

Augmenting BDI Agents with Deliberative Planning Techniques

A. Walczak, L. Braubach, A. Pokahr, W. Lamersdorf

Distributed Systems and Information Systems
Department of Informatics, University of Hamburg
D-22527 Hamburg, Germany

Abstract. Belief-Desire-Intention (BDI) agents are well suited for autonomous applications in dynamic environments. Their precompiled plan schemata contain the procedural knowledge of an agent and contribute to its performance. The agents generally are constrained to a fixed set of action patterns that are based on their current goals not on the future of the environment. Planning techniques can provide dynamic plans regarding the predicted state of the environment. We augment a BDI framework with a state-based planner for operational planning in domains where BDI itself is not well applicable. For this purpose, the requirements for the planner and for the coupling with a BDI system are investigated. An approach is introduced where a BDI system takes responsibility for plan monitoring and re-planning and the planner for the creation of plans. A fast state-based planner utilizing domain specific control knowledge retains the responsiveness of the system. In order to facilitate integration with BDI systems programmed in object-oriented languages, the *planning problem* is adopted into the BDI conceptual space with object-based domain models. The application of the hybrid system is illustrated using a propositional puzzle and a multi agent coordination scenario.

1 Introduction

BDI is a well established model of agency [1] based on the Theory of Practical Reasoning [2]. Early BDI-systems have been devised with the idea in mind to overcome the poor performance of propositional planners controlling agents in dynamic environments at that time. The systems are based on two central ideas. One of them is the reactive planning, comparable with hierarchical planning systems [3], the other is deliberation [4, 5].

Planning, an approach central to Artificial Intelligence (AI) research, is substantial for rational agent behavior. It is a method that aids agents in solving complex problems in synthetic and natural environments. Although planning systems are devised for means-end reasoning and are capable to find actions that achieve goals, they are less useful to decide, which goals to pursue [6].

Due to advances in planning techniques and understanding of *planning problems*, it seems reasonable and interesting to combine the strength of flexible means-end reasoning given by deliberative planners with the timely reactivity

and goal deliberation capabilities carried by BDI systems. It is also interesting to analyze suitability of planning approaches to BDI agents in real world applications.

In order to benefit from both paradigms one needs to consider the strengths of both paradigms and where they could be applied to a hybrid system in a practical setting. There are multiple ways to compose the systems and the outcome is different dependent on their properties and features. As stated above, hierarchical planning techniques are comparable with BDI. Their strength lies in the evaluation of future environmental states and a constructed proof that a course of action will achieve goals under the preconditions. On the other hand, BDI systems handle dynamic environments more efficiently and are capable of both: reactive behavior, and keeping long term goals in their intentional structure. They sacrifice the optimality and correctness of their planning algorithms for performance.

Both paradigms deal with action generation and share common ideas, so there is concern which parts of a control and planning problem will be delegated to a planner and to the BDI subsystem. This determines the choice of the BDI component and the planning algorithm. The overall architecture depends strongly on those choices.

A hybrid system can be built twofold. The planner may be applied to produce long term plans and to handle single parts to a reactive BDI subsystem for the execution. This approach invokes serious performance concerns especially in dynamic environments where continuous changes force the planner to re-plan - a process with performance penalties comparable to planning itself. Generally, planning algorithms have been devised for one shot planning and are well suited for a solution of a particular planning problem. They are rather less useful to maintain long term agent intentions.

The other way is to augment the BDI system with a relatively simple planner that is invoked from the BDI controller and used for the purpose of creating short-term plans that need a proof of correctness. The last approach is used in this work to join the best from both paradigms.

The remainder of this paper proceeds as follows. In Section 2 we define the concept of a *planning problem* used in this paper. Section 3 discusses the choice of a planning algorithm. In Section 4 we propose a way to integrate a planning component into a BDI framework. Section 5 presents two application examples of the hybrid system. Related work is presented in Section 6 and a conclusion is given in Section 7.

2 Planning Concepts

The basis for planning is given in the form of a *planning problem*. In order to represent a planning problem one needs at least to describe states of the world and how these states may change due to agent's actions. In a restricted classical view, this can be given by a model of a state-transition system $\Sigma = (S, \mathbb{A}, \gamma)$ where S is the set of states, \mathbb{A} - the set of actions and $\gamma : S \times \mathbb{A} \rightarrow S \cup \{\perp\}$

is the transition function mapping a state and action to another state. \perp is the illegal state being a result of a not applicable action. The planning problem is given by a triple $\mathcal{P} = (\Sigma, s_0, g)$ where $s_0 \in S$ is the initial state and g is the description of a goal state inducing the set $S_g := \{s \in S \mid s \text{ satisfies } g\}$ [7].

The following definition of a planning problem has been found useful for the purpose of this work. It deviates from a standard definition introducing utility functions interpreted as agent desires D . The other difference is an object-based representation of states build upon the sets O , A and V . The initial state s_0 has been understood as the current state s_c to reflect the fact that the planning takes place at runtime.

Definition 1. A state is a tuple $s = \langle G, \sigma, s_p, a_s \rangle$ where:

- G is a stack of agent goals in that state and each goal is a difference function $g_{\in G} : S \rightarrow \mathbb{R}$ revealing an approximate distance from the state to the goal in a common weighted measure.
- $\sigma : O \times A \rightarrow V \cup O$ is a partial function assigning values to the attributes of objects. O , A and V are taken from the model of a planning problem described below.
- s_p is the parent state of this one.
- a_s is the action applied to s_p state yielding s .

The planner reasons about states of the environment and agent’s own inner mental states. The environment is represented by object-based models given by an assignment function σ over a set of objects O and attributes A given below. For short-term planning only the immediate goals are interesting for the planner. A stack of goals G allows for ordered hierarchical decomposition. It is assumed that goals at this level of planning have been filtered through the deliberative BDI process and are consistent with each other.

Definition 2. A planning problem is a tuple $P = \langle \mathbb{A}, s_c, D, O, A, V \rangle$ where:

- \mathbb{A} is a set of all actions available to the agent.
- s_c is the current state of the environment.
- D contains agent desires, in respect to possible solutions, assumed not to change within the short scope of operational planning.
- O contains the objects from the planning domain with attributes from the set A taking values from the set V .

Desires are inverse utility functions $d_{\in D} : S \rightarrow \mathbb{R}$ on the states and reflect (not necessarily coherent) mental attitudes of the agent. Both desires and goals influence actions chosen by the agent, but only goals represent concrete points in the future state space, that the agent has to achieve. They also have direct impact on the choice of future goals.

Each action $a \in \mathbb{A}$ is a tuple $a = \langle p_a, \gamma_a, \omega_a \rangle$ with p_a being a predicate over a state telling if the action is applicable. γ_a and ω_a are transition functions over the set of goals and attribute assignment functions respectively. If action a is applicable to s , the application transforms it to a new child state $s' = \langle \gamma_a(G), \omega_a(\sigma), s, a \rangle$. This yields a new set of goals $G' = \gamma_a(G)$ and new attribute assignment function $\sigma' = \omega_a(\sigma)$.

3 Planning Algorithm

Planning has been assumed to be a higher cognitive activity than reacting and controlling behavior and has been granted more computational resources. We introduce a planner at a level below BDI control and deliberative behaviors. Given such an architectural design, the choice of a planning algorithm is restricted to a particular subset. With an advanced BDI system, one is equipped with reasoning and a strong conceptual framework, so there is no need to duplicate the functionality of both. To guide our choice of planning algorithms, the latest results from planning competitions [8] have been used. Planners entering such competitions have been tested on many standardized planning examples. Better performance indicates the right choice of an algorithm. Changing demands of successive planning competitions favored approaches that are easily expandable to new planning concepts.

In order for the system to remain responsive to circumstances that induced the planning task, the planner needs to operate under tight timing constraints. This fact emphasizes performance, not the generality or cognitive adequacy of a planning algorithm. State-based planners augmented with domain specific knowledge have been shown superior to partial-order planners, in that respect [8]. They are also easily applicable to many planning domains including propositional, numeric, timed, continuous and contingent domains.

The following planning algorithm is a state-based search algorithm working on an agenda of states. The main loop examines states from the agenda and expands them searching through the state space for one that achieves some or all of the goals. It terminates, when the agenda gets empty or when the time limit intended for planning is exceeded (cf. line 4). Please recall that states are tuples in the form $\langle G, \sigma, s_p, a_s \rangle$. Each state has an assigned inverse utility estimate through the function $e : S \rightarrow \mathbb{R}$.

```
PLAN( $\mathbb{A}, s_c, D, T$ )
1   $best \leftarrow s_c$ 
2   $e(best) \leftarrow \infty$ 
3   $agenda \leftarrow \{s_c\}$ 
4  while  $|agenda| > 0$  and  $t_c < T$ 
5  do  $s \leftarrow pop(agenda)$ 
6     if  $e(s) < e(best)$  and  $|G_s| \leq |G_{best}|$ 
7     then  $best \leftarrow s$ 
8      $Options \leftarrow generateOptions(s, \mathbb{A})$ 
9     for each  $\{a = \langle p_a, \gamma_a, \omega_a \rangle \mid a \in Options\}$ 
10    do  $G' \leftarrow \gamma_a(G)$ 
11        $s' \leftarrow \langle G', \omega_a(\sigma), s, a \rangle$ 
12        $removeSatisfiedGoals(s')$ 
13        $e(s') \leftarrow inverseUtility(s', D) + goalsDistance(s', G')$ 
14        $insert(s', agenda)$ 
15 return  $best$ 
```

For many real world problems it is impossible for an agent to enumerate all action instances. Even in discrete domains, the number of possible actions becomes prohibitive. In a concrete design one would delegate the task of generating a set of applicable actions to a separate *option generator* component. Based on a set of action schemata and the actual situation it generates all applicable actions (cf. line 8). Lines 10 and 11 apply the transition function. In line 12 all satisfied goals are removed from the top of the goal stack of the new state. In order to sort states by their utility, a new estimate is calculated in line 13 before the state is inserted into the agenda. The variable implementation of *pop* and *insert* methods allows for different state exploration strategies.

There are two basic types of actions. *Concrete actions* modify object models in a state by changing the σ assignment function. In this case, the goal manipulation function is an identity function $\gamma_a = id$. The other type of actions are abstract ones called *decomposition methods* that remove a goal from the top of the goal stack G and replace it by a list of totally ordered subgoals. Such an action has generally no effect on the models of environment.

The domain specific knowledge used to guide the planner is hidden in the action applicability predicates p_a , in the goal distance functions $g \in G$, and in the inverse utility functions $d \in D$ (cf. sec. 5). Including a reference to a parent in a state allows for complex temporal conditions, like safety and liveness ones, over the course of actions [9]. The estimate is computed using the heuristic function below:

$$e(s) = \sum_{d \in D} d(s) + \sum_{g \in G} g(s)$$

This choice of heuristic embodies the judgments that optimal solutions are anyway computationally expensive.

On success, the planning algorithm returns a state where every immediate goal of the agent posed to the planner is satisfied. In case, where the algorithm fails to find a complete plan, the best plan in respect to the estimate and number of unachieved goals is returned.

The emphasis has been put on performance and use of domain specific knowledge. The difference to other linear planning algorithms based on state-space search is the explicit representation of agent intentional structure manipulated during the process of planning. E.g. the STRIPS planner reasoned with a goal stack, this planner reasons *about* a goal stack in each planning step. This makes this planner similar to a state-based HTN planner with ordered decomposition. The use understanding of agent desires as heuristic and utility functions used to guide the planning process is specific to this work.

4 Integration with a BDI framework

A generic BDI agent architecture (based on [10,4]) is illustrated in Figure 1. One of the central components is the goal hierarchy housing higher level goals including desires and important events. The goals may depend on each other in

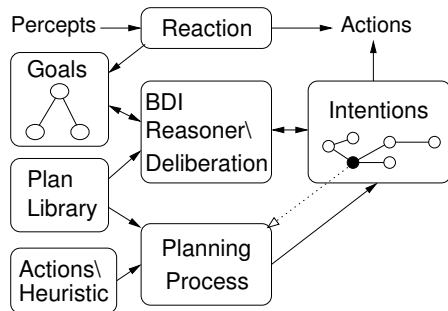


Fig. 1. Generic BDI architecture with a planner. Planning is an activity of the agent (filled circle) and produces new intentions.

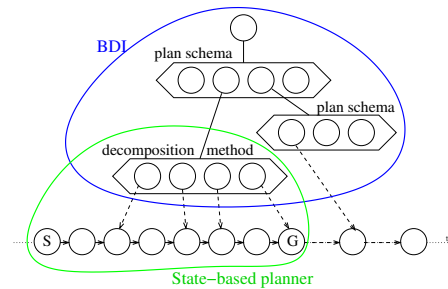


Fig. 2. Schematic illustration of a plan in the hybrid system.

a forest like structure. This component includes the upper part of the intentional structure of an agent and provides it with motivation and reason for action.

Goals are used by the BDI reasoner to chose among plan schemata and create the action structure on the agenda of intentions. In object-oriented BDI frameworks plan schemata stored in a plan library are defined by classes of plans. The intentional structure is given by current plan instances. Another common component of the architecture is a reactive subsystem. Triggered by belief changes or a percept, it generates new goals for the reasoner. At the meta-level a goal deliberation process may influence the reasoning and execution. The responsibility of goal deliberation is to analyze dependencies and to modify the intentional structure accordingly to agent's preferences.

4.1 Preparing a planning problem instance

For a specific goal the BDI reasoner may use the planner to create a dynamic plan. This is done, whenever all existing plan schemata have failed or there is an explicit preference for the planner in respect to the goal as may be specified in the agent description (cf. sec. 5). First an instance of a *planning problem* must be created. This requires mapping of agent goals and beliefs to the specific planner representation. All planner goals represent a distance measure defined over states having object based models described above. There is no BDI system, known to the authors, that define goals using a distance measure with respect to different attribute types and dimensions. The mapping can be done automatically, but this does not utilize the domain knowledge of the programmer, who may provide adequate weights for attribute dimensions and distance measure functions for goals that cannot be represented in the attribute-value form.

The states in the planning process are stored in a tree like database, that may consist of a large number (millions) of states, each storing a subset of beliefs. For planning being efficient in space and time only the relevant beliefs need to be stored in a state. The choice of action schemata determines, which beliefs

will be changed, and which will remain unaffected by the process. Object instances must be decomposed into their attributes in order to avoid copying solid objects including not relevant attributes. Due to the complexity, this process is performed manually in the prototype implementation.

E.g. in the loader dock scenario (cf. sec. 5) the position of a worker and the packet being held by it change. They are affected by agent actions used in its plans and must be reflected in a state. On the other hand, the domain time model stores movements of other workers and their different positions with respect to time. This model is not affected by plan actions and may be accessed as a belief of an agent. In the blocks' world example, the relative position of a block is relevant for the planning process, but its color remains unregarded.

The actions work directly on the state representation. The central point of an action is the application method, including the applicability predicate and transition functions on the stack of goals and object descriptions (cf. sec. 5). This method summarizes all domain knowledge needed for planning with respect to the action. A wide range of conditions and search control knowledge types may be specified this way, because they are represented in the programming language of choice. The same is true in respect to effects of an action. It is possible to derive this knowledge from already existing plan schemata of the BDI systems, but for concrete examples studied, it showed not to be sufficient to aid the planner in an effective way. The actions are implemented in our approach by the application designer. Heuristics reflect the desires of an agent regarding the created plans. They must be devised and implemented for the particular planning domain.

4.2 Processing created plans

Prepared problem instances are handled to the planner and executed in a planning process like other agent's plans. The planner delivers created plans directly into the intentions structure and does not store them in a plan library. If plans generated by the planner were general enough in their nature, the designer of agent's knowledge could also precompile such plans in advance. Because plans are created at low-level, their parameters are tightly bound and they are applicable only to a particular situation. Storing such plans in the plan library would clutter it with instances that are only executed once.

The resulting intentional structure may be seen in Figure 2. The upper fragment contains a partially expanded branch of the BDI goal structure. One of the goals has been assigned to the planner. It starts in the current state S and uses a decomposition method to place intermediate points in the search space as subgoals on the goal stack. The subgoals are removed from left to right as they are achieved. The created plan is placed in the intentional structure for execution and has a BDI goal as the parent node. The lower sequence represents the envisioned sequence of states, which should be attained at the execution time successively.

In a dynamic environment, conditions change in an unexpected way. Monitoring is an activity testing for correct plan execution. This activity is delegated to the BDI control system. In order to prove that the remainder of a plan is

correct it needs to be proved against the current situation. A component similar to the planner is used to evaluate the remainder in a simulated environment given by the planning domain. The simulator is a greedy planner with the option generator replaced by an iterator over plan's tail.

If the simulation or the execution of a plan step reports an error, the plan is abandoned like a BDI plan instance. The goal responsible for invoking the planner is still hold in the intentional structure of the BDI agent. The BDI reasoner may mark the goal with failure and abandon it or if the goal was marked, to retry plans the planner will be invoked again and asked for a new plan. In this respect, the planning process may be seen as a single agent plan schema or as a presumably infinite set of plan instances. The choice is taken at the design time and marked using BDI properties on agent goals specified in the agent description. In the loader dock example the `PickupPacket` is excluded on a plan failure (cf. sec. 5).

4.3 Managing plan failures

The case, when plans are created ad infinitum for a goal, poses a difficulty for most BDI engines, because they have been designed for relatively small number of schemas corresponding to a goal. Given the capability of a planner to create an infinite number of plan instances, the BDI reasoning engine would instruct the planner to create plans even if there is none. On the other hand as the situation changes the planner may find a plan in the future. There are four cases that need a decision on the part of a BDI reasoner:

- I. The planner cannot find any way to improve the agent's situation. In this case, it returns an empty plan with no actions. The BDI reasoner may wait a specific time and retry finding a plan. The BDI goal is marked with a BDI flag carrying the delay time between failed and new planning process.
- II. The planner could not find a correct plan satisfying all goals and subgoals. Following this plan would allow the agent to reach a state where the goals are satisfied, but it could also lead it into a dead-end if the plan contains irreversible decisions. On the other hand, the plan may be executed with the hope the future planning, starting from a better situation, may find a complete plan. The description of planning problem given to the planner, should include a flag specifying if incomplete plans are allowed.
- III. The planner could not find a complete plan, because all of the time designated for planning has been used up. The time limit is specified in the description of the planning problem given to the planner. If a plan cannot be found because the problem exceed the planning horizon of an agent, an incomplete plan will be returned that fits specified problem best. This case can be handled the same way like case II.
- IV. A number of correct plan instances is returned in successive trials but they fail to reach the goal. In this case the domain description is too abstract and lacks the knowledge needed by the planner to recognize specific reasons for failure. E.g. the speed of a robot depends on the load carried, but domain

designer specified constant speed. Such failures are seldom, but must be accounted for in this design. When plans fail because of a more demanding setting than stated in the domain description a counter on the goal for failed plans may be the most simple solution.

BDI systems with elaborate goal representation including failure and retrieval semantic [11], already offer the provisions to handle the cases at goal and plan levels. The description of goals and the semantics of the reasoner may be extended to provide BDI flags described above. When goals are merely events processed by the system this task can be delegated to an additional controller component wrapping around the planner.

5 Examples

The planner presented in this paper has been implemented in the JAVA™ language [12]. To evaluate it with a BDI framework it was integrated with JADEx [13] – a BDI reasoning framework developed at the University of Hamburg. Jadex incorporates many ideas from BDI-systems, like PRS [10] or JACK™ [14] and provides new unique facilities to deal with goal deliberation.

Two domains have been designed and implemented to demonstrate the dynamics and reasoning capabilities of the hybrid system. In the standard blocks-world example a simple but though propositional domain is used for testing. The Loader Dock example contains a continuous and highly dynamic domain demanding real-time performance.

5.1 Blocks' World

The blocks-world domain is a standard testing domain for planners. The problem consists of a bunch of blocks that must be moved into a final configuration. It is one of the first problems investigated with planners and from the beginning it has been a challenge as the problem is clearly exponential with respect to the number of blocks used.

The control knowledge is borrowed from the *TACPLANNER* [9] and adapted to JAVA™. The desire of an agent is to keep the number of moved blocks low. The distance to a goal is the number of blocks in *bad towers* (i.e. towers of blocks not conforming with target state).

Figure 3 shows an implementation of the control knowledge in JAVA™. Here all the applicability predicate and transition functions are arranged together in a single method. This way, the code is kept in one place and much of redundant computation is abandoned.

E.g. the control knowledge of `PutToTable` demonstrates the use of a temporal condition. Whenever in the foregoing plan a block was placed already on the table, the action is no more applicable to the state considered. This assertion prevents blocks from being placed on table again and again.

Using global search with an agenda of 10, it requires about 10 seconds on an i586 400MHz machine to stack 100 blocks. These results are not surprising as

```

public boolean applyTo(State state) {
    Block block=(Block)state.get(LOAD);
    if (block==null) return false;

    State previous=state.getPrevious();
    while(previous!=null) {
        if (previous.get(DOWN, block)==null) return false;
        previous = previous.getPrevious();
    }
    st.set(DOWN, block, null);
    st.set(LOAD, null);
    return true;
}

```

Fig. 3. Control knowledge for the PutToTable operator with a temporal condition requiring, that a block is put to table only once.

the planner has been build upon reliable and approved planning techniques. It compares well with the state of art planners (cf. [8]).

5.2 Loader Dock

In the loader dock there are several workers wandering around and carrying packets between incoming trucks and shelves. Their job is to unload packets from full trucks, or to deliver packets to empty trucks arriving at the store. The dock itself contains shelves where packets can be temporally placed. The shelves are separated by corridor ways, which are used by workers to transport the packets (cf. fig. 4).

This particular domain is fully observable in respect to a certain update interval of mutual beliefs that is performed by the store agent. It is almost deterministic because agents cannot be sure if created plans will be executed timely as it was predicted. The environment changes because of other agents and processes including trucks coming in and going out at various time intervals. Most quantities like the number of packets, trucks, workers and places are finite, but attributes of domain objects like speed, direction, arrival or departure time are real valued. There are many processes and agents acting in this domain concurrently. Agents do have common goals to handle the job at the storehouse, but share resources like time and corridor space. The storehouse depicted in Figure 4 is modeled using a discrete grid of points connected with each other through pathways. The workers and packets can take any position in the store and may head towards any real valued direction.

Planning takes very little time for this small domain. The approximate time of path planning on an i686 3GHz machine is from 1ms to 10ms. Because way points in the grid are static, all movement actions are precomputed in advance, which speeds the planning process.

When a worker makes an intention to move somewhere, it communicates this intention to other ones for the purpose of coordination. A trajectory of the

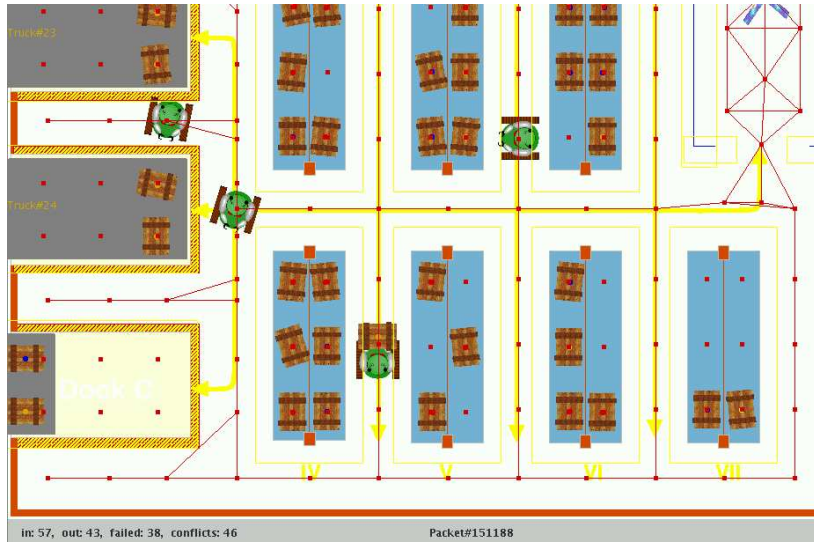


Fig. 4. The graphical interface of the loader dock example.

movement, described by points and exact time-values, is sent to each worker, so it can update its beliefs about future changes in the environment. This information is stored in a `domain time model` representing the beliefs concerning prospective workers' positions. This information guides the planning process of each worker, so none of them intersects a path of another one.

Goals to load or unload truck packets are distributed using the FIPA Contract Net Interaction Protocol [15]. Such a goal contains packet identification and a deadline. The worker responding with the fastest plan schema wins and commits to the execution. Other workers are informed about this intention. Figure 5 demonstrates a pickup goal. It represents an condition, where the agent is carrying a packet. `exclude="when_failed"` requests the BDI reasoner not to start the planner again after the initially created plan fails. The deliberation section tells the reasoner to suspend pursuing of other goals with types: `Go`, `PickupPacketC`, `PutdownPacketC`. It prevents an agent from trying to go in different directions and to respond to calls for proposals, while processing this goal. The goal represents the BDI sphere of responsibility. It contains the provisions, as mentioned earlier, for re-planning (specified with flags), deliberation and monitoring (using the target condition).

The corresponding planning domain is specified in Figure 6. The planning process takes place in an instance of `PickupPlan` triggered by a `PickupPacket` goal. This part of the agent description file (ADF) specifies the responsibility of the planning component. The type of search chosen is global search and the planning agenda may include up to 200 planning alternatives. The time given to the planner is 2000 ms and by default the planner should return correct plans only. The heuristic used expresses agent desire for the plans being short in time.

```

<achievegoal name="PickupPacket" exclude="when_failed">
  <parameter name="packet" class="Packet"/>
  <deliberation cardinality="1">
    <inhibits ref="Go"/>
    <inhibits ref="PickupPacketC"/>
    <inhibits ref="PutdownPacketC"/>
  </deliberation>
  <targetcondition>$beliefbase.packet!=null</targetcondition>
</achievegoal>

```

Fig. 5. Specification of the PickupPacket goal.

The operator MoveOp changes the position of a worker and the operator PickupOp picks up the packet specified in the goal.

```

<plan name="pickup" search="global" agenda="200" time="2000">
  <body>new PickupPlan()</body>
  <trigger>
    <goal ref="PickupPacket"/>
  </trigger>
  <heuristic>new TimeHeuristic()</heuristic>
  <operator>new MoveOp($to)</operator>
  <operator>new PickupOp()</operator>
</plan>

```

Fig. 6. Specification of the Pickup plan.

The use of a planner at the operational level helped worker agents to plan their activities and coordinate among themselves by communicating their intentions. They were successful to plan in a dynamic cooperative scenario with discrete and continuous resources. The BDI part has been effectively applied to control the dynamics of execution and has kept constraints over goals satisfied. The hybrid system has remained reactive and was able to adapt to different load factors.

6 Related Research

Architectures with strong emphasis on artificial intelligence include a planning component as their central part. An example system designed with BDI and planning in mind is INTERRRAP [16]. It includes a local planning layer utilizing a hierarchical planner. The representation of procedures and goals follows that of a hierarchical task network (HTN) planner and has a declarative form. In the layered architecture, the planner takes control over a reactive subsystem.

The topic of joining procedural BDI reasoning engines with decision theoretic planners is not new. System have been built that proved especially useful in domains featuring enough time for planning, like in the example of PROPICE-PLAN

[17] featuring a blast furnace domain. PROPICE-PLAN extends the dMars system with a state based planner IPP. CYPRESS is a system composed of the hierarchical planner SIPE-2 upon the Procedural Reasoning System (PRS) [10] also following a layered approach. Both systems are glued together with The ACT Formalism [18].

In the work of De Silva [19] a hierarchical planner JSHOP [20], is used to produce plans for the JackTMBDI system. Domain descriptions for the planner are created from the BDI plan schemata at compile time. The approach limits the programmer to a subset of possible programming solutions given by the intersection of the planning language and the BDI language and their possible transformations. It is also unclear, how to generalize this approach to a wider class of BDI systems.

BDI internal mental states can be mapped to a STRIPS [21] notation forth and back [22]. This has been done on an abstract BDI interpreter called X-BDI [23] and augmented with the GRAPHPLAN. The mapping is a structure transformation of beliefs, desires and intentions into a propositional notation that is used by the planner so beliefs and actions are constrained by the propositional STRIPS domain representation.

These approaches aimed at technical or theoretical feasibility. There was no concern about generality or performance. In our opinion, they are not well applicable to planning of low-level control tasks. GRAPHPLAN and IPP are generic state-based planners using generic heuristics. The range of problems solved by these planners is limited and they are overpowered by planners applying domain specific knowledge [8]. They require to state the planning problem declaratively as a set of propositions. Object-based domain modeling approaches fit better with newer BDI frameworks designed in object-oriented languages such as JACKTM [14] or JADEX [13].

SIPE-2 and INTERRRAP planning layer use hierarchical decomposition in the space of partially ordered plans. The use of this planning space gives more degrees of freedom to the planner. The choice of such an algorithm has to be carefully elaborated. It is generally not the question, how to give the agent all the possible options, but how to restrict the choice to a minimal subset that needs to be considered. In fact, this is one of the advantages of BDI systems that constrain the options to a small number precompiled plan schemata. BDI systems also use deliberation and filtering techniques to further decimate the choices.

7 Conclusion

We have investigated the composition of a deliberative planner with a BDI framework forming a new hybrid system with combined characteristics. The use of a state-based planner on a planning problem extended by BDI concepts space allows to easily merge those two paradigms. Implementing the planner in an object-oriented language and representing the planning domain with object-based models further facilitates the integration with BDI frameworks devised

with such concepts in mind. The main requirement for the planner was performance at the low-level of execution. The planning problem representation included many places for application of domain specific knowledge including action preconditions, goal distance functions and utility functions.

Further an integration schema was proposed, where the BDI system is used as system controller responsible for the upper part of the intentional structure. It uses a deliberative planner in situations where precompiled plans are hard to devise in advance. In this schema the planner performs short term planning and produces plans with a constructed proof of correctness. The resulting plans are handled directly to the scheduler. Four cases have been identified that require changes in the semantics of goal handling in the BDI framework. These situation may trigger a re-planning process in order to create plans that better suit the problem on hand.

The approach has been verified in a puzzle domain to test the scalability of the planner showing comparable results with existing planning systems. The whole hybrid system has been investigated on the basis of the loader dock scenario. Here multiple agents had to plan their way through a packet store and coordinate their activities in order to cope with the load introduced by trucks coming in and going out. Worker agents could handle this domain because their intentional structure has been completed by plans created at runtime. On the other hand, the BDI reasoning could retain its reactive and deliberative characteristic.

The future of this work includes advances on the part of the planner. Techniques should be investigated to include concurrent planning and planning under uncertainty and embed it into a BDI reasoning framework. We anticipate that development of complex planning domains would require modeling and debugging tools. The abstraction given by knowledge representation is particularly important to this planning approach. Techniques for the representation of planning domains should be used to facilitate it.

References

1. Georgeff, M.P., Pell, B., Pollack, M., Tambe, M., Wooldrige, M.: The belief-desire-intention model of agency. In: Intelligent Agents, 5th International Workshop, ATAL'98. Springer, Paris (1998) 1–10
2. Bratman, M.E.: *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA (1987)
3. De Silva, L., Padgham, L.: A comparison of BDI based real-time reasoning and HTN based planning. In: AI 2004: Advances in Artificial Intelligence, 17th Australian Joint Conference on Artificial Intelligence, Cairns, Australia, Springer (2004) 1167–1173
4. Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. *Computational Intelligence* **4** (1988) 349–355
5. Pokahr, A., Braubach, L., Lamersdorf, W.: A goal deliberation strategy for BDI agent systems. In: Third German conference on Multi-Agent System TEchnologieS (MATES-2005). (2005)

6. Shut, M., Wooldridge, M.: The control of reasoning in resource-bounded agents. *The Knowledge Engineering Review* **16**(3) (2001)
7. Ghallab, M., Nau, D., P.Traverso: *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers (2004)
8. Edelkamp, S., Hoffmann, J., Littman, M., Younes, H.: The 4th international planning competition 2004 (IPC-2004) (2004) Hosted at the International Conference on Automated Planning and Scheduling 2004 (ICAPS-2004).
9. Kvarnström, J., Magnusson, M.: TALplanner in IPC-2002: Extensions and control rules. *Journal of Artificial Intelligence Research (JAIR)* **20** (2003) 343–377
10. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning: An experiment with a mobile robot. In: *Proceedings of the sixth National Conference on Artificial Intelligence (AAAI-87)*, Seattle, Washington (1987) 677–682
11. Braubach, L., Pokahr, A., Moldt, D., Lamersdorf, W.: Goal representation for BDI agent systems. In: *The Second International Workshop on Programming Multi Agent Systems*. (2004) 9–20
12. Walczak, A.: Planning and the belief-desire-intention model of agency. Master's thesis, University of Hamburg (2005)
13. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. In Bordini, R., Dastani, M., Dix, J., Seghrouchni, A., eds.: *Multi-Agent Programming*, Kluwer Academic Publishers (2005)
14. Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agent - components for intelligent agents in Java (1999)
15. FIPA: FIPA Contract Net Interaction Protocol Specification. FIPA. (2001)
16. Fischer, K., Müller, J.P., Pischel, M.: Unifying control in a layered agent architecture. Technical Report TM-94-05, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Kaiserslautern, DE (1994)
17. Despouys, O., Ingrand, F.F.: Propice-plan: Toward a unified framework for planning and execution. In Biundo, S., Fox, M., eds.: *Recent Advances in AI Planning*, 5th European Conference on Planning, ECP'99, Durham, UK, Springer (1999) 278–293
18. Wilkins, D.E., Myers, K.L., Wesley, L.P.: Cypress: Planning and reacting under uncertainty. In Burstein, M.H., ed.: *ARPA/Rome Laboratory Planning and Scheduling Initiative Workshop Proceedings*. Morgan Kaufmann Publishers Inc., San Mateo, CA (1994)
19. De Silva, L., Padgham, L.: Planning on demand in BDI systems. In: *International Conference on Automated Planning and Scheduling*, Monterey, California (2005)
20. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* **20** (2003) 379–404
21. Fikes, R.E., Nilsson, N.J.: STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **2**(3–4) (1971) 189–208
22. Meneguzzi, F.R., Zorzo, A.F., da Costa Móra, M.: Propositional planning in BDI agents. In: *Proceedings of the 2004 ACM symposium on Applied computing*, ACM Press (2004) 58–63
23. Móra, M.C., Lopes, J.G., Viccari, R.M., Coelho, H.: BDI models and systems: Reducing the gap. In: *Proceedings of the 5th International Workshop on Intelligent Agents*, Springer (1999)