Propositional Planning in BDI Agents*

Felipe Rech Meneguzzi
HP/PUCRS
6681, Ipiranga Avenue
Porto Alegre, Brazil
felipe@cpts.pucrs.br

Avelino Francisco Zorzo
Faculty of Informatics
6681, Ipiranga Avenue
Porto Alegre, Brazil
zorzo@inf.pucrs.br

Michael da Costa Móra
Faculty of Informatics
6681, Ipiranga Avenue
Porto Alegre, Brazil
michael@inf.pucrs.br

ABSTRACT

This paper aims to describe the relationship between propositional planning systems and the process of means-end reasoning used by BDI agents. To show such relationship, we define a mapping from BDI mental states to propositional planning problems and from propositional plans back to mental states. In order to test the viability of such mapping, we have implemented it in an extension of a BDI agent model through the use of Graphplan as the propositional planning algorithm. The implementation was applied to model a case study of an agent controlled production cell.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Plan execution, formation, and generation*

General Terms

BDI Model, Planning

Keywords

Propositional Planning, BDI, X-BDI

1. INTRODUCTION

The development of rational agents, *i.e.* agents capable of performing useful actions given its world perception and knowledge [29], has been a major concern since the beginning of Computer Science research [25]. Such development originated various important approaches to the implementation of computational reasoning, starting with the General Problem Solver (GPS) and generic problem solving algorithms, and evolving into Planning Systems. Although these planning systems are capable of defining how goals are to be achieved, they do not deal with the problem of which goals are to be pursued. In other words, they are capable of performing means-end reasoning but not to deal with the problem of practical reasoning [23].

Planning systems gave way to the reasoning model based on deliberative agents as a means to deal with practical reasoning. These agents used, initially, a series of decision making mechanisms which are theoretically defined, like decision theory, but have proven themselves inadequate to implementation, since they assumed agents that had unlimited time and computational resources [23]. In order to solve the issue of limited resources, a philosophical practical reasoning model was formalized and computationally implemented [5], that, in theory, allows an agent to limit the time spent in deliberation. Nevertheless, implementations of these agents tend to avoid facing the complexity of means-end reasoning through the usage of plan libraries, defined for the agent prior to its execution. Therefore, this approach solves the problem of an agent's limited resources, but it delegates the responsibility of building plans to its developer.

Deliberation using a plan library is obviously more computationally efficient than performing plan formation at runtime. Nevertheless, the usage of a plan library ties the problem solving capability of an agent to the situations foreseen by its designer, while planning at runtime allows the agent to cope with a larger variety of situations. On the other hand, agent architectures that perform planning at runtime usually have tightly coupled planning methods that limit its improvement through the incorporation of novel planning strategies. Therefore, the purpose of our work is to show how to incorporate propositional planning in BDI agent models, providing agents with the ability of practical reasoning and means-end reasoning. Our approach is underpinned on a mapping among BDI mental states and propositional planning formalisms, thus allowing any algorithm based on a similar formalism to be used as a means-end reasoning process for a BDI agent. In order to demonstrate the viability of such approach we take a specific BDI agent model, namely the X-BDI model [19], and modify it to use propositional planning algorithms to perform means-end reasoning [20].

2. BASIC CONCEPTS

2.1 BDI Agents

As computer systems became more complex, abstraction mechanisms for these systems were developed. One abstraction mechanism that is becoming increasingly accepted is the notion of Computer Agents [23, 12, 9], so far as to be pro-

*This work is partially supported by HP Brazil and CNPq.

SAC '04, March 14 -17, 2004, Nicosia, Cyprus

posed as an alternative to the Turing Machine as an abstraction for the notion of computation [12, 26]. Although there is a variety of definitions for Computer Agents, an informal definition, which adequately captures important properties of agency, is the following [12]: *An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.*

In the context of multi-agent systems research, one of most widely known and studied models of deliberative agents uses Beliefs, Desires and Intentions as abstractions for the description of a system's behaviour. This model is called BDI (for Beliefs, Desires and Intentions) and was originated by a philosophical model of human practical reasoning [4], later formalized [8] and improved towards a more complete computational theory [22, 29].

One of the most important processes of the BDI model is the selection of the course of action the agent will take in order to satisfy its intentions, *i.e.* given an environment and a set of objectives, determine whether the agent is capable of satisfying its objectives through some sequence of actions. This problem is characterized as the *Agent Design Problem* [28]. The most widely known BDI agent implementations have been bypassing this problem through the use of plan libraries where the courses of action for every possible objective an agent might have are stored [11, 10]. The theories commonly used to underpin systems capable of dealing with the agent design problem assume an agent with unlimited resources, thus making its actual implementation impossible [23, 22]. On the other hand, recent works seek to deal with this problem in various ways, for instance, by defining alternate proof systems [19, 21] or using model checking in order to validate the agent's plan library [3]. An alternate approach to solve the problem is the use of planning algorithms to perform means-end reasoning at runtime [23]. It is important to point out that we do not tackle the issue of hierarchical planning at Intention and Action levels [16], though the architecture we propose is suitable to such extensions.

2.2 Planning Algorithms

Means-end reasoning is a fundamental component of any rational agent [4] and is useful in the resolution of problems in a number of different areas, for instance scheduling [24]. Therefore, the development of planning algorithms has been one of the main goals of AI research [29]. A planning problem is generically defined by formal description of the [27]: *i*) start state; *ii*) intended goals; *iii*) actions that may be performed. The planning system will take these components and will generate a set of actions ordered by some relation, that, when applied to the world where the initial state description is true, will make the goals description true.

It is known that planning is undecidable [7] and that planning problems, in the general case, are PSPACE [6]. Despite the high complexity proven for the general case of planning problems, recent advances in planning research led to the creation of planning algorithms that perform significantly better than previous approaches to solve various problem classes [27, 20]. These algorithms belong mainly to two classes: *i*) Graphplan based algorithms [2]; *ii*) algorithms based in the compilation of the planning problem into a formula whose satisfiability is tested (SAT) [13].

In this work, we focus on STRIPS-like (STanford Research Institute Problem Solver) formalisms [20]. Our description of the formalism is based on the one found in [20], and is, according to the author, a $\mathcal{S}_{\mathcal{IL}}$ formalism, *i.e.* the basic STRIPS plus the possibility to use incomplete specifications and literals in the description of world states. It is important to point out that the formalisms defined by Nebel [20] are more general, but since we do not intend to provide a detailed study of planning formalisms, we use a simpler version of the referred formalism. In particular, we use a propositional logical language with variables only in the specification of operators. Also, operators are not allowed to have conditional effects. Considering the STRIPS formalism, one can notice that the referred planners deal only with atoms. Nevertheless, within this paper more expressivity is desirable, in particular, the possibility to use first order ground literals. It is possible to avoid these limitation through the use of syntactic transformations so that the planners described operate over first order ground literals.

2.3 Graphplan

Graphplan [2] is a planning algorithm based on the construction and search in a graph. It is considered one of the most efficient planning algorithms created recently [27, 20, 24]. The efficiency of this algorithm was empirically proved through the significant results obtained by instances of Graphplan in the planning competitions of the AIPS (International Conference on AI Planning and Scheduling) [14, 18]. Planning in Graphplan is based on the concept of a *Planning Graph*, which is a data structure in which information regarding the planning problem is stored in such a way that the search for a solution can be accelerated. The Planning Graph is not a space state graph in which a plan is a path through the graph. Instead, a plan in the Planning Graph is essentially a flow, in the sense of a network flow. Planning Graph construction is efficient. It has polynomial complexity in graph size and construction time with regard to problem size [2]. The graph is then used by the planner in the search for a solution to the planning problem using data stored in the graph to speed up the process. The basic Graphplan algorithm (*i.e.* without the optimizations proposed by other authors) is divided into two distinct phases: Graph Expansion and Solution Extraction.

The Planning Graph in Graphplan is directed and levelled. Considering that a plan is composed of temporally ordered actions, and, in between these actions there are world states, graph levels are divided into alternating proposition and action levels. Proposition levels are composed of proposition nodes labelled with propositions. These nodes are connected to the actions in the subsequent action level through pre-condition arcs. Action nodes are labelled with operators and are connected to the nodes in the subsequent proposition nodes by effect arcs. Every proposition level denotes literals that are possibly true at a given moment, thus the first proposition level represents the literals that are possibly true at time 1, the next proposition level represents the literals that are possibly true at time 2 and so forth. Action levels denote operators that can be executed at a given moment in time in such a way that the first action level represents the operators that may be executed at time 1 and so forth. The graph contains mutual exclusion relations (*mutex*) between nodes in the same graph level. Mutex relations have a fundamental role in algorithm efficiency. A mutual exclusion relation between two nodes means that they can-

not be simultaneously present in the same graph level for the same solution. After graph expansion, the second phase of Graphplan, called solution extraction, takes place. It uses a backward chaining strategy to traverse the graph on a level by level basis trying to find a flow starting from the goals that leads to the initial conditions.

The algorithm chosen for implementation in the prototype is Graphplan. Its choice was due to the greater body of available work related to its improvement, *e.g.* real-time planning [24], additional inference of planning graph information [17], among others, that might be incorporated into an implementation.

3. PROPOSED SOLUTION

Considering that we will use propositional planners as a mechanism for means-end reasoning within BDI agents, we take the X-BDI agent model [19] and modify its operational definition to accommodate the use of planning algorithms external to its kernel.

3.1 Extending X-BDI with Graphplan

The X-BDI agent model was created in order to allow a formal agent specification to be directly executed [19]. That is possible because X-BDI's language is defined in terms of a formalism that has a reference implementation, which is called Extended Logic Programming with explicit negation (ELP) using the Well-Founded Semantics extended for the explicit Negation (WFSX), with a derivation procedure is called Selected Linear Derivation for extended programs (SLX) [1]. X-BDI uses ELP's ability to deal with contradiction to implement a variety of non-monotonic reasoning processes, necessary for the BDI model. Moreover, a modified form of Event Calculus [15] is used in order to allow X-BDI to deal with a dynamic world.

3.1.1 X-BDI Operation

An X-BDI agent has the traditional components of a BDI agent, *i.e.* a set of Beliefs, Desires and Intentions. Besides, given its extended logic definition, it has also a set of time axioms defined through a variation of the *Event Calculus* [19, 15]. The set of beliefs is simply a formalization of facts in ELP, individualized for a specific agent. The belief revision process in X-BDI is the result of the program revision process performed in ELP by the SLX procedure. From the agent's point of view, it is assumed that its beliefs are not always consistent, because whenever an event that makes the beliefs inconsistent, SLX will minimally revise the program, and therefore, the beliefs. Every desire in an X-BDI agent is conditioned to the body of a logic rule, which is a conjunction of literals called *Body*. Thus, *Body* specifies the pre-conditions that must be satisfied in order for an agent to desire a property. When *Body* is an empty conjunction, property P is unconditionally desired. Desires may be temporally situated, *i.e.* can be desired in a specific moment, or whenever its pre-conditions are valid. Besides, desires have a priority value used in the formation of an order relation among desire sets. There are two possible types of intentions: Primary Intentions, which refer to the intended properties, and Relative Intentions, which refer to actions able to bring about these properties. An agent may intend something in the past or that is already true. Besides, intentions may not be impossible, *i.e.* there must be at least one plan available to the agent whose result is a world state where the intended property is true. This definition represents the first change performed in X-BDI described in this work. In the original X-BDI the possibility of a property was verified through the abduction of an Event Calculus theory that would make the property true. In this work, we modified the planning process so that it is abstracted from the operational definition of X-BDI, allowing that any planning process that satisfies the conditions of Section 2.2 to be used by X-BDI. Thus, the notion of possibility of a desire is associated with the existence of a plan to satisfy it.

The reasoning process performed by X-BDI initiates with the selection of Eligible Desires, which represent the unsatisfied desires whose pre-conditions were satisfied. The elements of this set are not necessarily consistent among themselves. Candidate Desires are then generated, which represent a set of Eligible Desires that are both consistent and possible and will be later adopted as Primary Intentions. In order to satisfy the properties represented by Primary Intentions, the planning process generates a sequence of temporally ordered actions that constitute the Relative Intentions. Eligible desires have rationality constraints that are similar to those imposed over the intentions in the sense that an agent will not desire something in the past or something the agent believes will happen without his interference. Agent beliefs must also support the pre-conditions defined in the desire *Body*. Within the agent's reasoning process these desires will originate a set of mutually consistent subsets ordered by a partial order relation. The process of selecting Candidate Desires seeks to choose among the Eligible Desires one subset that contains only desires that are internally consistent and possible. A possible desire is one that has a property P that can be satisfied through a sequence of actions. In order to choose among multiple sets of Candidate Desires, X-BDI uses ELP constructs that allow the definition of preferred revisions. Thus, X-BDI defines a desire preference relation through a set of preferred revisions generated using the priorities expressed in the desires. Through this preference relation, a desire preference graph that relates all subsets of Eligible Desires is generated.

Candidate Desires represent the most significant modification made in this paper regarding the original X-BDI [19]. Originally, X-BDI verified the possibility of a desire through the abduction of an Event Calculus Theory in which the belief in the validity of a desired property P could be true. Such abduction process is, actually, a form of planning. Since our main objective in this paper is to separate the planning process previously hard-coded within X-BDI, the notion of desire possibility had to be re-defined. Therefore, we define that a set of Candidate Desires is the subset of Eligible Desires with the greater preference value, and whose properties can be satisfied. Satisfiability is verified through the execution of a propositional planner that processes a planning problem in which the initial state contains the properties that the agent believes at the time of planning. The P properties present in the Candidate Desires are used to generate the set primary intentions.

Primary Intentions can be seen as high-level plans, similar to the intentions in IRMA [5]. Hence, they represent the agent's commitment to a course of action, which will be performed through a series of refinements up to the point where an agent has a temporally ordered set of actions representing a concrete plan towards the satisfaction of its goals. Relative Intentions correspond to the temporally ordered steps of

the concrete plans generated to satisfy the agent's Primary Intentions. The notion of agent commitment results from the fact that Relative Intentions must be non-contradictory regarding Primary Intentions.

3.1.2 Intention Revision

The computational effort and the time required to reconsider the whole set of intentions of a resource-bounded agent is generally significant regarding the environment change ratio. Therefore, intention reconsideration should not occur constantly, but only when the world changes in such a way as to threaten the plans an agent is executing or when an opportunity to satisfy more important goals is detected. As a consequence, X-BDI uses a set of reconsideration "triggers" generated when intentions are selected, and causes the agent to reconsider its course of action when activated.

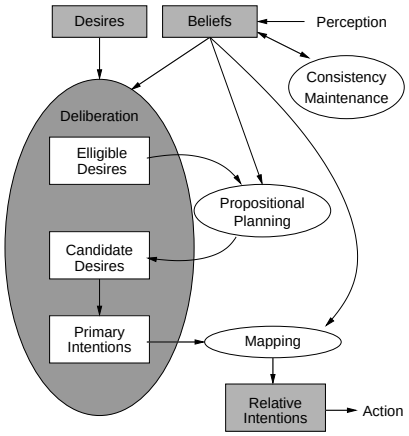

Figure 1: Modified X-BDI overview.

At this point, we verified that the modifications operated in X-BDI allow us to maintain the reconsideration conditions defined by Bratman [5]. In particular, if all of the agent's Primary Intentions are satisfied before the time planned for them to be satisfied, the agent will restart the deliberative process, for he has achieved his goals. On the other hand, if one of the Primary Intentions has not been achieved at the time planned for it, the agent will have to reconsider its intentions because its plans have failed. Moreover, if a desire with a higher priority than the currently selected desires becomes possible, the agent will reconsider its desires in order to take advantage of the new opportunity. Reconsideration is completely based on integrity constraints over beliefs. Therefore, considering that beliefs are revised at every sensoring cycle, it is possible that a reconsideration occurs due the "triggering" of a reconsideration restriction. The modifications implemented in X-BDI alter its operation so that it uses propositional planning algorithms as the underpinning of the means-end reasoning and as possibility verifiers in the practical reasoning process (Figure 1).

3.2 Solution Architecture

The prototype implemented for this work is essentially composed of three parts: the X-BDI kernel, implemented in Prolog, a planning system containing a C++ implementation of Graphplan, and a Java Graphical Interface used to ease the operation of X-BDI and to visualize its interaction with the environment. The *Agent Viewer* interface communicates with X-BDI through sockets by sending the input from the environment where the agent is embedded and receiving the result of the agent's deliberation. Through *Agent Viewer* the user can also describe the agent through its desires, actions and initial beliefs. Once X-BDI receives the agent description, it communicates with the planning library through Operating System files and the Prolog/C++ interface. The planner is responsible for generating a set of intentions for the agent. The modification applied into X-BDI essentially consists of, when the agent deliberates, convert subsets of the agent's desired properties into propositional planning problems and invoke the planning algorithm to solve these problems until either a plan that solve the highest priority desires is found, or the algorithm determines that it is not possible to solve any one of these problems.

4. A BDI PRODUCTION CELL

In this work we use a BDI agent in order to model a production cell as a case study, and as a means to verify the validity of the architecture described in Section 3. The proposed production cell is composed of seven devices, a Feed Belt, a Deposit Belt, four Processing Units and a Crane that can freely move components over all the devices in the cell. This Production Cell is illustrated in Figure 2.

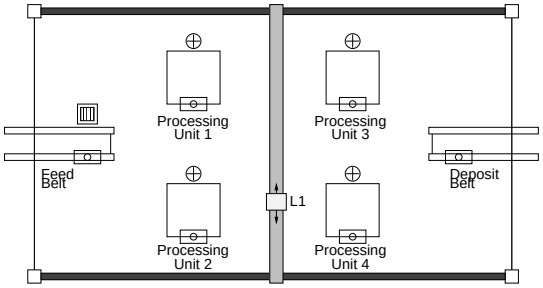

Figure 2: A BDI Production Cell.

Components enter the production cell for processing through the Feed Belt, and, once processed by all the appropriate Processing Units, they are removed from the cell through the Deposit Belt. Every Processing Unit is responsible for performing a different kind of operation in the component being processed, and can hold only one component at a given moment. Every component introduced in the cell can be processed by one or more Processing Units, which is determined by the type of component being processed. Different component types have different processing priorities. The control of this production cell is trusted to a BDI agent implemented using X-BDI, which should schedule the work of the production cell through its beliefs, desires and intentions, re-scheduling the work whenever some change occurs.

The first step in modelling any problem using a STRIPS-like formalism is the choice of the predicates used to represent the problem's object-types and its states. Hence, we have the following predicates representing objects in the cell:*i*) `bloc(B)` denotes that `B` is a Component to be processed; *ii*) `procUnit(P)` denotes that `P` is a Processing Unit, Processing Units are also Devices; *iii*) `device(D)` denotes that `D` is a Device; *iv*) `feedBelt` represents the Feed Belt;

v) `depositBelt` represents the Deposit Belt. Similarly, we have the following predicates representing system states: *i*) `over(B,D)` denotes that Component `B` is over Device `D`; *ii*) `empty(P)` denotes that Processing Unit `P` is empty, *i.e.* has not Component over it; *iii*) `processed(B,P)` denotes that Component `B` has already been processed by Processing Unit `P`; *iv*) `finished(B)` denotes that Component `B` has already been processed by all appropriate Processing Units and has been removed from the Production Cell. Next, we define the actions the agent is capable of performing in the context of the proposed problem: *i*) Action `process(B,P)` having as pre-conditions `procUnit(P)`, `bloc(B)` and `over(B,P)`, and as effect `processed(B,P)`, represents the processing that a Processing Unit `P` performs in a Component `B` over it; *ii*) Action `consume(B)` having as pre-conditions `bloc(B)` and `over(B,depositBelt)` and as effects ¬`over(B,depositBelt)`, `empty(depositBelt)` and `finished(B)`, represents the removal of component `B` from the production cell through the Deposit Belt; *iii*) Action `move(B,D1,D2)` having as pre-conditions `over(B,D1)`, `empty(D2)`, `bloc(B)`, `device(D1)` and `device(D2)` and as effects `over(B, D2)`, ¬`over(B,D1)`, ¬`empty(D2)` and `empty(D1)`, represents the motion of Component `B` from Device `D1` to Device `D2`.

The processing requirements of components and its priorities are modelled through the agent's desires. Therefore, we can model agent `pCell` necessity to process Component `bloc1` by Processing Units `procUnit1`, `procUnit2` and `procUnit3` as soon as this component is inserted into the production cell through the following desires:

```
des(pCell,finished(bloc1),Tf,[0.7])
 if bel(pCell, bloc(bloc1)),
    bel(pCell, processed(bloc1,procUnit1)),
    bel(pCell, processed(bloc1,procUnit2)),
    bel(pCell, processed(bloc1,procUnit3)),
    bel(pCell, -finished(bloc1)).
des(pCell,processed(bloc1,procUnit1),Tf,[0.6])
 if bel(pCell, bloc(bloc1)),
    bel(pCell, -processed(bloc1,procUnit1)).
des(pCell,processed(bloc1,procUnit2),Tf,[0.6])
 if bel(pCell, bloc(bloc1)),
    bel(pCell, -processed(bloc1,procUnit2)).
des(pCell,processed(bloc1,procUnit3),Tf,[0.6])
 if bel(pCell, bloc(bloc1)),
    bel(pCell, -processed(bloc1,procUnit3)).
```

Similarly, we can model the agent's need to process Component `bloc2` by Processing Unit `procUnit3` and `procUnit4` through the following desires:

```
des(pCell,finished(bloc2),Tf,[0.6])
 if bel(pCell, bloc(bloc2)),
    bel(pCell, processed(bloc2,procUnit3)),
    bel(pCell, processed(bloc2,procUnit4)),
    bel(pCell, -finished(bloc2)).
des(pCell,processed(bloc2,procUnit3),Tf,[0.5])
 if bel(pCell, bloc(bloc2)),
    bel(pCell, -processed(bloc2,procUnit3)).
des(pCell,processed(bloc2,procUnit4),Tf,[0.5])
 if bel(pCell, bloc(bloc2)),
    bel(pCell, -processed(bloc2,procUnit4)).
```

Finally, we model the agent's static knowledge regarding the problem domain, in particular the object's classes and the initial world-state with the following beliefs:

```
bel(pCell, procUnit(procUnit1)).
bel(pCell, procUnit(procUnit2)).
bel(pCell, procUnit(procUnit3)).
bel(pCell, procUnit(procUnit4)).
bel(pCell, device(procUnit1)).
bel(pCell, device(procUnit2)).
bel(pCell, device(procUnit3)).
bel(pCell, device(procUnit4)).
bel(pCell, device(depositBelt)).
bel(pCell, device(feedBelt)).
bel(pCell, empty(procUnit1)).
bel(pCell, empty(procUnit2)).
bel(pCell, empty(procUnit3)).
bel(pCell, empty(procUnit4)).
bel(pCell, empty(depositBelt)).
```

The arrival of a new component in the production cell is signaled by the sensors through the inclusion of `bloc(bloc1)` and `over(bloc1,feedBelt)` in the agent's beliefs database, activating the agent's reconsideration process. Given the desire's pre-conditions previously defined, only the desires related to the following properties become Eligible:

```
processed(bloc1,procUnit1)
processed(bloc1,procUnit2)
processed(bloc1,procUnit3)
```

These desires are then analyzed by the process of selecting Candidate Desires. In this process, the agent's Eligible Desires and beliefs are used in the creation of planning problems that are sent to Graphplan for resolution. The result of this processing is a plan that satisfies all the Eligible Desires, with the following steps:

```
1. move(bloc1,feedBelt,procUnit2)
2. process(bloc1,procUnit2)
3. move(bloc1,procUnit2,procUnit1)
4. process(bloc1,procUnit1)
5. move(bloc1,procUnit1,procUnit3)
6. process(bloc1,procUnit3)
```

The existence of this plan indicates to X-BDI that the specified set of Eligible Desires is possible, thus turning the previous set of desires into Candidate Desires, which will generate Primary Intentions, representing the agent's commitment. Next, Relative Intentions are generated using the steps in the plan recently created, one Intention for each step of the plan. These will lead the agent to perform the appropriate actions. Once the actions are executed, the Candidate Desires from the previous deliberation are satisfied. Moreover, the pre-condition of the desire to accomplish `finished(bloc1)` becomes valid, reactivating the agent's deliberative process and generating the following plan:

```
1. move(bloc1,procUnit3,depositBelt)
2. consume(bloc1)
```

Once more, this plan will originate the agent's intentions and, eventually, lead it to act. A possible situation during this agent's operation would be the arrival of a new component in the Production Cell. This could take place right after the deliberation which created the first plan, and would be signaled by the agent's sensors through the inclusion of `bloc(bloc2)` and `over(bloc2,feedBelt)` in the beliefs database, which would modify the Eligible Desires chosen in the second deliberation cycle to:

```
finished(bloc1);
processed(bloc2,procUnit3);
processed(bloc2,procUnit4);
```

These desires would cause the agent to generate a new plan to verify their validity, as well as the mental states required for the agent to eventually act.

5. CONCLUSIONS

In this paper, we described the relationship between propositional planning algorithms and means-end reasoning in BDI agents. To test the viability of such approach we describe a modification in the X-BDI agent model. Throughout this modification, new definitions of desires and intentions were created in order for the BDI model to maintain the theoretical properties present in its original version, especially regarding the definition of desires and intentions impossibility. Moreover, it was necessary to define a mapping between the structural components of a BDI agent and propositional planning problems. The result of implementing these definitions in a prototype can be seen in the case study of Section 4, which represents a problem that the means-end reasoning process of the original X-BDI could not solve. Considering that most implementations of BDI agents use a plan library in the means-end reasoning in order to bypass the inherent complexity of performing planning at runtime, X-BDI offers an innovative way of implementing more flexible agents. Its main drawback was the fact that the original planning mechanism is notably inefficient. For example, the case study described in Section 4 was not tractable by the original X-BDI planning process. Thus, the main contribution of our work consists in the definition of a mapping from BDI means-end reasoning to fast planning algorithms. Moreover, such approach enables the agent architecture to be extended with any propositional planning algorithm that uses a formalism compatible with the proposed mapping. Thus allowing an agent to use more powerful planners as they become available, or to use more suitable planning strategies for different problem classes.

Some ramifications of this work are foreseen as future work, in particular, the incorporation of the various Graphplan improvements, as well as the conduction of tests using other propositional planning algorithms, SAT being an example. Moreover, the determination of the class of problems for which some combination of BDI Agent and planning algorithm is capable of dealing with, represents an interesting theoretical contribution to our work.

¨

6. REFERENCES

[1] J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*. Springer, 1996.

[2] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 1997.

[3] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *Proc. of the 2nd AAMAS*, pages 409–416, 2003. ACM Press.

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

[5] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[6] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[7] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[8] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[9] W. V. der Hoek and M. Wooldridge. Towards a logic of rational agency. *Logic Journal of the IGPL*, 11(2):133–157, 2003.

[10] M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):233–260, 1998.

[11] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):33–44, 1992.

[12] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[13] H. Kautz and B. Selman. Planning as satisfiability. In *Proc. of the 10th ECAI*, pages 359–363, 1992. Wiley.

[14] J. Köhler. Solving complex planning tasks through extraction of subproblems. In R. Simmons, M. Veloso, and S. Smith, editors, *Proc. of the 4th AIPS*, pages 62–69, 1998. AAAI Press.

[15] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[16] N. Lacey, H. Hexmoor, and G. Beavers. Planning at the intention level. In *Proc. of the 15th FLAIRS*, pages 8–13, 2002. AAAI Press.

[17] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, 10:87–115, 1999.

[18] D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proc. of the 5th AIPS*, pages 196–205, 2000. AAAI Press.

[19] M. C. Móra, J. G. Lopes, R. M. Viccari, and H. Coelho. BDI models and systems: Reducing the gap. In *Proc. of the 5th International Workshop on Intelligent Agents*, 1999. Springer.

[20] B. Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research (JAIR)*, 12:271–315, 2000.

[21] N. Nide and S. Takata. Deduction systems for BDI logics using sequent calculus. In *Proc. of the AAMAS*, pages 928–935. ACM Press, 2002.

[22] A. S. Rao and M. P. Georgeff. Formal models and decision procedures for multi-agent systems. Technical Report 61, AAII, 1995. Technical Note.

[23] M. Schut and M. Wooldridge. The control of reasoning in resource-bounded agents. *The Knowledge Engineering Review*, 16(3), 2001.

[24] D. E. Smith and D. S. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. of the 17th IJCAI*, pages 326–337, 1999.

[25] A. M. Turing. Intelligent machinery. *Machine Intelligence*, 5:3–23, 1948.

[26] P. Wegner. Why interaction is more powerful than algorithms. *Comm. of the ACM*, 40(5):80–91, 1997.

[27] D. S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

[28] M. Wooldridge. The computational complexity of agent design problems. In E. Durfee, editor, *Proc. of the 4th ICMAS*, pages 341–348. IEEE Press, 2000.

[29] M. Wooldridge. *Reasoning about Rational Agents*. The MIT Press, 2000.