Declarative & Procedural Goals in Intelligent Agent Systems

Michael Winikoff
School of Computer Science
and Information Technology
RMIT University
Melbourne, Australia.
winikoff@cs.rmit.edu.au

Lin Padgham
School of Computer Science
and Information Technology
RMIT University
Melbourne, Australia.
linpa@cs.rmit.edu.au

James Harland
School of Computer Science
and Information Technology
RMIT University
Melbourne, Australia.
jah@cs.rmit.edu.au

John Thangarajah
School of Computer Science
and Information Technology
RMIT University
Melbourne, Australia.
johnt@cs.rmit.edu.au

Abstract

An important concept for intelligent agent systems is *goals*. Goals have two aspects: declarative (a description of the state sought), and procedural (a set of plans for achieving the goal). A declarative view of goals is necessary in order to reason about important properties of goals, while a procedural view of goals is necessary to ensure that goals can be achieved efficiently in dynamic environments. In this paper we propose a framework for goals which integrates both views. We discuss the requisite properties of goals and the link between the declarative and procedural aspects, then derive a formal semantics which has these properties. We present a high-level plan notation with goals and give its formal semantics. We then show how the use of declarative information permits reasoning (such as the detection and resolution of conflicts) to be performed on goals.

1 Introduction

Intelligent agents are an important technology. Based on foundational work in AI and Philosophy, agent technology has significant applications in a wide range of domains (Jennings and Wooldridge, 1998) and are considered by some to be a natural successor to object oriented programming (Jennings, 2001). Although there has been much debate on what constitutes an agent, and which features are important, the consensus is that an *intelligent agent* is situated, autonomous, reactive, pro-active, and social (Wooldridge, 1998).

A clearly central concept for pro-active agents is that of *goals* (Winikoff et al., 2001). Goals have two aspects: *declarative*, where a goal is a description s of the state of the world which is sought ($Env \models s$); and *procedural*, where a goal is a set of procedures P which are executed (in an attempt) to achieve the goal[1]. Although the declarative aspect is perhaps more natural, the procedural is important for realisable agents: an important property of goals is that the agent have the capability of bringing about the goal, for example having the goal to make it stop raining isn't sensible, since (presumably) the weather isn't under the control of the agent (Padgham and Lambrix, 2000). Providing explicitly a procedural aspect which specifies how the agent might bring about the desired goal is one way of ensuring that the agent doesn't adopt goals which it has no way of bringing about.

Intelligent agent implementation platforms (such as PRS (Ingrand et al., 1992), dMARS (d'Inverno et al., 1998), JAM (Huber, 1999), JACK (Busetta et al., 1998), 3APL (Hindriks et al., 1999), and ConGOLOG (Giacomo et al., 2000)) are intended for deployment in highly dynamic environments and as a result adopt the procedural view of goals in order to avoid lengthy deliberation. For example, systems in the BDI (Belief Desire Intention) family treat goals as events which trigger plans, and 3APL defines a goal as being simply a procedure.

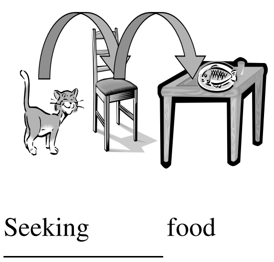
The use of the procedural aspect of goals is crucial to the practicality of these systems in highly dynamic environments. *However, by omitting the declarative aspect of goals the ability to reason about goals is lost.* Without knowing what a goal is trying to achieve, one cannot check whether the goal has been achieved, check whether the goal is impossible, or check for interference between goals (Thangarajah et al., 2002a). This lack of intelligence also constitutes a gap between BDI theories and implementations: an ideal BDI agent is required to drop goals when they are either achieved, or become unachievable (Rao and Georgeff, 1992). This cannot be done without declarative information. Further, declarative information allows the correct realisation of commitments to goals in BDI systems by decoupling plan failure from goal failure. A goal

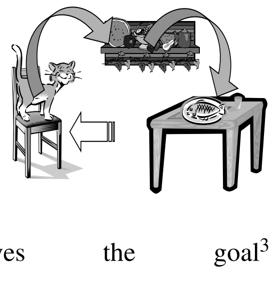
[1] There is often more than one means of achieving a goal.

should not be dropped merely because a plan to achieve it has failed. Declarative information can specify a condition for dropping the goal which is independent from plan failure. By controlling our selection of goal discharge condition we can realise different commitment strategies.

We seek to develop a representation of goals which allows for both declarative and procedural aspects to be specified and used. Our aim is to allow agent implementation platforms to be more faithful to their theoretical foundations, and to provide better handling of goals. In doing this, it is important that the execution semantics for goals is consistent with the desired properties of (declarative) goals. For example, given a goal to achieve condition s using a set of procedures (or recipes or plans) P, if s holds, then P should not be executed.

We now present a simple example which illustrates the use of goals and the importance of the properties discussed. Consider a hungry cat. The cat knows there is food on the table and formulates a plan to jump on a chair, and then jump from the chair to the table (left diagram below). The cat leaps on to the chair successfully. At this point, a nearby human moves the chair so that the cat can no longer leap to the table. The cat (being an intelligent agent[2]) realises this, and revises its plans: the new plan is to leap from the chair to a shelf, then walk along the shelf, and finally leap from the shelf down to the table (right diagram below). However, fortune smiles upon the feline: after leaping up to the shelf it finds that food has been left on the shelf. The opportunistic cat abandons its plan to continue to the table and eats upon the shelf instead. This scenario illustrates the importance of retrying upon the failure of a plan, and of detecting when a goal (reaching food) is fortuitously achieved and dropping the goal and the associated plan. Another important property is that goals are not dropped when an associated plan succeeds, unless the goal's condition is met. For example, suppose the nearby human had moved the food rather than the chair. Our cat would then successfully perform it's original plan and reach the table, only to find that there was no food on the table. Although the plan succeeded, the goal of reaching food has not been achieved.





Seeking food involves the goal[3] $\mathsf{Goal}(\text{atfood}, \textit{findfood}, \text{nofood})$ where *findfood* is a set of plans for locating food, planning a path to reach the food, and following the path and where $\text{atfood} \equiv \exists X.\text{foodat}(X) \wedge \text{location}(X)$ and $\text{nofood} \equiv \neg\exists X.\text{foodat}(X)$.

The correct behaviour of the cat relies on goal execution having certain desired properties. In the next section we explore what these properties are and how the execution mechanism for goals can be derived so that it possesses these properties. In section 3 we present a plan notation with goals and give its formal semantics. Section 4 discusses how interaction between goals (both positive and negative - i.e. conflicts) can be reasoned about and managed by using declarative information.

2 Representing Goals

What properties should goals have? In specifying a goal construct this question is of obvious importance and we shall begin by looking at desired properties of goals. After identifying the desired properties of goals, we then derive a procedural interpretation for goals which meets these desired properties.

The BDI work of Rao and Georgeff (Rao and Georgeff, 1992) requires that successful and impossible attitudes are dropped[4] (that is, goals must not already be achieved and they must be possible), that goals are known to the agent (axiom *AI*5 (Rao and Georgeff, 1991; Rao and Georgeff, 1998)) and that the set of goals held by an agent be consistent (Rao and Georgeff, 1992). Although persistence is left up to the commitment strategy, rather than being a property of the framework, the common commitment strategies require persistency of goals.

The formalisation of (van Linder et al., 1995) defines goals in terms of preferences. Preferences are persistent and known. Goals are selected from those preferences which are unfulfilled and realisable. Goals are also required to be consistent, so that where $\mathsf{G}A$ denotes that A is a goal, we have that $(\mathsf{G}A \Rightarrow \neg\mathsf{G}\neg A)$.

The GOAL language (Hindriks et al., 2001) requires that goals not be entailed by beliefs (i.e. that they be unachieved) and that goals be satisfiable. The set of goals $\mathcal{G}$ is *not* required to be consistent: $\mathcal{G}$ can contain both p and $\neg p$. This is handled by not requiring that all goals be achieved simultaneously, so that if $\mathcal{G} = \{p, \neg p\}$ then p can be achieved and discharged and $\neg p$ can be achieved at a later time. A consequence of this is that the goal $\mathsf{G}a \wedge b$ is not implied by the two goals $\mathsf{G}a$ and $\mathsf{G}b$ (since a and

[2] Or a robot: *www.necoro.com*

[3] The notation $\mathsf{Goal}(s, P, f)$ is read as "achieve condition s using the set of plans P; failing if f becomes true".

[4] The abstract BDI interpreter on page 441 includes the steps drop-successful-attitudes(B,G,I) and drop-impossible-attitudes(B,G,I) which correspond to an *open minded* commitment strategy.

b could be achieved at different times). Further, the goal operator does not distribute over implication.

Table 1: Properties of goals

	BDI	van Linder et al.	GOAL
Drop successful	✓	✓	✓
Drop impossible	✓	✓	✓
Known	✓	✓	
Consistent	✓	✓	
Persistent	(✓)	✓	

We thus require goals held by a rational agent to be *persistent*, *unachieved*, *possible*, *consistent*, and *known*:

Persistent: A goal should *only* be deleted when it succeeds or where there is a good reason for dropping it. A rational agent should not drop its goals without a good reason.

Unachieved: A goal to achieve s should be dropped when s is true. Note that this focuses on the desired outcome rather than the process - if we are in the middle of executing a plan P in order to bring about s and s becomes true, then the goal is dropped (with success) and the plan P aborted. A corollary is that a goal to achieve s where s already holds should trivially succeed without doing anything.

Possible: We specify a failure condition f which defines when a goal should be dropped with failure. In the same way that specifying s decouples the success of a goal from the success of its plans, specifying f decouples goal failure from plan failure. One advantage of allowing f to be specified is that it allows for a range of commitment strategies to be realised by specifying appropriate conditions for a goal to be dropped. We assume open minded commitment (Rao and Georgeff, 1992) as a default; in this case a goal is dropped with failure when it becomes *impossible* to achieve, i.e. f corresponds to an impossibility condition. We thus describe this desired property (goals are dropped with failure when f becomes true) as *possible*.

Consistent: A rational agent's goals are required to be *consistent*; an agent should not simultaneously pursue goals that conflict. For example, if an agent has a goal of eating the food on the table and another goal of eating food on the shelf at the same time, then the agent should recognise the conflict and resolve it in favour of the more important goal.

Known: Finally, a rational agent should know what goals it has; that is, goals should be *known*. This is an essential prerequisite to being able to reason about interactions between goals (see section 4).

Based on these properties we propose the construct $\mathsf{Goal}(s,P,f)$ which is read as "achieve s using P; failing if f becomes true" and which is viewed as being an executable statement in a plan language. Both s and f are logical formulae over about the agent's beliefs. We assume that s and f are exclusive, in that there is no world state S such that $S \models s \wedge f$. The procedural aspect, P, is a *set of guarded plans* of the form[5] $\llparenthesis C_1 : P_1, \ldots, C_n : P_n \rrparenthesis$ where each C_i is a condition and each P_i is a plan designed to achieve s. If the condition C_i is true, then P_i may be executed; if there is more than one executable plan (i.e. more than one condition is true) than a non-deterministic choice is made between them.

The guiding intuition is that given an initial state of the world S_0, pursuing a goal $\mathsf{Goal}(s,P,f)$ will result in a (possibly infinite) sequence of states $\mathcal{S} = \langle S_0, S_1, S_2, \ldots \rangle$. Note that this sequence of states, which includes environmental changes, cannot be wholly determined by $\mathsf{Goal}(s,P,f)$ and S. However, it is clearly constrained by $\mathsf{Goal}(s,P,f)$, and it is the nature of those constraints that we elicit.

In particular, we are concerned with ensuring that this sequence behaves appropriately with respect to s and f (i.e. the declarative aspects of goals) and in particular that the *persistence*, *unachieved* and *possible* properties are reflected this sequence. The precise relationship between these states is discussed in more detail in Section 3 where we give the operational semantics for goals.

We denote by $exec(\mathsf{Goal}(s,P,f),S)$ a sequence of states that results from executing $\mathsf{Goal}(s,P,f)$ in the state S. Note that there can be more than one such sequence (as P may contain a number of possible courses of action). However, in such cases an implementation will have to select a particular course of action to follow, and the properties discussed in the remainder of this section are independent of this choice.

Thus the *persistent*, *unachieved* and *possible* properties are captured as the following properties of the sequence of states:

- If there is a state S_n in $\mathcal{S}$ such that $S_n \models s \vee f$, then $\mathcal{S} = \langle S_0, S_1, \ldots S_n \rangle$ (i.e. $\mathcal{S}$ is finite and S_n is the final state in the sequence) and $\forall\, 0 \leq i \leq n-1$ we have $S_i \not\models s \vee f$.
- Otherwise, $\mathcal{S}$ is infinite (and hence $\forall i \geq 0$ we have $S_i \not\models s \vee f$).

The *persistent* property is captured by the sequence being infinite unless there is a state in which s or f is true. The *unachieved* and *possible* properties are captured by the requirement that if there is such a state in the sequence, then

[5] More generally, P can be any construct of a plan-language.

it is the final such state and neither s nor f is true in any previous state. In other words, the execution of a goal stops if (and only if!) either s or f become true.

The *known* condition corresponds to maintaining a data structure containing a record of the goals held by the agent; this is done in Section 3; consistency is discussed in Section 4.

We now define *exec*. This provides the link between the sequence discussed above and the plans P used to achieve the goal. As discussed above, the desired behaviour of *exec* is that if s or f hold in S then it should do nothing, otherwise it executes P. There are now three cases: (i) $s \vee f$ becomes true during the execution of P, (ii) $s \vee f$ is true when P completes executing (but not before), and (iii) $s \vee f$ remains false while P executes, and is false when P completes. In the first case, we abort the execution of P when $s \vee f$ becomes true and return the result of partially executing P (*pexec*); in the second case we return the result of executing P. In the third case we cannot complete the execution of $\mathsf{Goal}(s,P,f)$ since neither s nor f are true; we thus continue to execute $\mathsf{Goal}(s,P,f)$ in the situation resulting from P's execution.

Hence we denote by $pexec(X, \mathsf{Goal}(s,P,f), S)$ the sequence of states resulting from executing *no further than the completion of P*, but halting as soon as any condition in the set X becomes true (formally, $pexec(X,P,S)$ is a prefix of $exec(P,S)$); thus $pexec(X, \mathsf{Goal}(s,P,f), S)$ has three possible cases:

- $\langle S = S_0, S_1, S_2, \ldots S_n \rangle$ where $S_n \models s \vee f \vee \bigvee_{x \in X} x$ and $\forall\, 0 \leq i < n$ we have $S_i \not\models s \vee f \vee \bigvee_{x \in X} x$
- $\langle S = S_0, S_1, S_2, \ldots S_m \rangle$ where $\forall\, 0 \leq i \leq m$ we have $S_i \not\models s \vee f \vee \bigvee_{x \in X} x$ and P terminates in state S_m
- $\langle S = S_0, S_1, S_2, \ldots \rangle$ where $\forall\, i \geq 0$ we have $S_i \not\models s \vee f \vee \bigvee_{x \in X} x$

Note that in the third case, P does not terminate and there is no state in which $s \vee f$ becomes true.

We then define $exec(\mathsf{Goal}(s,P,f))$ as the result of repeatedly applying $pexec(P)$ until one of s or f becomes true, as below.

Definition 1 *We say a sequence of states* $\mathcal{S}$ *is* s,f-compatible *if either*

- $\forall S_i \in \mathcal{S} S_i \not\models s \vee f$
- $\mathcal{S} = \langle \mathcal{S}_{/}, \mathcal{S}_{\infty}, \ldots \mathcal{S}_{\backslash} \rangle$, $S_n \models s \vee f$ *and* $\forall 0 \leq i \leq n-1$ *we have* $S_i \not\models s \vee f$

We define $pexec(\mathsf{Goal}(s,P,f),S)$ *as a* maximal s,f-compatible *sequence of states that results from the execution of P commencing in state S.*

We define $exec(\mathsf{Goal}(s,P,f),S)$ *as*

- $\langle S \rangle$ *if* $S \models s \vee f$
- $pexec(\{s,f\},P,S)$ *if* $pexec(\{s,f\},P,S) \models s \vee f$ *or* $pexec(\{s,f\},P,S)$ *is infinite*
- $pexec(\{s,f\},P,S) \circ tl(exec(\mathsf{Goal}(s,P,f),S'))$ *otherwise, where* S' *is the final state of* $pexec(\{s,f\},P,S)$, *"$\circ$" denotes sequence concatenation, and* $tl(-)$ *is the sequence without its first element; this is needed since* S' *is also the first state of* $exec(\mathsf{Goal}(s,P,f),S')$.

We say that the goal $\mathsf{Goal}(s,P,f)$ succeeds *from S if* $exec(\mathsf{Goal}(s,P,f),S)$ *is finite and its final state* S_n *is such that* $S_n \models s$. *We say that the goal* $\mathsf{Goal}(s,P,f)$ fails *from S if* $exec(\mathsf{Goal}(s,P,f),S)$ *is finite and its final state* S_n *is such that* $S_n \models f$.

It is then straightforward to show the following result.

Proposition 1 *For any* S_i *in* $exec(\mathsf{Goal}(s,P,f),S)$ *we have that*

- *If* $S_i \not\models s \vee f$ *then there is a successor state* S_{i+1} *in* $exec(\mathsf{Goal}(s,P,f),S)$
- *Otherwise* S_i *is the final state in* $exec(\mathsf{Goal}(s,P,f),S)$

Thus the derived execution semantics for goals satisfies the *unachieved*, *possible* (via f), and *persistent* conditions. The *known* condition is addressed in Section 3 and *consistency* is discussed in Section 4.

3 Operational Semantics

We now present a detailed operational semantics for a high-level plan language incorporating as first-class citizens goals with both declarative and procedural aspects. The CAN[6] notation and its semantics clearly (and formally) illustrate how a possible plan language could be implemented. *Our reason for doing this is that we intend that the notion of goal developed be realisable in implemented agent systems*. The notation we present below is illustrative of the plan languages of typical agent languages, both

[6] CAN = **C**onceptual **A**gent **N**otation

in the BDI tradition, and elsewhere. In particular, it is similar to Rao's AgentSpeak(L) (Rao, 1996) and to Kinny's Ψ (Kinny, 2001), both of which attempt to extract the essence of a class of implemented BDI agent platforms.

3.1 The CAN Notation

An agent program (denoted by Π) consists of a collection of plan clauses of the form[7] $e : c_1 \leftarrow c_2 : P$ where e is an event, c_i are context conditions (logical formulae over the agent's beliefs) which must be true in order for the plan to applicable[8], and P is the plan body.

Beliefs (b) are first order terms but could be orthogonally extended to other logics. All that we assume are that we have operations to check whether a condition follows from a belief set ($B \models c$), to add a belief to a belief set ($B \cup \{b\}$), and to delete a belief from a belief set ($B \setminus \{b\}$). Traditionally, agent systems have implemented addition and deletion of beliefs as literal addition and deletion from a set of terms, however, there is no reason why belief revision could not be applied. A condition is a logical formula over belief terms:

$$C ::= b \mid C \wedge C \mid C \vee C \mid \neg C \mid \exists x.C$$

The plan body P is built up from the following constructs. We have the basic constants $true$, $fail$ and $ex(X)$ (exception), and the primitives act (an action not further specified), operations to add ($+b$) and delete ($-b$) beliefs, a test for a condition ($?c$), and events[9] $!e$. We also have a number of compounds: sequencing ($P_1;P_2$), parallelism ($P_1 \| P_2$), and goals ($\mathsf{Goal}(s,P,f)$).

In addition to these compounds there are also a number of auxilliary compound forms which are used internally when assigning semantics to constructs: for example when an event matches a set of guarded plans these are collected into a set of guarded alternatives ($(\!|B : P, \ldots|\!)$). The other auxilliary compound form is a choice operator dual to sequencing ($P_1 \triangleright P_2$), which executes P_1 and then executes P_2 only if P_1 failed. The language is described by the following grammar:

$$\begin{array}{lcl} P & ::= & true \mid fail \mid ex(X) \\ & \mid & act \mid +b \mid -b \mid ?c \mid !e \\ & \mid & P;P \mid P \parallel P \mid \mathsf{Goal}(s,P,f) \\ & \mid & P \triangleright P \mid (\!|B : P, \ldots, B : P|\!) \end{array}$$

[7]An omitted c_i is equivalent to $true$.

[8]More precisely, c_1 is an eager condition and c_2 is lazy; both must be true for the plan to be applied, but they are tested at different times: c_1 is tested when the event is matched against the plans, whereas c_2 is tested when the plan is being considered for execution.

[9]Where it is obvious that e is an event we shall sometimes elide the exclamation mark, in the interests of readability.

3.2 Formal Semantics

We assume that we are given operations to check whether a condition follows from a belief set ($B \models c$), to add a belief to a belief set ($B \cup \{b\}$), and to delete a belief from a belief set ($B \setminus \{b\}$). In the case of beliefs being a set of ground atoms these operations are respectively consequence checking, and set addition/deletion.

We define a basic configuration $S = \langle B, \mathcal{G}, P \rangle$ where B is the beliefs of the agent, $\mathcal{G}$ is a set of goals being pursued (used for consistency checking – see section 4), and P is the plan being executed. A transition $S_0 \longrightarrow S_1$ specifies that executing S_0 a single step yields S_1. We define $S_0 \longrightarrow^* S_n$ in the usual way: S_n is the result of one or more single step transitions. The transition relation is defined using rules of the form $\dfrac{}{S \longrightarrow S'}$ or of the form $\dfrac{S' \longrightarrow S_r}{S \longrightarrow S'_r}$; the latter are conditional with the top (numerator) being the premise and the bottom (denominator) being the conclusion. In order to make the presentation more readable we use the convention that where a component of S isn't mentioned it is the same in S and S' (and in S_r and S'_r). We also assume that B refers to the agent's beliefs, and elide angle brackets. Thus each of the left rules is shorthand for the corresponding right rule. The full set of rules are given in figure 1.

$$\frac{B \models c}{?c \longrightarrow true}\ ?c \qquad \frac{B \models c}{\langle B, \mathcal{G}, ?c \rangle \longrightarrow \langle B, \mathcal{G}, true \rangle}\ ?c$$

$$\frac{P_1 \longrightarrow P'}{P_1;P_2 \longrightarrow P';P_2}\ ; \qquad \frac{\langle B, \mathcal{G}, P_1 \rangle \longrightarrow \langle B', \mathcal{G}', P' \rangle}{\langle B, \mathcal{G}, P_1;P_2 \rangle \longrightarrow \langle B', \mathcal{G}', P';P_2 \rangle}\ ;$$

The first rule above specifies that the condition test $?c$ transitions to true if the condition c is a consequence of the agent's beliefs ($B \models c$). The second rule specifies that $P_1;P_2$ transitions to $P';P_2$ where P' is the result of a single execution step of P_1.

The operational semantics carry around a set of conditions being watched for ($\longrightarrow_X$). This is used by goals to interrupt the execution of a sub-plan should a condition become true. In order for this to work correctly we require that the *check* rule take precedence over other rules – it must be applied if it is applicable. We add rules to propagate exceptions, interrupting further processing ($;_x$, $\|_{x1}$, $\|_{x2}$, $\triangleright_x$, and G_{xi}).

The rules for executing goals are based on the definitions of *exec* and *pexec*. The rules G and G_{copy} add s and f to the conditions being watched for (X) and execute P. This relies on the *check* rule to stop executing if s or f hold, throwing an exception $ex(x)$ when this occurs. The exception is handled appropriately by one of the three G_{xi} rules. The rule G_{tf} continues attempting to achieve the goal in the case that executing P did not lead to s or f becoming true.

We extend simple configurations (which correspond to a single thread of execution within an agent) to agent configurations $S_A = \langle N, B, G, \mathbb{P}P \rangle$ which consist of a name, a single (shared) belief set, a goal set, and a set of executing plans. The following rule defines the operational semantics over agent configurations in terms of the operational semantics over simple configurations.

$$\frac{P \in \Gamma \quad \langle B, G, P \rangle \longrightarrow \langle B', G', P' \rangle}{\langle N, B, G, \Gamma \rangle \longrightarrow \langle N, B', G', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle} \ Agent$$

Note that there are elements of nondeterminism in CAN, such as the choice of plan to execute from a set of (multiple) applicable plans. In addition, the *Agent* rule nondeterministically selects an executing plan. In general, this choice should be constrained by the desired scheduling policy (such as a round-robin strategy to ensure fairness of some description).

One of the fundamental features of CAN is that it is a high-level plan language, in the spirit of process algebras such as the π-calculus and agent systems such as Ψ, rather than a programming language per se. This means that we can concentrate on the important issues, such as plan selection, event handing, belief updates, etc. rather than potentially cumbersome details such as data structures and mechanisms for passing data around. As a result, CAN is agnostic with respect to such issues.

Proposition 2 *Let $S^{(')}$ be approximated by $B^{(')}$ (i.e. $S \models c$ iff $B \models c$) and let $Q \in \{true, fail\}$ and $R \in \{true, fail, ex(x)\}$. Then $exec(P,S) = \langle \ldots, S' \rangle$ iff $\langle B, \mathcal{G}, P \rangle \longrightarrow^* \langle B', \mathcal{G}', Q \rangle$ and $pexec(X,P,S) = \langle \ldots, S' \rangle$ iff $\langle B, \mathcal{G}, P \rangle \longrightarrow_X{}^* \langle B', \mathcal{G}', R \rangle$.*

Proposition 3 *Let $S^{(')}$ be approximated by $B^{(')}$. The goal G succeeds (resp. fails) from state S iff $\langle B, \mathcal{G}, G \rangle \longrightarrow^* \langle B', \mathcal{G}', true \rangle$ (resp. if $\langle B, \mathcal{G}, G \rangle \longrightarrow^* \langle B', \mathcal{G}', fail \rangle$).*

These rules specify a precise and implementable operational semantics for a plan language including goals with both declarative and procedural aspects. The semantics of goal execution respect the desired properties of goals (namely *persistent*, *unachieved*, *possible*, and *known*; *consistency* is discussed in the next section). These rules have been translated into Prolog, yielding a prototype CAN interpreter.

3.3 An Example

Let us return to the cat example given in section 1. A set of plans suitable for sating hunger are given in figure 2. These plans include explicitly human intervention, as described in section 1.

We assume the following[10]:

$$\begin{array}{rcl}
G & = & \mathsf{Goal}(\text{atfood}, !\mathit{findfood}, \text{nofood}) \\
af & = & foodat(X) \wedge location(X) \\
nf & = & not(foodat(X)) \\
P & = & \mathit{findfood} \\
G_1 & = & \mathsf{Goal}_P(\text{atfood}, (\!|!survey_terrain; reachfood|\!), \\
 & & \text{nofood}) \\
rf & = & planpath(Y,X); ?path(A); followpath \\
G_2 & = & \mathsf{Goal}_P(\text{atfood}, !survey_terrain; reachfood \rhd (\!||\!), \\
 & & \text{nofood}) \\
G_3 & = & \mathsf{Goal}_P(\text{atfood}, true; reachfood \rhd (\!||\!), \text{nofood}) \\
G_4 & = & \mathsf{Goal}_P(\text{atfood}, reachfood \rhd (\!||\!), \text{nofood}) \\
G' & = & \mathsf{Goal}(\text{atfood}, \text{nofood}) \\
X & = & \{\text{atfood}, \text{nofood}\} \\
B_0 & = & \{location(floor), canjump(floor, chair), \\
 & & canjump(chair, table), canjump(ledge, table), \\
 & & canjump(chair, ledge)\} \\
B_1 & = & \{location(floor), foodat(table), \\
 & & canjump(floor, chair), canjump(chair, ledge), \\
 & & canjump(chair, table), canjump(ledge, table)\}
\end{array}$$

We consider the cat seeking food, i.e. executing the goal G. Note that the execution consists of multiple derivations, each advancing the agent a single execution step. The first step unfolds the goal, adding it to the goal set and noting that *nofood* and *atfood* are conditions that need to be watched, then posts the event *findfood* which matches a single plan which first surveys the terrain then (attempts to) reach food.

$$\cfrac{\cfrac{\cfrac{\overline{B_0 \models foodat(table) \wedge location(floor)}}{B_0, \{G'\}, !\mathit{findfood} \longrightarrow_X B_0, \{G'\}, (\!|t : !st; !rf|\!)} \ Ev}{B_0, \{G'\}, G_P \longrightarrow_\emptyset B_0, \{G'\}, G_1} \ \mathsf{G}}{B_0, \emptyset, G \longrightarrow_\emptyset B_0, \{G'\}, G_1} \ \mathsf{G}_{copy}$$

The second step selects the (only) plan.

$$\cfrac{\cfrac{\overline{B_0 \models true}}{B_0, \{G'\}, (\!|t : !st; !rf|\!) \longrightarrow_X B_0, \{G'\}, !st; !rf \rhd (\!||\!)} \ Sel}{B_0, \{G'\}, G_1 \longrightarrow_\emptyset B_0, \{G'\}, G_2} \ \mathsf{G}$$

The third step performs the first step of the plan, namely surveying the terrain[11]. We assume that this succeeds and

[10] Due to space limitations we use the following abbreviations; st: survey_terrain, rf: reachfood, nf: nofood, af: atfood, t: true, f: fail.

[11] We have compressed matching the event against a plan, selecting that plan, and executing it into a single step.

$$\frac{B \models c}{?c \longrightarrow \mathit{true}}\ ?c_t \qquad \frac{B \not\models c}{?c \longrightarrow \mathit{fail}}\ ?c_f \qquad \frac{}{B,+b \longrightarrow B \cup \{b\},\mathit{true}}\ +b \qquad \frac{}{B,-b \longrightarrow B \setminus \{b\},\mathit{true}}\ -b$$

$$\frac{}{a \longrightarrow \mathit{true}}\ \mathit{act} \qquad \frac{B \models x \in X}{P \longrightarrow_X \mathit{ex}(x)}\ \mathit{check}$$

$$\frac{(t_i{:}c_i{\leftarrow}b_i{:}P_i) \in \Pi \quad B \models c_i}{!e \longrightarrow (\!|b_1{:}P_1,\ldots,b_n{:}P_n|\!)}\ \mathit{Ev} \qquad \frac{\neg\exists b_i{:}P_i \in \Delta. B \models b_i}{(\!|\Delta|\!) \longrightarrow \mathit{fail}}\ \mathit{Sel}_f \qquad \frac{b_i{:}P_i \in \Delta \quad B \models b_i}{(\!|\Delta|\!) \longrightarrow P_i \triangleright (\!|\Delta \setminus \{b_i{:}P_i\}|\!)}\ \mathit{Sel}$$

$$\frac{P_1 \longrightarrow P'}{P_1;P_2 \longrightarrow P';P_2}\ ; \qquad \frac{}{\mathit{true};P \longrightarrow P}\ ;_t \qquad \frac{}{\mathit{fail};P \longrightarrow \mathit{fail}}\ ;_f \qquad \frac{}{\mathit{ex}(X);P \longrightarrow \mathit{ex}(X)}\ ;_x$$

$$\frac{P_1 \longrightarrow P'}{P_1\|P_2 \longrightarrow P'\|P_2}\ \|_1 \qquad \frac{P_2 \longrightarrow P'}{P_1\|P_2 \longrightarrow P_1\|P'}\ \|_2 \qquad \frac{}{\mathit{true}\|P \longrightarrow P}\ \|_{t1} \qquad \frac{}{P\|\mathit{true} \longrightarrow P}\ \|_{t2}$$

$$\frac{}{\mathit{fail}\|P \longrightarrow \mathit{fail}}\ \|_{f1} \qquad \frac{}{P\|\mathit{fail} \longrightarrow \mathit{fail}}\ \|_{f2} \qquad \frac{}{\mathit{ex}(X)\|P \longrightarrow \mathit{ex}(X)}\ \|_{x1} \qquad \frac{}{P\|\mathit{ex}(X) \longrightarrow \mathit{ex}(X)}\ \|_{x2}$$

$$\frac{P_1 \longrightarrow P'}{P_1 \triangleright P_2 \longrightarrow P' \triangleright P_2}\ \triangleright \qquad \frac{}{\mathit{true} \triangleright P \longrightarrow \mathit{true}}\ \triangleright_t \qquad \frac{}{\mathit{fail} \triangleright P \longrightarrow P}\ \triangleright_f \qquad \frac{}{\mathit{ex}(X) \triangleright P \longrightarrow \mathit{ex}(X)}\ \triangleright_x$$

$$\frac{G \cup \{\mathsf{Goal}(s,f)\}, \mathsf{Goal}_P(s,P,f) \longrightarrow G',P'}{G, \mathsf{Goal}(s,P,f) \longrightarrow G',P'}\ \mathsf{G}_{copy} \qquad \frac{P \longrightarrow_{X \cup \{s,f\}} P'}{\mathsf{Goal}_P(s,P,f) \longrightarrow_X \mathsf{Goal}_P(s,P',f)}\ \mathsf{G}$$

$$\frac{Q \in \{\mathit{fail},\mathit{true}\}}{\mathsf{Goal}_P(s,Q,f) \longrightarrow \mathsf{Goal}_P(s,P,f)}\ \mathsf{G}_{tf} \qquad \frac{}{G, \mathsf{Goal}_P(s,\mathit{ex}(s),f) \longrightarrow G \setminus \{\mathsf{Goal}(s,f)\}, \mathit{true}}\ \mathsf{G}_{x1}$$

$$\frac{}{G, \mathsf{Goal}_P(s,\mathit{ex}(f),f) \longrightarrow G \setminus \{\mathsf{Goal}(s,f)\}, \mathit{fail}}\ \mathsf{G}_{x2} \qquad \frac{y \notin \{s,f\}}{G, \mathsf{Goal}_P(s,\mathit{ex}(y),f) \longrightarrow G \setminus \{\mathsf{Goal}(s,f)\}, \mathit{ex}(y)}\ \mathsf{G}_{x3}$$

$$\frac{P \in \Gamma \quad \langle B,G,P\rangle \longrightarrow \langle B',G',P'\rangle}{\langle N,B,G,\Gamma\rangle \longrightarrow \langle N,B',G',(\Gamma \setminus \{P\}) \cup \{P'\}\rangle}\ \mathit{Agent}$$

Note that in rules such as *act* we assume that actions always succeed. In order to take potential failures into account, it is not difficult to modify such rules to explicitly check a given success (or failure) condition. Note also that the *Ev* rule does not allow for multiple solutions of context conditions, modifying it to allow for multiple solutions is straightforward. Likewise, adding explicit substitutions is not hard.

Figure 1: Operational Semantics

```
human_intervention ← -canjump(chair,table) ; +canjump(chair,ledge).
becomehungry ← +hungry ; goal(not(hungry), satehunger, fail).
satehunger ← reachfood ; eatfood.
eatfood : foodat(X) ∧ location(X) ← act(eatfood) ; -foodat(X) ; sated.
sated : hungry ← -hungry.
sated ← true.
reachfood ← goal(foodat(X) ∧ location(X),findfood, not(foodat(X))).
findfood : foodat(X) ∧ location(Y) ← survey_terrain ; planpath(Y,X) ; ?path(A) ; followpath.
survey_terrain ← act(survey_terrain).
planpath(X,Y): plan(P) ← -plan(P) ; planpath(X,Y).
planpath(X,Y): canjump(X,Y) ← +path([X,Y]).
planpath(X,Y): canjump(Y,X) ← +path([X,Y]).
planpath(X,Y): canjump(X,Z) ∧ canjump(Z,Y) ← +path([X,Z,Y]).
planpath(X,Y): canjump(X,Z) ∧ canjump(Z,Q) ∧ canjump(Q,Y) ← +path([X,Z,Q,Y]).
followpath: path([P1]) ← -path([P1]).
followpath: path([P1,P2|Ps]) ← jump(P1,P2) ; -path([P1,P2|Ps]) ; +path([P2|Ps]) ; followpath.
jump(floor,chair): location(floor) ∧ canjump(floor,chair)
        ← act(jump(floor,chair)) ; -location(floor) ; +location(chair) ; human_intervention.
jump(P1,P2): location(P1) ∧ (canjump(P1,P2) ∨ canjump(P2,P1)) ← act(jump(P1,P2)) ; -location(P1) ; +location(P2).
```

Figure 2: Sample Plans for the Cat

updates the agent's beliefs from B_0 to B_1.

$$\dfrac{\dfrac{\dfrac{\vdots}{B_0,\{G'\},!st \longrightarrow_X B_1,\{G'\},t}}{B_0,\{G'\},!st;!rf \longrightarrow_X B_1,\{G'\},t;!rf}\;;}{\dfrac{B_0,\{G'\},!st;!rf \rhd (\!|\,|\!) \longrightarrow_X B_1,\{G'\},!st;!rf \rhd (\!|\,|\!)}{B_0,\{G'\},G_2 \longrightarrow_\emptyset B_1,\{G'\},G_3}\;\mathsf{G}}\;\rhd$$

The fourth step simply removes the *true*.

$$\dfrac{\dfrac{\dfrac{}{B_1,\{G'\},t;!rf \longrightarrow_X B_1,\{G'\},!rf}\;;_t}{B_1,\{G'\},t;!rf \rhd (\!|\,|\!) \longrightarrow_X B_1,\{G'\},!rf \rhd (\!|\,|\!)}\;\rhd}{B_1,\{G'\},G_3 \longrightarrow_\emptyset B_1,\{G'\},G_4}\;\mathsf{G}$$

The execution continues by:

- planning a path from the floor to the table (via the chair),
- jumping onto the chair
- having the human move the chair
- realising that the next step in the plan (jumping from the chair to the table) cannot be done
- surveying the terrain
- planning a path from the chair to the table (via the ledge)
- jumping to the ledge, jumping to the table, and then eating[12].

[12] In this run there isn't any food on the ledge

4 Reasoning about Goals

When an agent has more than one goal to pursue, there are a number of ways in which the pursuit of these goals can fail to be independent. At one extreme an agent has as goals both A and $\neg A$, here it is clearly irrational for the agent to attempt to satisfy both goals (or at least both goals simultaneously). It is also possible for goals to be logically consistent but not simultaneously achievable (for example they might require the same resources). A third possibility is that only some of the plans for each goal conflict, and hence it is possible to achieve both goals simultaneously by an appropriate choice of plan. For example, consider a goal to eat food (and there is food on the table and on a shelf) and a goal to scratch the table (an obviously naughty cat). These two goals can be simultaneously pursued and achieved (e.g. jumping onto the table and eating the food on it while scratching it), but some choice of plan (e.g. planning to eat the food on the shelf) will create a conflict between the goals[13]. In some instances it may also be appropriate to show that pursuit of a given set of goals can be achieved independently of each other — in other words, that there is no interference at all, and hence the plans to achieve each goal can be freely interleaved. Finally, we may wish to determine instances of positive interference, i.e. situations in which the achievement of one goal can

[13] Of course, it is possible that all plans to achieve the goals conflict. We are currently working on solutions to such instances by identifying such conflicts via summary information based on the work of Clement and Durfee (Clement and Durfee, 1999b; Clement and Durfee, 1999a).

assist in the achievement of other goals.

In this section we discuss various possibilities for performing these kinds of reasoning in our framework. For space reasons, this will necessarily be illustrative of the possibilities rather than a detailed prescription.

Our framework for goals simplifies the development of this kind of reasoning, such as being able to identify sub-goals $sg^n(G)$ which *necessarily* appear in the pursuit of G and sub-goals $sg^p(G)$ which *possibly* so appear. The definition for $sg^n(G)$ is thus

$$\begin{aligned} sg^n(P_1;P_2) &= sg^n(P_1) \cup sg^n(P_2) \\ sg^n(\llparenthesis B_1 : P_1, \ldots, B_n : P_n \rrparenthesis) &= \bigcap_{1 \le i \le n} sg^n(P_i) \\ sg^n(\mathsf{Goal}(s,P,f)) &= \{\mathsf{Goal}(s,P,f)\} \cup sg^n(P) \\ sg^n(a) &= \emptyset \end{aligned}$$

and similarly for $sg^p(G)$ with $\bigcap$ replaced with $\bigcup$. Note that $sg^n(G) \subseteq sg^p(G)$.

In general an agent will have a set of goals $\mathcal{G}$ and it wants to ensure that the addition of a given goal G does not conflict with $\mathcal{G}$. We say that G and $\mathcal{G} = \{G_1, \ldots, G_n\}$ are *necessarily consistent* iff all possible subgoals of G and all possible subgoals of all G_i in $\mathcal{G}$ do not conflict, i.e. $\forall g \in sg^p(G). \forall G_i \in \mathcal{G}. \forall g_i \in sg^p(G_i). g \natural g_i$, where $G_1 \natural G_2$ indicates that the s and f conditions for G_1 and G_2 are compatible[14]. Similarly, we say that G and $\mathcal{G} = \{G_1, \ldots, G_n\}$ are *necessarily inconsistent* iff some necessary subgoal of G and some necessary subgoal of any G_i in $\mathcal{G}$ conflict, i.e. $\exists g \in sg^n(G). \exists G_i \in \mathcal{G}. \exists g_i \in sg^n(G_i). \neg(g \natural g_i)$.

The definition of necessarily (in)consistent only considers *goal compatibility* in detecting conflict between a new goal G and the existing set of goals $\mathcal{G}$. However, we also need to consider the resource requirements of goals and ensure that there are no conflicts with respect to the resource requirements of G and that of $\mathcal{G}$.

We can define notions of necessary and possible resources as we did for sub-goals, but there are some differences in the way we deal with goal compatibility and *resource compatibility*. In goal compatibility we compare each sub-goal of G with every sub-goal of every goal G_n in $\mathcal{G}$ and this is sufficient. However this would not suffice for resource compatibility. For example assume that there are 50 units of the resource *energy* available, G requires 20 units of *energy*, and G_n has two *necessary* sub-goals SG_1 and SG_2 that require 25 units of energy each. G is compatible with SG_1 and SG_2 individually (since $20+25 \le 50$) but not with G_n because G_n requires the combined resources of SG_1 and SG_2 (i.e. $20+(25+25) \nleq 50$). Therefore it is necessary to combine the resource requirements of the sub-goals of a goal in order to check for resource compatibility.

[14] For example, that achieving one goal does not cause the other goal to fail.

In order to combine resource summaries we need to identify the different types of resources. Similar to sub-goals resources can also be classified as either *necessary resources* or *possible resources*. Some resources are no longer available after use (*consumable resources*, e.g. food), whilst others are still available after use (*reusable resources*, e.g. chair). The manner in which we derive resources summaries for a goal depends on the classification of the resource and the way in which the sub-goals are combined (i.e. whether they are sequential sub-goals or parallel sub-goals). Due to space limitation we cannot provide formal definitions and algorithms for deriving and using resource requirement summaries for goals, however we have addressed them in other work (Thangarajah et al., 2002b) which we will discuss in section 5.

If G and $\mathcal{G}$ are neither necessarily consistent nor necessarily inconsistent then they are *possibly consistent* (or, for that matter, *possibly inconsistent*). In the case where two goals are necessarily consistent we can achieve them concurrently. If they are necessarily inconsistent then we need to choose between them (Thangarajah et al., 2002a). *If they are possibly inconsistent then we need to choose consistent means of achieving these goals.*

Our framework also provides a suitable foundation for reasoning about positively interacting goals. We say that two goals G_1 and G_2 *necessarily support* each other if there is common necessary subgoal, i.e. $sg^n(G_1) \cap sg^n(G_2) \neq \emptyset$. For example, driving to the beach and driving to the garage might both have the necessary subgoal of getting more petrol ("Gas" in American English). However, this requires that any plans chosen must generate the *same* single sub-goal, which appears to be too strong, where it seems more natural to require that for any plan choice, a common sub-goal, possibly depending on the plan choice made, can be made to arise. We thus define a weaker condition and say that two goals G_1 and G_2 *possibly support* each other ($G_1 \rightleftharpoons G_2$) if they either necessarily support each other or if (i) given a choice of P_i from the set of plans $\{P_1, \ldots, P_n\}$ associated with G_1 there exists a choice of Q_j from the set of plans $\{Q_1, \ldots, Q_m\}$ associated with G_2, such that $P_i \rightleftharpoons Q_j$; and (ii) given a choice of Q_j there exists a choice of P_i such that $P_i \rightleftharpoons Q_j$.

Note that it is entirely possible for both positive and negative interactions to occur simultaneously. Consider the following example (based on (Horty and Pollack, 2001)): we need to travel to the airport to catch a 4pm flight and are considering catching either a taxi (expensive but faster) or a bus (cheaper but slower). We need to decide whether to adopt the goal of attending a meeting at 2:30pm. Assuming the meeting is likely to run late and is being held at the university this would constrain us to catch a taxi and thus attending the meeting is seen as less desirable *in con-*

text. If the meeting were to be held at 3:30 at the university then it would prevent us from reaching the airport in time, which makes attending the meeting impossible (assuming the flight is more important than the meeting). On the other hand, if the meeting were held at the airport, then, since we are already intending to travel to the airport, the cost of attending the meeting is *lower* than it would be, and hence in this context, the meeting is seen as possible and desirable. This example illustrates the sorts of reasoning that we want to be able to perform, namely assessing how goals can both hinder and help other goals.

5 Discussion

We have proposed an explicit goal construct for agent systems, based on the desired (declarative) properties of goals, and we have given an operational interpretation, via the rules of Section 3, which realises these properties. We have also discussed how reasoning about interactions between goals, both positive and negative, can be performed within our goal framework. We anticipate that this work will form the basis of significant development of agent systems with explicit representation of goals, including those based on the popular BDI model, thus reducing the gap between theory and practice, and enhancing the intelligent capabilities of such systems.

Due to space limitations we did not provide details on how to derive resource summaries and the details of how they can be used to detect conflict between goals with respect to resource requirements. We have however addressed this in (Thangarajah et al., 2002b). In that work 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 performing dynamic updates of resource requirements; we also discuss how conflict can be resolved.

Although the operational behaviour of goals can be realised in BDI systems (using maintenance conditions), we believe that goals are a sufficiently key concept for intelligent agents that they deserve to be represented directly. Furthermore, the declarative aspects are important and by representing goals we enable reasoning about interactions (both positive and negative) to be done.

5.1 Related Work

AgentSpeak (Rao, 1996) and Ψ (Kinny, 2001) both capture in a formal way the semantics of a plan language interpreter. They focus on capturing the essence of current practice and as a result capture the weaknesses of (current) BDI systems including the lack of declarative goals and associated problems.

The GOAL language (Hindriks et al., 2001) uses declarative goals but lacks a sufficiently powerful notion of procedural goals: plans cannot use sequences and are limited to being reactive.

Horty and Pollack (Horty and Pollack, 2001) formalise the reasoning process for positively interacting (i.e. assisting) goals by defining a notion of compatible plans and of the merging of (compatible) plans. The cost of a plan $\mathcal{P}$ *in context* $\mathcal{C}$ (where the two are compatible) is the difference between the cost of the context merged with the plan and the cost of the context on its own: $\kappa(\mathcal{P}/\mathcal{C}) = \kappa(\mathcal{P} \cup \mathcal{C}) - \kappa(\mathcal{C})$. Their work only addresses positive interactions and assumes that plans can be (at least roughly) simulated and that the cost of executing a plan can be at least approximated[15]. By comparison, we reason directly about conflicts between goals and address both negative and positive interactions.

There is a considerable amount of work on conflict in agent systems which addresses many different types of conflicts at various levels (see for example (Tessier et al., 2000)). By contrast, the main contribution of this paper isn't ways of dealing with conflict, but rather a realisation of goals in BDI-like systems. This realisation provides an implementable *foundation* upon which conflict resolution can be built, and alternative approaches compared.

5.2 Future Work

There is a wide range of directions for future work including: reasoning about plans; looking at attaching declarative information to program elements other than goals (in particular to plans); investigating the use of other forms of declarative assertions in addition to postconditions (e.g. preconditions, in-conditions (Clement and Durfee, 1999b; Clement and Durfee, 1999a)); investigating how $\mathsf{Goal}(s, P, f)$ allows for the derivation of potential plans P' from s (and f) should the given plan P fail in achieving s; extending to richer forms of goals including temporal logic and resources; and investigating the use of declarative aspects of goals in debugging agent systems.

We have addressed the issue of dealing with resource requirements in detecting conflicts between goals. Future work includes extending this concept to handle positive interaction (co-operation) between goals with respect to resource requirements (e.g. there is a positive interaction between two goals if one goal renews a consumable resource that is necessary for the other goal).

[15]For practical reasons they do not compute the precise cost, but rather maintain an approximation interval which contains the true cost. In some cases decisions can be made based on the interval without having to know the precise cost.

Acknowledgements

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

References

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

Clement, B. J. and Durfee, E. H. (1999a). Identifying and resolving conflicts among agents with hierarchical plans. In *AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities, Technical Report WS-99-12*. Available from *http://ai.eecs.umich.edu/people/bradc/*.

Clement, B. J. and Durfee, E. H. (1999b). Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 495–502. Available from *http://ai.eecs.umich.edu/people/bradc/*. A longer technical report version is also available.

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

Giacomo, G. D., Lespérance, Y., and Levesque, H. (2000). ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169.

Hindriks, K., de Boer, F., van der Hoek, W., and Meyer, J.-J. (2001). Agent programming with declarative goals. In *Intelligent Agents VI - Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000)*. Springer Verlag.

Hindriks, K. V., Boer, F. S. D., der Hoek, W. V., and Meyer, J.-J. C. (1999). Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401.

Horty, J. and Pollack, M. (2001). Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127:199–220.

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

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

Jennings, N. and Wooldridge, M. (1998). Applications of intelligent agents. In Jennings, N. R. and Wooldridge, M. J., editors, *Agent Technology: Foundations, Applications, and Markets*, chapter 1, pages 3–28. Springer.

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41.

Kinny, D. (2001). The psi calculus: an algebraic agent language. In *Agent Theories, Architectures, and Languages (ATAL-2001)*.

Padgham, L. and Lambrix, P. (2000). Agent capabilities: Extending BDI theory. In *Proceedings of Seventeenth National Conference on Artificial Intelligence - AAAI 2000*, pages 68–73.

Rao, A. and Georgeff, M. (1998). Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):292–342.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In de Velde, W. V. and Perrame, J., editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, pages 42–55. Springer Verlag. LNAI, Volume 1038.

Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-Architecture. In Allen, J., Fikes, R., and Sandewall, E., editors, *Principles of Knowledge Representation and Reasoning, Proceedings of the Second International Conference*, pages 473–484.

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

Tessier, C., Chaudron, L., and Müller, H.-J., editors (2000). *Conflicting Agents: Conflict Management in Multi-Agent Systems*. Kluwer Academic Publishers. ISBN 0-7923-7210-7.

Thangarajah, J., Padgham, L., and Harland, J. (2002a). Representation and reasoning for goals in BDI agents. In *Australasian Computer Science Conference*.

Thangarajah, J., Winikoff, M., Padgham, L., and Fischer, K. (2002b). Avoiding resource conflicts in BDI agents. Submitted to the European Conference on Artificail Intelligence (ECAI 2002).

van Linder, B., van der Hoek, W., and Meyer, J.-J. (1995). Formalising motivational attitudes of agents: On preferences, goals and commitments. In Wooldridge, M., Müller, J., and Tambe, M., editors, *Intelligent Agents II: Agent Theories, Architectures, and Languages, IJCAI'95 Workshop (ATAL)*, pages 17–32. Springer, LNAI 1037.

Winikoff, M., Padgham, L., and Harland, J. (2001). Simplifying the development of intelligent agents. In Stumptner, M., Corbett, D., and Brooks, M., editors, *AI2001: Advances in Artificial Intelligence. 14th Australian Joint Conference on Artificial Intelligence*, pages 555–568. Springer, LNAI 2256.

Wooldridge, M. (1998). Agent-based computing. *Interoperable Communication Networks*, 1(1):71–97.