Programming Agent Deliberation

An Approach Illustrated Using the 3APL Language

Mehdi Dastani Frank de Boer Frank Dignum John-Jules Meyer
Institute of Information and Computing Sciences
Utrecht University
The Netherlands
{ mehdi , frankb , dignum , jj }@cs.uu.nl

ABSTRACT

This paper presents the specification of a programming language for implementing the deliberation cycle of cognitive agents. The mental attitudes of cognitive agents are assumed to be represented in an object language. The implementation language for the deliberation cycle is considered as a meta-language the terms of which denote formulae from the object language. Without losing generality, we use the agent programming language 3APL as the object language. Using the meta-deliberation language, one can program the deliberation process of a cognitive agent. We discuss a set of programming constructs that can be used to program various aspects of the deliberation cycle including the planning constructs.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multi-agent systems*; D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*Syntax, Semantics*

General Terms

Design, Languages, Theory

Keywords

Agent Programming, Agent Deliberation, Planning

1. INTRODUCTION

For many realistic applications, a cognitive agent should have both reactive and deliberative behavior [6, 12]. The first type of behavior concerns the recognition of emergency situations in time and provides rapid responses whereas the second type of behavior concerns the planning of actions to achieve its long term goals. In order to implement an agent's deliberative behavior, its mental attitudes such as goals and beliefs as well as the interaction between these attitudes should be implemented. Issues related to the implementation of agents' mental attitudes can be considered as object-level concerns while issues related to the implementation of the interaction between mental attitudes form meta-level concerns.

To illustrate these levels, consider the scenario [3] where a cognitive mobile robot has a goal to transport boxes in an office like environment as illustrated in figure 1. In particular, the goal is to transport boxes that are placed at rooms 2,3, and 4 to room 1. The beliefs of the robot include

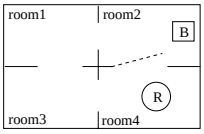

Figure 1: ***An example of a mobile robot environment.***

world knowledge such as the existence of a door between two rooms, perceptual information such as the position of itself and the boxes, the task knowledge such as how to transport a box, the gain of transporting a box, and the cost of the transportation. In this example, the gain of transporting a box is assumed to be 5, the cost of moving between rooms 2 and 4 is 5, and the cost of moving between other rooms is 1. However, the robot aims to transport a box if its gain is more that its costs. Therefore, the robot should deliberate on different transportation routes (plans) and transport a box if there is a route with higher gain than loss. The goals and the beliefs of the robot form its object level concerns while planning form its meta-level concerns.

Agent deliberation is not limited to the planning of the tasks, but it includes various types of decisions at each moment of time [10] such as how to select a goal from a set of possible goals, whether revising a plan or executing it, or behaving more reactively or more deliberatively. These types of decisions, which constitute the agent's deliberation process and determine the types of agent behavior, are usually implemented statically in an interpreter and are not explicitly and directly programmable in an agent module. For example, the decision on goal selection is usually based on an assumed predefined ordering on goals such that the interpreter always selects the goal which has the highest rank. To illustrate the problems related to this issue, imagine that the robot in Figure 1 has an additional goal to clean the rooms and that the decision to select a goal (transport or clean)

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

depends on the agent's belief on a particular situation rather than a predefined ordering. Of course, it may still be possible to implement some of these decisions implicitly in the agent's mental attitudes (object level) despite the way the interpreter is implemented. For example, the decision on goal selection can be implemented by making goals conditional on beliefs, but as we will argue, this type of approach shifts the problem rather than solves it. Moreover, we believe that doing so violates the basic programming principle called separation of concerns. Therefore, we aim to distinguish the object-level concerns, which are related to the mental attitudes of agents, from the meta-level concerns, which are related to the agent's deliberation process.

In this paper, we present the specification of a programming language to implement the deliberation cycle of cognitive agents. Although this programming language is designed to implement the deliberation cycle of 3APL agents [9], it can be used for other types of cognitive agents by some modifications related to how mental attitudes are represented. The 3APL programming language provides a set of programming constructs to implement agents' mental attitudes such as its beliefs, goals, basic actions, and planning rules. The aim of this paper is to extend this language with a second set of programming constructs by means of which the deliberation cycle becomes programmable. This extension should make 3APL a programming language for cognitive agents that respects the separation of concerns principle.

This paper is organized as follows. In section 2, we present a cognitive agent architecture for which we develop a programming language for its deliberation cycle. In section 3, we briefly explain the syntax and semantics of the 3APL programming language which can be used to implement mental attitudes of cognitive agents. An example of a 3APL program is proposed to explain the behavior of the transport robot and to illustrate the problem of deliberation. In section 4, we propose the meta-language that provides programming constructs to implement the deliberation cycle of cognitive agents. In section 5, the semantics of the programming language for the deliberation cycle is presented. Finally, in section 6, we conclude the paper.

2. AN ARCHITECTURE FOR COGNITIVE AGENTS

The cognitive agent that we consider has a mental state consisting of beliefs, goals, basic actions, reasoning rules, and an interpreter. The beliefs represent the agent's general world knowledge as well as its knowledge about the surrounding environment. In contrast to the BDI tradition [11], where goals are to-be-goals and thus considered as states to be reached by the agent, the goals here are to-do-goals that can be considered as processes to be executed by the agent. The basic actions are the actions that the agent can perform. These actions may be cognitive actions such as belief updates or external actions such as moving in a certain direction. The reasoning rules can be used for various ends such as generating, revising, optimizing, dropping, and planning goals. Finally, the interpreter determines the flow of control among the above mentioned ingredients, the selection of goals, actions, and reasoning rules, and querying the belief base. The interpreter is usually based on a deliberation cycle [6, 7]. This cognitive architecture, which we call the 3APL architecture, is illustrated in Figure 2.

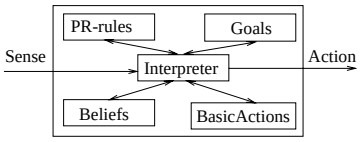

Figure 2: ***The 3APL architecture.***

Although the 3APL architecture has many similarities to other cognitive architectures such as PRS, proposed for the BDI agents [7], they differ from each other in many ways. For example, the PRS architecture is designed to plan agents' goals (desires) while the 3APL architecture is designed to control and revise agents' to-do-goals. In fact, a goal in the 3APL architecture is a program which can be compared to partial plans in the PRS architecture. Moreover, there is no ingredient in the PRS architecture that corresponds to the practical reasoning rules, which are a powerful mechanism to revise mental attitudes. Finally, the deliberation cycle in 3APL is supposed to be a programmable component while the deliberation cycle in PRS is integrated in the interpreter.

In the basic implementation of 3APL the interpreter component is not programmable but implemented statically. Like the PRS interpreter this interpreter is a simple fixed loop consisting of the following steps:

1- Find Reasoning Rules of which the head Matches a Goal
2- Find Reasoning Rules of which Beliefs satisfy the Guard
3- Select Practical Reasoning Rules to Apply to Goals
4- Apply Practical Reasoning Rules to Goals
5- Find Goals to Execute
6- Select Goal to Execute
7- Execute the prefix of the Goal consisting of Basic Actions

This loop illustrates very clearly that the interpreter consists of two parts. One part is the deliberation about goals and rearranging of the goals by rewriting them using the reasoning rules (steps 1 to 4). The second part deals with the execution of selected goals (steps 5 to 7). The first part is closely related to planning. However, in 3APL we only apply one reasoning rule at the time. We do not apply new reasoning rules on the resulting new goals of the agent. This means that we only generate a first part of a plan, which is immediately executed. Only after this execution a next planning step is performed by applying (newly) applicable reasoning rules to the remaining goal of the agent. The advantage of this method is that the agent is very reactive to the environment because execution and planning are interleaved. A disadvantage is that no "real" planning is possible. The advantage of a real planning stage is that plans can be developed and evaluated and subsequent backtracking over the plans can take place when the plan was not satisfactory. In 3APL there is no backtracking! Whenever a PR rule is applied, the goal has changed and the agent has to move from that new situation. We cannot roll back the reasoning rules and try alternative rules if the first rules do not lead to the desired result.

In the next sections we will elaborate a bit more on this feature. Here we only mention it because it highlights a hidden choice that is made in the basic 3APL implementation and other agent systems in which the deliberation cycle is predefined and implemented statically. Besides fixing the

order in which reasoning rules and goals are selected and executed it was also determined that the interpreter has to interleave the application of reasoning rules with the execution of goals (and we cannot iterate those components a number of times before switching to the other component). From the above very simple loop we do already find most of the basic elements needed to program the deliberation of the agents. We need to select reasoning rules to rearrange the current goals, to apply one or more reasoning rules, to select goals to execute, and to execute the goals. Later on we will argue that we also need an explicit step to form a plan on the deliberation level.

In this paper we argue that we cannot just fix the way in which these steps are used in the interpreter, but that the interpreter itself should also be programmed. Of course we are not the first to introduce this type of meta-level programming. Some traditional programming languages already incorporate this feature. For example, Haskell but even Prolog have self referencing and modifying features. But also in the area of agent research this feature can be found in, for instance, the agent design tool DESIRE [5]. In DESIRE one can explicitly program a component to control a number of other components. However, this meta-level technique has not been proposed in agent programming. Our work can be considered as introducing meta-level techniques in an agent-programming language 3APL. Of course, one might argue that the interpreter itself could be programmed using 3APL. In this case the basic actions would be things like select-goal, apply-rule, etc. The goal would be to achieve the goals of the goal base or to apply as many reasoning rules or something like that. However, we feel that the number of actions on this level is too small and the programs (for the moment) are too simple to warrant the use of a powerful mechanism like 3APL on this level. Instead we will introduce a restricted number of instructions with which most situations that we encountered up till now can be easily and compactly programmed. An explicit argument is also the programmability of the complete system. One should have enough expressive power on each level, but having too much expressive power makes it harder to program those things one needs to do. (e.g. although Java is a nice and powerful language it takes an awful amount of code to program simple things like a "hello world" program.) So, one should provide the right tools for the right goals.

3. 3APL PROGRAMMING LANGUAGE

3APL [9] consists of languages for beliefs, basic actions, goals, and practical reasoning rules. A 3APL agent can be programmed by expressions of these languages. A set of expressions of a language implements a 3APL module. Below is an overview of the languages to program each module.

DEFINITION 1. *Given a set of domain variables and functions, the set of domain terms T_D is defined as usual. Let $t_1, \ldots, t_n \in T_D$, and $Pred_b$ be the set of predicates that constitute the belief expressions. A belief language $\mathcal{L}_\mathcal{B}$ is defined as follows: $\mathcal{L}_\mathcal{B} = \phi ::= p(t_1, \ldots, t_n) \mid \neg\phi \mid \phi \wedge \psi$. All variables in $\phi \in \mathcal{L}_\mathcal{B}$ are universally quantified with maximum scope. The belief-base module of a 3APL program is a set of belief formulae.*

The set of basic actions is a set of (parameterized) actions that can be executed if certain preconditions hold. After execution of an action certain post-conditions must hold.

DEFINITION 2. *Let Act be an action name $t_1, \ldots, t_n \in T_D$, and $\phi, \psi \in \mathcal{L}_B$. Then, an action langauge $\mathcal{L}_A$ is defined as follows: $\mathcal{L}_A = \alpha ::= \langle \phi, Act(t_1, \ldots, t_n), \psi \rangle$. The basic action module of a 3APL program is a set of basic actions.*

The set of goals consists of different types of goals: Basic action goals (BactionGoal), predicate goal (PredGoal), Test goal (TestGoal), skip goal (SkipGoal), sequence goal (SeqGoal), if-then-else goal (IfGoal), and while-do goal (WhileGoal).

DEFINITION 3. *Let $t_1, \ldots, t_n \in T_D, Pred_g$ be the set of predicates such that $Pred_b \cap Pred_g = \emptyset, q \in Pred_g$, $\alpha \in \mathcal{L}_A, \phi \in \mathcal{L}_B$. Then, the set of goals $\mathcal{L}_G$ is defined as follows:* $\mathcal{L}_G = \pi ::= \alpha \mid q(t_1, \ldots, t_n) \mid \phi? \mid skip \mid \pi_1 \ ; \ \ldots \ ; \ \pi_n \mid$ `IF` ϕ `THEN` π_1 `ELSE` $\pi_2 \mid$ `WHILE` ϕ `DO` π. *The goal base module of a 3APL program is a set of goals.*

Before we define practical reasoning rules, a set of goal variables, $GVAR$, is introduced. These variables are different from the domain variables used in the belief language. The goal variables may occur in the head and the body of practical reasoning rules and will be instantiated with a goal. Note that the domain variables are instantiated with the belief terms. We extend the language $\mathcal{L}_G$ with goal variables. The resulting language $\mathcal{L}_{G_v}$ extends $\mathcal{L}_G$ with the following clause: if $X \in GVAR$, then $X \in \mathcal{L}_{G_v}$.

DEFINITION 4. *Let $\pi_h, \pi_b \in \mathcal{L}_{G_v}$ and $\phi \in \mathcal{L}_B$, then a practical reasoning rule is defined as: $\pi_h \leftarrow \phi \mid \pi_b$.*
This practical reasoning rule can be read as follows: if the agent's goal is π_h and the agent believes ϕ, then π_h can be replaced by π_b. The practical reasoning module of a 3APL program is a set of practical reasoning rules.

A practical reasoning rule can be applied to a goal by unifying the head of the rule with (a part of) the goal. Since goal variables may occur in the head and the body of practical reasoning rules, the unification results in a substitution for goal variables. The resulting substitution will be applied to the body of the practical reasoning rule and the resulting goal will replace (part of) the goal to which the rule was applied. For example, consider the practical reasoning rule $A(); X; C() \leftarrow \top \mid X; X$ and the goal $\pi = A(); B(); C()$. The application of the rule to π results the substitution $[X/B()]$ which, when applied to the body of the rule, result the goal $B(); B()$. This goal will replace π since the head of the rule was unified with the whole π and not with a part of it. Of course, there are many different choices for how to unify the head of a rule with a goal. A discussion of this issue is out of the scope of this paper.

3.1 3APL Semantics

In [9] an operational semantics for the 3APL language is proposed which is defined by means of a transition system. This semantics specifies transitions between the agent's states by means of transition rules. The state of an agent is defined as follows:

DEFINITION 5. *The state of a 3APL agent is defined as a 4-tuple $< \Pi, \sigma, \Gamma, \theta >$, where Π is the set of the agent's goals, σ is the agent's beliefs, Γ is the set of practical reasoning rules, and θ is a substitution consisting of binding of variables that occur in the agent's beliefs and goals. (In the sequel we leave out Γ from the agent's states since this part does not change by transitions.)*

The bindings of variables θ are passed through from one state to the next state by means of transition rules. Some transition rules update the variable binding before passing them through the next state, other transition rules pass it through the next state without an update. For example, the bindings are updated by the transition rules for the execution of test goals and the application of practical reasoning rules and they are not updated by the transition rules for sequence and choice operators. In this section, we illustrate only some of the transition rules that help the understanding of this paper. The reader will find the complete set of transition rules in [9].

The first transition rule is for the execution of the set Π of agents' goals. This transition rule states that the execution of a set of goals can be accomplished by the execution of each goal separately. Let $\Pi = \{\pi_0, \ldots, \pi_i, \ldots \pi_n\} \subseteq G$, and θ and θ' be ground substitutions. Then,

$$\frac{< \{\pi_i\}, \sigma, \theta > \rightarrow < \{\pi_i'\}, \sigma', \theta' >}{< \{\pi_0, \ldots, \pi_i, \ldots \pi_n\}, \sigma, \theta > \rightarrow < \{\pi_0, \ldots, \pi_i', \ldots \pi_n\}, \sigma', \theta' >}$$

The transition for basic actions updates the belief base but does not update the variable bindings. Let $A(\overrightarrow{t})$ be a basic action with a sequence of domain terms $\overrightarrow{t}$ and τ be an update function that specifies the effect of the basic action on agent's beliefs. The semantics of basic actions is captured by the following transition rule.

$$\frac{\tau(A(\overrightarrow{t})\theta, \sigma) = \sigma'}{< \{A(\overrightarrow{t})\}, \sigma, \theta > \rightarrow < \emptyset, \sigma', \theta >}$$

3.2 Example of a 3APL Program

The example of the mobile robot, mentioned in the introduction, can be implemented in 3APL as follows.

```
PROGRAM "transport_robot_1"
CAPABILITIES:
  {pos(R1),cost(X),not forbid(R1, R2)}
      Go(R1, R2)
  {not pos(R1),pos(R2),cost(X+1)},

  {pos(R1),cost(X),forbid(R1, R2)}
      Go(R1, R2)
  {not pos(R1),pos(R2),cost(X+5)},

  {box(self),delpos(R),pos(R)}
      PutBox()
  {not Box(self),box(R)},

  {pos(R),box(R)}
      GetBox()
  {not box(R), box(self)}
BELIEFBASE:
  pos(r4),box(r2),delpos(r1),gain(5),cost(0),forbid(r4, r2),
  door(r1, r2),door(r1, r3),door(r2, r4),door(r3, r4),
  door(R1, R2):-(R2, R1),forbid(R1, R2):-forbid(R2, R1)
GOALBASE:
  transportBox()
RULEBASE:
  transportBox() <-- pos(R1),box(R2),delpos(R3) |
      goxy(R1, R2);GetBox();goxy(R2, R3);PutBox(),
  goxy(R1, R2) <-- pos(R1),door(R1, R3),not R1 = R2 |
      Go(R1, R3);goxy(R3, R2),
  goxy(R1, R2) <-- R1 = R2 | SKIP.
```

The first basic action states that the robot can go from room R_1 to room R_2 if the robot is in room R_1 (pos(R_1)) and if it is not forbidden to go from room R_1 to R_2 (not forbid(R_1, R_2)), and its accumulated costs before this action (cost(X)). After the execution of the action, the robot is not in room R_1 (not pos(R_1)), but in room R_2 (pos(R_2)) and its accumulated cost is increased by one (cost($X + 1$)). The second basic action differs from the first since going from one to the other room while it is forbidden increases the accumulated cost with 5. Other basic actions should be interpreted in similar ways.

The statement in the belief base module states that the robot is initially in room r_4 (pos(r_4)), box is in room r_2 (box(r_2)), the box should be delivered in room r_1 (delpos(r_1)), the utility gained by transporting a box is 5 (gain(5)), the initial accumulated cost of the transport is 0 (cost(0)), it is forbidden to go from room r_4 to room r_2 (forbid(r_4, r_2)), and between which rooms is a door. Note also that the belief base can also contain rules such as door(R_1, R_2):-door(R_2, R_1).

The robot has only one goal which is to transport boxes (i.e. transportBox()). Note that this is a predicate goal which can be achieved by applying practical reasoning rules to it.

The practical reasoning rules are in fact planning rules that indicate how the transport goal can be achieved. The first planning rule states that the robot should go first to the room where there is a box, get the box, then go to the delivery room and put the box in the delivery room. The guard in this rule is used to instantiate a number of domain variables needed to determine how to achieve the transport goal. Other planning rules can be interpreted in a similar way.

In order to execute an agent (3APL program) many issues should be decided continually. For example, at each moment of the execution it should be decided which rules to select and apply and which goals to select and execute. Also, it should be decided if rules should be applied before goals are executed or vice versa. These decisions are usually implemented in the interpreter which implies that many deliberation choices are decided statically and thus hardwired in the interpreter. The fact that these decisions are pre-programmed in the interpreter makes it hard, if not impossible, to program or influence the deliberation process of an agent. As the deliberation process involves various processes such as reasoning about plans, norms, goals, and commitments and since agent programmers may want to influence these processes, it is essential to allow agent programmers to program the agent deliberation process. With a static deliberation cycle it is not easy to program agents with reasonable complex behavior.

For example, consider the 3APL program for the transport robot where the planning rules are specified based on a general model of the environment. This model is defined in terms of the relation between rooms and the movements through the doors. If this program is executed by selecting the beliefs, goals and rules in the order they occur, the robot may move between two adjacent rooms indefinitely. This is related to the recursive way the subgoal $goxy(R_1, R_2)$ is defined and the way the guards are unified with belief formulae. Of course, this problem can be solved by modelling the environment and following more specific planning rules.

RULEBASE:
transportBox() $\longleftarrow$ pos(R_1) , box(R_2) , delpos(R_3) |
goxy(R_1, R_2);GetBox();goxy(R_2, R_3);PutBox(),
goxy(r_1, r_2) $\longleftarrow$ pos(r_1) | Go(r_1, r_2),
goxy(r_3, r_2) $\longleftarrow$ pos(r_3) | Go(r_3, r_1);Go(r_1, r_2),
goxy(r_2, r_1) $\longleftarrow$ pos(r_2) | Go(r_2, r_1),
goxy(r_4, r_2) $\longleftarrow$ pos(r_4) | Go(r_4, r_2),
goxy(r_4, r_2) $\longleftarrow$ pos(r_4) | Go(r_4, r_3);Go(r_3, r_1);Go(r_1, r_2).

This set of planning rules solves this new problem of indefinite moves between adjacent rooms, but the result is a less general program since it can only behave in the specific environment as illustrated in Figure 1. Moreover, neither of the two 3APL programs will transport the box from room r_4 to room r_1 with the lowest cost. This problem is related to the order of planning rules. Of course, this program can still be modified in the following way such that the robot will transport the box with the lowest cost.

RULEBASE:
transportBox() $\longleftarrow$ pos(R_1) , box(R_2) , delpos(R_3) |
goxy(R_1, R_2);GetBox();goxy(R_2, R_3);PutBox(),
goxy(r_1, r_2) $\longleftarrow$ pos(r_1) | Go(r_1, r_2),
goxy(r_3, r_2) $\longleftarrow$ pos(r_3) | Go(r_3, r_1);Go(r_1, r_2),
goxy(r_2, r_1) $\longleftarrow$ pos(r_2) | Go(r_2, r_1),
goxy(r_4, r_2) $\longleftarrow$ pos(r_4),not(forbid(r_4, r_2)) | Go(r_4, r_2),
goxy(r_4, r_2) $\longleftarrow$ pos(r_4),forbid(r_4, r_2) |
Go(r_4, r_3);Go(r_3, r_1);Go(r_1, r_2).

Like the previous formulation, this set of practical reasoning rules models the environment in a specific way. But, unlike the previous formulation, it models also a part of the task as well, i.e. it specifies the cheapest ways to transport boxes from any room to room r_1 and r_2. This model is thus specific to both the environment and the task.

There are a number of drawbacks for such solutions. The first problem is that the agent program becomes too specific to the given situation and therefore loses its generality such that the agent cannot operate in another office like environment and for other tasks. Also, the agent program becomes too complex and unreadable since too many details are included in the program. Finally, the agent programmer is forced to implement the desired behavior of the agent in an implicit way instead of programming them by explicit statements. For this example, an agent programmer would like to have explicit planning constructs in order to implement a search algorithm in the space of possible plans to find the best plan. Programming this type of agent activities implicitly in terms of details of the task and the environment does not respect the programming principle called the separation of concerns.

4. A PROGRAMMING LANGUAGE FOR THE DELIBERATION CYCLE

The specification of 3APL [9] includes a meta-language to implement some parts of the deliberation cycle. The proposed meta-language can express decisions such as whether goals should be revised before getting executed, as well as the selection mechanism, i.e. which goals should be selected and executed and which rules should be selected and applied. The meta-language is imperative and set-based. It is imperative because it is used to program the flow of control and it is set-based because it is used to select goals and rules from the set of goals and rules.

The proposed meta-language is not expressive enough to program a selection mechanism for goals and rules that is based on the agent's beliefs. According to the existing meta-language a goal or a rule can be selected only on the basis of an ordering that is defined on them; the beliefs can only be incorporated in the selection of rules (and not goals) by means of the guard of the rules in the object language. However, there are situations in which the deliberation process in general, and the selection mechanism in particular, depend on agent's beliefs. For example, consider the transport robot and suppose that it has, besides the transport goal, a number of other goals including the cleaning of the rooms. Suppose also that the robot can be in an emergency situation where it should only select and execute the transport goal without applying any revision rule. In a normal situation, the robot should select and apply revision rules and select and execute both goals without preferring one over the other.

As noted in the previous section, an important part of the agent deliberation concerns its capabilities to represent and reason about possible plans. The existing meta-language does not include artifacts to generate and manipulate plans. Therefore, we extend the 3APL meta-language with new programming constructs to generate and execute new plans, replanning an existing plan based on some criteria, and backtracking in the space of possible plans. In the following we consider a plan as a sequence of planning rules which, when applied to a goal consecutively, result in a sequence of actions.

DEFINITION 6. *A plan is a sequence $\langle \rho_1^x, \ldots, \rho_n^y \rangle$ of practical reasoning rules with $n \in N$. The practical reasoning rules are assumed to be planning rules, i.e. they have the form $\pi_h \leftarrow \phi \mid \pi_b$, where π_h is an predicate goal, ϕ is a belief formula, and π_b is a goal in which no goal variable occurs. The indices x and y are used to express the identity of the planning rules. The subscripts refer to their place in the plan.*

As in the existing 3APL meta-language and as we need means to access goals, rules, and plans, we introduce terms that denote goals, rules, and plans. Here we use sorted terms because we need terms to denote different entities such as sets of goals, individual goals, sets of rules, individual rules, numbers, etc. These terms can be used as arguments of programming constructs that implement the deliberation cycle.

DEFINITION 7. *Let S be a set of sorts including sorts for sets of goals (sg), set of practical reasoning rules (sr), individual goals (ig), individual practical reasoning rules (ir), individual plans (p), set of plans (sp), numbers (N), and boolean (B). Let also Var_s be a set of countably infinite variables for sort s, and F be a set of sorted functions. The sorted terms for the meta-language are then defined as follows:*
- $\Pi \in T_{sg}$ and $\Gamma \in T_{sr}$.
- $Var_s \subseteq T_s$ for all sort $s \in S$.
- if $s_1, \ldots, s_n \in S, t_i \in T_{s_i}$ and $f : s_1 \times \ldots \times s_n \rightarrow s \in F$, then $f(t_1, \ldots, t_n) \in T_s$.

Typical functions that are defined on goals are $max : sg \rightarrow sg$ that selects the subset of most preferred goals from the set of goals, based on an ordering on the goals, $empty : sg \rightarrow B$ that determines whether the set of goals is empty, $sel\_trans : sg \rightarrow ig$ that selects an individual goal (in this case the transport goal) from a set of goals, or $gain : ig \rightarrow N$

that determines the utility (a number) of an individual goal. Note that nested applications of functions can be used as well. For example, $gain(max(\Pi))$ indicates the utility of the most preferred goal. Similar functions can be defined for practical reasoning rules and individual plans.

In general, practical reasoning rules can be classified according to the purpose of their use. As mentioned, they can for example be used to revise goals that are not achievable, or actions that are blocked, to plan goals, or to optimize goals. The types of practical reasoning rules can be used to select them. For example, rules that generate reactive behavior can be selected preferably to make an agent behave more reactively. Similarly, rules that generate plans can be selected preferably to make an agent more deliberative. A different classification of rules can be defined based on their structural properties such as having a head, a body, or a condition. For example, a preference to apply rules without body makes the agent drop its goals preferably. It is clear that a more refined classification of practical reasoning rules results in a variety of agent types.

Given the meta-terms for goals, practical reasoning rules, and plans, the set of meta-formulae can be defined. Since the goal and the rule selection mechanism may depend on agent's beliefs, we include the belief base of the agent in the set of meta-formulae.

DEFINITION 8. *Let $s \in S$ be a sort, $t, t' \in T_s$ be meta-terms of sort s, φ be a belief sentence,*[1] *and ϕ and ψ be meta-formulae. We define the set of meta-formulae MF as follows:*
- $t = t', t \geq t', B(\varphi) \in MF$
- *if $\phi, \psi \in MF$, then $\neg\phi, \phi \wedge \psi \in MF$*

This set of programming constructs, which can be used to program a 3APL deliberation cycle, includes those proposed in [9], but also the programming constructs for the planning purposes. The programming constructs for implementing the deliberation cycle of cognitive agents are defined as follows:

DEFINITION 9. *Let $s \in S, t_s \in T_s$ and $V_s \in Var_s$. The set of meta-statements MS is defined as follows:*
- $V_s := t_s \in MS$
- $exgoal(t_{ig}, V_{ig})$,
  $selrule(t_{sg}, t_{sr}, V_{ig}, V_{ir})$,
  $applyrule(t_{ir}, t_{ig}, V_{ig}) \in MS$
- $plan(t_{ig}, V_p, t_N)$,
  $replan(t_{ig}, t_{sp}, V_p, f_c, t_N)$,
  $explan(t_p)$,
  $btplan(t_{ig}, t_p, V_p) \in MS$
- *if $\phi \in MF, \alpha, \beta \in MS$, then*
  $\alpha;\beta$,
  `IF` ϕ `THEN` α `ELSE` β,
  `WHILE` ϕ `DO` $\alpha \in MS$

The meta-statement $V_s := t_s$ is designed to assign a sorted term t_s to a variable V_s of the same sort. The meta-statement $exgoal(t_{ig}, V_{ig})$ is designed to execute an individual goal denoted by the term t_{ig}. As a goal may not be fully executable, some part of it may remain unexecuted. This remaining part is itself a goal which is assigned to the variable V_{ig}. The meta-statement $selrule(t_{sg}, t_{sr}, V_{ig}, V_{ir})$ selects a rule and a goal from the set of rules denoted by the terms t_{sr} and the set of goals denoted by the term t_{sg}, respectively. The selected rule should be applicable to the selected goal. The selected rule and goal are assigned to variables V_{ir} and V_{ig}, respectively. The meta-statement $applyrule(t_{ir}, t_{ig}, V_{ig})$ applies a rule denoted by the term t_{ir} to a goal denoted by the term t_{ig}. The resulting revised goal is assigned to variable V_{ig}.

The meta-statement $plan(t_{ig}, V_p, t_N)$ generates a plan V_p to achieve goal t_{ig}. The length of the plan is bounded to t_N. The meta-statement $replan(t_{ig}, t_{sp}, V_p, f_c, t_N)$ generates a new plan to replace plans t_{sp} that were generated to achieve goal t_{ig}. The new plan is assigned to variable V_p and satisfies the criteria f_c. The function f_c maps plans to boolean values indicating whether the plan satisfies a criterium c. The length of the new plan is bounded by t_N. The meta-statement $explan(t_p)$ executes the plan denoted by the term t_p. Finally, the last meta-statement for the planning purposes is $btplan(t_{ig}, t_p, V_p)$, which does the same as replan, except that btplan uses the order of planning rules and generates the next plan according to that order. The existing plan denoted by t_p, which was generated to achieve the goal denoted by t_{ig}, is replaced by a new plan which is assigned to variable V_p.

Let $trans(\Pi) = transportBox()$ be the term denoting the transport goal, $cost(P)$ be the term that determines how expensive is the plan that is assigned to variable P, and $gain(trans(\Pi))$ be the term that determines the utility of the goal $trans(\Pi)$. The deliberation cycle of our example 3APL agent can now be programmed as follows:

```
plan(trans(Π),P,maxlength)
WHILE (cost(P) > gain(trans(Π))) OR empty(Π) DO
BEGIN
   btplan(trans(Π),P,P',maxlength);
   P:=P';
END
IF gain(trans(Π)) > cost(P) THEN
   explan(P)
```

This deliberation cycle initially finds a plan P, whose length is bounded to maxlength, to achieve the goal denoted by the term trans(Π). While the cost of this plan is higher than the utility of achieving the goal ($gain(trans(\Pi))$), it attempts to find a new plan by backtracking in the space of possible plans.

5. SEMANTICS OF META-LANGUAGE

Before we specify the semantics for the meta-level programming constructs, the following auxiliary notations are introduced. The transition that is derived in the transition system by executing the basic actions that are the prefix of a goal is called an execution transition and it can be expressed as $\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \xrightarrow{\pi_1, \pi_2} \langle \Pi \cup \{\pi_2\}, \sigma_2, \theta_2 \rangle$. This transotion indicates that transition $\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \longrightarrow \langle \Pi \cup \{\pi_2\}, \sigma_2, \theta_2 \rangle$ can be derived in the transition system by executing the sequence of basic actions and tests that are the prefix of π_1. In this case, $\pi_1 = a_1; \ldots; a_n; \pi_2$ where $a_1, \ldots, a_n \in BactionGoal \cup TestGoal$.

The transition that is derived in the transition system by applying a practical reasoning rule to a goal is called an application transition and it can be expressed as follows:

[1] We restrict ourselves to belief *sentences* to simplify the semantic definition of the meta-language. This restriction allows us to ignore the problem of substitution of object-level terms.

$\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \xrightarrow{\pi_1, \rho_i^x, \pi_2} \langle \Pi \cup \{\pi_2\}, \sigma_1, \theta_2 \rangle$ which indicates that transition $\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \longrightarrow \langle \Pi \cup \{\pi_2\}, \sigma_1, \theta_2 \rangle$ can be derived in the transition system by applying PR-rule ρ_i^x to (the prefix of) π_1.

We consider a 3APL goal as a partial plan. Such a partial plan gets more specific by planning the predicate goals that occur in the partial plan. This can be done by the consecutive application of planning rules (PR-rule) to the predicate goals in the order of their occurrence in the partial plan. Since a PR-rule can be applied if and only if the condition in its guard is satisfied and because this condition can be affected by the actions that precede it, we need to execute the preceding actions. Therefore, we define a pair of transitions called a transition cycle. A transition cycle consists of an execution transition followed by an application transition.

DEFINITION 10. *Let $\pi_1 = a_1; \ldots; a_n; \pi_3$ and π_2 be goals. Then, a transition cycle, labelled by c, is defined as follows:*

$\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \xrightarrow{c(\pi_1, \rho_i^x, \pi_2)} \langle \Pi \cup \{\pi_2\}, \sigma_2, \theta_2 \rangle \Leftrightarrow$

$\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \xrightarrow{\pi_1, \pi_3} \langle \Pi \cup \{\pi_3\}, \sigma_2, \theta_3 \rangle \xrightarrow{\pi_3, \rho_i^x, \pi_2} \langle \Pi \cup \{\pi_2\}, \sigma_2, \theta_2 \rangle.$

In the following definitions, $\overrightarrow{\rho_n} =< \rho_1^x, \ldots, \rho_{n-1}^y >$ is a sequence of practical reasoning rules and L is the maximum depth of the tree obtained by applying the sequence $\overrightarrow{\rho_n}$ to the (partial plan) goal π_1. A partial plan gets more specific by a sequence of transition cycles.

DEFINITION 11. *A sequence of transition cycles, labelled by c^*, is defined as follows:*

$\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \xrightarrow{c^*(\pi_1, \overrightarrow{\rho_n}, \pi_n, L)} \langle \Pi \cup \{\pi_n\}, \sigma_n, \theta_n \rangle \Leftrightarrow$

$\langle \Pi \cup \{\pi_1\}, \sigma_1, \theta_1 \rangle \xrightarrow{c(\pi_1, \rho_1^x, \pi_2)} \langle \Pi \cup \{\pi_2\}, \sigma_2, \theta_2 \rangle$

$\vdots$

$\langle \Pi \cup \{\pi_{n-1}\}, \sigma_{n-1}, \theta_{n-1} \rangle \xrightarrow{c(\pi_{n-1}, \rho_{n-1}^y, \pi_n)} \langle \Pi \cup \{\pi_n\}, \sigma_n, \theta_n \rangle$

Let $\pi_1 \in \Pi$. We write $\langle \Pi, \sigma_1, \theta_1 \rangle \xrightarrow{c^*(\pi_1, \overrightarrow{\rho_n}, \pi_n, L)}$ to indicate that there exists a computation sequence transition for 3APL agent configuration $\langle \Pi, \sigma_1, \theta_1 \rangle$ where L is the maximum depth of the tree obtained by applying the sequence $\overrightarrow{\rho_n}$ to the goal π_1.

In order to define the semantics of the meta-language, we need to define meta-state and meta-configuration.

DEFINITION 12. *A meta-state is a tuple $\langle \langle \Pi, \sigma, \theta, \Gamma \rangle, D, O, I \rangle$, where $\langle \Pi, \sigma, \Gamma, \theta \rangle$ is 3APL object state consisting of the set of goals, beliefs, practical reasoning rules, and the substitutions, $D = \{D_{s_1}, \ldots, D_{s_n}\}$ is the set of domains of the sorts $s_1, \ldots, s_n$, $O = \{\geq_{s_1}, \ldots, \geq_{s_n}\}$ is a set of orderings defined on domains $D_{s_1}, \ldots, D_{s_n}$, and I is a variable valuation $I : Var \rightarrow D$. Note that $\Pi \in D_{sg}, \Pi = D_{ig}, \Gamma \in D_{sr}$, and $\Gamma = D_{ir}$. Note also that the domain D_{ip} of individual plans contains sequences $\overrightarrow{\rho}$ of planning rules, i.e. $D_{ip} = \{\overrightarrow{\rho_n} \mid \overrightarrow{\rho_n} = \langle \rho_1^x, \ldots, \rho_n^y \rangle$ & $\rho_i^v \in D_{ir}$ & $1 \leq i \leq n$ & $n \in N\}$.*

In the following, we will not mention the set of practical reasoning rules Γ, the set of domains D, and orderings O in the meta-states since they are assumed not to change. We present a meta-state as the nested tuple $\langle \langle \Pi, \sigma, \theta \rangle, I \rangle$. The terms and formulae of the meta-language are evaluated in meta-states. The evaluation function can be defined in a standard way which we will not present for the sake of space. The semantics of the meta-language is an operational semantics and the transition relation of the meta-transition system is denoted by $\Rightarrow$. The transition relation is defined on the meta-configuration which is a tuple $\langle \mathcal{S}, \langle \Pi, \sigma, \theta \rangle, I \rangle$ consisting of a meta-statement $\mathcal{S}$ and a meta-state $\tau = \langle \langle \Pi, \sigma, \theta \rangle, I \rangle$. In the following, we present only the transition for the planning meta-statements. Other transitions are already presented in [9].

Let g be the goal term that denotes the goal π_1, r_i be the rule term that denotes the planning rule ρ_i^x for $1 \leq i \leq n$, and $[\langle r_1, \ldots, r_n \rangle / P]$ indicates that the sequence $\langle r_1, \ldots, r_n \rangle$ is assigned to the plan variable P. The following transition states that if the sequence of planning rules $\overrightarrow{\rho_n} = \langle \rho_1^x, \ldots, \rho_n^y \rangle$ can be applied consecutively to the predicate goal π_1 to achieve it, then the meta-statement $plan(g, P, L)$ can be applied successfully and the result is an assignment of the goal variable P. Note that planning does not change the object state, but only the meta-variable bindings.

DEFINITION 13. *The semantic rule for the planning construct can now be defined as follows:*

$$\frac{\langle \Pi, \sigma, \theta \rangle \xrightarrow{c^*(\pi_1, \overrightarrow{\rho_n}, \pi_n, L)}}{\langle plan(g, P, L), \langle \Pi, \sigma, \theta \rangle, I \rangle \Rightarrow \langle E, \langle \Pi, \sigma, \theta \rangle, I' \rangle}$$

where $||[g]||_\tau = \pi_1 \in \Pi$ and $I' = I \cup \{[\langle r_1, \ldots, r_n \rangle / P]\}$, and $||[r_i]||_\tau = \rho_i^v$ for $i = 1, \ldots, n$, and $\overrightarrow{\rho_n} = \langle \rho_1^x, \ldots, \rho_n^y \rangle$. A plan P is called a concrete plan whenever $\pi_n = E$.

The idea of replanning is to generate, for a predicate goal π_1 and a set of plans $\{\overrightarrow{\rho_k}, \ldots, \overrightarrow{\rho_l}\}$ that achieve it, a new but different plan $\overrightarrow{\rho_m}$ which achieves π_1 and satisfies the criteria encoded by the function f_c. Note that replanning does not change the object state, but only the meta-variable bindings.

DEFINITION 14. *Let LP denote the set $\{\overrightarrow{\rho_k}, \ldots, \overrightarrow{\rho_l}\}$ and $k \leq i \leq l$. The semantic rule for the replanning construct can now be defined as follows:*

$$\frac{\forall i \langle \Pi, \sigma, \theta \rangle \xrightarrow{c^*(\pi_1, \overrightarrow{\rho_i}, \pi_n, L)} \& \langle \Pi, \sigma, \theta \rangle \xrightarrow{c^*(\pi_1, \overrightarrow{\rho_m}, \pi_m, L)} \& \; ||[f_c(P')]||_\tau}{\langle replan(g, LP, P', f_c, L), \langle \Pi, \sigma, \theta \rangle, I \rangle \;\Rightarrow\; \langle E, \langle \Pi, \sigma, \theta \rangle, I' \rangle}$$

where $||[g]||_\tau = \pi_1 \in \Pi$, and $\overrightarrow{\rho_i} = \langle \rho_1^x, \ldots, \rho_i^y \rangle$ and $\overrightarrow{\rho_m} = \langle \rho_1^{x'}, \ldots, \rho_m^{y'} \rangle$, and $\forall i \; \overrightarrow{\rho_i} \neq \overrightarrow{\rho_m}$, and $|[r_j]|_\tau = \rho_j^v$ and $||[r'_j]||_\tau = \rho_j^{v'}$, and $I' = I \cup \{[\langle r'_1, \ldots, r'_m \rangle / P']\}$, and $[\langle r_1, \ldots, r_n \rangle / P_n] \in I$.

The execution of a plan can be specified in a step by step fashion. The execution of the plan does change the object state, but leaves the meta-state unchanged. The idea of executing a plan $\overrightarrow{\rho_n}$ is to apply the first planning rule ρ_1 to achieve goal π_1. The application of this rule transforms the object state $\langle \Pi, \sigma, \theta \rangle$ to a next object state $\langle \Pi', \sigma', \theta' \rangle$ and proceeds with executing with the rest of the plan which is $\langle \rho_2, \ldots, \rho_n \rangle$. The sequence of terms $\langle r_2, \ldots, r_n \rangle$ that denote the rest of the plan is assigned to the plan variable P'.

DEFINITION 15. *The semantic rule for the plan execution statement is defined based on executing one planning rule per step as follows:*

$$\frac{\langle \Pi, \sigma, \theta \rangle \xrightarrow{c^*(\pi_1, \overrightarrow{\rho_n}, \pi_n, L)} \& \langle \Pi, \sigma, \theta \rangle \xrightarrow{\pi_1, \rho_1, \pi_2} \langle \Pi', \sigma', \theta' \rangle}{\langle explan(P), \langle \Pi, \sigma, \theta \rangle, I \rangle \Rightarrow \langle explan(P'), \langle \Pi', \sigma', \theta' \rangle, I \rangle}$$

where $[\langle r_1, \ldots, r_n \rangle / P] \in I$ and $[\langle r_2, \ldots, r_n \rangle / P'] \in I'$ and $||[r_i]||_\tau = \rho_i$ for $i = 1, \ldots, n$ and $\overrightarrow{\rho_n} = \langle \rho_1, \ldots, \rho_n \rangle$, and $\Pi' = (\Pi \backslash \{\pi_1\}) \cup \{\pi_2\}$.

The backtrack plan is a meta-statement that, given a goal π_1 and a plan $\overrightarrow{\rho_n}$ that achieves it, finds the next plan $\overrightarrow{\rho_m}$ to achive π_1 in the space of possibe plans. In order to specify the next plan, we use the ordering in the space of possible plans and demand that there is no other plan $\overrightarrow{\rho_k}$ which is closer to the $\overrightarrow{\rho_n}$.

DEFINITION 16. *The semantic rule for the backtrack construct defined as follows:*

$$\frac{\langle \Pi, \sigma, \theta \rangle \overset{c^*(\pi_1, \overrightarrow{\rho_n}, \pi_n, L)}{\longrightarrow} \ \& \ \langle \Pi, \sigma, \theta \rangle \overset{c^*(\pi_1, next(\overrightarrow{\rho_n}), \pi_m, L')}{\longrightarrow}}{\langle btplan(g, P, P'), \langle \Pi, \sigma, \theta \rangle, I \rangle \ \Rightarrow \ \langle E, \langle \Pi, \sigma, \theta \rangle, I' \rangle}$$

where $|[g]|_\tau = \pi_1 \in \Pi$ *and* $\overrightarrow{\rho_n} =< \rho_1^x, \ldots, \rho_i^{y_1}, \rho_{i+1}^{y_2}, \ldots, \rho_n^z >$ *and* $next(\overrightarrow{\rho_n}) =< \rho_1^x, \ldots, \rho_i^{y_1}, \rho_{i+1}^{y_2'}, \ldots, \rho_m^{z'} >$ *and with* $y_2' > y_2$ *and* $\nexists \rho_{i+1}^{y_2''}$ *with* $y_2' > y_2'' > y_2$ *and* $|[r_i]|_\tau = \rho_i^v$ *and* $|[r_i']|_\tau = \rho_i^{v'}$ *and* $I' = I \cup \{[\langle r_1, \ldots, r_i, r_{i+1}', \ldots, r_m' \rangle / P']\}$, *and* $[\langle r_1, \ldots, r_n \rangle / P] \in I$.

The proposed semantics of the planning constructs assume a certain planning algorithm, which is based on systematic search through the space of all possible plans specified by the set of planning rules. Of course, this is only one possible way to define the semantics of the planning constructs. Alternative planning algorithms and planning styles can be assumed, and consequently alternative semantics for the proposed planning constructs can be given [1, 2, 4]. Our claim is not that the assumed planning algorithm is desirable, but that it is desirable to make the proposed planning constructs available to the agent programmers in order to implement various types of agent deliberation.

6. CONCLUSION

In this paper, we presented the specification of a programming language to implement the deliberation cycle of cognitive agents such as 3APL agents. The programming language can be used to implement high level control of cognitive agents and software agents. The proposed language is considered as a meta-language the terms of which are formulae from an object language. We have assumed 3APL as the object language which provides programming constructs to implement object level concerns such as the mental attitudes of the agent.

In contrast to other programming languages designed to implement high level control of cognitive agents, such as ConGolog, the 3APL goals are deterministic programs and do not allow the choice and parallel operator. Instead, deterministic goals can be revised by applying practical reasoning rules to those goals. The application of practical reasoning rules to deterministic goals is equivalent to allowing the choice operator in goals and using backtracking mechanism to meet the choices. Note that elsewhere [8] it is proven that ConGolog can be embedded in 3APL.

At this moment we have a static 3APL interpreter implemented as a JAVA class, which has private attributes for each of the 3APL modules. This class has at least two methods, one method for creation of the 3APL object through which the private attributes are set, and one method for running the object. The latter method corresponds to the 3APL interpreter which implements the deliberation cycle. We also have a prototype version of a meta-interpreter written in Haskell. This prototype version contains the meta-level constructs which can be used to select and execute goals, select and apply rules, plan a goal, replan an existing plan, backtrack in the space of possible plans, and execute plans.

7. REFERENCES

[1] J. Allen, J. Hendler, and A. Tare, editors. *Readings in Planning*. Morgan Kaufmann, Palo Alto CA, 1990.

[2] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[3] G. Boella and R. Damiano. An architecture for normative reactive agents. In K. Kawabara and J. Lee, editors, *Intelligent Agents and Multi-Agent Systems, LNAI 2413*, pages 1–17. Springer Verlag, 2002.

[4] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999.

[5] F. Brazier, B. D. Keplicz, N. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997.

[6] G. De Giacomo, Y. Lespérance, and H. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[7] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, 1987.

[8] K. Hindriks, Y. Lespérance, and H. Levesque. An embedding of congolog in 3APL. Technical Report UU-CS-2000-13, Department of Computer Science, University Utrecht, 2000.

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

[10] D. Kinny. The psi calculus: An algebraic agent language. In J.-J. C. Meyer and M. Tambe, editors, *Intelligent Agents VIII, Agent Theories Architectures and Languages, LNAI 2333*, pages 32–50. Springer-Verlag, 2001.

[11] A. Rao and M. Georgeff. Modeling rational agents within a BDI architecture. In *Proceedings of the KR91*, 1991.

[12] M. Shanahan. Reinventing shakey. In J. Minker, editor, *Workshop on Logic-Based Artificial Intelligence*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.