Detecting & Exploiting Positive Goal Interaction in Intelligent Agents

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

Lin Padgham
RMIT University
Melbourne, Australia
linpa@cs.rmit.edu.au

Michael Winikoff
RMIT University
Melbourne, Australia
winikoff@cs.rmit.edu.au

ABSTRACT

Rational agents typically pursue multiple goals in parallel. However most existing agent systems do not have any infrastructure support for reasoning about either positive or negative interactions between goals. Negative interactions include such things as competition for resources, which if unrecognised can lead to unnecessary failure of both goals requiring the resource. Positive interactions include situations where there is potentially a common subgoal of two goals. This paper looks at mechanisms for identifying potential common subgoals, and attempting to schedule the actions of the agent to take advantage of this. Potential common subgoals are identified by maintaining summaries of definite and potential effects of goals and plans to achieve those goals. Template summaries for goal types are produced at compile time, while instance summaries are maintained and updated at execution time to allow the agent to choose and schedule its plans to take advantage of potential commonality where possible. This increases the ability of the agent to act in a rational manner, where rational is loosely defined as the sensible behaviour exhibited by humans.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling, Plan execution, formation, and generation*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence— *Intelligent agents*

General Terms

Algorithms

Keywords

Goal Interactions, Co-ordinating Plans, Rational Agents

*Student primary author.

AAMAS '03 Melbourne, Australia

1. INTRODUCTION

Goals are a central concept to pro-active intelligent agents [16]. Typically an intelligent agent will have multiple goals which are active simultaneously. Goals can interact both negatively and positively. Intelligent agents should be *rational* in the way they go about achieving these goals. One aspect of rationality is to avoid conflicting goals, for example 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. In [14, 15] we presented a framework for detecting and resolving conflicts related to resources.

Goals can interact not only negatively but also positively [16]. For example a Mars rover agent that has a goal to perform soil experiments at location A and a goal to perform rock experiments at location A could perform the goals sequentially by going to A, performing the soil experiments, return back to lander, then go to A, perform the rock experiments and return back to lander. Alternatively it could go to A, perform the soil experiments, perform the rock experiments and then return to lander. Clearly the second approach is more intelligent, and is brought about by the agent reasoning about the positive interaction between the goals. This makes the agent actions more efficient. The contribution of this paper is to provide a framework that allows the agent to detect and make use of potential positive interactions, thus contributing to the theory of rational choice of intelligent agents.

Much work has been done on merging plans, where the plans to be executed are determined before execution [7, 5, 10]. However these are not applicable to the type of agent systems that we work with, which operate in highly dynamic environments where it is generally not possible to decide which plans to use in advance as they depend on (dynamic) environmental conditions.

Our approach identifies positive interaction between goals without requiring that each goal has a fully elaborated plan to achieve it. Plan merging is done through scheduling. For example, suppose that the goals G_1 and G_2 have a common sub-goal G' that could potentially be performed only once for both goals rather than once for each goal. In order to do this we synchronise the execution of the two goals such that G' is executed only once.

Horty and Pollack in [7] present a theory of rational choice where an agent evaluates its options in the context of its existing plans. They discuss methods of merging type-identical[1]

[1]A *Type* is a function associating each plan step s with Type(s), its action type.

EF − Effects DE − Definite Effect PE − Potential Effects

DE − { (EF, [plan, plan ..]), (EF, [..]) ..} each effect has the set of plans that bring about the effect. Similar for PE .

SE (G) − Effect−summary of goal G. It is a two tuple < DE, PE >

DMP − [(EF, goal{plan, plan}, goal{plan, plan}) , (EF , goal{..}, ..)] is the data structure that contains plans that can be definitely merged. With each effect the goals and the plans of the goals that bring about the effect is stored.

PMP − [(EF, goal d/p {plan, plan}, goal d/p {plan, plan}) , ..)] similar to DMP except that the plans here are only potentialy mergable. For each goal of an effect the flag (d or p) indicates if the effect is definite or potential.

Common effect of G1 and G2 − an effect that is an effect of G1 and an effect of G2

WGL − The Waiting Goals List. A list of goals currently suspended due to scheduling. Assists in deadlock prevention.

Figure 1: Legend of terms

plan steps and how plan merging reduces the cost of the overall goal. Their work is interesting in that they evaluate potential goals in the context of other goals being pursued, which has some similarity to our approach. For example, assume an agent pursuing a goal of buying a shirt and considering a new goal of buying a tie. In this case even if buying a tie is not as important as some other goal, because buying a tie and buying a shirt have many aspects in common such as going to the shopping centre, going to a clothing shop etc, it may be pursued.

Their work does however, like many others, require complete plans. They also assume that plans do not fail, which is inappropriate for the kinds of systems we are dealing with. They also use explicit notions of temporal constraints, and causal links in their representation of plans. In contrast our approach performs online scheduling and monitoring of plans, taking plan failure into consideration and although we have no notion of explicit time constraints we generate sufficient information to perform reasoning about future positive interactions.

Pollack in [12] discusses ways in which intentions may be overloaded and argues that this can improve plan generation as well as plan recognition. Although [12] does not provide any detailed mechanisms, they provide a good theoretical foundation for overloading intentions. Our work then provides detailed mechanisms for exploiting positive goal interactions that can be directly implemented in agent development platforms such as PRS[9], JAM[8], dMARS [4], and JACK [1].

2. PLAN AND GOAL REPRESENTATION

The type of agents that we develop have a library of programmer defined plans which are utilised to satisfy goals at run-time. Goals are achieved by executing plans. Each goal has a number of alternative plans that can be used to achieve it. Each plan has a number (possibly zero) of sub-goals, all of which must be achieved to successfully execute the plan. This naturally defines a *goal-plan tree* where the children of each goal are the alternative plans that achieve it and the children of each plan are its sub-goals. The children of a goal node are alternatives, and thus are "or'd", whereas the children of a plan node are "anded". Given this goal-plan tree we denote nodes by a path expression from the root to the node. For example from figure 2 the node $MoveToPlan(A)$ is denoted as $PerformSoilExperimentGoal(A) \rightarrow SoilExperimentPlan \rightarrow MoveToLocation(A) \rightarrow MoveToPlan(A)$.

A goal-type is a template for a goal (similar to a class in Object-oriented terms). When an agent decides to pursue a goal, a goal-instance of the goal-type is created (instantiated). The same applies to a plan-type and plan-instance. Each goal-type will have a template goal-plan tree at compile time which gets instantiated at run-time with each goal. Sub-goals share the goal-plan tree of their respective top-level goal.

The execution cycle of an agent consists of the steps [15]:

1. Instantiate a goal-type creating a goal-instance.
2. Match the goal instance against plans in the plan library obtaining a set of *relevant* plan-types.
3. For each relevant plan-type, evaluate its context condition giving a plan instance for each context condition which evaluates to true.
4. Remove any plan instances which are equivalent to previously failed plan instances for this goal-instance. The remaining set of plan instances are the *applicable plans*.
5. 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-instance fails.

The above description characterises what are known as BDI (belief Desire Intention) systems [13], such as PRS [9], dMARS [4], and JACK [1].

In the work that we are doing we require the plans and goals of the agent to contain particular information.[2]

We require each plan-type to contain the following:

- **Type** - A label for the type of plan. e.g. MoveToPlan ⟨some location⟩
- **Pre-Condition** - A logical condition that must be true in order for an instance of the plan to begin execution. The condition need not hold once the plan has begun execution. Pre-conditions can be viewed as *context* conditions in systems such as JACK.
  e.g. TransmitDataPlan.preCondition(DataCollected)
- **In-Condition** - A logical condition that must be true for the duration of a plan-instance. The pre-condition is implicitly extended with the in-condition: a plan with pre-condition P and in-condition I is treated as though it really had pre-condition $P \wedge I$. The reason is that if P is true but I false then the plan will be chosen for execution and immediately fail because the in-condition is violated! e.g. AnalyseSoilPlan.inCondition (RoverStationary)

[2]Not all this information is required for the reasoning described in this paper but it is provided here for completeness and for consistency with our other work.

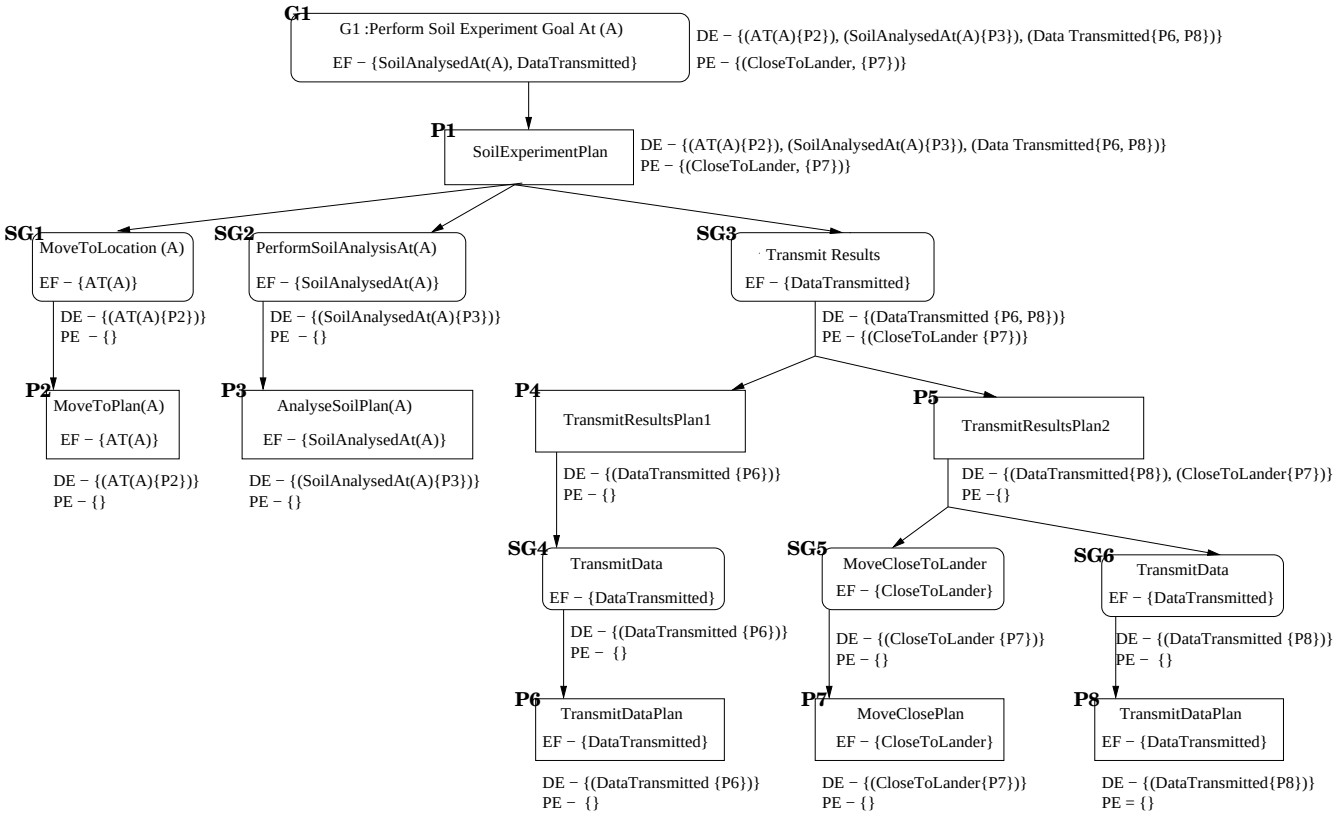

Figure 2: Interaction Tree example

- **Effects** - The effects of executing the actions within the plan (excluding subgoal effects) represented as logical conditions, i.e. what becomes true as a result of executing the plan's actions. The effects of a plan that are not effects of a goal are side-effects, or secondary expectations [12][3].
  e.g. RockExperimentPlan.Effects(RocksAnalysed and StorageSpaceFull[4]).

- **plan body** - The body of the plan which contains actions and sub-goals. 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 "$G_1 \| G_2$").

Similarly each goal-type consists of:

- **Type** - A label which indicates the type of the goal.

- **In Condition** - a logical condition that must be true during the entire execution of the goal.

- **Effects** (i.e. Success condition [16]) - logical condition that the goal is trying to achieve. The goal is satisfied when the success condition evaluates to true. This is used to identify the *primary expectations* [12][3] of a goal.

- **Plans** - The possible set of plan-types that can satisfy the goal [16].

In addition, each plan and goal instance have an **InstanceName** which is a label created at run-time to provide a handle to the plan/goal. This is basically a path expression in a goal-plan tree instance. e.g. $G_1 \rightarrow PerformRockExpGoal \rightarrow RockExperimentPlan$,
where G_1 is the label of the relevant goal-plan tree instance.

3. A SIMPLE EXAMPLE

Consider a mars rover that is deployed on the Martian surface to perform soil and rock analysis experiments at various designated locations. For illustration purposes we assume that the rover has storage space for results of four experiments, and regularly uploads stored data to the lander.

Assume that one such location is location A and the two goals related to it are PerformSoilExperimentAt(A) (lets call it G_1) and PerformRockExperimentAt(A) (G_2). Some of the sub-goals in the plans to achieve G_1 may include, moving to location A, analysing the soil, and transmitting the results. Plans to achieve G_2 may have sub-goals of moving to location A, analysing rock samples and transmitting the results. It can be seen that there are some common elements in the execution of G_1 and G_2, that could possibly be merged, allowing plans and/or sub-goals to be executed only once.

[3]Pollack defines expectations as beliefs about future activities and circumstances. *Primary expectations* are those that are directly intended by the agents actions and *secondary expectations* are those that are side effects of intended actions.

[4]In this example RocksAnalysed is a primary expectation and StorageSpaceFull is a secondary expectation

Figure 2 shows a simple goal-plan tree template for the goal-type $PerformSoilExperimentGoalAt(A)$ where A is a location. The *effect summaries* of the goal, which we will define and discuss in the next section, are attached to the nodes to the tree. We have labelled the nodes in the tree (e.g. $G1$, $P1$, $SG1$, etc), so that we use the label for the node instead of the full path name when making reference to that node. For example $SG3$ is a label for $PerformSoil$-$ExperimentGoalAt(A) \rightarrow$
$SoilExperimentPlan \rightarrow TransmitResults$.

In the following sections we will use this example to illustrate how the rover agent can identify such interactions and how it can facilitate merging of plans.

4. EFFECT SUMMARIES

We use the *effects* of goals and plans to identify positive interactions. In this section we will define the notion of effect summaries, which are based partly on our own work with resource summaries [15] but which are also significantly influenced by the plan summaries of Clement and Durfee [2]. We then use these effect summaries in later sections where we describe our algorithms for recognising and taking advantage of possible positive interactions.

We classify effects into *definite effects* and *potential effects*. These definitions are analogous to the definitions of necessary and possible resources in our previous work [15], as well as being very similar to the definitions of "must" and "may" conditions in Clement and Durfee [2].

We classify effects into *definite effects* and *potential effects*. We define a **definite effect** to be an effect that will *definitely* be met at some point along every possible path of achieving the goal. In other words this effect cannot be avoided if the goal/plan doesn't fail [5]. It is important to note that this effect need not be required by all plans, but is required by at least one plan in every possible way of achieving the goal.

We define a **potential effect** to be an effect that will *possibly* be met at some point in time during the execution of the goal. This effect is achieved by at least one plan in at least one path (but not all paths) to achieve the goal. Unlike our work on necessary and possible resources [15], we define definite and potential effects to be exclusive – definite effects are not a subset of the potential effects.

We use the notion of summary information [3, 15] to define effect summaries which basically summarises the definite and potential effects of goals and plans. We also define and use a structure called an *interaction tree*. The interaction tree is a goal-plan tree where each node is annotated with the respective effects summary. This is similar to the resource tree detailed in [15], which is a goal plan tree with resource summaries attached to each node.

Effect summaries are calculated by propagating the effects of plans and sub-goals up the tree in order to perform reasoning about interactions the goal might have with other goals. These summaries are used to reason about possible plan merges.

Figure 2 shows the interaction tree of a simple goal of a Mars rover agent to perform soil experiments at location A.

The effects summary of a goal G ($S_E(G)$), is a two tuple $\langle D_E, P_E \rangle$ where D_E and P_E are the definite effects and potential effects of G respectively.

The data structure for definite effects (D_E) and potential effects (P_E) is a set of effects and associated with each effect the set of plans that bring about that effect.
$D_E = \{(effect, \{plan, plan..\}), (effect, \{plan, ..\})..\}$
For example the DE and PE of subgoal
$TransmitResults(SG3)$ in figure 2 are
$DE = \{(DataTransmitted, \{P6, P8\})\}$ and
$PE = \{(ClosetoLander, \{P7\})\}$

4.1 Deriving Effect Summaries

In this section we describe how we can derive the effect summary for each plan and each goal within the agent system. This effect summary will assist in identifying and facilitating positive interactions.

At each node in the tree we derive the effect summary of that node by combining the effects of the children nodes with the local effects of the node we are at. Since it can be assumed that there are no direct effects of a goal, other than those resulting from the plans associated with it, calculation of the effect summary of a goal node results from combining the effect summaries of its children plans.

Plans on the other hand may have both effects resulting from their sub-goals and also effects resulting from direct actions within the plan. Thus the effect summary of a plan node combines the direct effects of the plan with the effect summaries of the sub-goals within the plan, i.e. the children nodes in the goal-plan tree. Unlike in calculation of resource summaries [15] we need not differentiate between sequential and parallel sub-goals of a plan as all will be achieved (if the plan runs successfully) and therefore all effects will be achieved.

We will first define a few useful operators that are necessary for the derivation of effect summaries. We define the intersection operator $\overset{*}{\cap}$ on two sets of effects E_1 and E_2 as

$$E_1 \overset{*}{\cap} E_2 = \{ (e_1, p_1 \cup p_2) \mid (e_1, p_1) \in E_1 \wedge (e_2, p_2) \in E_2 \wedge e_1 = e_2 \}$$

Remember that p_1 and p_2 here are sets of plans.

The next operator is the union operator $\overset{*}{\cup}$, which on E_1 and E_2 can be defined as:

$$E_1 \overset{*}{\cup} E_2 = \{ (e,p) \mid ((e,p) \in E_1 \wedge (e,p) \notin E_2) \vee ((e,p) \in E_2 \wedge (e,p) \notin E_1) \vee ((e,p) \in E_1 \overset{*}{\cap} E_2)\}$$

where e is an effect, and p is a set of plans. This definition says that $E_1 \overset{*}{\cup} E_2$ contains (i) anything in E_1 that isn't in E_2, (ii) anything in E_2 that isn't in E_1, and (iii) anything that is in $E_1 \overset{*}{\cap} E_2$.

The subtraction of two set of effects E_1 and E_2 can be defined as:

$$E_1 - E_2 = \{(e,p) \mid ((e,p) \in E_1) \wedge \neg(\exists p' \cdot (e,p') \in E_2)\}$$

We now can define the $\otimes$ operator which combines the effect summaries of two plans. In pursuing a goal, one of these plans will be chosen to be executed to achieve that goal, if this plan fails then alternative plans will be selected, but only one plan will be selected and executed at any given time. The combined definite effects are the effects that are

[5] Definite effects of a goal will include the primary effects of the goal, i.e. what the goal is trying to achieve, as well as side-effects of the goal that cannot be avoided.

definite effects of both plans. The combined potential effects are the union of the potential effects of both plans and the definite effects that are not common to both plans. Formally,

$$\langle D_{E1}, P_{E1}\rangle \otimes \langle D_{E2}, P_{E2}\rangle = \langle (D_{E1} \overset{*}{\cap} D_{E2}) , (P_{E1} \overset{*}{\cup} P_{E2} \overset{*}{\cup} ((D_{E1} \overset{*}{\cup} D_{E2}) - (D_{E1} \overset{*}{\cap} D_{E2})))\rangle$$

For example from figure 2
$S_E(TransmitResPlan2) \otimes S_E(TransmitResPlan1) =$
$\langle\{(DataTrasmitted\{P8\}), (CloseToLander\{P7\})\}\rangle$
$\otimes \langle\{(DataTransmitted\{P6\})\}, \{\}\rangle =$
$\langle\{(DataTrasmitted\{P6, P8\})\}, \{(CloseToLander\{P7\})\}\rangle$

The $\oplus$ operator is used to combine the summaries of two goals. This operator is used when combining the sub-goals of a plan to produce an effect summary for that plan. The combined definite effects of two goals are the union of the definite effects of each goal. Similarly for potential effects. Where an effect is, for example, in both D_{E1} and in P_{E2} then it is a definite effect of the resulting summary, and hence should not be also considered to be possible (recall that definite and possible are defined to be mutually exclusive). The definition thus has the possible set explicitly exclude the resulting definite set.

$$\langle D_{E1}, P_{E1}\rangle \oplus \langle D_{E2}, P_{E2}\rangle = \langle (D_{E1} \overset{*}{\cup} D_{E2}), ((P_{E1} \overset{*}{\cup} P_{E2}) - (D_{E1} \overset{*}{\cup} D_{E2})) \rangle$$

For example from figure 2
$S_E(MoveCloseToLander) \oplus S_E(TransmitData) =$
$\langle\{(CloseToLander\{P7\})\}, \{\}\rangle$
$\oplus \langle\{(DataTransmitted\{P8\})\}, \{\}\rangle =$
$\langle\{(DataTrasmitted\{P8\}), (CloseToLander\{P7\})\}, \{\}\rangle$

Given the above the effect summary of a goal G can be formally derived as follows:

$$S_E(G) = \bigotimes_{p \in G.plans} S_E(p)$$
$$S_E(p) = \bigoplus_{g \in p.subgoals} S_E(g) \oplus \langle\{(e, \{p.name\}) | e \in p.EF\}, \{\}\rangle$$

Figure 2 shows the derived interaction summaries of all the nodes of the goal-plan tree for the goal to perform soil experiments.

Often the agent may want to consider sets of goals for adoption instead of just single goals. In order to determine possible interactions between sets of goals we need to derive the effect summary for the set of goals. This can be done by applying the $\oplus$ operator on all of the goals in the set as follows

$$S_E(GoalSet) = \bigoplus_{g \in GoalSet} S_E(g)$$

The algorithms that we present in following sections are based on effect summaries and they can be applied to individual goals or sets of goals with no distinction.

4.2 Dynamic Update of the Interaction Tree

The interaction tree template for each goal-type is built at compile time and a copy is instantiated for every goal-instance. This copy of the interaction tree is updated dynamically as the agent executes its goals and plans. This provides the agent with up-to-date information enabling it to make more informed rational decisions. The basic idea is to update the tree at runtime by deleting plans and goals as they complete and recalculating the effect summaries each time a node is deleted.

When a plan completes execution either by successful completion or failure, the tree is updated as follows:

- the corresponding node is removed from the tree.
- if the plan successfully completed then the goal node that is the parent is also complete and hence removed from the tree, and the change propagated up the tree.
- if the plan failed and the parent goal has no other means of success (i.e. if all other alternate plans have also failed, or if there is no other alternate plan) then the parent goal fails, which in turn fails its parent plan. All three failed nodes are removed and the change propagated up the tree.
- if the plan failed, but the parent goal has other plans (children) to attempt, then the effect-summary of the parent node is recomputed and this change is propagated up the tree.

When a sub-goal completes execution the tree is updated as follows:

- the corresponding node is removed from the tree.
- the effect-summary of the parent node is re-computed and propagated up the tree.

Note that the change propagation is a recursive procedure. The algorithms for performing the above is similar to the algorithms provided in [15] for updating resource summaries in a goal-resource tree.

5. IDENTIFYING AND FACILITATING POSITIVE INTERACTION

The guiding intuition here is that plans of two goals that bring about the same effect could possibly be combined (merged) hence reducing the overall cost and increasing efficiency of the pursuit of goals. This section describes mechanisms for identifying situations when plans of different goals can be merged and the process of scheduling the pursuit of sub-goals enabling such plans to be merged.

We identify positive interactions between two goals by using the effect summaries of the two goals, and facilitate plan merging by using data structures that we term *Definitely Mergeable Plans*(DMP) and *Possibly Mergeable Plans*(PMP). These structures are used to store plans that could definitely and possibly be merged respectively in the pursuit of the agent's current goals. In order to avoid deadlocks when synchronising execution we use a data structure which we shall term the *WaitingGoalsList*(WGL) which is a list of goals that are currently suspended.

Each entry in the DMP is an effect followed by the goals that bring about this effect and associated with each goal the set of plans of the goal that cause the effect [6].
DMP = [(effect, (goal{plan, plan, ..}),(goal{plan, ..}),..) ,
(effect, ...), ...]
All the effects in the DMP are definite effects of their respective goals in the DMP.

[6] i.e. the plans for which the effect is a direct effect.

The data structure for the possibly Mergeable plans, PMP, is a similar structure to the DMP. They differ in that whilst e is a definite effect of all its associated goals in DMP, in PMP, e can be either a potential or definite effect of the associated goals. This is explicitly represented by a flag as follows;
PMP=[(effect,(goal,p/d,{plan, ..}),(goal,p/d,{plan, ..})..)..]

We shall now describe how these data structures are made use of. Consider two goals G_1 and G_2 which the agent has just begun to pursue concurrently where e is a common effect of the two goals. Let the plans[7] that achieve e be $\{P_1, P_2\}$ in G_1, and $\{P_3, P_4\}$ in G_2.

There are three ways in which e can be common to both goals. e can be:

1 - a definite effect of G_1 and G_2

2 - a definite effect of G_1 and a potential effect of G_2

3 - a potential effect of G_1 and G_2

Let us first consider **case 1**: In this situation, when the goal pusuit begins the plans that will be executed are unknown (as they depend on the context of the agent's environment). However we do know that at least[8] one of P_1,P_2 and at least one of P_3,P_4 will be attempted, since e is a definite effect of G_1 and G_2. In this case we place this information in the DMP as follows;
DMP = [(e, (G_1 {P_1, P_2}), (G_2 {P_3, P_4})), ..]
which indicates to the agent that there is a definite interaction between the plans of the two goals and that the goals should be scheduled. The basic idea is to synchronize the plans by flagging them as ready to be executed when they are reached, wait for other plans that could be merged with them to be reached, and then perform a merge.

This is done by incorporating the following steps into the agent execution cycle.

Before a plan P of goal G is executed the agent checks if it is in the DMP.
If it is in the DMP **then**
  let e be the associated effect in the DMP
  If all the goals associated with e (other than G) have an associated plan that has been flagged as ready to be executed **then**
    perform a merge between the plans that are ready to be executed for e
  Else if any of the goals(other than G) associated with e, that does not have a plan flagged as ready (i.e. the goals that G would have to wait for) is already in the WaitingGoalsList **then**
    P is executed and not suspended (in order to avoid any potential deadlocks)
  Else
    P is flagged as ready to be executed, and the execution of G is suspended. The name of the goal instance (G) is placed in the WGL.

For example in case 1, assume that P_1 is reached in the execution of G_1. Since it is in the DMP, it is flagged but not executed. The agent then waits for either P_3 or P_4 to be reached in the execution of G_2. Lets assume that P_3 is reached. Now since all the goals that have e as a definite effect have reached a plan that brings about e, the agent would attempt to merge these plans.

It is important to avoid deadlocks when suspending goal pursuit. For example consider the following in the DMP; (x, (G_a {p_1}), (G_b {p_2})) and (y, (G_a {p_3}), (G_b {p_4})). If p_1 is reached and G_a waits for p_2 in G_b, and then p_4 is reached and G_b waits for p_3 in G_a then there is a deadlock. In the algorithm above G_b will not be allowed to be suspended because a goal that it is waiting for (G_a) is already suspended (i.e. in the WGL). This method prevents deadlocks but is not the most optimal mechanism for deadlock detection and prevention. Introducing more complex deadlock detection mechanisms would increase the complexity of the agent system. If the additional complexity is acceptable then deadlock prevention mechanisms such as the resource allocation models found in operating systems [6] or transaction models used in distributed networks [11] can be adopted. The details will not be presented in this work.

Case 2 is when the effect (e) is a definite effect of one goal (G_1) and a potential effect of the other (G_2). In this situation the agent is unsure whether P_3 or P_4 will be reached in the execution of G_2. Therefore the information is stored in the PMP rather than in the DMP. This information in the PMP is monitored as follows;

If P_3 is reached **then**
  since e is a definite effect of G_1, P_1 or P_2 will definitely be reached, hence this entry is moved to the DMP with P_3 flagged as ready to be executed.
If P_1 is reached **then**
  since e is only a potential effect of G_2 the agent cannot know if either of P_3 or P_4 will be reached. In this case the agent has two choices;
  1. begin to execute P_1 and remove this entry from the PMP, because neither P_3 nor P_4 may be reached. This approach is *cautious*, which would prevent the agent from delaying the execution of P_1 unnecessarily if P_3 nor P_4 is reached. However, if one of them is reached then this approach will fail to capitalise on this.
  2. flag P_1 as ready to be executed and move this entry into the DMP waiting for either P_3 or P_4 to be reached. This approach is *optimistic*. If either P_3 or P_4 is reached then this entry would attempt to exploit this interaction. However if neither P_3 nor P_4 is reached then P_1 would have to wait until G_2 is either satisfied or no longer relevant, to begin execution. By moving the entry into the DMP, the effect although potential is treated as if it was a definite effect.

In addition to the above monitoring the agent must also monitor for situations where the effects in PMP move from being potential effects to definite effects of a goal and also when the effects are no longer applicable. This can be done when the interaction tree is updated as described in section 4.2. If effects that were potential change to definite effects, and all the goals associated with the effect have it as a definite effect then the relevant entry is moved to the DMP. If the effect is no longer applicable for a particular goal, then the goal is removed from the entry associated with the effect in PMP. If there is not at least two other goals left in the

[7]These are plan instances and it is possible to have, say, P_1 and P_3 be different instances of the same plan type, for example a MoveTo plan.

[8]If a plan fails then alternatives are tried.

entry after this deletion, the entry is removed, because for a merge there must at least be two goals involved.

In **case 3**, e is a potential effect of the goals, G_1 and G_2. Since the effect is uncertain, neither of the plans, P_1, P_2, P_3, P_4, are guaranteed to be executed (reached). In such a situation the agent can take one of the following options;

- be cautious and execute G_1 and G_2 independantly.
- be optimistic and place them in the DMP.

If the type of application system is more resource bound then the optimistic approach would suit better since it will try to merge plans in all possible situations, however it may waste time. If the application system is more time-constrained then the cautious approach would suit better as it would ensure that the agent would not delay the execution of its plans for a merge that may never happen.

In the event that two goals have no common effect (neither definite nor potential) then there is no possibility of merging plans with respect to effects.

The above mechanisms will identify and facilitate plan merging. However there are instances where although plans could possibly be merged they should not be merged. For example consider a mars rover with a goal G_1 to perform task $task_1$ and then *refuel*, and a goal G_2 to perform task $task_2$ and then *refuel*. Although these two goals have a common plan (which maps to the sub goal of) of *refuel*, this must be performed once for each goal, because if it performs $task_1$ and doesn't *refuel*, it may not have sufficient fuel to complete $task_2$. In order to prevent such plans from being merged, we introduce a boolean attribute *mergeable* into the representation of the plan-type discussed in section 2. This attribute is assigned a default value of *true* for each plan-type and when the plan instances that should not be merged are instantiated the mergeable attribute of the plan instance is set to *false*.

5.1 Plan Merging

When the agent decides to merge two plans it needs to perform reasoning about the feasibility (i.e. is it possible) and choice of merge (i.e. which plan to execute).

Consider two plans P_1 and P_3 that the agent wants to merge.

- **if** P_1 and P_3 are of the same type (e.g. they are both MoveTo(A))[9]
  then either of the plans can be executed.
- **if** P_1 and P_3 are not the same type but they achieve only e and no other effect,
  then either of the plans can be executed.
- **if** P_1 achieves e and x where x is a potential effect of the goal G_1 (i.e. side effect), and P_3 only achieves e,
  then either of the plans can be executed since x is not a necessary effect.
- **if** P_1 achieves e and x where x is also *a definite effect* of the goal G_1, and P_3 only achieves e,
  then P_1 must be executed because P_3 will not bring about x which is necessary for G_1.
- **if** P_1 achieves e and x, and P_3 achieves e and y where x is a definite effect of G_1 and y is a definite effect of G_2, and $x \neq y$,
  then it is not possible to merge these plans because if only one of P_1 or P_3 is executed then one of x or y will not be achieved and this is not rational as they are definite effects of the goals. In such a situation, which we will term *useless wait*, the plans must be executed individualy. Section 5.2 deals with ways of minimising the occurence of this situation.

Everytime the above merge is performed the DMP is modified as follows.

- The entry in question is removed from the DMP; and
- When the plan (in the case where a merge is performed) or plans (in the case where the plans were executed independantly) complete, the effect-summary of the associated goals are re-computed as discussed in section 4.2. These updated summaries are used to check for interaction between the goals with respect to the effect e.

The reason for re-checking only for the effect e is because other entries in the DMP are not effected by the merge. The reason for re-checking for interactions with respect to e is because there may be another pair of plans (one from each goal), that could be merged for the same effect.

5.2 Avoiding useless wait

The *useless wait* situation can cause the system to be inefficient. However, under certain assumptions, this situation is unlikely to occur. If we assume that plans are cohesive, i.e. are written to achieve a single effect, rather than a conjunction of effects, then the useless wait condition will not arise. Plans in BDI-style systems appear to meet this condition. Nevertheless if required the following methods can be applied. The first approach is to apply the following filter before adding these goals to DMP. If **all** of the goals in question have **at least one** plan that has a definite effect that is not common to all the goals, then these goals are not added to the DMP. Whilst this method prevents the problem it is too strict and will disallow a lot of potential merges.

The second approach is to filter goals as follows. If **all** the plans in **every** goal has a definite effect that is not common to all the goals, then these goals are not added to the DMP. This approach will remove goals that will definitely result in a *useless wait*, however this does not guarantee that the situation will never arise.

A better method would be to apply the second filter and from the goals that pass the filter detect those that will never bring about an *useless wait* (i.e. goals that pass through the first filter) and place them in the DMP. The other goals are placed in the PMP and each time one of them are updated with respect to interactions, the process is repeated on the set of goals in the PMP by applying the two filters. So the basic idea is that those that definitely result in an useless wait are not considered to be merged, those that will never result in an useless wait will be moved to the DMP and the others will remain in the PMP and monitored further.

6. CONCLUSION

[9]For the purposes of this work the parameter is included in the plan type as we do not yet deal with logical variables.

This paper has focussed on the ways in which an intelligent agent can recognise, and take advantage of, possibilities for merging plans to more efficiently achieve multiple, potentially interacting, goals.

We have described the representation we use for goals and plans, and the effect-summaries calculated, which are similar to summaries used by Durfee et al. [2] and analogous to resource summaries in our earlier work [15]. These are then used to identify positive interactions between goals. We have described the algorithm whereby we compute these summaries initially at compile time, and the mechanisms for updating them at execution time. Finally we have shown how this information is used to recognise opportunities for merging of plans, and the mechanisms by which we schedule and synchronise execution of goals and plans, to take advantage of these opportunities where possible.

In contrast to existing work on plan merging [7, 5, 10], our work does not require complete knowledge of the plans to be executed in advance, which makes this work suitable for intelligent agents in highly dynamic domains. We construct data structures at compile time which are then updated dynamically as the agent pursues its goals. Therefore the agent always reasons about its goals in its current situation. The dynamic information about partially achieved goals also allows us to avoid an explicit notion of time. The representations and algorithms that we have provided in this work are simple and efficient and are easily implemented in agent implementation platforms.

We have implemented the representations and mechanisms described in this work in a prototype which extends JACK [1] to incorporate the notion of goals and the reasoning mechanisms described. The test bed application is the Mars rover example used in this paper. We are in the process of doing experiments to analyse the costs and benefits of the reasoning mechanims described in this and related work.

Future work also includes; investigating more optimal mechanisms for avoiding deadlock situations when scheduling the execution of goals, whilst maintaining the simplicity of our algorithms; investigating a similar notion of positive interactions in a multi-agent setting; and investigating other means of positive interactions other than common effects between goals.

7. ACKNOWLEDGEMENTS

We would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant CO0106934.

8. REFERENCES

[1] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998.

[2] B. J. Clement and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-99)*, pages 495–502, July 1999.

[3] B. J. Clement and E. H. Durfee. Top-down search for coordinating the hierarchical plans or multiple agents. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 252–259, Seattle, WA, USA, 1999. ACM Press.

[4] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages*, pages 155–176. Springer-Verlag LNAI 1365, 1998.

[5] D. E. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2-3):143–181, 1992.

[6] A. N. Habermann. Prevention of system deadlocks. *Communications of the ACM*, 12(7):373–377, 1969.

[7] J. F. Horty and M. E. Pollack. Evaluating new options in the context of existing plans. In *Artificial Intelligence*, volume 127, pages 199–220. 2001.

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

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

[10] D. S. Nau, Q. Yang, and J. Hendler. Optimization of multiple-goal plans with limited interaction. In *Proc. of the Workshop on Innovative Approaches to Planning*, pages 160–165, Scheduling and Control, San Diego, CA, 1990.

[11] R. Obermarck. Distributed deadlock detection algorithm. *ACM Transactions on Database Systems (TODS)*, 7(2):187–208, 1982.

[12] M. E. Pollack. Overloading intentions for efficient practical reasoning. *Noûs*, 25(4):513–536, 1991.

[13] A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 439–449, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[14] J. Thangarajah, L. Padgham, and J. Harland. Representation and reasoning for goals in BDI agents. In M. Oudshoorn, editor, *Proceedings of the Twenty-Fifth Australasian Computer Science Conference (ACSC 2002)*, Melbourne, Australia, 2002.

[15] J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In van Harmelen, editor, *Proceedings of the 15th European Conference on Artifical Intelligence 2002 (ECAI 2002)*, Lyon, France, 2002.

[16] 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 (KR2002)*, Toulouse, France, Apr. 2002.