PRACTIONIST: A NEW FRAMEWORK FOR BDI AGENTS

V. Morreale[a] S. Bonura[a] G. Francaviglia[a] M. Cossentino[b]

S. Gaglio[cb]

[a] *R&D Laboratory - Engineering Ingegneria Informatica S.p.A.*
[b] *ICAR-Italian National Research Council*
[c] *DINFO-University of Palermo*

Abstract

In this paper, we present PRACTIONIST (PRACTIcal reasONIng sySTem), a new framework built on the Bratman's theory of practical reasoning to support the development of BDI agents in Java (using JADE) with a Prolog belief base. We aims at reducing the gap between the expressive power of the BDI model and the difficulty of efficiently implementing its features.

In PRACTIONIST we adopt a goal-oriented approach and stress the separation between the deliberation process and the means-ends reasoning, with the abstraction of goal used to formally define both desires and intentions during the deliberation phase. Moreover, PRACTIONIST agents are able to reason about their beliefs and other agent's beliefs, since beliefs are not simple grounded literals or data structures but modal logic formulas.

1 Introduction

One of the most interesting and popular agent theories is the Belief-Desire-Intention (BDI) model. According to the philosophical viewpoint, the internal states and the decision process of a BDI agent are modeled in terms of mental states, such as beliefs, desires and intentions, which respectively represent the information, motivational, and deliberative attitudes of the agent.

The BDI model derives from the philosophical tradition of human practical reasoning, which was first developed by Bratman [2]. It states that humans decide, moment by moment, which actions to perform in order to pursue their goals.

Practical reasoning involves two processes: (1) *deliberation*, to decide what states of affairs to achieve; and (2) *means-ends reasoning*, to decide how to achieve these states of affairs. Bratman's theory stresses the role of intentions in human reasoning, as it states that intentions are important because they affect the selection of next actions to be executed.

In the context of rational agents, the BDI model appears very attractive for several reasons. Firstly, the abstractions used in the model are really intuitive: the reader will agree that it is easy to understand the distinction between the process of deciding what to do and the one involving how to; similarly, the notions of belief, desire and intention are very familiar in human reasoning, so they could be easily used in BDI agent design.

Moreover, this model provides a clear functional decomposition, which indicates what sort of subsystems might be required to build an agent. Nevertheless, the main issue in developing BDI agents is to figure out how to efficiently implement the above-mentioned processes [13].

The best-known BDI agent architecture is the Procedural Reasoning System (PRS), developed by Georgeff and Lansky [7]. Some concrete implementations of PRS have been developed, such as dMARS [6], developed at the Australian AI Institute, the UM-PRS implemented in C++ at the University of Michigan [9], and a Java version of PRS called JAM [8]. Finally, it is worth mentioning JACK [3], which is a commercially available programming language that extends the Java language with BDI features.

It should be remarked that in the PRS and all above-mentioned descendant implementations, beliefs are treated as a collection of arbitrary data structures, desires are usually treated as events,

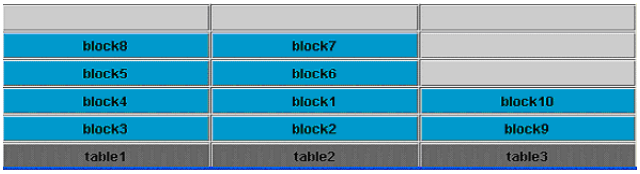

Figure 1: Initial situation for the blocks world problem.

and intentions are realised as executing plans. Thus, the implementation of mental states, deliberation, and means-ends reasoning somewhat differs from their philosophical meaning.

However, in the JADEX agent framework [10], that is an add-on to the JADE agent platform [1], some of these issues are addressed. Indeed, JADEX supports an explicit and declarative representation of goals, so that a deliberation mechanism is able to manage the state transitions of all adopted goals.

Besides, 3APL [5], a programming language for implementing cognitive agents, allows to implement agents' beliefs, goals, belief updates, external actions, communication actions and a set of practical reasoning rules by which agents' goals can be updated. Each 3APL program is executed on the 3APL platform by means of an interpreter that deliberates on the cognitive attitudes of agents. Nevertheless, in 3APL goals and plans represent tha same entity, as goals are considered as abstract plans and actual plans as to-do goals.

In this paper we present PRACTIONIST (PRACTIcal reasONIng sySTem), a new framework we have been developing, which adopts a goal-oriented approach and stresses the separation between the deliberation process and the means-ends reasoning. Indeed the abstraction of goal is used to formally define both desires and intentions during the deliberation phase.

Unlike most of existing BDI implementations, in our approach we actually adopt the plans as recipes to achieve the intentions. Besides, PRACTIONIST allows developers to implement agents that are able to reason about their beliefs and the other agent's (including humans') beliefs, since beliefs are not simple grounded literals or data structures but modal logic formulas.

Throughout this paper we show how PRACTIONIST agents actually work by means of an example, i.e. the well-known *blocks world problem*. Therefore, we developed a *blocks world agent*, which, starting from an initial situation (see figure 1), is supposed to order some numbered blocks according to a criteria that the agent itself receives by means of an ACL message.

This paper is organized as follows: in section 2 we give an overview of the PRACTIONIST agent architecture and its components (i.e. beliefs, plans, goals, desires, intentions, actions, etc.), while in section 3 the execution flow is described in details; finally, we point out our intended future work and give some conclusions.

2 PRACTIONIST Agents

The main design objective of the PRACTIONIST framework is to support the programmer in developing agents *(i)* endowed with a symbolic representation about the environment in which they live, *(ii)* able to plan their activities in order to pursue some objectives, and *(iii)* provided with both proactive and reactive behaviours.

We chose to define the PRACTIONIST framework on top of JADE [1], a widespread platform that implements the FIPA[1] specifications. JADE provides some core services, such as a communication infrastructure, agent life-cycle management, and so forth. Apart from the aforementioned services, JADE alone does not endow agents with specific capabilities. The behavior abstraction of the JADE agent model allows simple integration of external software into one of the agent tasks.

PRACTIONIST agents are an extention of JADE agents and are deployed within JADE containers. Moreover the deliberation process and the means-ends reasoning are implemented within a JADE cyclic behaviour, while each intended means to pursue agents' objectives is executed in a separate thread.

In this paper we mainly focus on the conceptual model of PRACTIONIST agents, even though some implementation notes are given.

[1] http://www.fipa.org

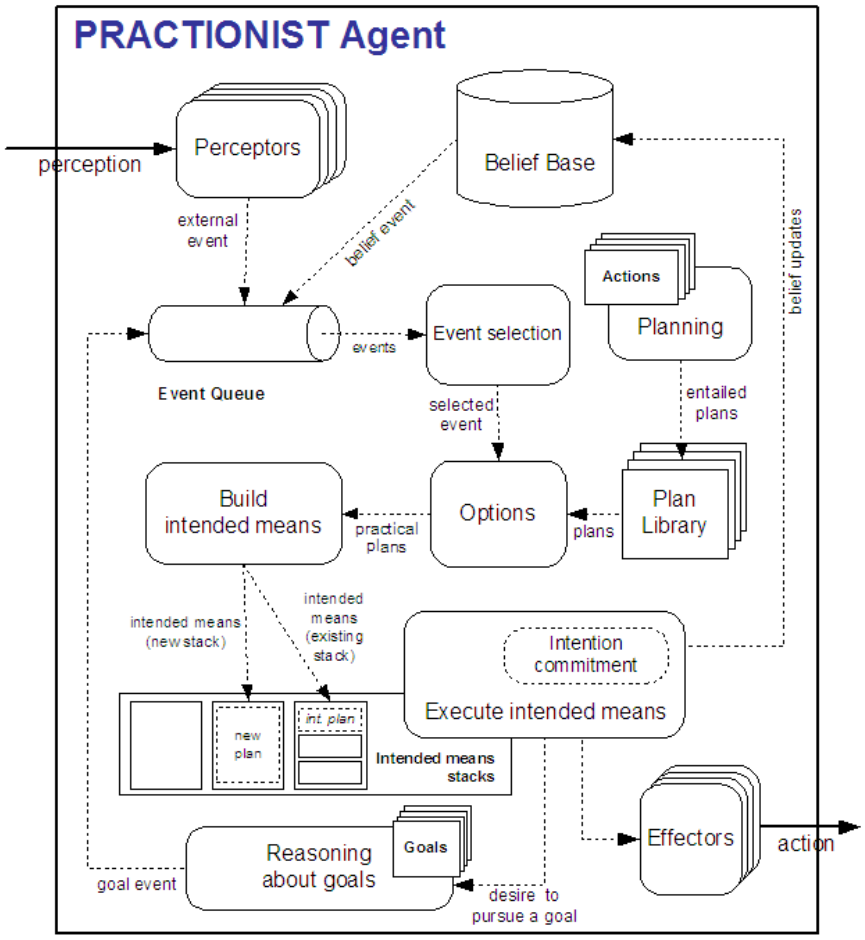

Figure 2: PRACTIONIST Agent Architecture

A PRACTIONIST agent is a software component with the following elements (figure 2): *(i)* a set of *perceptions* and the corresponding *perceptors* that listen to some relevant external stimuli; *(ii)* a set of *beliefs*, which represents the information the agent has got about both its internal state and the external world; *(iii)* a set of *goals* the agent wishes or wants to pursue, which represent some states of affairs to bring about or activities to perform; *(iv)* a set of *plans* that are the means to achieve its intentions; *(v)* a set of *actions* the agent performs to act over its environment; and *(vi)* a set of *effectors* that can actually execute the actions.

In this section we give an overview of the structure and the features of such components, while in section 3 we show how they interact to provide the overall behaviour of PRACTIONIST agents. As stated, both aspects will be referred to the above-mentioned blocks world example.

2.1 Beliefs

Rao and Georgeff's model [11] explicitly acknowledges that agents' information about the world is incomplete or incorrect, due to uncertainty and problems with perceptions and communication in their dynamic and possibly unpredictable environments. For this reason the BDI model refers to beliefs instead of knowledge, as beliefs are not necessarily true, while *knowledge* usually refers to something that is true [4]. According to this, an agent may believe true something that is false from the other agents' or the designer's point of view, but the idea is just to provide the agents with a subjective window upon the world.

Informally, in PRACTIONIST beliefs can be about either propositions or other beliefs. In other words, an agent may believe or not something described as a proposition (e.g. "it is raining in Rome"). It may also believe that some agent (it could be itself) believes something (e.g. "the agent Jim believes that it is raining in Rome").

In the PRACTIONIST framework beliefs are expressed through the modal operator

$$Bel(\alpha,\ \varphi)$$

which has two arguments, i.e. the agent α (the believer) and what it believes (φ, the fact). More formally, let F be a set of closed formulas of first-order logic (FOL) and $\hat{F}$ the set of formulas of the modal logic[2] based on F. Then, the fact believed by an agent is in turn expressed by elements of $\hat{F}$, such as φ, $\Diamond\vartheta$, $\Box\Diamond\tau$, $\forall x\,\psi\,(x)$ etc., where φ, ϑ, and τ are FOL closed formulas belonging to F, ψ is a predicate symbol, and $\Box$ and $\Diamond$ are respectively the classical modal operators of *necessitation* and *possibility*.

In PRACTIONIST predicates can also be represented by specifying the role of their arguments, as follows:

$$predicate(role1 : element1, role2 : element2, ..., roleN : elementN).$$

Moreover, we defined a special function for unifying terms that takes into account the role of terms within a predicate, instead of their order. Thus, as an example, $on(over : block4, under : block3)$ and $on(under : block3, over : block4)$ represent the same predicate.

We also assume the usual Hintikka-style schemata for *Bel*, that is the KD45 axioms corresponding to a "Weak S5 modal logic" (see [4] for more details), which for any agent α states that:

- (K) $Bel(\alpha, \varphi \Rightarrow \psi) \Rightarrow (Bel(\alpha, \varphi) \Rightarrow Bel(\alpha, \psi))$
- (D) $Bel(\alpha, \varphi) \Rightarrow \neg Bel(\alpha, \neg\varphi)$
- (4) $Bel(\alpha, \varphi) \Rightarrow Bel(\alpha, Bel(\alpha, \varphi))$
- (5) $\neg Bel(\alpha, \varphi) \Rightarrow Bel(\alpha, \neg Bel(\alpha, \varphi))$

In any moment, the set of the agent's beliefs forms its *Belief Set (BS)*, which corresponds to a snapshot of the world as perceived by the agent. Moreover for any $\varphi \in \hat{F}$, a PRACTIONIST agent α may express directly or not (i.e. through inference) three distinct belief states:

- $Bel(\alpha, \varphi)$: the agent α believes that φ is true;
- $Bel(\alpha, \neg\varphi)$: the agent α believes that φ is false;
- $Ubif(\alpha, \varphi) \equiv \neg Bel(\alpha, \varphi) \wedge \neg Bel(\alpha, \neg\varphi)$: the agent α does not have any belief about φ.

In PRACTIONIST it is also possible to define *nested beliefs*, such as:

$$Bel(\alpha, Bel(\beta, \varphi))$$

$$Ubif(\alpha, Bel(\beta, \neg\varphi))$$

which respectively mean that the agent α believes that the agent β believes that φ is *true* and the agent α does not have any belief about the fact that the agent β believes that φ is *false*.

In PRACTIONIST it is possible to link beliefs of an agent with others' beliefs or other elements, obtaining new entailed beliefs, through the *belief formulas* (BFs). In other words, BFs allow to define implicitly sets of beliefs that change generally over the time and dependencies among them. An example of belief formula follows:

$$Bel(tom, Bel(john, raining(where : rome))) \wedge Bel(tom, trust(who : john)) \Rightarrow Bel(tom, raining(where : rome))$$

Each PRACTIONIST agent is endowed with a Prolog belief base, where beliefs are asserted, removed, or entailed through inference on the basis of KD45 rules and user-defined belief formulas. Therefore, in any moment the agent's BS is composed of the beliefs that have been both directly asserted and inferred by means of the BFs and the other built-in theorems.

Referring to the blocks world example, in table 1 a subset of the initial agent's beliefs is shown. As the reader can notice, the initial set up of the blocks shown in figure 1 is formally represented in terms of facts that the blocks world agent (*self* from its point of view) believes true. Moreover, some general rules are included in terms of beliefs; their form is $bel(\alpha, \psi \Leftarrow \varphi)$ which is read "α believes that if φ is true, then ψ is also true".

[2]The syntax of the modal logic is defined by the following rules: (1) if $\varphi \in F$ then φ is a formula (that is $\varphi \in \hat{F}$); (2) if φ is a formula then so are $\Box\varphi$ (necessarily φ) and $\Diamond\varphi$ (possibly φ).

$bel(self, table(name : table1))$ % *The agent believes that table1,*
$bel(self, table(name : table2))$ % *table2 and table3 are tables.*
$bel(self, table(name : table3))$

$bel(self, on(over : block4, under : block3)))$ % *The agent believes that block4 is on block3,*
$bel(self, on(over : block5, under : block4)))$ % *block5 is on block4, and block8 is on block5.*
$bel(self, on(over : block8, under : block5)))$
...
$bel(self, clear(obj : block10))$ % *The agent believes that block10, block7 and block8 are clear.*
$bel(self, clear(obj : block7))$
$bel(self, clear(obj : block8))$

% *Any agent believes that B1 is over B2 if B1 is just on B2 or*
% *if B1 is just on B3, which in turn is over B2*
$bel(Whoever, over(up : B1, down : B2) \Leftarrow on(over : B1, under : B2))$
$bel(Whoever, over(up : B1, down : B2) \Leftarrow on(over : B1, under : B3) \wedge over(up : B3, down : B2))$

Table 1: A subset of blocks world agent's beliefs.

2.2 Plans

In PRACTIONIST framework plans represent an important container in which developers define the actual behaviors of agents. Each agent may own a declared set of plans (the *plan library*), each specifying the course of acts the agent will undertake in order to pursue its intentions, or to handle incoming perceptions, or to react to changes of its beliefs. The structure of a plan is described in table 2.

Identifier	Unambiguous (within each agent) identifier of plans
Trigger event	If this event matches the selected event, this plan can be activated. In this case the plan is defined as *practical*.
Context	A formula of the set $\hat{F}$ that, when believed true by the agent, makes *applicable* a practical plan, so that the agent may select it to pursue its objectives.
Success condition	When the agent believes that this condition holds, the plan ends with success, regardless its execution state.
Cancel condition	When the agent believes that this condition holds, the plan ends with failure, regardless its execution state.
Body	Set of acts that are performed during the execution of the plan. The body define the actual behavior of the plan.
Invariant	Condition that must remain true during the execution of the plan. As soon as it becomes false (at least according to the agent's point of view), it will try to restore it.
Belief updates in case of success	Effects of this plan, in terms of belief updates in case the plan ends with success.
Belief updates in case of failure	Effects of this plan, in terms of belief updates in case the plan ends with failure.

Table 2: The structure of PRACTIONIST plans.

The body is an *activity*, which is a set of *acts*, such as *desiring* to bring about some states of affairs or to perform some action, *adding* or *removing* beliefs, *sending* ACL messages, *doing* an action and so forth.

Regarding the blocks world running example, one of the plans we developed to provide the agent with the capability of receiving and handling the request for ordering the blocks is the *TopLevelPlan* that is shown in table 3 where the body is expressed in pseudo code as example. As the reader can notice, this plan will get practical as soon as the blocks world agent (endowed with a specific perceptor to receive ACL messages) is requested to order the blocks.

In section 3 we illustrate how the plans and their components are used during the execution

Identifier	TopLevelPlan
Trigger event	$msg(request(action(order(blocks : BlockList))))$
Context	$ready(who : self)$
Success condition	$on(over : BlockList[i], under : BlockList[i-1])\ i = 1, 2, ..., N$
Cancel condition	$not(ordering(blocks : BlockList))$
Body	$add(ordering(blocks : BlockList))$ $Under = BlockList[0];$ $for(i = 1; i \leq BlockList.length; i++)\{$ $Over = BlockList[i]$ $desire(achieve(on(over : Over, under : Under)))$ $Under = Over$ $\}$
Invariant	$ableToOrder(who : self)$
Belief updates in case of success	$add(happy(who : self))$
Belief updates in case of failure	$del(ordering(blocks : BlockList))$

Table 3: A plan of the blocks world agent.

flow and how the execution of plans affects the overall behaviour of the agents.

2.3 Actions

In PRACTIONIST, actions are described by tuples of five elements: (1) the *name* of the action; (2) *arguments*, which are the objects each action acts over; (3) *results*, which are some kind of direct responses received from the environment; (4) *preconditions*, which should be satisfied before executing the action; and (5) *effects* (for both successfully and failing action execution), which are the state of affairs that will be true or false after executing the action (as long as preconditions were satisfied). It should be noticed that arguments and results are objects, while preconditions and effects are elements of the set $\hat{F}$ defined above.

Actions are implemented in PRACTIONIST through Prolog structures or Java classes that include the above-mentioned elements. An example of action description from the blocks world agent follows:

```
action(move(block: Block, to: To),
       inputs: [Block, To],
       outputs: [],
       preconditions: [ on(over: Block, under: From),
                        clear(obj: To), clear(obj: Block) ],
       success: [ -clear(obj: To), -on(over: Block, under: From),
                  +clear(obj: From), +on(over: Block, under: To) ],
       failure: [])
```

It states that the action *move* takes two arguments as inputs (that is respectively the block to move and the block where it has to be moved over). Moreover, the preconditions *on(over: Block, under: From) clear(obj: To)*, and *clear(obj: Block)* must be satisfied before performing the action in order to have a proper execution. Finally, once the action has been executed, in case of success the agent will believe that both *clear(obj: To)* and *on(over: Block, under: From)* are false, while it will believe that both *clear(obj: From)* and *on(over: Block, under: To)* are true. Otherwise, in case of failure in executing the action, no updates of the agent's beliefs has to be done.

It is worth mentioning that planning attitudes and decision making of PRACTIONIST agents rely on such action description elements, especially preconditions and effects.

2.4 Goals

In the PRACTIONIST framework, a goal is an objective to pursue and we refer to it as an abstraction to make the distinction between the state of affairs to achieve and the way to do it. Besides,

we use goals as a means to transform desires into intentions through the satisfaction of some properties. In other words, our agents are programmed in terms of goals, which then will be related to either desires or intentions according to whether some specific conditions are satisfied or not.

In PRACTIONIST the following two families of goals were defined: *state goals*, which refer to some states of affairs the agent desires/intends to bring about, or cease, or preserve, or avoid; *perform goals*, which are not related to some world states but to some activities the agent desires/intends to perform. Moreover, we provided PRACTIONIST agents with the capability of managing and reasoning about the following state goals:

- *achieve*, which represents what kind of world state to bring about;
- *cease* (as opposed to achieve), which represents a world state an agent wants to stop;
- *maintain*, which has the purpose to observe some world state and continuously re-establish this state when it does not hold;
- *avoid* (as opposed to maintain), which has the purpose to observe some world state and continuously prevent it.

We represent states of affairs by means of goal formulas, which formally describe what the agent wants to achieve, cease, maintain, or avoid. More precisely, goal formulas are represented by closed formulas of the set F defined above. Moreover, let $\hat{F}$ be the set of formulas of the modal logic based on the set F and Φ be the set of all goals the agent can pursue (i.e. according to its design). Then, the *success condition function (scf)* is defined as follows:

$$\sigma : \Phi \rightarrow \hat{F} \tag{1}$$

such that for each goal there is only one success condition which is defined by a modal logic closed formula. Therefore, for any $\varphi \in F$ we propose the following success condition for the state goals defined above:

$$\sigma(achieve(\varphi)) = \varphi \tag{2}$$

$$\sigma(cease(\varphi)) = \neg\varphi \tag{3}$$

$$\sigma(maintain(\varphi)) = \Box\varphi \tag{4}$$

$$\sigma(avoid(\varphi)) = \Box\neg\varphi \tag{5}$$

We also defined the properties of *inconsistency* and *entailment* for goals. More precisely, we define a goal G_1 as *inconsistent* with a goal G_2 if and only if when G_1 succeeds, then G_2 fails. On the other hand, a goal G_1 *entails* a goal G_2 (or equivalently G_2 *is entailed by* G_1) if and only if when G_1 succeeds, then also G_2 succeeds. These properties of goals are used by PRACTIONIST agents when reasoning about goals during the deliberation phase, as explained in section 3.

Moreover, since PRACTIONIST agents are compliant to the BDI model, which means that it is relevant what they believe, we defined goals and their properties according to what agents believe, that is, they will reason about goals on the basis of what they believe about their properties. Thus, as an informal example, an agent will believe that a goal has succeeded if it believes that the success condition returned by the *scf* is true. This holds for all goal properties.

3 Execution Flow

In this section, we illustrate the execution flow of PRACTIONIST agents in terms of relationships among the abstractions described above (i.e. plans, goals, beliefs, etc.). Referring to the agent architecture (see figure 2), an agent cyclically performs the following steps:

1. it searches for stimuli from the environment through the perceptors, which transform perceptions into *events* (we call them *external events*), which are put into the *Event Queue*. Belief updates and commitments to some goal (see below) produce *internal events*, which are collected in the Event Queue as well.

   In the blocks world example, the agent will receive the following ACL message, by which another agent requests it for ordering the blocks in a specified order, that is *block*1 over the *table*3, *block*2 over *block*1, and so forth.

```
(request
    :language FIPA-SL0 :protocol FIPA-Request
    :ontology blocks-world-ontology
    :content (action
               (agent-identifier :name bwa@foo.com
                                 :addresses (sequence iiop://foo.com/acc))
               (order(blocks: [table3, block1, block2, ... , block10])))))
```

2. it selects an event from the queue, through the logic provided in the *Event Selection*. As a default choice, events are extracted from the queue according to the order of insertion, so that the first in is the first out (FIFO). However, the programmer can specify his/her own criterion to select events in the queue or can program the agent to dynamically change such a logic at runtime. Once selected, each event is removed from the queue.

   In the blocks world example, let us suppose that the event corresponding to the above-mentioned received message is extracted from the queue and then handled as follows;

3. for each selected event, it selects practical plans from the *Plan Library*, which are those plans whose trigger event matches the selected event (*Options* in figure 2); if the selected event is related to a goal and no plan has been triggered, an automatic planning (described in some details below) is performed on the basis of available action descriptions in order to build a new dynamically-generated plan which is able to pursue that goal (*Planning* in figure 2). If the planner is not able to figure out any plan, the calling plan will fail, so that the selected event will have not been successfully managed.

   In the blocks world example, we have defined a plan (i.e. $TopLevelPlan$) that will be activated by the above-mentioned message, as it has the following trigger event (see in table 3):

   ```
   msg(request (action
                  (agent-identifier :name bwa@foo.com)
                  (order(blocks: BlockList))))
   ```

   Once the plan is triggered, the following substitution is made:

   ```
   BlockList = [table3, block1, block2, block3, ..., block8, block9, block10]
   ```

4. among practical plans, the agent detects the applicable ones, which are those plans whose context is believed true by the agent. Then, the agent selects one of such plans (we call it the main plan). As a default, the firstly declared applicable plan is selected, even though the logic for selecting applicable plans can be customized by the programmer. Then the agent builds the intended means (*Build Intended Means*), which represents the means the agent has just chosen and committed to in order to satisfy a goal, or to react to a perception or a change in its beliefs. Therefore, the intended means will contain the main plan and the other alternative practical plans (see figure 3).

   If the event selected at the step (2) concerns with a goal, this means that some executing intended means has generated it after the deliberation phase (see below). Thus, the selected plan is put on top of such an intended means (figure 3a). On the other hand, in case of external events or belief update events, a new intented means stack is created (figure 3b);

5. all intended means stacks are concurrently executed in separate threads. Particularly, the main plan at the top of each stack is extracted and executed (see *Execute intended means* in figure 2). Therefore, the agent will behave according to the type of acts within the executing plan, which in turn will fail as soon as one of its acts fails. Possible acts are:

   (a) *desire* to bring about some state of affairs or perform an action, which is expressed as a goal. The agent processes such a desire in order to figure out whether it can be promoted to an intention or not. If it can, a corresponding internal event is created, put in the Event Queue, and then considered at the step (2) in one of the following cycles. Then, the corresponding intended means is built and nested inside the one that contains the

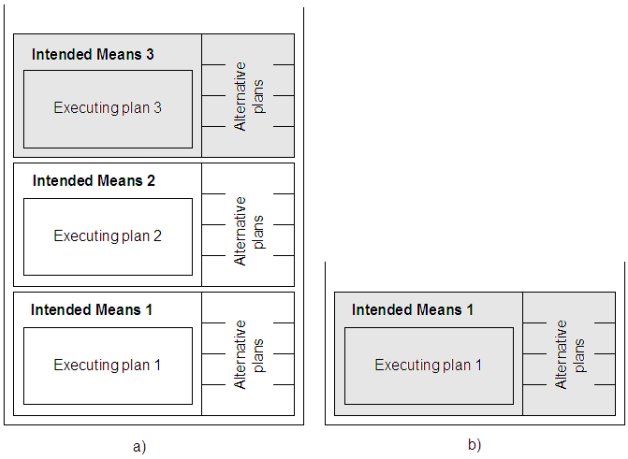

Figure 3: The structure of Intended Means Stacks: a) the intended means is put on top of executing stack; b) the intended means is put in a new stack.

above-mentioned desire invocation. After executing the nested intended means, if the agent does not believe that the goal has succeeded, then the desire act fails; otherwise, the goal will be satisfied and the act will succeed.

Suppose that the agent α starts its deliberation process and generates the desire to pursue the goal $G = achieve(on(over : T1, under : T2))$, as it is shown in the body of the plan in table 3. This means that it would like to commit to G, whose success condition is $on(over : T1, under : T2)$. The agent α will perform the following steps in the context of its deliberation activities:

i. it will check if the goal G is inconsistent with some of current active goals, which are those goals which the agent is already committed to. Then, if G is inconsistent with at least one of those goals, G will remain just a desire and the agent will not "intend" it.

ii. If the goal G is not inconsistent with active goals, then it can be actually pursued. But before making any means-ends reasoning, the agent α will check if it believes that the goal G has just succeeded or if the goal G is entailed by some of current active goals. If so, there is no reason to really pursue the goal, that is the agent does not need to make any means-ends reasoning to figure out how to achieve such a goal (refer to steps 3 and 4).

Otherwise, wishing to bring about G will become an intention of the agent α, which will try to figure out a plan to achieve this intention.

(b) *do an action*: if its preconditions are satisfied (at least according to the agent's beliefs), the agent executes it with the proper inputs and gets both the outputs and the general overcome of the action (that can be *true* or *false*). If the preconditions are not satisfied or the outcome is *false*, the action fails. Finally, the effects of such an action are applied to the Belief Base after the execution. It should be noted that actions are actually executed by some effectors. Thus, the agent will search for a proper effector that is able to execute such an action and then delegate the actual execution to it;

(c) *add* or *remove* beliefs: as soon as it is executed, the corresponding *belief updated event* is generated. This event will be then handled in the following cycles;

(d) *query* over the Belief Base according to several criterion;

(e) *send an ACL message* to other agents;

(f) *wait for external messages*, by specifying an abstract structure (a template) of them. Thus, as soon as a message is received by the agent, if the message matches the template, this intended means will continue its execution;

(g) *wait for successful goals*, which lets the agent synchronize its activities with other intended means, but at the goal level. In other words, there is not an explicit synchronization among plans or intended means. Instead, some intended means and their executing plans can be directly synchronized with some ends (the intentions).

Actually, several other acts are possible when executing intended means and the corresponding plans. Here for conciseness, only some of them have been reported.

During the execution of a plan, if the agent believes that the success condition (see table 2) holds, the plan ends with success regardless its execution point and state. On the other hand, if the agent believes that the cancel condition is true, the plan ends with failure. Thus, referring to the blocks world agent and the *TopLevelPlan* described in table 3, if during the execution of such a plan the required final order of blocks is achieved (for example due to some external reasons, e.g. other agents doing the work of ordering the blocks), the agent will stop executing the plan, being successful in achieving its goals. Analogously, if the agent suddenly believes that the predicate $ordering(blocks : BlockList)$ is false, the agent will stop executing the plan, but in this case it will have failed in pursuing its objectives.

Moreover, while executing the plan the agent checks the condition to be maintained (e.g. $ableToOrder(who : self)$ in table 3): if it is believed false, the agent will try to bring it about, by desiring and possibly intending to achieve it. If the agent succeeds in doing that, the plan will continue executing, otherwise it will fail. When the plan has completed its execution, the agent will update some beliefs, according to whether the plan has succeeded or not.

It should be noticed that, when the executing plan fails, since a PRACTIONIST agent has a strong commitment to handling the selected event, it selects another plan from the above-mentioned alternative practical plans and checks if it is applicable, that is the context is believed *true*. Then, if some other applicable plan exists, the agent replaces the failed plan with it (which becomes the new main plan) in the executing intended means; otherwise, the whole intended means fails and in turn the plan below fails too, and so forth.

4 Planning capabilities

As stated, PRACTIONIST agents are able to dynamically build plans in order to pursue state goals (see *Planning* function in figure 2), in case of no plan of the library can be activated by some selected event. These planning capabilities are based on a backward search algorithm in the states space [12] implemented in Prolog. Thus, the issue of pursuing a state goal can be viewed as a planning problem where the initial state is represented by the agent's beliefs and the domain is represented by its actions. The planning component solves such a problem by producing a sequence of actions to be performed in order to satisfy that goal. These actions will be part of the body of a dynamically-generated plan, which temporarily becomes an element of the agent's plan library. This plan may be composed by a set of either abstract or concrete actions, according to whether they have some variables as inputs or not. In abstract actions, variables will be instantiated with some outputs of previous actions.

As soon as each action is performed, its postconditions are applied to update the agent's beliefs. The actual effects of the overall plan should be the satisfaction of the initial goal (the ends), as the plan is the right means to pursue it.

The plan can fail when action preconditions are not satisfied due to some unexpected changes of beliefs or the execution of some action fails.

It should be noticed that since the planning component could be time-consuming, the developer can disable it. Moreover, the developer can set the maximum number of actions for dynamically-generated plans and how planned actions will be performed, i.e. through either effectors or perform goals.

As an example, regarding to the blocks world problem, and specifically referring to the plan in table 3, once selected and executed it, the agent will 'desire' to $achieve(on(over : block1, under : table3))$, $achieve(on(over : block2, under : block1))$, and so forth. Let us suppose that the goal $achieve(on(over : block1, under : table3))$ has just become an intention. The planning component

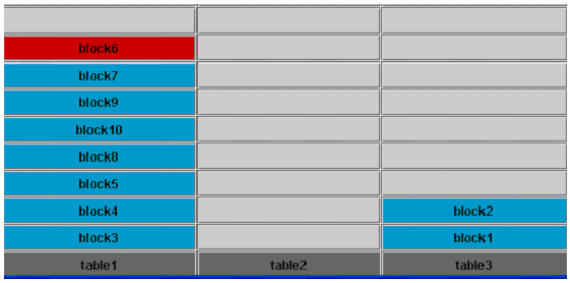

Figure 4: An intermediate situation of blocks world example

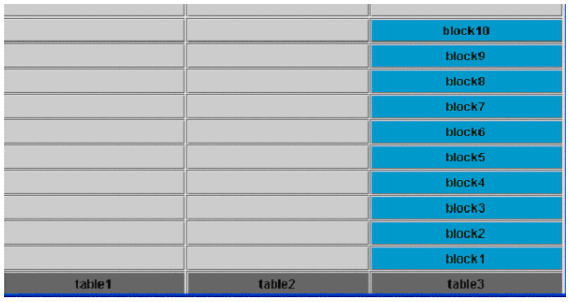

Figure 5: The intended final situation in the blocks world problem.

of PRACTIONIST agents is able to figure out a sequence of *move* actions (defined in section 2.3) that lets it $achieve(on(over : block1, under : table3))$, as follows:

```
move(block: block10, to: block8)
move(block: block9, to: block10)
move(block: block7, to: block9)
move(block: block6, to: block7)
move(block: block1, to: table3)
```

Then the agent will desire to pursue the goal $achieve(on(over : block2, under : block1))$. Thus, as soon as the goal becomes an intention, the agent will figure out a plan with only one action, that is

```
move(block: block2, to: block1)
```

In figure 4 the situation of blocks after executing the above two dynamically-generated plans is shown, while in figure 5 the final requested configuration of blocks is depicted.

5 Conclusions and Future Work

In this paper we presented PRACTIONIST, a framework we have been developing for the implementation of agents according to the BDI model of agency. PRACTIONIST implements the practical reasoning theory proposed by Bratmam, trying to provide developers with usable abstractions and processing capabilities.

In this direction, we believe that with respect to some other BDI agent platforms, our approach provides a more clear separation between the ends the agent wishes/wants to bring about and the means to do it. We assume an important role of goals as an abstraction to let the agent reason about both desires and intentions. Actually we have developed a formal theory (not presented here) for PRACTIONIST goals based on modal logic over first-order logic.

Moreover, our framework provides a very expressive way to represent and reasoning about beliefs through modal logic formulas. The framework allows agents to dynamically build plans in case of no plan available in the library can be activated by some selected event.

Some further work should be done with respect to the several issues that a BDI model involves; our intention is to improve the execution flow by adding some functionalities like timing, new acts, and so on, that could help in the successful application of our framework in real problems.

Acknowledgments. This work is partially supported by the Italian Ministry of Education, University and Research (MIUR) through the project PASAF.

References

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - a FIPA-compliant agent framework. In *Proceedings of the Practical Applications of Intelligent Agents*, 1999.

[2] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, 1987.

[3] P. Busetta, Ralph Rnnquist, Andrew Hodgson, and Andrew Lucas. Jack intelligent agents - components for intelligent agents in java, 1999.

[4] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, Cambridge, 1980.

[5] M. Dastani, B. van Riemsdijk, F. Dignum, and J. Meyer. A programming language for cognitive agents: Goal-directed 3apl, 2003.

[6] Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dMARS. volume 1365 of *LNAI*, pages 155–176. Springer, 1997.

[7] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proc. of AAAI-87*, pages 677–682, Seattle, WA, 1987.

[8] Marcus J. Huber. Jam: A bdi-theoretic mobile agent architecture. In *Agents*, pages 236–243, 1999.

[9] Jaeho Lee, Marcus J. Huber, Patrick G. Kenny, and Edmund H. Durfee. UM-PRS: An Implementation of the Procedural Reasoning System for Multirobot Applications. In *Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS)*, pages 842–849, Houston, Texas, 1994.

[10] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: Implementing a bdi-infrastructure for jade agents. *EXP - in search of innovation (Special Issue on JADE)*, 3(3):76–85, 9 2003.

[11] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[12] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[13] Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.