Detecting & Avoiding Interference Between Goals in Intelligent Agents

John Thangarajah and **Lin Padgham** and **Michael Winikoff**
RMIT University
Melbourne, AUSTRALIA
{johthan,linpa,winikoff}@cs.rmit.edu.au

Abstract

Pro-active agents typically have multiple simultaneous goals. These may interact with each other both positively and negatively. In this paper we provide a mechanism allowing agents to detect and avoid a particular kind of negative interaction where the effects of one goal undo conditions that must be protected for successful pursuit of another goal. In order to detect such interactions we maintain summary information about the definite and potential conditional requirements and resulting effects of goals and their associated plans. We use these summaries to guard protected conditions by *scheduling* the execution of goals and plan steps. The algorithms and data structures developed allow agents to act rationally instead of blindly pursuing goals that will conflict.

1 Introduction

Pro-active intelligent agents are those agents that pursue goals [Thangarajah *et al.*, 2002a]. These agents typically have multiple goals that are active simultaneously. These goals can interact with each other both positively and negatively. For example consider a Mars rover agent that has a goal to perform soil analysis at location A and another goal to perform rock analysis at location B. Clearly the agent cannot pursue these goals in parallel since they require the agent to be at different locations. This is an example of negative interaction.

An intelligent agent should not blindly pursue conflicting goals or unintentionally make its own goals unattainable if such is easily avoidable. In earlier work [Thangarajah *et al.*, 2003] we provided mechanisms for identifying and facilitating positive interactions with respect to the effects of goals. Horty and Pollack's work on evaluating an agent's options in the context of its existing plans [Horty and Pollack, 2001] and Pollack's work on overloading intentions [Pollack, 1991] also contribute to managing positive interactions.

In pursuing their goals intelligent agents often use intermediate steps which enable later steps to achieve the goal. This is often managed using a series of plan steps where the effect of an earlier step achieves the pre-condition for a later step. A rational agent engaged in this kind of pro-active behaviour should not allow the pre-conditions that have been achieved by an earlier step to be interfered with before the later step that relies on the pre-condition is done. For example suppose an agent is using a plan to cook dinner, involving going to the store, buying some items, coming home and preparing the meal. Having arrived at the location of the store, a rational agent would not allow a plan which took the agent elsewhere to intervene, prior to actually purchasing the items at the store.

This paper focuses on recognising situations where these linkages exist in a plan and controlling plan interleaving to ensure that an agent does not allow negative interference in the pursuit of separate parallel goals. We provide detailed mechanisms that can be easily implemented in agent platforms such as PRS [Ingrand *et al.*, 1992], JAM [Huber, 1999], dMARS [d'Inverno *et al.*, 1998], and JACK [Busetta *et al.*, 1998].

There has been significant work in the area of conflicts in agent systems [Tessier *et al.*, 2000]. However, the focus has been on multi-agents and identifying various types of conflicts. Our work focuses rather on providing algorithms and representations that allow detection and resolution of conflicts due to multiple parallel goals in a single agent.

There is also work in plan scheduling to avoid conflicts such as the work of [Boutilier and Brafman, 1997] and [Clement and Durfee, 1999b; 1999a]. Boutilier and Brafman provide a representation of concurrent interacting actions by extending the STRIPS action representation language, and provide an algorithm for planning with concurrent plans. However this work, like traditional planning approaches, requires the agent to have a completed plan, and perform the scheduling prior to execution (i.e. off-line). This is not useful for the type of agents that we deal with which are situated in highly dynamic environments where it is generally not possible to determine which plans to use in advance.

Clement et al. use the notion of summary information to co-ordinate plans at abstract levels. They have provided a formalism which defines correct interleaved plan execution [Clement and Durfee, 1999b] and a scheduling algorithm which uses this information to appropriately interleave plans to avoid resource conflicts [Clement *et al.*, 2001]. The work presented here focusses on recognising the way in which early steps in a plan typically pave the way for later steps, setting up "dependency links" and provides mechanisms to ensure that an agent recognizes these in managing its execu-

tion of parallel goals. We also provide mechanisms for continually updating summary information so that agents are also able to reason about partially completed goals.

In [Thangarajah *et al.*, 2002b] we provided means for an agent to detect and avoid resource conflicts by using summary information about resources.[1] In the current paper we use a notion of interaction summaries, based on the effect summaries developed by Clement and Durfee [Clement and Durfee, 1999b], containing information about effects, pre-conditions and in-conditions of goals and plans. Effects are the conditions that a goal or plan aims to achieve, pre-conditions are conditions that must be true for a plan to begin execution, and in-conditions are conditions that must remain true while the relevant goal or plan is active.

Specifically, we protect in-conditions and pre-conditions that have been achieved by earlier steps in a plan. We do not protect all pre-conditions: if the pre-condition has not been achieved by a previous step then there is no clear justification for protecting it. In particular, when defining the notion of *preparatory effects* in section 3, we only consider dependencies between nodes that are part of the same top-level goal.

In this work we consider only "achievement goals"[2], that is goals where the agent attempts to achieve some condition, and when this condition is achieved the goal is removed from the agent's mental state.

2 Plan and Goal representation

This work builds on our previous work presented in [2002a; 2002b; 2003]. The representation of goals and plans are therefore similar to what we presented in [Thangarajah *et al.*, 2003]. To summarise these agents have a pre-defined set of plans which are used to satisfy goals at run-time. A goal-type is a template for a goal which consists of (i) a label that indicates the type of the goal, e.g. TransmitResultsGoal (ii) in-conditions of the goal[3] (iii) the effects of the goal[4] and (iv) the possible set of plan-types that can satisfy the goal [Winikoff *et al.*, 2002].

Similarly a plan-type is a template for a plan and consists of (i) a label for the type of plan (ii) pre-conditions for the plan to begin (iii) in-conditions that must be maintained during plan execution (iv) the direct effects of the plan and (v) a plan body which specifies what the plan does. 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 "$G_1 \| G_2$").

The pre-conditions include in-conditions[5]. The reason for this is that in-conditions are stronger than pre-conditions: if an in-condition fails to hold when a plan begins executing then the plan fails immediately.

Each goal-type G also has a *goal-plan tree* template where a goal-plan tree is a tree structure of goals whose children are the plans that achieve it and the children of a plan are the subgoals of the plan, with G as the root. All of the subgoals of a plan must be executed in order for the plan to succeed ("AND"). However, it is not necessary that all plans of a goal be executed ("OR").

When the agent decides to pursue a goal, a goal-plan tree instance is created and updated at run-time to contain goal and plan instances. In addition to the above properties, goal and plan instances will also have a label (instance-name) that gives a unique handle to it at run-time. This is basically a path expression in a goal-plan tree instance (e.g. in figure 1 $G \rightarrow P \rightarrow SG3 \rightarrow P4$ is the instance-name for plan $P4$).

The execution cycle of the agent is similar to the well-known and developed BDI (Belief Desire Intention) [Rao and Georgeff, 1995] style of agents that map to agent implementation systems such as PRS, JAM, dMARS, and JACK, where a plan is selected from an applicable plan set and if it fails an alternative applicable plan is tried if available. We extend this model by requiring that a goal with an in-condition that is false is delayed until the in-condition becomes true[6].

3 Preparatory Effects and Dependency links

Typically, when developing a set of plans to achieve a goal, a developer is not able to consider all the ways in which the pursuit of other goals may interfere with this goal. There are two important ways in which parallel goals can interfere, which we try to capture and reason about in this work. Considering the logical outcomes of goals and plans, interference can occur:

1. When an in-condition is made false while a plan or goal is executing, causing the plan or goal to fail.
2. When a previously achieved effect is made false before a plan or goal that relies on it begins executing, preventing the goal or plan from being able to execute.

We thus define a preparatory-effect (*p-effect*) of a goal G as follows. Consider two plans P_1 and P_2 which are utilised to satisfy G and P_1 is executed before P_2 as shown in figure 1. If P_1 brings about an effect e which is a pre-condition of P_2, then there is a *dependency-link* between the *preparatory-effect* e and the *dependent-plan* P_2 ($e \rightarrow \{P_2\}$). We shall term this a *dependency-planlink*. In the event that an effect of a plan serves as a pre-condition to all the plans that satisfy a sub-goal (as is the case with effect e and sub-goal SG_3) then there is a dependency-link between the p-effect and the *dependent-subgoal* ($e \rightarrow \{SG_3\}$). We shall term this a *dependency-subgoallink*.

A dependency-planlink is *complete* when the dependent plan begins execution. A dependency-subgoallink is complete when either (a) the dependent-subgoal is complete or (b) the last possible plan to satisfy the sub-goal begins execution (all other plans have been tried but failed).

[1] This work was done independently from, but is quite similar to that of [Clement and Durfee, 1999b].

[2] Another type of goal is a "maintenance goal" where the agent ensures that a particular condition is never violated.

[3] Pre-conditions, in-conditions and effects are all represented as logical conditions.

[4] These effects are the direct, desired effects of the goal, similar to the notion of primary expectations of [Pollack, 1991].

[5] Formally, if i is the in-conditions then we consider the pre-conditions to be $i \wedge p$.

[6] Some of the systems will begin executing the goal and have it immediately fail.

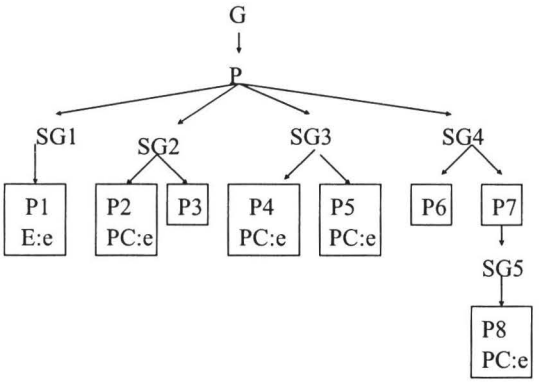

Figure 1: Dependency-links example

When a p-effect is achieved the agent should protect the effect from being undone until all the dependency-links associated with the effect are complete.

All dependency-planlinks are *potential* by definition because the agent has other means of achieving the dependent-plan's parent sub-goal.

A dependency-subgoallink is either

- *definite* if the dependent-subgoal will definitely be pursued as is the case with SG_3 (figure 1); or
- *potential* if the dependent-subgoal is not definite. SG_5 is an example of such a sub-goal as it depends on the path chosen for SG_4.

The above categorisations leads to a ranking of importance with respect to guarding a p-effect. A definite dependency-subgoallink is strongest because the dependent-subgoal will definitely be pursued and all ways of achieving it require the p-effect as a pre-condition. A potential dependency-subgoallink is stronger than a dependency-planlink because although they are both potential the dependent-subgoal of the dependency-subgoallink will definitely fail if the p-effect is not true. In contrast the parent sub-goal of the dependent-plan of a dependency-planlink has other means of being achieved. This ranking is useful for defining agents that are more or less careful with respect to protecting preparatory effects from interference.

Dependency-entry:
In order to reason about and protect p-effects we attach dependency information at each plan-node of a goal-plan tree. This is done at compile time[7] and each entry which we shall term a *dependency-entry* consists of a set of dependency-links, *(p-effect, dependent-plan/subgoal)*, where the p-effect is an effect of the plan. We attach dependency-entries only to plans because p-effects are achieved by plans, goals only declare what the plans will achieve.

4 Interaction Tree

In order to detect if goals can be executed in parallel without any interference from each other we define the notion of interaction summaries. When there is potential interference these interaction-summaries help us to schedule appropriately.

Interaction-summaries are similar to effect-summaries developed in [Thangarajah *et al.*, 2003] and based on the work of Clement and Durfee [Clement and Durfee, 1999b] on condition summaries. The effect-summaries of a goal summarise the effects that will definitely be achieved irrespective of the path chosen to satisfy the goal and those that may be achieved depending on the path chosen. These were used in [Thangarajah *et al.*, 2003] to detect situations where the plans of two goals that bring about the same effect could be merged and to facilitate this merging.

We use interaction summaries to ensure that the effects of one goal do not interfere with active in-conditions and dependency-links of other goals. Therefore we need to maintain summary information of (a) effects as they could cause the interference, and (b) in-conditions and preparatory-effects as these are what need to be protected when active. Only those pre-conditions that are set-up by p-effects are protected so we do not need to maintain pre-condition summaries separate from the information in the dependency-entries.

For the discussion ahead the general term *condition* refers to either in-conditions or effects (including p-effects). Similarly to [Thangarajah *et al.*, 2003], conditions can be either *definite* or *potential*. A **definite condition** will *definitely* be achieved/required (achieved for effects and required for in-conditions) at some point along every possible path of achieving the goal (i.e. this condition need not be required by all plans, but is required by at least one plan in every possible way of achieving the goal). A **potential condition** will *possibly* be achieved/required at some point in the pursuit of the goal (i.e. this condition is part of at least one plan in at least one path of achieving the goal but **not** achieved/required in all possible paths of achieving the goal). The set of potential conditions and definite conditions are exclusive of each other.

Interaction Summary:
An *interaction-summary* of a goal/plan contains the definite and potential in-conditions, effects and p-effects. Formally the interaction-summary of a goal/plan n is a 3 tuple.

$$I_{summary}(n) = \langle S_i(n), S_e(n), S_{pe}(n) \rangle$$

where S_i, S_e and S_{pe} are the summaries of in-conditions, effects and preparatory-effects respectively. The in-conditions summary contains the definite and potential in-conditions of the node. $S_i(n) = \langle D_i, P_i \rangle$ and each of D_i and P_i are simply sets of conditions, $D_i = \{ic_1, ic_2, \ldots, ic_n\}$. Similarly for the effects and p-effects summaries. We shall see how we can derive interaction-summaries of a goal/plan in section 4.1.

Interaction-Tree:
The *interaction-tree* structure is a goal-plan tree where attached at each node is the interaction-summary and in addition for plan-nodes the dependency-entry[8] of the plan. This tree is built at compile time for each goal-type and instantiated at run-time for each goal-instance. Figure 2 shows the interaction tree of a simple goal of a Mars rover agent to perform soil experiments at location A.

The interaction-tree for any active goal is updated dynamically as the goal is pursued. When a plan/sub-subgoal com-

[7] We do not describe the computation method in detail – the definitions above form a specifi cation that can be easily realised.

[8] as described at the end of section 3.

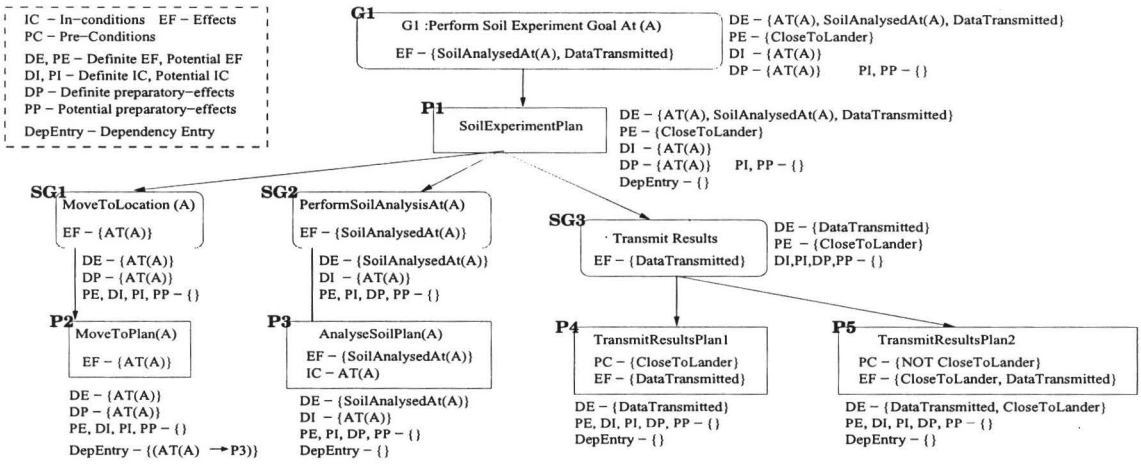

Figure 2: Interaction Tree example

pletes the corresponding node is removed and the interaction summary of the parent-node is re-computed. Any changes to the parent node are propagated up the tree. Dependency-entries are updated when plans (dependent-plans) begin execution.

4.1 Deriving Interaction Summaries

Interaction summaries are derived at compile time and updated at runtime, similarly to the mechanism used in [Thangarajah *et al.*, 2002b]. In this section we use the following notation: $G(P)$ is the set of subgoals of the plan P, and *in-cond*(n) is the in-conditions of n (with n being either a goal or a plan).

The interaction-summary of a node of a goal-plan tree is derived by combining the local conditions of that node with the interaction-summaries of all its child nodes. Each piece of the summary (the in-condition summary, the effect summary and the p-effect summary) is calculated by combining the relevant pieces of the local and children nodes. The interaction summaries of goal nodes and plan nodes are calculated slightly differently and the details are as follows.

The **effects summary** of a plan is computed by taking the union of the local effects of the plan with the effects of each of the goals within the plan, using the addition operator $\oplus$ defined as follows:

$$\langle D_1, P_1\rangle \oplus \langle D_2, P_2\rangle = \langle (D_1 \cup D_2), (P_1 \cup P_2)\rangle$$

where D and P are respectively sets of definite and potential conditions (either effects or in-conditions).

The effects summary of a plan is then:

$$S_e(P) = \langle \textit{effects-of}(P), \{\}\rangle \oplus \bigoplus_{g \in G(P)} S_e(g)$$

The **p-effect summaries** are analogous to effect summaries.

The **in-condition summary** of a plan is similar to the above and is defined as:

$$S_i(P) = \langle \textit{in-cond}(P), \{\}\rangle \oplus \bigoplus_{g \in G(P)} S_i(g)$$

The summaries of goals however need to take into account that the plans whose conditions are being combined, are alternative ways to achieve a goal. The agent will typically[9] only execute one of these alternative plans and thus the combined definite conditions are those that are definite for all plans, while the combined potential conditions are those potential for some plan plus those that are definite for some plan, but not for all. The conditions of plans are combined using a merge operator $\otimes$, defined as follows:

$$\langle D_1, P_1\rangle \otimes \langle D_2, P_2\rangle = \langle (D_1 \cap D_2), ((D_1 \cup D_2 - D_1 \cap D_2) \cup P_1 \cup P_2)\rangle$$

Goals do not have effects other than the effects of their plans, so the effect summary[10] of a goal is:

$$S_e(G) = \bigotimes_{p \in \textit{plans-of}(G)} S_e(p)$$

The in-condition summary of a goal is then defined as the merged in-conditions of its plan nodes, combined with the local in-conditions of the goal as follows:

$$S_i(G) = \langle \textit{in-cond}(G), \{\}\rangle \oplus \bigotimes_{p \in \textit{plans-of}(G)} S_i(p)$$

In order to determine the interaction-summary (IS) of a set of goals, the $\uplus$ operator is applied to all of the goals in the set as follows

$$IS(GoalSet) = \biguplus_{g \in GoalSet} IS(g)$$

where $IS(g_1) \uplus IS(g_2) =$
$\langle S_i(g_1) \oplus S_i(g_2), S_e(g_1) \oplus S_e(g_2), S_{pe}(g_1) \oplus S_{pe}(g_2)\rangle$.

5 Executing goals simultaneously

Having calculated interaction summaries, we then use these to detect and avoid potential interference. We will first look at ways of determining whether two goals will definitely not interfere with each other, in which case they can be pursued in parallel without any monitoring or scheduling of plans. Then we will discuss how goals that may have potential interference can be scheduled to avoid such conflicts whilst being pursued simultaneously.

As we have discussed in section 3, when two goals are executed in parallel, we track and protect the following two constraints: (**a**) the in-conditions of each goal and its active plan instances must not be violated and (**b**) the preparatory-effects (p-effects) that have been achieved must not be undone until the dependency-links are complete.

[9]More than one plan may be executed in the case of plan failure. This does not change the rationale for the combination operators.

[10]The p-effect summary is similar

We can determine that a new goal G_{new} will not interfere with an existing set of goals $GSet$ if all (i.e. both definite and potential) the effects of G_{new} are compatible with all the derived in-conditions and p-effects of $GSet$. Two conditions[11] are compatible if it is possible for them to be simultaneously true. The details depend on the specific representation of conditions.

5.1 Scheduling

If there is interference between two goals this can be avoided by pursuing the goals in sequence. However, this restriction is too strong since it requires that the agent never have more than one goal that is being pursued. A more reasonable restriction is to pursue the goals in parallel, but monitor these goals for the steps which cause conflict and schedule them such that they do not interfere with each other. We do this by *guarding* the in-conditions and dependency-links that are active when adopting new goals and executing new plans.

In order to do this we use the interaction-trees and we further require a data structure that we shall call the *GuardedSet*[12] (GS) which includes in-conditions and dependency-links that are currently active and need to be guarded by the agent. Dependency-links are obtained from the dependency-entry of the associated plan-node from the interaction-tree. In-conditions are placed together with the name of the associated goal or plan instance. The GuardedSet can be defined by the following simple grammar:

GuardedSet → *dependency-entry* InCondEntry**
dependency-entry → *p-effect PGName*
InCondEntry → *in-condition PGName*
PGName → *planInstanceName* | *goalInstanceName*

We avoid interference between goals by (i) protecting the in-conditions while a plan or goal is executing, and (ii) protecting the p-effects that are achieved by a plan from the effects of new plans until the relevant dependency-links complete. However, in doing this, we must be careful to avoid deadlocks. For example, imagine we have the following two goals:

G_1.incondition(x), G_1 → .. →subgoal1.effect(NOT y),
G_2.incondition(y), G_2 → .. → subgoal2.effect(NOT x).

Assume that the agent is pursuing G_1, but has not yet reached *subgoal1*. The GS contains x, as it is an in-condition of G_1. In considering G_2, if the agent checks only the conditions in the GS there appears to be no problem and G_2 could be pursued, placing y also in the GS. However when *subgoal1* and *subgoal2* are respectively reached, they would be suspended as they affect y and x respectively, both of which are in the GS, thus creating a deadlock. Consequently, before adopting a goal/plan, we look ahead at effects that can be achieved by existing goals (using their interaction summaries) and ensure they don't interfere with the in-conditions of the new goal/plan to be executed.

Interference can then be avoided between goals by incorporating the following steps into the agent execution cycle.

1. When the agent **begins the pursuit of a goal** the in-conditions of the goal (i.e. the direct in-conditions, not the derived in-conditions) are added to the GS.
2. When the agent **satisfies a goal** or the **goal fails** any in-conditions of the goal in the GS are removed. If the goal is part of a dependency-subgoallink in the GS this entry is also removed.
3. When a **plan begins execution**
   (a) In addition to placing the in-conditions into the GS, if the plan has any p-effects then the relevant dependency-entry of the p-effects are placed in the *GuardedSet*. The reason we do this at the *start* of the plan, instead of the end (when the p-effect is achieved) is because the plan step that achieves the effect is not known in advance; and the agent should not begin another plan that could possibly undo the p-effect while the plan is executing.
   (b) If the plan either completes a dependency-planlink or a dependency-subgoallink[13] that is protected in the GS, then delete that dependency entry.
4. When a **plan completes or fails** execution, the in-conditions of the plan that are in the GS are removed. Further, in the event of plan failure any dependency-entries associated with p-effects of the plan are removed from the GS.
5. **Before the agent adopts a goal** G_{new}, it should perform the following to avoid deadlocks. Check for each goal G that has an associated condition in the GS, whether the in-conditions of G_{new} are compatible with the **combined definite and potential effects** of G.

   If they are compatible then G_{new} can be pursued in parallel with G. Else check for compatibility with the **definite effects** of G.

   If they are *not* compatible then unless G_{new} completes before G they will conflict. Since we have no notion of time G_{new} should not be pursued in parallel with G, but should be re-considered when the incompatible effects of G have been achieved.

   Else the agent can either **(a)** choose to be cautious and not allow G_{new} to execute until the incompatible effects in G are achieved, or **(b)** since potential conditions depend on the path chosen, be optimistic and allow G_{new} to execute hoping that the incompatible effects (and resulting goal/plan failure) will be avoided.
6. **Before the agent executes a plan** P_{new}, it should check whether the effects of P_{new} are compatible with the conditions protected in the GS. If they are not compatible then the plan must wait until they are compatible. Also in order to prevent deadlocks the in-conditions and p-effects of the plan must be checked against the derived effects of existing goals that have a guarded condition as done before adopting a new goal.

Above we have given priority to the goal that is already being pursued as opposed to the new goal. This however is totally dependent on the priority mechanism of the application. If the new goal is more important then existing goals

[11] More generally, sets of conditions

[12] Actually, this is a pair of multisets.

[13] By being the last plan to be attempted.

can be preempted. However goal priorities and mechanisms for goal preemption are beyond the scope of this paper.

6 Conclusion

In this paper we argued that intelligent agents often have plans involving several steps, where the results of one step set up conditions for a later step to be successful. Intelligent agents typically pursue multiple goals in parallel, but if intelligent or rational, they should manage this parallelism in a way which avoids undoing of preparatory effects too early. They should also not do something intentionally which causes immediate failure of another currently executing plan/goal.

We defined a notion of preparatory effects and dependency links and described how summary information could be computed and then used to protect both preparatory effects (before a plan/goal executes) and in-conditions (while a plan/goal executes). These mechanisms allow an intelligent agent to pursue multiple goals in parallel without stupidly and unnecessarily acting in a way that causes failure of some goal or plan. These mechanisms capture the kind of reasoning that is done effortlessly by humans, but which is not yet evident in intelligent computer agents.

Our analysis and experimentation indicates that this additional reasoning is not computationally expensive. However we plan to also test the costs and benefits empirically in further work. We also plan to do work to clarify the relationship between this work and earlier work by Clement and Durfee et al.

Acknowledgements

We would like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant CO0106934. We would also like to thank Ed Durfee for bringing to our attention the similarities between this work and work previously published by his group.

References

[Boutilier and Brafman, 1997] Craig Boutilier and Ronen I. Brafman. Planning with concurrent interacting actions. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, 1997.

[Busetta *et al.*, 1998] 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.

[Clement and Durfee, 1999a] Bradley J. Clement and Edmund H. Durfee. Identifying and resolving conflicts among agents with hierarchical plans. In *AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities, AAAI Technical Report WS-99-12*, 1999.

[Clement and Durfee, 1999b] Bradley J. Clement and Edmund 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.

[Clement *et al.*, 2001] 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.

[d'Inverno *et al.*, 1998] 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*, 1998.

[Horty and Pollack, 2001] John F. Horty and Martha E. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127:199–220, 2001.

[Huber, 1999] Marcus 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.

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

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

[Rao and Georgeff, 1995] Anand S. Rao and Michael P. Georgeff. BDI-Agents from theory to practice. In *Proceedings of the International Conference on Multi-Agent Systems ICMAS-95*, San Francisco, USA, 1995.

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

[Thangarajah *et al.*, 2002a] John Thangarajah, Lin Padgham, and James Harland. Representation and reasoning for goals in BDI agents. In *Proceedings of the Twenty-Fifth Australasian Computer Science Conference (ACSC 2002)*, Melbourne, Australia, 2002.

[Thangarajah *et al.*, 2002b] John Thangarajah, Michael Winikoff, Lin Padgham, and Klaus Fischer. Avoiding resource conflicts in intelligent agents. In *Proceedings of the 15th European Conference on Artifical Intelligence 2002 (ECAI 2002)*, Lyon, France, 2002.

[Thangarajah *et al.*, 2003] John Thangarajah, Lin Padgham, and Michael Winikoff. Detecting and exploiting positive goal interaction in intelligent agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, Melbourne, Australia, 2003.

[Winikoff *et al.*, 2002] 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.