A BDI Agent Architecture for a POMDP Planner

Gavin Rens[1,2]
Alexander Ferrein[3]
Etienne van der Poel[1]
[1] School of Computing, Unisa, Pretoria, South Africa
[2] Knowledge Systems Group, Meraka Institute, CSIR, Pretoria, South Africa
[3] Robotics and Agents Research Laboratory, University of Cape Town, South Africa
grens@csir.co.za, alexander.ferrein@uct.ac.za, evdpoel@unisa.ac.za

Abstract

Traditionally, agent architectures based on the Belief-Desire-Intention (BDI) model make use of *pre-compiled* plans, or if they do *generate* plans, the plans do not involve stochastic actions nor probabilistic observations. Plans that *do* involve these kinds of actions and observations are generated by partially observable Markov decision process (POMDP) planners. In particular for POMDP planning, we make use of a POMDP planner which is implemented in the robot programming and plan language Golog. Golog is very suitable for integrating beliefs, as it is based on the situation calculus and we can draw upon previous research on this. However, a POMDP planner on its own cannot cope well with dynamically changing environments and complicated goals. This is exactly a strength of the BDI model; the model is for reasoning over goals dynamically. Therefore, in this paper, we propose an architecture that will lay the groundwork for architectures that combine the advantages of a POMDP planner written in the situation calculus, and the BDI model of agency. We show preliminary results which can be seen as a proof of concept for integrating a POMDP into a BDI architecture.

Introduction

Traditionally, plan-based agents that include generative planning (as opposed to utilizing pre-compiled plans) would generate a complete plan to reach a *specific fixed* goal, then execute the plan. If plan execution monitoring is available, the agent would replan from scratch when the plan becomes invalid. Due to the time requirements for generating complete plans, the plan may be invalid by the time it is executed. This is because the world may change substantially during plan generation.

Therefore, Belief-Desire-Intention (BDI) architectures take a different approach. BDI theory is based on the philosophy of practical reasoning (Bratman 1987). It offers flexibility in planning beyond traditional planning for agents, by reasoning over *different goals*. That is, an agent based on BDI theory can adapt to changing situations by focusing on the pursuit of the most appropriate goal at the time. Typically, an appropriate plan to achieve an adopted goal is then selected from a data base of plans. Although a plan that satisfies certain constraints (e.g., does not conflict with other adopted plans, is executable, etc.) will be adopted, it may not be the most appropriate plan in existence. A plan that is generated with the agent's current knowledge for guidance, may be more appropriate. BDI agents can also make rational decisions as to when to replan if a plan becomes invalid, reducing the amount of replanning, thus increasing the agent's reactivity. Note that the BDI model is, however, not the only approach to replanning (cf. (Likhachev et al. 2005)).

In general, BDI architectures do not make use of plan generation, they rather draw on plan libraries. While with BDI approaches, an agent can reason over several goals, the agent lacks some flexibility by not being able to generate suitable plans on demand. Therefore, in this paper, we aim at integrating a POMDP planner into a BDI architecture to combine its benefits with the ability to *generate* plans. Moreover, we want to supply models that are as realistic as possible. We therefore decided on employing partially observable Markov Decision Processes (POMDPs).

In this paper we describe our approach for combining BDI theory with a POMDP planner. Combining the two formalisms can be viewed from two perspectives. One, to enhance an existing planner for use in real-time dynamic domains by incorporating the planner into a BDI agent architecture so that the management of goal selection, planning and replanning is handled in a principled way. Two, to enhance the classical BDI agent architecture by incorporating a POMDP planner into the BDI architecture so that the agent can reason (plan) with knowledge about the uncertainty of the results of its actions, and about the uncertainty of the accuracy of its perceptions. We employ the POMDP planner described in our previous work (Rens, Ferrein, and Van der Poel 2008). This planner is implemented in Golog (Levesque et al. 1997), which in turn is based on the situation calculus (McCarthy 1963; Reiter 2001). An advantage of using a Golog implementation for the planner is that the integration of beliefs into the situation calculus has previously been done (e.g., (Bacchus, Halpern, and Levesque 1999)) and this work can be used for formulating POMDPs. Further, given a background action theory, an initial state and a goal state (or reward function in POMDPs), Golog programs essentially constrain and specify the search space (the space of available actions).

The resulting plan (or *policy* in POMDPs) is a Golog program which can be executed directly by the agent. To the

best of our knowledge, till present, no BDI-based agent architecture has implemented its planning function so as to generate plans that take stochastic action and partial observation into account. Therefore, this work can be seen as a first proof of this concept.

The rest of the paper is organized as follows. In the next section we introduce the plan generator used in this study. Then, we briefly introduce the BDI theory, after which we explain our hybrid BDI/POMDP-planner architecture in detail. Before we conclude, we show some preliminary results from an implementation of our architecture, which gives a first proof of our approach.

The Planning Module

The POMDP Model

In partially observable Markov decision processes (POMDPs) actions have nondeterministic results, yet may be predicted with a probability of occurrence. And observations are uncertain: the world is not directly observable, therefore the agent *infers* how likely it is that the world is in some specific state. The agent thus believes to some degree—for each possible state—that it is in that state. Furthermore, a POMDP is a *decision* process and thus facilitates making decisions as to which actions to take, given its previous observations and actions. Formally, a POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, b_0 \rangle$ with: $\mathcal{S}$, a finite set of states of the world; $\mathcal{A}$, a finite set of actions; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ is the *state-transition function*, where Π is a probability distribution; $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, the *reward function*; Ω, a finite set of observations the agent can experience; $\mathcal{O} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\Omega)$, the *observation function*; and b_0, the initial probability distribution over all world states in $\mathcal{S}$ (see e.g., (Kaelbling, Littman, and Cassandra 1998)). In the model, b is a *belief state*, i.e. a set of pairs (s, p) where each state $s \in \mathcal{S}$ is associated with a probability p. The *state estimation* function $SE(b, a, o)$ updates the agent's beliefs. Now the aim of the agent deploying a POMDP model is to determine a policy, that is, the actions or decisions that will maximize its rewards. Formally, a policy π is a function from a set of belief states B to the set of actions: $\pi : B \rightarrow A$. That is, actions are *conditioned* on beliefs. This means that the agent takes its next decision not only based on a stochastic action model, but also on a stochastic observation model. In this sense, a policy can be represented as a *policy tree*, with nodes being actions and branches being observations.

Planning over Degrees of Belief

In this section we describe our POMDP planner, an extension to the decision-theoretic language, DTGolog (Boutilier et al. 2000).

DTGolog is based on Reiter's variant of the situation calculus (McCarthy 1963; Reiter 2001), a second-order language for reasoning about actions and their effects. According to this calculus, changes in the world are due only to actions, so that a situation is completely described by the history of actions starting in some initial situation—$do(a, s)$ is the term denoting the situation resulting from doing action a in situation s. Properties of the world are described by *fluents*, which are situation-dependent predicates and functions. For each fluent the user defines a successor state axiom specifying precisely which value the fluent takes on after performing an action. These, together with precondition axioms for each action, axioms for the initial situation, and foundational and unique names axioms, form a so-called *basic action theory* (Reiter 2001).

Decision-theoretic planning in DTGolog works roughly as follows. Given an input program that leaves open several action alternatives for the agent, the DTGolog interpreter generates an optimal policy. Formally, the interpreter solves a Markov Decision Process (MDP, cf. e.g., (Puterman 1994)) using the forward search value iteration method—searching (to a specified horizon) for the actions that will maximize the total expected reward. Programs are interpreted as follows: All possible outcomes of the intended nondeterministic, stochastic action are expanded. For each choice point, the action resulting in the optimal value at the particular point in the MDP, is determined. These values are calculated relative to the world situation associated with the point in the MDP. The policy is calculated with an optimization theory consisting of a reward and a transition function (cf. also (Boutilier et al. 2000)). The transition function describing transition probabilities between states of the Markov chain is given by Reiter's variant of the basic action theory formalized in the underlying situation calculus (McCarthy 1963; Reiter 2001). Formally, the $BestDo$ macro defines the process described above: it evaluates an input program and recursively builds an optimal policy.

The POMDP planner we use here is $BestDoPO$ (Rens, Ferrein, and Van der Poel 2008); an extension of $BestDo$ ($BestDo$ *P*artially *O*bservable), which calculates an optimal policy for the partially observable case (Rens, Ferrein, and Van der Poel 2008). The main difference is that $BestDoPO$ operates on a belief state rather than on a world state. $BestDoPO(p, b, h, \pi, v, pr)$ takes as arguments a Golog program p, a belief state b and a horizon h, which determines the solution depth sought by the interpreter. The policy π as well as its value v and the success probability pr are returned. After a certain action a is performed and the associated observation o is perceived, the next belief state is determined via a belief state transition function (similar in vein to the state estimation function of the previous subsection, and the successor-state axiom for likelihood weights as given in (Bacchus, Halpern, and Levesque 1999)):

$$
\begin{aligned}
&b_{new} = BU(o, a, b) \doteq \\
&\quad b_{temp} = \{(s^+, p^+) \mid (\exists n, s^+, p^+).(s^+, p^+) \in b_{temp} : \\
&\qquad s^+ = do(n, s) \wedge choiceNat(n, a, s) \wedge PossAct(n, s) \wedge \\
&\qquad p^+ = p \cdot probObs(o, a, s^+) \cdot probNat(n, a, s)\} \\
&\quad b_{new} = normalize(b_{temp}).
\end{aligned}
$$

$choiceNat(n, a, s)$ specifies the possible outcomes n of the agent's intention to perform action a. $PossAct(n, s)$ denotes the possibility of performing action n in situation s. $probObs(o, a, s^+)$ and $probNat(n, a, s)$ are functions that

return the probability of observing o in the situation s^+—the situation resulting from doing action a, and respectively, the probability of action n being the outcome of the intention to execute action a in situation s.

For $BestDoPO$ to be integrated as required for the present work, two arguments are added to the list: $BestDoPO$ as defined in (Rens, Ferrein, and Van der Poel 2008) is modified to return δ and to take nom. The input program may provide information for a sequence of actions of length greater than the policy horizon. Call the remaining program δ—the portion of the program that was not used for policy generation. δ becomes the new program from which future policies will be generated. nom—the name of the input program—is used to select the reward function associated with the input program. Two clauses that are part of the definition of the modified $BestDoPO$ appear below.

$$BestDoPO(p, \delta, nom, b, h, \pi, v, pr) \stackrel{def}{=}$$
$$h = 0 \wedge \delta = p \wedge$$
$$\pi = stop \wedge \exists v.believedReward(nom, v, b) \wedge pr = 1.$$
$$BestDoPO(a : p, \delta, nom, b, h, \pi, v, pr) \stackrel{def}{=}$$
$$\neg actionBelievedPossible(a, b) \wedge$$
$$\delta = p \wedge \pi = stop \wedge v = 0 \wedge pr = 0 \vee$$
$$actionBelievedPossible(a, b) \wedge$$
$$\exists\, obs.setofAssocObservations(a, obs) \wedge$$
$$\exists \pi', v', pr.Aux(obs, a, p, \delta, nom, b, h, \pi', v', pr) \wedge$$
$$believedReward(nom, r, b) \wedge \pi = a; \pi' \wedge v = r + v'.$$

Please refer to (Rens, Ferrein, and Van der Poel 2008) for more detail.

BDI Theory

A desire is understood as what an agent ideally wants to achieve, that is, what motivates it. In reality, agents are resource-bounded, and hence should rationally choose the desires to pursue whose achievement are most valuable to the agent and that are achievable according to the agent's current situation and capabilities. The desires that have been committed to pursuing through a rational process of reasoning may be called *intentions*. The Belief-Desire-Intention (BDI) model of agency takes intentions—in addition to beliefs and desires—as first-class mental states. Traditional agent architectures either simply do not consider intentions, or do not consider them as explicit operands within the processes of an agent's reasoning system.

The value of taking intentions seriously is that they manage the agent's resources in a rational way. Intentions induce the agent to act and intentions persist. As such, they focus the agent's activity to commit resources and thus pursue a desire more effectively. Also, because intentions persist, new intentions are not constantly being adopted: new intentions are constrained by current intentions, and hence, future deliberation is constrained (Wooldridge 2000).

It is useful to distinguish between *deliberation*: to decide on what ends (e.g., reward functions; goal states) to pursue and *means-ends reasoning*: how to achieve the ends. Deliberation may be further divided into (i) reasoning to generate options from beliefs, i.e., 'wishing' to decide on current desires; (ii) reasoning to select intentions, i.e., 'focusing' on a subset of those desires and committing to achieve them. Committed-to goals, or plans for achieving them, are *intentions*.

A BDI agent has at least these seven components (Wooldridge 1999):

- A knowledge base of beliefs.
- An option generation function ($wish$), generating the options the agent would ideally like to pursue (its desires).
- A set of desires $Dess$ returned by the $wish$ function.
- A function ($focus$) that filters out incompatible, impossible and less valuable desires, and that focuses on a subset of the desire set.
- A structure of intentions $Ints$—the most desireable options/desires returned by the $focus$ function.
- A belief change function ($update$): given the agent's current beliefs and the latest percept sensed, the belief change function returns the updated beliefs of the agent.
- A function (*execute*) that selects some action(s) from the plan the agent is currently executing, and executes the action(s).

In most of the well known implementations of agents based on the BDI model (e.g., PRS (Georgeff and Ingrand 1989), IRMA (Bratman, Israel, and Pollack 1988) and dMARS (Rao and Georgeff 1995)), the $plan$ function returns plans from a plan library; a set of pre-compiled plans. An *intention structure* then structures various plans into larger hierarchies of plans. An intention in the intention structure in the classical BDI theory is a partial plan structured as a hierarchy of subplans. Furthermore, subplans may at some point be abstract, waiting to be 'filled in' (Bratman, Israel, and Pollack 1988). Some BDI architectures are designed to let the $plan$ function generate plans from atomic actions (Sardina, De Silva, and Padgham 2006; Walczak et al. 2007) (or it may possibly use a combination of pre-compiled and generated plans). However, none of the architectures that have a generative component employ a planner that produces plans for a POMDP model.

Combining POMDP Planning with the BDI Model

In this section we see how an agent controller in the BDI model can incorporate the $BestDoPO$ POMDP planner into its practical reasoning processes. We took the prototypical control loop of the BDI model as a reference and modified it to accommodate planning with POMDP policies. The proposed architecture is called BDI-POP (BDI with POmdp Planner).

First we introduce some terms and their relationships with the aid of Figure 1 (next page). Implicitly included in the "BELIEF" data store, is a fixed set of behaviors $behs$ and a fixed set of reward functions $rwds$ ($rwds$ is considered globally accessible). $behs$ is the agent's primitive goals; its innate drive. The idea is that each behavior refers to a unique goal that the agent is designed to achieve. Each behavior is *defined* by the set of programs and reward functions that can achieve the behavior. The $wish$ function is

omitted from our architecture (for now) because the options the agent would pursue at any time are its behaviors $behs$. The agent also has a fixed set of desires d. Each $des \in d$ is a triple $(nom, prog, ach)$: nom is a reference to the Golog program $prog$, and ach is a reference to the behavior $beh \in behs$ that $prog$ can potentially achieve, thus $ach \in behs$. The reward functions $rf \in rwds$ take as argument a nom that refers to the program that rf is associated with. The following holds: $\forall beh.[beh \in behs \rightarrow (\exists des).des = (nom, prog, ach) \wedge ach = beh]$: for each behavior, there exists at least one program to achieve it.

To understand the controller, we also need to consider the agent's deliberation process. $deliberate$ is the procedure that calls and controls the $focus$ predicate and that operates on the intention stack. We write $focus(b, d, i, behs, h^-)$ to be the predicate that selects one $des \in d$ for each $beh \in behs$, placing these desires in a stack, in ascending order, ordered by the desires' values. The desire selected for a behavior is the one that can achieve the behavior ($ach = beh$) and that has the highest value. A desire's value is estimated as the value v of the policy found, generated to a depth h^-: $BestDoPO$ is called with b, h^- and the applicable $prog$ as arguments; v is used and the policy is discarded. We keep $h^- < h$ to save on time spent deliberating. $focus$ 'returns' the stack i of selected desires.

$$\begin{aligned} &deliberate(b, d, i, behs, ai, i', h^-) \stackrel{def}{=} \\ &\quad (isEmpty(i) \wedge \exists i'.focus(b, d, i, behs, h^-) \vee \\ &\quad \neg isEmpty(i) \wedge \exists i'.i' = i) \wedge \\ &\quad \exists ai, i''.popIntentionStack(i', ai, i''). \end{aligned}$$

BDI-POP tests whether a usable policy could be generated, that is, whether the planner returns the $stop$ policy: When every outcome of an intended action (according to the input program) is illegal (according to the background action theory), $BestDoPO$ returns $stop$, and we say that the input program is *impossible*. An intention $i = (nom, prog, ach)$ with $prog$ being impossible is thus defined as an *impossible intention*.

The strategy used in $deliberate$ to deal with an impossible intention is extremely simple: it is dropped and the next intention on the stack is popped. This is a reasonable strategy because the next intention on the stack has the highest value, and should thus be pursued next. Calling $focus$ to refill the intention stack at this time would defeat the principle of *commitment* to intentions. Other strategies are possible, for example, replacing the impossible intention with another intention that achieves the same behavior, if one exists.

A logical high-level specification of BDI-POP follows, after which, it is explained in words.

$$\begin{aligned} &Agent(b, d, i, behs, ai, \pi, h, h^-) \stackrel{def}{=} \\ &\quad (nom, p, ach) = ai \wedge \\ &\quad \pi \neq stop \wedge p \neq nil \wedge \\ &\quad \exists \pi''.\pi = a; \pi'' \wedge execute(a) \wedge \\ &\quad \exists sv.getPercep(a, sv) \wedge \exists o.recognize(a, o) \wedge \\ &\quad \exists \pi'''.getSubPolicy(\pi'', o, \pi''') \wedge \\ &\quad \exists b'.b' = BU(o, a, b)) \wedge \\ &\quad Agent(b', d, i, behs, ai, \pi''', h, h^-). \end{aligned}$$

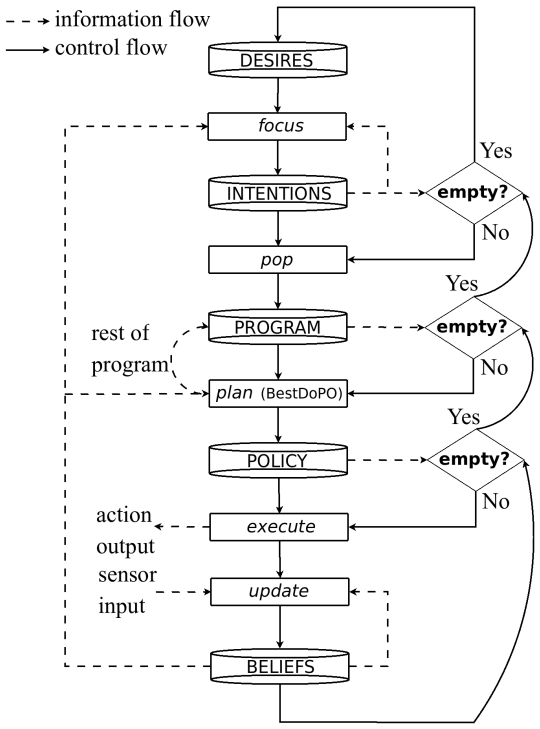

Figure 1: Schematic diagram of a sketch of the BDI architecture with the POMDP planner.

$$\begin{aligned} &Agent(b, d, i, behs, ai, \pi, h, h^-) \stackrel{def}{=} \\ &\quad (nom, p, ach) = ai \wedge \\ &\quad \pi = stop \wedge p = nil \wedge \\ &\quad deliberate(b, d, i, behs, ai', i', h^-) \wedge \\ &\quad Agent(b, d, i', behs, ai', \pi, h, h^-). \end{aligned}$$

$$\begin{aligned} &Agent(b, d, i, behs, ai, \pi, h, h^-) \stackrel{def}{=} \\ &\quad (nom, p, ach) = ai \wedge \\ &\quad \pi = stop \wedge p \neq nil \wedge \\ &\quad \exists \delta, \pi', v, pr.BestDoPO(p, \delta, nom, b, h, \pi', v, pr) \wedge \\ &\quad ai' = (nom, \delta, ach) \wedge \\ &\quad (\pi' = stop \wedge \\ &\qquad \exists ai'', i'.deliberate(b, d, i, behs, ai'', i', h^-) \wedge \\ &\qquad Agent(b, d, i', behs, ai'', \pi', h, h^-) \vee \\ &\quad \pi' \neq stop \wedge \\ &\qquad \exists \pi''.\pi' = a; \pi'' \wedge execute(a) \wedge \\ &\qquad \exists sv.getPercep(a, sv) \wedge \exists o.recognize(a, o) \wedge \\ &\qquad \exists \pi'''.getSubPolicy(\pi'', o, \pi''') \wedge \\ &\qquad \exists b'.b' = BU(o, a, b)) \wedge \\ &\qquad Agent(b', d, i, behs, ai', \pi''', h, h^-)). \end{aligned}$$

The agent follows the intention with the highest value—the intention popped from the stack. Call this the *active intention*. Initially, the intention stack is empty, so $deliberate$ is called and the active intention is instantiated. Whenever the controller needs a new plan to execute, $BestDoPO$ is

called to generate a policy with horizon h using the program specified by the active intention. The agent executes the policy until the end of the policy is reached, then $BestDoPO$ is called again for the rest of the program. If there is no rest of program (the program is empty), $deliberate$ is called. If the program has become impossible, $deliberate$ is called.

$getPercept$ returns a sensor value, given the action executed / sensor activated. The agent processes the sensor data and decides what it observed—the agent recognizes the sensor reading via the $recognize$ predicate, which outputs an observation. With this observation, the correct subpolicy is extracted from the current policy, and this (possibly empty) subpolicy becomes the new current policy.

After the action recommended by the policy is executed, the agent's beliefs must be updated according to what it 'knows' about the effects of its actions. The same belief update function used during planning by $BestDoPO$ is used to update the agent's beliefs. The current belief state of the agent will be the 'initial' belief state required as argument to $BestDoPO$ the next time the planner is called.

Given our present definition of $deliberate$ and given that we shall allow only finite programs for achieving intentions, the agent is guaranteed to deliberate at regular intervals. However, this interval period is fixed (to the degree that intentions become impossible). Adding a $reconsider$ predicate that tells the agent once every control cycle whether to deliberate, is a more sophisticated method. $reconsider$ is described by, for example, Wooldridge (2000) and "was examined by David Kinny and Michael Georgeff, in a number of experiments," (Wooldridge 1999, p. 57). Because we are investigating the feasibility of the basic idea of the hybrid architecture in this paper, we have left out the $reconsider$ predicate from the present investigation.

A somewhat significant difference of our hybrid architecture from the perspective of control via POMDP policy generation, is that—as stand-alone controller—the POMDP planner takes a single plan with a single associated reward function, to generate a policy. The new hybrid architecture takes several programs, each with an associated reward function. This aspect of the agent being able to reason over multiple behaviors has the advantage that the agent designer can separately specify behaviors that should—at least intuitively—be considered separately.

$BestDoPO$ expands Golog programs into hierarchically structured plans (policies), and only programs that have been selected as intentions are expanded into policies. Each program can generate a policy—or several policies if the program is expanded piece-wise. Viewing a policy tree as an intention structure in the sense of traditional BDI architectures, each program in the intention stack represents (at least one) intention structure. BDI-POP, therefore, maintains *several* unexpanded intention structures, only expanded when popped from the intention stack.

Implementation and First Experiments

To validate the BDI-POP architecture and to gain a sense for its performance potential, we observe one agent based on the architecture, in a simulation. The simulation environment is inspired by Tileworld (Pollack and Ringuette 1990), a testbed for agents. We designed and implemented the FireEater world, a dynamically changing grid world (a 5×5, two dimensional grid of cells) in which our agent is situated. There are obstacles that change position and fires that can be 'eaten'. Space prohibits a detailed explanation of FireEater world.

The agent gets one 'fire-point' for eating one fire. It can only eat a fire if it is in the same cell as the fire. There are two agent behaviors: `findFood`, `eat` $\in behs$. `findFood` may be realized by two available programs, and `eat` is forced to be achieved by one (other) program. The agent can go left, right, up or down—locomotive actions which are stochastically nondeterministic; it can sense its location (probabilistically) and it can eat fire (deterministically).

In order to have a base-line against which the performance of the new hybrid architecture can be compared, a simple or 'naive' architecture (called Naive-POP) was implemented. It has no explicit intentions or desires as defined for the BDI model. The agent is provided with a single Golog program and associated reward function. In this implementation, the program loops continuously over a nondeterministic action—nondeterministic between all available actions. If there is no rest of program, that is, the agent has executed the whole program, the agent will stop its activity.

BDI-POP, in contrast, does not employ programs that loop infinitely (in the experiments): programs were designed so that they become empty as soon as a policy is generated from the program. Hence an intention will be regarded as 'achieved' as soon as its policy becomes empty. Then the next intention will be popped from the stack. Because the size of the stack equals $|behs|$ and because all programs are finite, it is guaranteed that periodically all intentions will be achieved, that is, the stack is empty, and the agent is forced to deliberate to fill the intention stack with a fresh set of intentions.

The two agents as implemented by the two architectures, have identical knowledge bases, except for their programs and reward functions. That is, they believe the same actions are possible, with the same effects and associated probabilities. They both employ the exact same planner: $BestDoPO$. The graph in Figure 2 compares the performance of the two

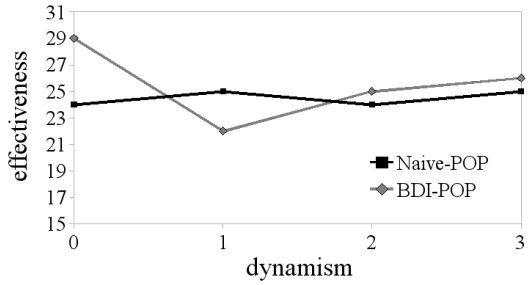

Figure 2: Performance of the two architectures.

architectures. Given the restrictions inherent in their respective architectures, each agent was roughly optimized to give them equal advantage.

Both architectures generate policies of horizon depth 3 ($h = 3$). In BDI-POP, we set $h^- = 1$—the search hori-

zon that *focus* uses to determine program values. Throughout the experiments, the strategy for the time allowed to the agent was the same. There are 6 obstacles and initially 9 fires in each trial. We allow the agent to perform 3 actions each time before the obstacle positions change. The parameter for the number of obstacle changes per simulation cycle is the only parameter varied during experiments. Fourty trials per setting of this parameter were performed. *Effectiveness* is the total fire-points collected for the 40 trials. *Dynamism* is defined as 'number of obstacle changes per number of agent actions.

Conclusion

Compared to Naive-POP, the performance of BDI-POP in our experiments is not that impressive. This does not show that a BDI agent architecture should not be imbued with a POMDP planner; several enhancements to the simple BDI architecture used here are still possible: In particular, to build on this groundwork, we want to add the $reconsider$ predicate to deal with cases where intentions have become inappropriate to some degree, and utilizing partial/abstract plan structures.

Moreover, the relative sophistication of BDI-POP may not be applicable in very simple worlds such as FireEater. We would thus like to deploy our agents in a larger world, perhaps with more complicated tasks for the agent to perform. This will also give more scope for the variety of programs that would be applicable, and the real power of the BDI model could come into play.

What *has* been shown is that the proposed architecture is implementable; there is no obvious fundamental conflict in synthesizing our POMDP planner and the BDI model for agent control. The groundwork has thus been laid for the development of more sophisticated planning processes in the BDI-POP framework.

What remains unclear is how practical this approach might be in realistically complex domains. With probabilistic outcomes and events in the world, the policy searches blow up very quickly with depth. For complete and optimal policies, POMDP solvers can deal with just a modest number of easily enumerated states. Policy trees of a fixed depth (as generated by $BestDoPO$) are not complete policies and thus less costly to generate. Realistically though, due to belief states being extremely numerous and the equivalence problem for states in the situation calculus, $BestDoPO$ seems to be intractable if not undecidable. Furthermore, the situation calculus, in principle, provides a good deal of expressivity (including quantified reasoning), which brings its own computational complexity issues. For a hybrid BDI/POMDP architecture to scale up to a domain more meaningful than a microdomain, the integration of more 'common sense' reasoning techniques into the architecture may have benefits. And the latest advances in POMDP solvers (e.g., (Toussaint, Charlin, and Poupart 2008)) should be investigated for ideas to improve the efficiency of $BestDoPO$.

References

Bacchus, F.; Halpern, J.; and Levesque, H. 1999. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence* 1-2(111):171–208.

Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI-00*. AAAI Press. 355–362.

Bratman, M.; Israel, D.; and Pollack, M. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4:349–355.

Bratman, M. 1987. *Intention, Plans, and Practical Reason*. Massachusetts/England: Harvard University Press.

Georgeff, M., and Ingrand, F. 1989. Decision-making in an embedded reasoning systems. In *Proc. IJCAI-89*. San Fransisco, CA: Morgan Kaufmann. 972–978.

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 1-2(101):99–134.

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *J. of Log. Progr.* 31(1-3).

Likhachev, M.; Ferguson, D.; Gordon, G.; Stentz, A.; and Thrun, S. 2005. Anytime dynamic A*: An anytime, replanning algorithm. In *Proc. Intl. Conf. on Automated Planning and Scheduling (ICAPS)*.

McCarthy, J. 1963. Situations, actions and causal laws. Technical report, Stanford University.

Pollack, M., and Ringuette, M. 1990. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proc. AAAI-90*. AAAI Press. 183–189.

Puterman, M. 1994. *Markov Decision Processes: Discrete Dynamic Programming*. New York, USA: Wiley.

Rao, A., and Georgeff, M. 1995. BDI agents: From theory to practice. In *Proc. ICMAS-95*. AAAI Press. 312–319.

Reiter, R. 2001. *Knowledge in Action*. MIT Press.

Rens, G.; Ferrein, A.; and Van der Poel, E. 2008. Extending DTGolog to deal with POMDPs. In *Proc. PRASA-08*. PRASA. 49–54.

Sardina, S.; De Silva, L.; and Padgham, L. 2006. Hierarchical planning in BDI agent programming languages: A formal approach. In *Proc. AAMAS-06*. ACM Press. 1001–1008.

Toussaint, M.; Charlin, L.; and Poupart, P. 2008. Hierarchical POMDP controller optimization by likelihood maximization. In *Workshop on Advancements in POMDP Solvers, Tech. Report WS-08-01, AAAI-08*. AAAI Press. url:http://www.aaai.org/Library/Workshops/ws08-01.php.

Walczak, A.; Braubach, L.; Pokahr, A.; and Lambersdorf., W. 2007. Augmenting BDI agents with deliberative planning techniques. In *Proc. ProMAS-06*. Springer. 113–127.

Wooldridge, M. 1999. Intelligent agents. In Weiss, G., ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Massachusetts/England: MIT Press. chapter 1.

Wooldridge, M. 2000. *Reasoning About Rational Agents*. Massachusetts/England: MIT Press.