding its adoption of goals.

The following section describes a mechanism for maintaining information that efficiently calculates the remaining resource requirements for a partially completed goal.

5 Dynamic Updates of Resource Requirements

Each of an agent's goals is represented by a tree consisting of nodes representing goals and plans, with resource summaries attached to each node. The initial goal-plan tree for each goal type can be created at compile time (where resource amounts are unknown until runtime a range of 0 to ∞ is used). This tree can be updated at runtime by deleting plans and goals as they complete and recalculating resource summaries each time a node is deleted. For example in the figure below, if *delegateB* is performed and deleted then summaries in *expA* and *merge & store* can be combined to provide an updated summary for π_2, which can in turn be combined with the summary of π_1 to provide an updated resource summary for the top level goal.

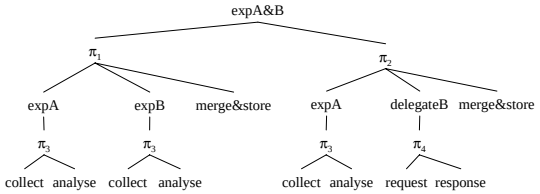
Although only the path from the root to the modified portion of the tree needs to be traversed for each update, this may be costly if done routinely (depending on the number of goals being pursued in parallel and the branching factor of the tree). Consequently we can simply flag nodes as requiring update, but delay the actual propagation of changes to resource summaries until we know that there is some need for monitoring or recalculation. This occurs when a potentially non-safe goal is being considered (requiring recalculation) or when such a goal is adopted (requiring monitoring). We can easily monitor only with respect to resources identified as contributing to the goal set being non-safe.

[8] This only means that the set of goals are safe with respect to resources – failure can still be caused by other factors including logical conflict.

[9] It may also wish to reason about whether to drop existing goals but we do not deal with that here.

An additional simple optimisation which we could use is to maintain a count at each node of the number of children which use each resource. As a child node completes, this count can be decremented. When it reaches zero the need for that resource can be set to zero for goal nodes and to the value of that resource in $P^{\mathcal{R}}$ for plan nodes.[10] This can in some cases allow faster updating and propagation as once a resource need is zero or the value of it in $P^{\mathcal{R}}$, remaining child nodes can be disregarded with respect to that resource.

With these optimisations any cost beyond a couple of trivial operations is incurred only when a need is identified.



6 Conclusion

A rational agent should not adopt goals that clearly conflict. We have presented mechanisms that allow an agent to be aware of the possible and necessary resource requirements of goals and to use these to detect and avoid conflicts. We also recommend that intelligent agents engage in reasoning behaviour about the current situation with regard to partially achieved goals, and we provide algorithms to make this computationally feasible.

The strengths of our approach of maintaining dynamic information about partially achieved goals are that it allows us to avoid explicitly representing time, and it has efficient runtime algorithms and simple representations, amenable to use in real applications. This work contributes towards achieving the consistency of goals which is a basic assumed property of BDI agents [8] but where there is surprisingly little existing practical work.

We have extended JACK [1] to incorporate goals, resource summaries, mechanisms for creating the goal-plan tree, mechanisms for dynamically updating this tree, and the algorithm for detecting the status of adopting a new goal set. The resulting system has been used to detect resource conflicts in the Mars rover example used in this paper. Currently the agent is cautious and only executes safe goals.

Although our work targets single agents, it can be extended to a multi-agent setting. A key issue is deciding when to communicate changes to the resource summaries of goals: on the one hand we want to communicate frequently so that each agent is working with up-to-date information, on the other hand we don't want to be sending a message after every execution step.

Although the area of conflict in agent systems has seen significant work (see for example [10]) the focus has been on understanding the many types of conflict that can occur, rather than on providing algorithms and data structures allowing conflict detection and resolution. Also, much of the work has focussed on conflicts *between* agents, rather than on conflict *within* an agent as explored in this paper. The latter allows for considerably richer solutions since more information is available to the agent and since the inability to influence other (autonomous) agents is not an issue.

In the area of databases and operating systems resource conflict is also an issue. However, the requirements lead to different solutions: whereas "classical" resource management prevents conflicts by using locking, situated intelligent agents cannot always use locking, and cannot always prevent conflict. However, since intelligent agents are designed to detect and recover from various failures, it is *not necessary* to guarantee that no conflicts will occur – it is more important to *detect* conflicts so that appropriate recovery steps can be taken. Note that deadlock is not an issue since preemption is always possible (a plan can be aborted).

Future work includes detecting and avoiding conflicts based on issues other than resource needs, further investigation of the representation of resources, and extending the presented algorithms to better deal with cases where resource requirements are not known in advance (at the moment these cases are handled by specifying infinite possible resources and letting the dynamic update of resource requirements make this more precise at runtime).

[10] recall that $P^{\mathcal{R}}$ is the user defined resource needs of the actions within a plan.

ACKNOWLEDGEMENTS

We would like to acknowledge the support of Agentis International, of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (under grants C49906798 and CO0106934).

REFERENCES

[1] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas, 'JACK Intelligent Agents - Components for Intelligent Agents in Java', Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, (1998).

[2] Bradley J. Clement, Anthony C. Barrett, Gregg R. Rabideau, and Edmund H. Durfee, 'Using abstraction in planning and scheduling', in *Proceedings of the Sixth European Conference on Planning (ECP-01)*, (September 2001).

[3] Bradley J. Clement and Edmund H. Durfee, 'Identifying and resolving conflicts among agents with hiearchical plans', in *AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities, AAAI Technical Report WS-99-12*, pp. 6–11, (1999).

[4] Bradley J. Clement and Edmund H. Durfee, 'Performance of coordinating concurrent hierarchical planning agents using summary information', in *Intelligent Agents VII : Agent Theories, Architectures, and Languages (ATAL-2000)*, eds., Cristiano Castelfranchi and Yves Lespérance, number 1986 in LNAI. Springer-Verlag, (July 2000).

[5] Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge, 'A formal specification of dMARS', in *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages*, eds., M.P. Singh, A.S. Rao, and M. Wooldridge, pp. 155–176. Springer-Verlag LNAI 1365, (1998).

[6] Marcus J. Huber, 'JAM: A BDI-theoretic mobile agent architecture', in *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pp. 236–243, (May 1999).

[7] F. F. Ingrand, M. P. Georgeff, and A. S. Rao, 'An architecture for real-time reasoning and system control', *IEEE Expert*, **7**(6), (1992).

[8] Anand S. Rao and Michael P. Georgeff, 'Modeling rational agents within a BDI-Architecture', in *Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference*, eds., James Allen, Richard Fikes, and Erik Sandewall, pp. 473–484, (April 1991).

[9] Anand S. Rao and Michael P. Georgeff, 'An abstract architecture for rational agents', in *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, eds., C. Rich, W. Swartout, and B. Nebel, pp. 439–449, San Mateo, CA, (1992). Morgan Kaufmann Publishers.

[10] *Conflicting Agents: Conflict Management in Multi-Agent Systems*, eds., Catherine Tessier, Laurent Chaudron, and Heinz-Jürgen Müller, Kluwer Academic Publishers, 2000. ISBN 0-7923-7210-7.

[11] Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah, 'Declarative & procedural goals in intelligent agent systems', in *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, Toulouse, France, (April 2002).