INCORPORATING PLANNING INTO BDI SYSTEMS

FELIPE RECH MENEGUZZI
AVELINO FRANCISCO ZORZO
MICHAEL DA COSTA MÓRA
MICHAEL LUCK

Abstract. Many architectures of autonomous agent have been proposed throughout AI research. The most common architectures, BDI, are procedural in that they do no planning, seriously curtailing an agent's ability to cope with unforeseen events. In this paper, we explore the relationship between propositional planning systems and the process of means-ends reasoning used by BDI agents and 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 a mapping, we have implemented it in an extension of a BDI agent model through the use of Graphplan as the propositional planning algorithm. The implemented prototype was applied to model a case study of an agent controlled production cell.

Key words. Propositional Planning, Agent Models and Architectures, BDI, X-BDI

1. Introduction.

Development of autonomous rational agents has been one of the main drivers of artificial intelligence research for some time [37]. Initial efforts focused on disembodied means-ends reasoning with the development of problem-solving systems and generic planning systems, such as STRIPS [15], later evolving into the idea of embodied problem solving entities (*i.e.* agents) [37]. In this line of research, one of the most widely studied models of autonomous agents has been that supported by the mental states of beliefs, desires and intentions [7], or the BDI model. While efforts towards defining BDI architectures have been sustained and significant, resulting in both theoretical [34] and practical architectures [14], they have also led to a disconnect between them.

In particular, theories of autonomous BDI agents often rely on logic models that assume infinite computational power, while architectures defined for runtime efficiency have curtailed an agent's autonomy by forcing the agent to rely on a pre-compiled plan library. Although simple selection of plans from a plan library is computationally efficient, at compile time an agent is bound to the plans provided by the designer, limiting an agent's ability to cope with situations not foreseen at design time. Moreover, even if a designer is able to define plans for every conceivable situation in which an agent finds itself, such a description is likely to be very extensive, offsetting some of the efficiency benefits from the plan library approach. The absence of planning capabilities thus seriously curtails the abilities of autonomous agents. In consequence, we argue that planning is an important capability of any autonomous agent architecture in order to allow the agent to cope at runtime with unforeseen situations.

Though the efficiency of planning algorithms has been a major obstacle to their deployment in time-critical applications, many advances have been achieved in planning [43], and developments are ongoing [2]. Considering that planning is an enabler of agent flexibility, and that there have been significant advances in planning techniques, it is valuable and important for autonomous agent architectures to employ planning to allow an agent to cope with situations that the designer was not able to foresee. This article describes and demonstrates one such architecture, which integrates propositional planning with BDI, allowing agents to take advantage of the practical reasoning capabilities (*i.e.* selecting and prioritising goals) of the BDI model, and replacing the BDI means-ends reasoning (*i.e.* selecting a course of action to achieve goals) with the flexibility of generic planning. Our approach is underpinned by a mapping among BDI mental states and propositional planning formalisms, allowing any algorithm based on a similar formalism to be used as a means-ends reasoning process for a BDI agent. In order to demonstrate the viability of such an approach we take a specific BDI agent model, namely the X-BDI model [27], and modify it to use propositional planning algorithms to perform means-ends reasoning [30].

The paper is organised as follows: Section 2 contains an overview of the related work and main concepts used throughout this paper; Section 3 describes X-BDI and the extensions that allow it to use an external planning algorithm; Section 4 contains a case study used to demonstrate the implemented prototype; finally, Section 5 contains concluding remarks about the results obtained in this work.

2. Agents and Planners. In this section we review background work on agents and planning systems, and conclude with a discussion of the integration of these technologies in an agent architecture, laying the groundwork for the remainder of this article. Section 2.1 provides an overview of computer agents and the BDI model, used in the agent architecture described later in this article; Section 2.2 introduces generic planning algorithms and problem representation; Section 2.3 describes the particular planning algorithm used in the prototype described in Section 3; finally, we discuss how these technologies can be pieced together in order to address some of their individual limitations.

2.1. Agents. The growing complexity of computer systems has led to the development of increasingly more advanced abstractions for their representation. An abstraction of growing popularity for representing parts of complex computer systems is the notion of *computer agents* [13], so far as to be proposed as an alternative to the Turing Machine as an abstraction for the notion of computation [19, 42]. Although there is a variety of definitions for computer agents, most researchers agree with Jennings' definition of an agent as *encapsulated* computer system, *situated* in some environment, and capable of *flexible*, *autonomous* action in that environment in order to meet its *design objectives* [19].

In the context of multi-agent systems research, one of the most widely known and studied models of deliberative agents uses *beliefs*, *desires* and *intentions* (BDI) as abstractions for the description of a system's behaviour. The BDI model originated from a philosophical model of human practical reasoning [6], later formalised [11] and improved towards a more complete computational theory [34, 44]. Though other approaches to the design of autonomous agents have been proposed [16], the BDI model or variations of it are used in many new architectures of autonomous agents [13, 31, 4, 40]. More specifically, the components that characterise the BDI model can be briefly described as follows [28]:

- **beliefs** represent an agent's expectation regarding the current world state or the possibility that a given course of action will lead to a given world state;
- **desires** represent a set of possibly inconsistent preferences an agent has regarding a set of world states; and
- **intentions** represent an agent's commitment regarding a given course of action, constraining the consideration of new objectives.

The operation of a generic BDI interpreter can be seen as a process that starts with an agent considering its sensor input and updating its belief base. With this updated belief base, a set of goals from the agent's desires is then selected, and the agent commits itself to achieving these goals. In turn, plans are selected as the means to achieve the goals through intentions which represent the commitment. Finally, these intentions are carried out through concrete actions contained in the instantiated plans (or intentions). This process is illustrated in the activity diagram of Figure 2.1, which shows the components of an agent that are used in each of the main processes of BDI reasoning, namely: obtaining sensor input and updating beliefs; selecting a goal from among the desires; and adopting intentions to carry out the actions required to achieve the selected goal.

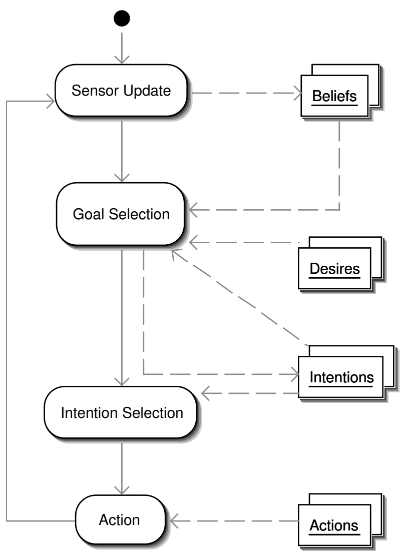

FIG. 2.1. *Activities of a generic BDI interpreter.*

This last process of selecting and adopting intentions to achieve a goal is one of the most important processes of the BDI model, since it affects not only the actions an agent chooses, but also the selection of goals, as an agent must drop goals deemed impossible. This problem of determining whether an agent is capable of satisfying its objectives through some sequence of actions given an environment and a set of objectives is sometimes characterised as the *agent design problem* [45]. The most widely known BDI agent implementations [18, 33, 14] bypass this problem through the use of plan libraries in which the courses of action for every possible objective an agent might have are stored as encapsulated procedures. Agents using these approaches are said to pursue *procedural goals*. However, the *theories* commonly used to underpin the creation of new plans of action at runtime assume an agent with unlimited resources, thus making their actual implementation impossible [37, 34]. When an agent selects target world-states and then uses some process at runtime to determine its course of action, it is said to pursue *declarative goals*. Recent efforts seek to deal with this problem in various ways, for instance by defining alternate proof systems [27, 31] or using model checking in order to validate the agent's plan library [5]. An alternative approach to solving the problem is the use of planning algorithms to perform means-ends reasoning at runtime [37, 26, 47].

2.2. Planning Algorithms. Means-ends reasoning is a fundamental component of any *rational* agent [6] and is useful in the resolution of problems in a number of different areas, such as scheduling [38], military strategy [39], and multi-agent coordination [12]. Indeed, the development of planning algorithms has been one of the main goals of AI research [35]. In more detail, a planning problem is generically defined by three components [43]:

- a formal description of the start state;
- a formal description of the intended goals; and
- a formal description of the actions that may be performed.

A planning system takes these components and generates a set of actions ordered by some relation which, when applied to the world in which the initial state description is true, makes the goals' description true. Despite the high complexity proven for the general case of planning problems[1], recent advances in planning research have led to the creation of planning algorithms that perform significantly better than previous approaches to solving various problem classes [43, 2]. These new algorithms make use of two main techniques, either combined or separately:

- expansion and search in a planning graph [3]; and
- compilation of the planning problem into a logical formula to be tested for satisfiability (SAT) [20].

One such planning algorithm is Graphplan, which we consider in more detail below.

2.3. Graphplan. Graphplan [3] is a planning algorithm based on the first of these techniques, expansion and search in a graph. It is considered to be one of the most efficient planning algorithms created recently [43, 38, 17], having been refined into a series of other algorithms, such as IPP (Interference Progression Planner) [22] and STAN (STate ANalysis) [24]. The efficiency of Graphplan was empirically demonstrated through the very significant results obtained by instances of Graphplan in the planning competitions of the AIPS (International Conference on AI Planning and Scheduling) [21, 25].

Planning in Graphplan is based on the concept of a graph data structure called the *planning graph*, in which information regarding the planning problem is stored in such a way that the search for a solution can be accelerated. Planning graph construction is efficient, having polynomial complexity in graph size and construction time with regard to problem size [3]. A plan in the planning graph is essentially a flow, in the sense of a network flow, and the search for a solution to the planning problem is performed by the planner using data stored in the graph to speed up the process. The basic Graphplan algorithm (*i.e.* without the optimisations proposed by other researchers [21, 25]) is divided into *graph expansion* and *solution extraction*, which take place alternately until either a solution is found or the algorithm can prove that no solution exists. The way these two parts of Graphplan are used throughout planning is summarised in the activity diagram of Figure 2.2, and explained below.

Since 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, making it a directed and levelled graph, as shown in Figure 2.3. Proposition levels are composed of proposition nodes labelled with propositions, and connected to the actions in the subsequent action level through pre-condition arcs. Here, 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, so that the first proposition level represents the literals that are possibly true at time

[1]Planning is known to be undecidable [10] and planning problems, in the general case, have PSPACE complexity [9].

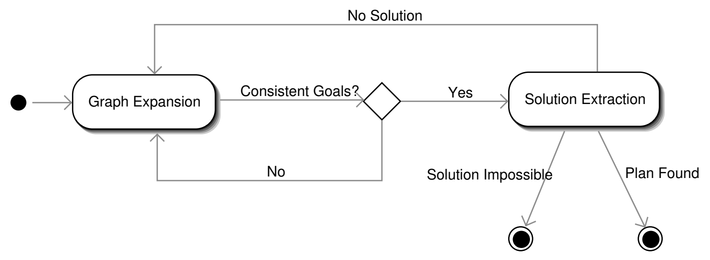

Fig. 2.2. *Graphplan algorithm overview.*

1, the next proposition level represents the literals that are possibly true at time 2 and so forth. Similarly, 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, the second action level represents the operators that may be executed at time 2 and so forth. The graph also contains mutual exclusion relations (*mutex*) between nodes (at the same graph level) so that they cannot be simultaneously present at the same graph level for the same solution. This gives them a fundamental role in algorithm efficiency, as they allow the search for a solution to completely ignore a large number of possible flows in the graph.

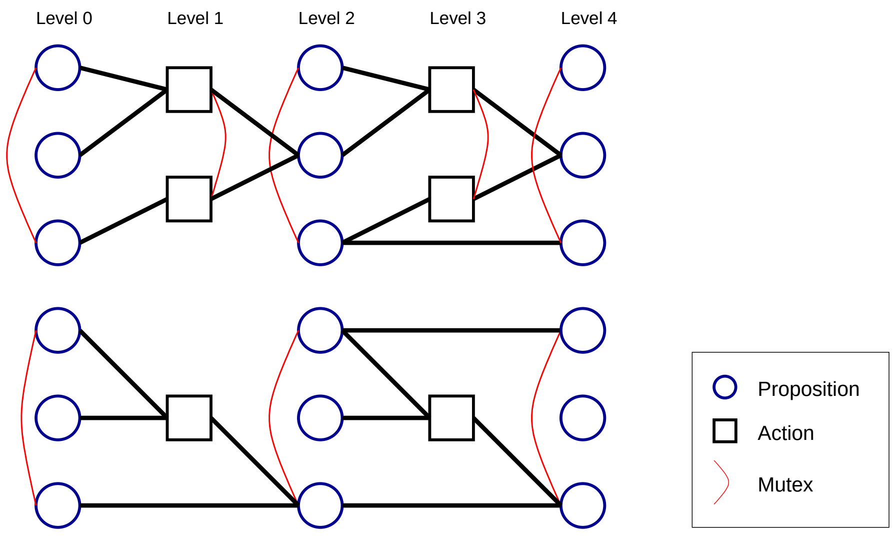

Fig. 2.3. *A planning graph example.*

After graph expansion, the graph is analysed by the solution extraction part of the algorithm, which uses a backward chaining strategy to traverse the graph, level by level, trying to find a flow starting from the goals and leading to the initial conditions. An important optimising factor in this phase is never to search for a solution unless all the *goal* propositions are present and consistent, since *they cannot be mutually exclusive at the last graph level.* Fundamental to Graphplan is its assurance that, whenever a plan for the proposed problem exists, the algorithm will find it, otherwise the algorithm will determine that the proposed problem is unsolvable [3].

2.4. Discussion. When one considers how BDI reasoning operates, it is straightforward to perceive that propositional planning can be used as a means-ends reasoning component. From a representational point of view, BDI mental states can be converted to planning problems without complication: beliefs translate into an initial state specification, actions and capabilities translate into operator specifications and selected goals translate into a goal state specification. At this simple level, the delegation of means-ends reasoning to an external planning process can improve the runtime efficiency of existing BDI interpreters by leveraging advances in planning algorithms research.

3. Introducing procedural planning into X-BDI.

3.1. Introduction. Given the shortcomings of traditional BDI architectures in terms of runtime flexibility, and the performance problems of alternative architectures, we define an extended version of the X-BDI agent model [27], modified to accommodate the use of an external planning component. Here, we focus on STRIPS-like (STanford Research Institute Problem Solver) formalisms [15]. Our formalism is based on the one introduced by Nebel [30], and, according to the author, is 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 formalism defined by Nebel [30] is more general, but since we do not aim to provide a detailed study of planning formalisms, we use a simpler version. In particular, we use a propositional logical language with variables only in the specification of operators, and with operators not being allowed to have conditional effects. In Nebel's description of the the STRIPS formalism, one can notice that it deals 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 limitations through the use of syntactic transformations so that planners can operate over first order ground literals. The main contribution of our work lies in the efficiency improvement of a *declarative* agent architecture. The fact that this type of agent architecture has traditionally been notoriously inefficient highlights the relevance of this efficiency gain.

3.2. X-BDI. An X-BDI agent has the traditional components of a BDI agent, *i.e.* a set of *beliefs*, *desires* and *intentions*. The agent model was originally defined in terms of the *Extended Logic Programming with explicit negation* (ELP) formalism created by Alferes and Pereira [1], which includes a revision procedure responsible for maintaining logic consistency. We do not provide a description of the formalism here, though we assume the presence of its revision procedure in our description of X-BDI. Given its extended logic definition, X-BDI also has a set of time axioms defined through a variation of the *Event Calculus* [27, 23].

The set of beliefs is simply a formalisation of facts in ELP, individualised for a specific agent. From the agent's point of view, it is assumed that its beliefs are not always consistent, and whenever an event makes the beliefs inconsistent, they must be revised. The details of this process are unimportant in the understanding of the overall agent architecture, but can be found in [1]. The belief revision process in X-BDI is the result of the program revision process performed in ELP.

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, some property P is unconditionally desired. Desires may be temporally situated, *i.e.* can be desired at a specific moment, or whenever their pre-conditions are valid. Moreover, a desire specification contains 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 not intend something in the past or that is already true, and intentions must in principle be possible, *i.e.* there must be at least one plan available whose result is a world state where the intended property is true.

Now, we diverge from the original X-BDI architecture in several respects. First, the original reasoning process verified the possibility of a property through the abduction of an event calculus theory to validate the property. In brief, the logic representation of desires in the original X-BDI included clauses specifically marked for revision in such a way that sequences of actions (whose preconditions and effects were described in event calculus) could be found true in the process of revising these clauses. This abduction process was necessary for the implementation of X-BDI planning framework in extended logic, but the implementation of the logic interpreter was notably inefficient for abductive reasoning. In this work, the planning process is abstracted out from the operational definition of X-BDI, allowing any planning component that satisfies the conditions of Section 2.2 to be invoked by the agent. 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 begins with the selection of *eligible desires*, which represent unsatisfied desires whose pre-conditions are valid, though the elements of this set of desires are not necessarily consistent among themselves. A set of eligible desires that are both consistent and possible is then selected as *candidate desires*, to which the agent commits itself to achieving by adopting them as *primary intentions.* In order to achieve the primary intentions, the planning process generates a sequence of temporally ordered actions that constitute the *relative intentions.* This process is summarised in Figure 3.1.

Eligible desires have rationality constraints that are similar to those imposed by Bratman [6] over intentions in the sense that an agent will not desire something in the past or something the agent believes will happen without its interference. Agent beliefs must also support the pre-conditions defined in the desire *Body*. Within the agent's reasoning process these desires give rise to a set of mutually consistent subsets ordered by a partial order relation.

The process of selecting candidate desires seeks to choose from the eligible desires one subset that contains only desires that are internally consistent and possible. A possible desire in this sense 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, the original X-BDI uses ELP constructs that allow desires to be prioritised in the revision process. Although we depart from the original abduction theory, we still use these priority values to define a desire preference relation. 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 [27]. 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 an abduction process is, actually, a form of planning. Since our main objective in this paper is to distinguish the planning process previously hard-coded within X-BDI, the notion of desire possibility must be re-defined. Therefore, we define the set of candidate desires to be the subset of eligible desires with the greater preference value, and whose properties can be satisfied.

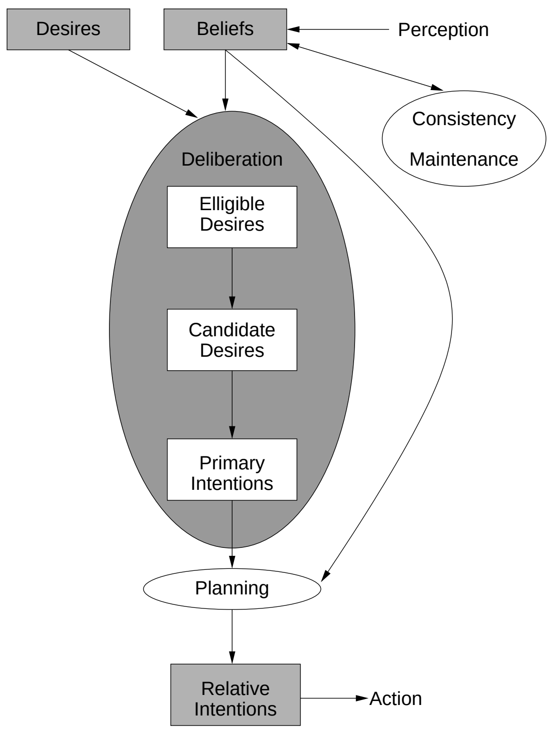

FIG. 3.1. *X-BDI operation overview.*

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 of primary intentions. The modified reasoning process for X-BDI is illustrated in Figure 3.2.

Primary intentions can be seen as high-level plans, similar to the intentions in IRMA [7], and representing the agent's commitment to a course of action. These primary intentions are systematically refined 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 then correspond to the temporally ordered steps of the concrete plans generated to satisfy the agent's primary intentions. Thus the notion of agent commitment results from the fact that relative intentions must not contradict or annul primary intentions.

3.3. 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. Intention reconsideration should therefore 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.

These trigger conditions are defined to enforce Bratman's [6] rationality conditions for BDI components, as follows. If all of the agent's primary intentions are satisfied

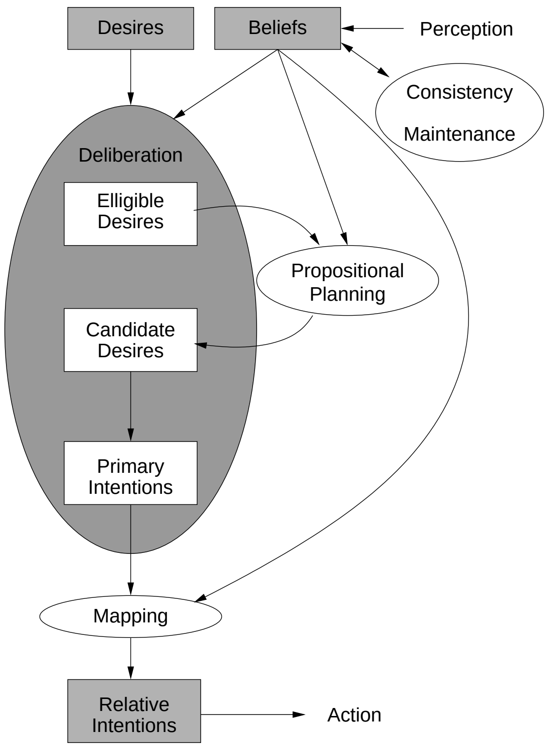

FIG. 3.2. *Modified X-BDI overview.*

before the time planned for them to be satisfied, the agent restarts the deliberative process, since it has achieved its goals. On the other hand, if one of the primary intentions is not achieved at the time planned for it, the agent must 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 reconsiders its desires in order to take advantage of the new opportunity. Reconsideration is completely based on integrity constraints over beliefs, and since beliefs are revised at every sensoring cycle, it is possible that reconsideration occurs due the *triggering* of a reconsideration restriction.

3.4. Implementation. The prototype implemented for this work is 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 visualise its interaction with the environment. The architecture is outlined in Figure 3.3.

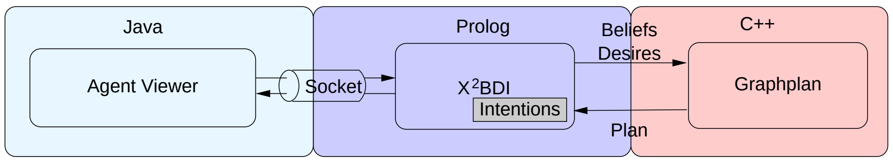

FIG. 3.3. *Solution Architecture*

Here, the *Agent Viewer* interface communicates with X-BDI through sockets by sending the input from the environment in which the agent is embedded and receiving the result of the agent's deliberation. Through the *Agent Viewer* the user can also specify the agent in terms of its desires, actions and initial beliefs. Once X-BDI receives the agent specification, it communicates with the planning module through operating system files and the Prolog/C++ interface. The planner is responsible for generating a set of intentions for the agent. When the agent deliberates, it converts subsets of the agent's desired properties into propositional planning problems and invokes the planning algorithm to solve these problems until either a plan that solves 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. In particular, the rational utilisation of equipment in industrial facilities is a complex problem, especially scheduling its use. This problem is complicated when the facility produces multiple component types, where each type requires a subset of the equipment available. In our test scenario, the proposed production cell [46], illustrated in Figure 4.1, is composed of seven devices: a *feed belt*, a *deposit belt* and four *processing units* upon which components are moved to be processed.

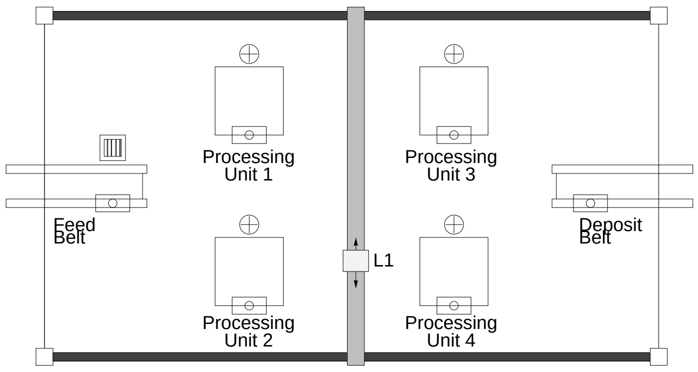

FIG. 4.1. *A BDI Production Cell.*

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 on the component being processed, and can hold only one component at a given moment. Each component introduced into the cell can be processed by one or more processing units, determined by the type of component being processed, and different component types have different processing priorities. The control of the production cell is entrusted to a BDI agent implemented using X-BDI, which should schedule the work of the production cell in relation to its beliefs, desires and intentions, re-scheduling whenever some change in the system 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. We define the following predicates representing objects in the cell:

- `component(C)` denotes that `C` is a *component* to be processed;

- `procUnit(P)` denotes that `P` is a *processing unit*, which is also a device;
- `device(D)` denotes that `D` is a *device*;
- `feedBelt` represents the *feed belt*;
- `depositBelt` represents the *deposit belt*.

Similarly, we have the following predicates representing system states:

- `over(C,D)` denotes that component `C` is over device `D`;
- `empty(P)` denotes that processing unit `P` is empty, *i.e.* has no component over it;
- `processed(C,P)` denotes that component `C` has already been processed by processing unit `P`;
- `finished(C)` denotes that component `C` 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, these are summarised in Table 4.1. Informally, action `process(C,P)` represents the processing that a processing unit `P` performs on a component `C` over it; `consume(C)` represents the removal of component `C` from the production cell through the deposit belt; and `move(C,D1,D2)` represents the motion of component `C` from device `D1` to device `D2`.

Action	Preconditions	Effects

`process(C,P)`	`procUnit(P)`
	`component(C)`
	`over(C,P)`
		`processed(C,P)`

`consume(C)`	`component(C)`
	`over(C,depositBelt)`
		¬`over(C,depositBelt)`
		`empty(depositBelt)`
		`finished(C)`

`move(C,D1,D2)`	`over(C,D1)`
	`empty(D2)`
	`component(C)`
	`device(D1)`
	`device(D2)`
		`over(C, D2)`
		¬`over(C,D1)`
		¬`empty(D2)`
		`empty(D1)`

TABLE 4.1
Action specification for the production cell agent.

The processing requirements of components and their priorities are modelled through desires. Thus, we can model an agent, `pCell`, which needs to process component `comp1` by processing units `procUnit1`, `procUnit2` and `procUnit3` as soon as this component is inserted into the production cell using the specification of Listing 1[2].

Similarly, we can model the agent's need to process component `bloc2` by processing unit `procUnit3` and `procUnit4` by adding to the agent specification the desires of Listing 2.

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 beliefs specified in Listing 3.

The arrival of a new component in the production cell is signalled by the sensors through the inclusion of `component(comp1)` and `over(comp1,feedBelt)` in the agent's belief database, activating the agent's reconsideration process. Given the desire's

[2]`Tf` is the time at which the desire is valid, and the values `0.7` and `0.6` are the desires priorities.

Listing 1
Specification of desires related to processing `comp1`.

```
des(pCell,finished(comp1),Tf,[0.7])
 if bel(pCell, component(comp1)),
    bel(pCell, processed(comp1,procUnit1)),
    bel(pCell, processed(comp1,procUnit2)),
    bel(pCell, processed(comp1,procUnit3)),
    bel(pCell, -finished(comp1)).

des(pCell,processed(comp1,procUnit1),Tf,[0.6])
 if bel(pCell, component(comp1)),
    bel(pCell, -processed(comp1,procUnit1)).

des(pCell,processed(comp1,procUnit2),Tf,[0.6])
 if bel(pCell, component(comp1)),
    bel(pCell, -processed(comp1,procUnit2)).

des(pCell,processed(comp1,procUnit3),Tf,[0.6])
 if bel(pCell, component(comp1)),
    bel(pCell, -processed(comp1,procUnit3)).
```

Listing 2
Specification of desires related to processing `comp2`.

```
des(pCell,finished(comp2),Tf,[0.6])
 if bel(pCell, component(comp2)),
    bel(pCell, processed(comp2,procUnit3)),
    bel(pCell, processed(comp2,procUnit4)),
    bel(pCell, -finished(comp2)).

des(pCell,processed(comp2,procUnit3),Tf,[0.5])
 if bel(pCell, component(comp2)),
    bel(pCell, -processed(comp2,procUnit3)).

des(pCell,processed(comp2,procUnit4),Tf,[0.5])
 if bel(pCell, component(comp2)),
    bel(pCell, -processed(comp2,procUnit4)).
```

pre-conditions previously defined, only the desires related to the following properties become eligible:

- `processed(comp1,procUnit1)`;
- `processed(comp1,procUnit2)`;
- `processed(comp1,procUnit3)`;

These desires are then analysed 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:

Listing 3
Domain knowledge for the production cell.

```
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)).
```

1. `move(comp1,feedBelt,procUnit2)`
2. `process(comp1,procUnit2)`
3. `move(comp1,procUnit2,procUnit1)`
4. `process(comp1,procUnit1)`
5. `move(comp1,procUnit1,procUnit3)`
6. `process(comp1,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 generate primary intentions representing the agent's commitment. Next, relative intentions are generated using the steps in the recently created plan, with one intention for each step of the plan. These 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(comp1)` becomes true, reactivating the agent's deliberative process and generating the following plan:

1. `move(comp1,procUnit3,depositBelt)`
2. `consume(comp1)`

Once more, this plan brings about some intentions and, eventually, leads the agent to act. Now, suppose that during the agent's operation, a new component in the production cell arrives. If this occurred immediately after the deliberation that created the first plan, it would be signaled by the agent's sensors through the inclusion of `component(comp2)` and `over(comp2,feedBelt)` in the beliefs database, which would modify the eligible desires chosen in the second deliberation cycle to:

- `finished(comp1)`;
- `processed(comp2,procUnit3)`;
- `processed(comp2,procUnit4)`;

These desires become candidate desires because Graphplan is capable of generating a plan that satisfies all the desires. The new plan is:

1. `move(comp1,procUnit3,depositBelt)`
2. `move(comp2,feedBelt,procUnit4)`

3. `consume(comp1)`
4. `process(comp2,procUnit4)`
5. `move(comp2,procUnit4,procUnit3)`
6. `process(comp2,procUnit3)`
7. `move(comp2,procUnit3,depositBelt)`
8. `consume(comp2)`

The steps of this plan thus generate relative intentions, eventually leading the agent to the execution of its actions.

5. Conclusions. In this paper, we have discussed the relationship between propositional planning algorithms and means-end reasoning in BDI agents. To test the viability of using propositional planners to perform means-ends reasoning in a BDI architecture, we have described a modification to the X-BDI agent model. Throughout this modification, new definitions of desires and intentions were created in order for the agent 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 for 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 through its fully declarative specification. However, its planning mechanism is notably inefficient. For example, the case study described in Section 4 was not tractable in the original X-BDI planning process. Thus, the main contribution of our work consists in addressing this limitation through the definition of a mapping from BDI means-end reasoning to fast planning algorithms. Moreover, such an 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.

Other approaches to performing runtime planning have also been proposed, the most notable recent ones by Sardina *et al.* [36] and Walczak *et al.* [41]. Sardina proposes the tight integration of the JACK agent framework [8] with the SHOP hierarchical planner [29]. This approach relies on new constructs added to an otherwise procedural agent representation and takes advantage of the similarity of *hierarchical task network* (HTN) planning to BDI reasoning. The work of Walczak proposes the integration of JADEX [32] to a customised knowledge-based planner operating in parallel to agent execution, using a similar process of agent-state conversion to work of Meneguzzi *et al.* [26, 47], as well as the one presented in this paper.

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. It is clear that other agent architectures can benefit from the usage of planning components to allow agents to cope with unforeseen events at runtime, as demonstrated by recent efforts in planning agents [36, 41]. Therefore, investigating how to integrate planning capabilities to AgentSpeak-based agents could create agents that can take advantage of both the fast response of pre-compiled plans and the flexibility of being able to plan at runtime to cope with unforeseen situations.

REFERENCES

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

[2] S. Biundo, K. L. Myers, and K. Rajan, eds., *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, AAAI, 2005.

[3] A. L. Blum and M. L. Furst, *Fast planning through planning graph analysis*, Artificial Intelligence, 90 (1997), pp. 281–300.

[4] R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, *Multi-Agent Programming: Languages, Platforms and Applications*, vol. 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer, 2005.

[5] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge, *Model checking AgentSpeak*, in Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS-03), Melbourne, Australia, July 2003, ACM Press, pp. 409–416.

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

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

[8] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas, *Jack intelligent agents - components for intelligent agents in java*. AgentLink Newsletter, January 1999. White paper, http://www.agent-software.com.au.

[9] T. Bylander, *The computational complexity of propositional STRIPS planning*, Artificial Intelligence, 69 (1994), pp. 165–204.

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

[11] P. R. Cohen and H. J. Levesque, *Intention is choice with commitment*, Artificial Intelligence, 42 (1990), pp. 213–261.

[12] J. S. Cox, E. H. Durfee, and T. Bartold, *A distributed framework for solving the multiagent plan coordination problem*, in AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, 2005, ACM Press, pp. 821–827.

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

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

[15] R. Fikes and N. Nilsson, *STRIPS: A new approach to the application of theorem proving to problem solving*, Artificial Intelligence, 2 (1971), pp. 189–208.

[16] M. Georgeff, B. Pell, M. E. Pollack, M. Tambe, and M. Wooldridge, *The belief-desire-intention model of agency*, in Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98), J. Müller, M. P. Singh, and A. S. Rao, eds., vol. 1555, Springer-Verlag: Heidelberg, Germany, 1999, pp. 1–10.

[17] J. Hoffmann and B. Nebel, *The FF planning system: Fast plan generation through heuristic search*, Journal of Artificial Intelligence Research (JAIR), 14 (2001), pp. 253–302.

[18] F. F. Ingrand, M. P. Georgeff, and A. S. Rao, *An architecture for real-time reasoning and system control*, IEEE Expert, Knowledge-Based Diagnosis in Process Engineering, 7 (1992), pp. 33–44.

[19] N. R. Jennings, *On agent-based software engineering*, Artificial Intelligence, 117 (2000), pp. 277–296.

[20] H. Kautz and B. Selman, *Planning as satisfiability*, in Proceedings of the Tenth European Conference on Artificial Intelligence, Chichester, UK, 1992, Wiley, pp. 359–363.

[21] J. Köhler, *Solving complex planning tasks through extraction of subproblems*, in Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, R. Simmons, M. Veloso, and S. Smith, eds., Pittsburg, 1998, AAAI Press, pp. 62–69.

[22] J. Köhler, B. Nebel, J. Hoffmann, and Y. Dimopoulos, *Extending planning graphs to an ADL subset*, in Proceedings of the 4th European Conference on Planning, S. Steel, ed., vol. 1348 of Lecture Notes in Computer Science, Springer Verlag, Germany, 1997, pp. 273–285.

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

[24] D. Long and M. Fox, *Efficient implementation of the plan graph in STAN*, Journal of Artificial Intelligence Research (JAIR), 10 (1999), pp. 87–115.

[25] D. Long and M. Fox, *Automatic synthesis and use of generic types in planning*, in Pro-

ceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, Colorado, 2000, AAAI Press, pp. 196–205.

[26] F. R. Meneguzzi, A. F. Zorzo, and M. D. C. Móra, *Propositional planning in BDI agents*, in Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004, ACM Press, pp. 58–63.

[27] M. d. C. Móra, J. G. P. Lopes, R. M. Vicari, and H. Coelho, *BDI models and systems: Bridging the gap.*, in Intelligent Agents V, Agent Theories, Architectures, and Languages, Fifth International Workshop, ATAL '98, vol. 1555 of LNCS, Springer, Paris, France, 1999, pp. 11–27.

[28] J. P. Müller, *The design of intelligent agents: A layered approach*, in The Design of Intelligent Agents: A Layered Approach, vol. 1177 of Lecture Notes in Computer Science, Springer Verlag, Germany, 1996.

[29] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, *Shop: Simple hierarchical ordered planner.*, in IJCAI, 1999, pp. 968–975.

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

[31] N. Nide and S. Takata, *Deduction systems for BDI logics using sequent calculus*, in Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, 2002, pp. 928–935.

[32] A. Pokahr, L. Braubach, and W. Lamersdorf, *Jadex: Implementing a bdi-infrastructure for jade agents*, EXP - in search of innovation (Special Issue on JADE), 3 (2003), pp. 76–85.

[33] A. S. Rao, *AgentSpeak(L): BDI agents speak out in a logical computable language*, in Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, W. V. de Velde and J. W. Perram, eds., vol. 1038 of LNCS, Springer, Eindhoven, The Netherlands, 1996, pp. 42–55.

[34] A. S. Rao and M. P. Georgeff, *Formal models and decision procedures for multi-agent systems*, Tech. Report 61, Australian Artificial Intelligence Institute, 171 La Trobe Street, Melbourne, Australia, 1995. Technical Note.

[35] S. J. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, 1994.

[36] S. Sardina, L. de Silva, and L. Padgham, *Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach*, in AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, 2006, ACM Press, pp. 1001–1008.

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

[38] D. E. Smith and D. S. Weld, *Temporal planning with mutual exclusion reasoning*, in Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), 1999, pp. 326–337.

[39] J. R. Surdu, J. M. D. Hill, and U. W. Pooch, *Anticipatory planning support system*, in WSC '00: Proceedings of the 32nd conference on Winter simulation, San Diego, CA, USA, 2000, Society for Computer Simulation International, pp. 950–957.

[40] B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. C. Meyer, *Dynamics of declarative goals in agent programming.*, in LNCS, vol. 3476, 2004, pp. 1–18.

[41] A. Walczak, L. Braubach, A. Pokahr, and W. Lamersdorf, *Augmenting BDI Agents with Deliberative Planning Techniques*, in The Fifth International Workshop on Programming Multiagent Systems (PROMAS-2006), 2006.

[42] P. Wegner, *Why interaction is more powerful than algorithms*, Communications of the ACM, 40 (1997), pp. 80–91.

[43] D. S. Weld, *Recent Advances in AI Planning*, AI Magazine, 20 (1999), pp. 93–123.

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

[45] M. Wooldridge, *The Computational Complexity of Agent Design Problems*, in Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000), E. Durfee, ed., IEEE Press, 2000, pp. 341–348.

[46] A. F. Zorzo, L. A. Cassol, A. L. Nodari, L. A. Oliveira, and L. R. Morais, *Scheduling safety-critical systems using a partial order planning algorithm*, in Advances in Logic, Artificial Intelligence and Robotics, São Paulo, Brazil, 2002, IOS Press, pp. 33–40.

[47] A. F. Zorzo and F. R. Meneguzzi, *An agent model for fault-tolerant systems*, in SAC '05: Proceedings of the 2005 ACM symposium on Applied computing, New York, NY, USA, 2005, ACM Press, pp. 60–65.