Learning in BDI Multi-agent Systems

Alejandro Guerra-Hernández[1], Amal El Fallah-Seghrouchni[2], and Henry Soldano[1]

[1] Université Paris 13, Laboratoire d'Informatique de Paris Nord, UMR 7030 - CNRS, Institut Galilée, Av. Jean-Baptiste Clément, Villetaneuese 93430, France.
{agh,soldano}@lipn.univ-paris13.fr

[2] Université Paris 6, Laboratoire d'Informatique de Paris 6, UMR 7606 - CNRS, 8 rue du Capitaine Scott, Paris 75015, France.
Amal.Elfallah@lip6.fr

Abstract. This paper deals with the issue of learning in multi-agent systems (MAS). Particularly, we are interested in BDI (Belief, Desire, Intention) agents. Despite the relevance of the BDI model of rational agency, little work has been done to deal with its two main limitations: i) The lack of learning competences; and ii) The lack of explicit multi-agent functionality. From the multi-agent learning perspective, we propose a BDI agent architecture extended with learning competences for MAS context. Induction of Logical Decision Trees, a first order method, is used to enable agents to learn when their plans are successfully executable. Our implementation enables multiple agents executed as parallel functions in a single Lisp image. In addition, our approach maintains consistency between learning and the theory of practical reasoning.

1 Introduction

The relevance of the BDI (Belief, Desire, Intention) model of rational agency can be explained in terms of its philosophical grounds on intentionality [7] and practical reasoning [2], as well as its elegant abstract logical semantics [23, 25, 29]. In addition, different implementations of the model, e.g., IRMA [3], and the PRS-like systems [11], have led to successful applications, including diagnosis for space shuttle, factory process control, and business process management [12]. However, two limitations of the BDI model are well known [13]: Its lack of learning competences; and the lack of explicit multi-agent functionality. Both limitations constitute an issue of what is now known as MAS learning [28, 26], roughly characterized as the intersection of MAS and Machine Learning (ML). MAS learning is justified as follows: Learning seems to be the way to deal with the complexity inherent to agents and MAS, while at the same time, learning on the MAS context could improve our understanding of learning principles in natural and artificial systems.

From the MAS learning perspective, this paper shows how a BDI architecture, based in dMARS specification [15], was extended to conceive a BDI learning architecture. The design of this architecture is inspired by the definition of

a learning agent by Stuart Russell and Peter Norvig [24]. Design choices were constrained by the fact that BDI agents perform practical reasoning to behave. Practical reasoning, together with BDI semantics, pose a hard design problem: Learning methods directed towards action, very popular in MAS learning, use representations less expressive than those used in the BDI model, i.e., basically propositional representations as in classic reinforcement learning [27]; Learning methods with more expressive representations are usually conceived as isolated learning systems, directed towards epistemic reasoning. This may explain why in the abundant MAS learning literature, only Olivia et al. [21] has considered the problem of BDI learning agents, despite the relevance of the model in agency and MAS. The approach we use to solve this problem, applies Inductive Logic Programming (ILP) [20] methods, particularly Induction of Logical Decision Trees [4], to learn when plans are executable, as expressed by the context of plans, i.e., the components of the BDI model behind choices in practical reasoning.

This paper is organized as follows: Section 2 introduces the key BDI concepts that are necessary to explain our approach. Emphasis is set on the aspects of intentional agency and practical reasoning, involved in learning. Section 3 describes our BDI learning architecture and justifies the choices of design and implementation. Justifications include a hierarchy of learning MAS, built on the concept of awareness, to decide when agents should learn either by themselves or in cooperation with other agents. Also, the section briefly describes the learning method used on the architecture – Induction of Logical Decision Trees. Section 4 introduces an example of learning BDI agent at level 1 of the proposed hierarchy, i.e., centralized learning; Details of MAS learning at level 2, i.e., distributed learning, are considered in section 5. Finally, section 6 deals with related work and concludes this paper.

2 BDI agency

Software agents are usually characterized as computer systems that exhibit flexible autonomous behavior [29], which means that these systems are capable of independent, autonomous action in order to meet their design objectives. BDI models of agency approximate this kind of behavior through two related theories about the philosophical concept of intentionality: Intentional Systems, defined by Daniel Dennett [7] as entities which appear to be subject of s, desires and other propositional attitudes; and the Practical Reasoning theory, proposed by Michael Bratman [2] as a common sense psychological framework to understand ourselves and others, based on beliefs, desires, and intentions conceived as partial plans. These related notions of intentionality provide us with the tools to describe agents at the right level of abstraction, i.e., in terms of beliefs, desires and intentions (BDI), adopting the intentional stance; and to design agents in a compatible way with such intentional description, i.e., as practical reasoning systems. Different aspects of intentionality and practical reasoning have been formally studied, resulting in the so called BDI logics [23]. For a road map of

the evolution of these formalisms, see [25, 29]. Implementations make use of refinement techniques, e.g., using specifications in Z language [17].

This section sketches our BDI architecture, based on dMARS [15] specification, using a very simple scenario proposed by Charniak and McDermott [8]. This scenario (Fig. 1) is composed of a robot with two hands, situated in an environment where there is a board, a sander, a paint sprayer, and a vise. Different goals can be proposed to the robot, e.g., sand the board, or even get self painted! this introduces the case of incompatible goals, since once painted the robot is not operational (its state changes from `ok` to `painted`) for a while. The robot has different options, i.e., plans, to achieve its goals. It is possible to introduce other robots (see agent `r2`) in the environment to experiment social interactions [6], e.g., sharing goals, conflict for resources, etc. This scenario will be used in the examples in the rest of the paper.

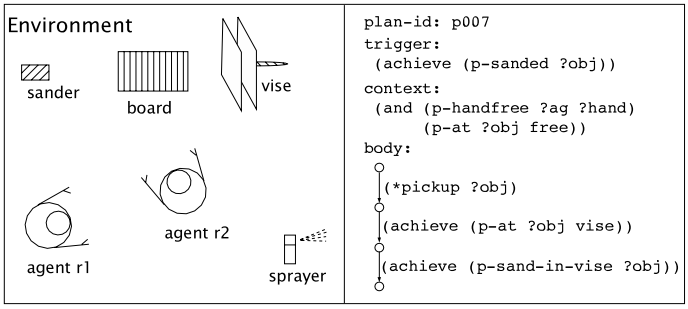

Fig. 1. A simple scenario for the examples in this paper and a simplified BDI plan.

2.1 The BDI model

In general, an architecture built on the BDI model of agency is specified in terms of the following data structures:

Beliefs. They represent information about the world. Each belief is represented as a ground literal of first-order logic. Literals not grounded, known as belief formulae, are used to define plans, but are not considered beliefs. They are updated by the perception of the environment, and the execution of intentions. The scenario shown in Fig. 1 can be represented with the following beliefs: `(p-state r1 ok)`, `(p-at sander free)`, `(p-at board free)`, `(p-handfree r1 left)`, `(p-handfree r1 right)`, `(p-at sprayer free)`. Where `free` is a constant meaning that the object is not at the vise or an agent has it. The rest is self explicative.

Desires. Also known as goals, correspond to the tasks allocated to the agent. Usually, they are considered logically consistent among them. Desires include to achieve a belief, and to test a situation, expressed as a situation formula,

i.e., a belief formula or a disjunction and/or a conjunction of them, e.g., `(test (and (p-state r1 ok) (p-freehand r1 ?x)))`. All strings starting by '?' are variables. Those starting by 'p-' are predicate symbols.

Event queue. Perceptions of the agent are mapped to events stored in a queue. Events include the acquisition or removal of a belief, e.g., `(add-bel (p-sand board))`, the reception of a message, e.g., `(told r2 (achieve (p-sand board)))`, and the acquisition of a new goal. These examples are simplified, events are implemented as structures keeping track of historical information. What is shown corresponds to a trigger, a component of events, used to identify them. The reception and emission of messages is used to implement MAS competences of our BDI agents. For the moment, no explicit agent communication language (ACL) is considered, but they can easily be included in our architecture since Lisp packages exist for them, at least for FIPA ACL and KQML.

Plans. BDI agents usually have a library of predefined plans. Each plan has several components, the most relevant for us are shown in the simplified plan on Fig. 1. The `plan-id` is used to identify a plan in the plan library. In our example, the plan is identified as `p007`. The trigger works like an invocation condition of a plan, it specifies the event a plan is supposed to deal with. Plan `p007` is triggered by an event of the form `(achieve (p-sanded ?obj))`. Observe that the use of variables is allowed here. If the agent registers an event like `(achieve (p-sanded board))` in the event queue, it will consider plan `p007` as relevant to deal with such event. The context specifies, as a situation formula, the circumstances under which a plan should be executed. Remember that a situation formula is a belief formula or a conjunction and/or disjunction of them. Plan `p007` is applicable if the agent has one hand free and the object to be sanded is free. The plan body represents possible courses of action. It is a tree which nodes are considered as states and arcs are actions or goals of the agent. The body of plan `p007` starts with an external action, identified by a symbol starting by '*', `(*pickup ?x)`. External actions are like procedures the agent can execute directly. Then the body continues with two goals. Goals are posted to the event queue when the plan is executed, then other plans that can deal with such events are considered, and so on. Additionally, plans have some maintenance conditions which describe the circumstances that must remain to continue the execution of the plan. A set of internal actions is specified for the cases of success and failure of the plan. Finally, some BDI architectures include some measure of the utility of the plan.

Intentions. They are courses of action an agent has commited to carry out. Each intention is implemented as a stack of plan instances. In our example, as seen above, in response to the event `(achieve (p-sanded board))`, plan `p007` is considered as relevant. If the context of a plan is a consequence of the beliefs of the agent, the plan is considered executable. A plan instance is composed of a plan, as defined in the plan library, and the substitutions that make it relevant and applicable, e.g., *(board/?obj, left/?hand, r1/?ag)*. If the event that triggered the plan is an external one, i.e., no plan has generated it,

then an empty stack is created and the plan instance is pushed on it. If the event is internal, i,e., generated by a plan, then the plan instance is pushed on the existing stack containing the plan that generated the event, e.g., imagine that a plan instance p005 deals with the event `(achieve (p-at(board, vise)))`, generated while executing `p007`, this plan instance will be pushed on the stack containing `p007`, resulting on `(p005 p007)`.

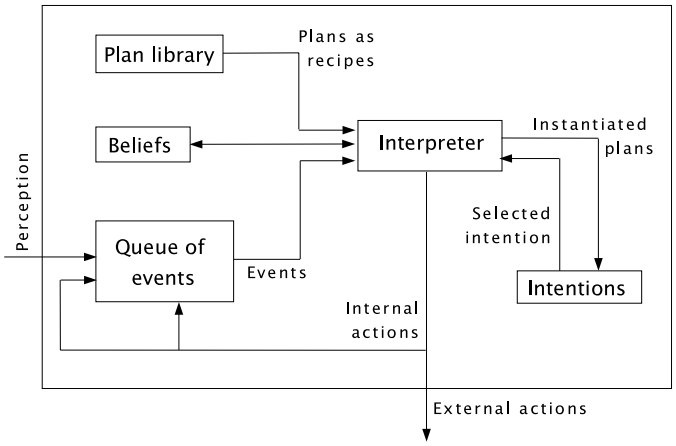

Fig. 2. Our BDI architecture inspired in dMARS specification

These structures interact with an interpreter, as shown in Fig. 2. Different algorithms for the interpreter are possible, the most simple is:

1. Update the event queue by perception and internal actions to reflect the events that have been observed;
2. Select an event, usually the first one in the queue, and generate new possible desires by finding the set of relevant plans in the library for the selected event, i.e., those plans whose trigger condition matches the selected event;
3. Select from the set of relevant plans an executable one, i.e., one plan whose context is a logical consequence of the beliefs of the agent, and create an instance plan for it.
4. Push this instance plan onto an existing or new intention stack, as explained before;
5. While the event queue is empty, select an intention stack, take the top plan, and execute its current step. If this step is an action, execute it, otherwise if it is a subgoal, post it to the event queue.

Michael Wooldridge [29] presents some algorithms for the BDI interpreter, corresponding to different commitment strategies, e.g., single-minded and open-minded commitments. David Kinny and Michael Georgeff [16] present a comparative study of two strategies identified as bold and cautious. Both strategies

perform well if the environment is not very dynamic, but if this is not the case, cautious agents out perform bold ones.

2.2 Some issues on implementation

Given the nature of the PRS-dMARS approach, we considered using a symbolic programming language to implement our architecture. Since we decided to do our own Lisp implementation of a BDI architecture, some arguments are in place. We knew that different implementations for BDI architectures already existed. For PRS [11], and its re-implementation dMARS [15], we only had access to formal specifications. A PRS-like system, Jam! [14], became available once we had started implementing our interpreter, but its semantics differs from the specification we were using, particularly the context of plans are defined as conjunctions of belief formulae. Most importantly, there is evidence [18] that adapting existing software, even when disposing of low level specifications, to produce a learning agent or to attach one to existing software, is not obvious at all. In addition, sometimes it is even impossible without depth changes in the design of the original software. We found out that the structures and procedures used in the dMARS, fit well the features of Lisp, i.e., uniformity of representation for data and procedures as lists, data an procedure abstraction, etc. Note that PRS was originally developed using Lisp, so that dMARS, at least at the specification level, is pretty much influenced by this language.

Our architecture already provides functions to define primitive actions available to the agents in the system; to define plans that may use these primitive actions; to define and assign different competences to agents, in terms of a plan library; to bootstrap goals for each agent in terms of initial events; and to execute agents under different commitment strategies. These can be considered standard BDI features, together with syntax verification tools for the BDI language used to define agents, and built-in functions to test if BDI formulae are a logical consequence of a set of beliefs. The interface with the OS is provided by Lisp.

Non standard BDI features in our architecture include a set of functions to simulate agents in a MAS, as parallel processes running in the same Lisp image; and an interface to use DTP theorem prover [10] in the case agents need to perform more sophisticated epistemic reasoning, beyond the built-in logical competences. DTP does refutation proofs of queries from databases in First-Order predicate calculus, using a model elimination algorithm and domain independent control reasoning. The use of sub-goaling inference with model elimination reductions, makes the inference of DTP sound and complete.

3 BDI learning agents

Based on the definition of a well posed learning problem as proposed in ML [19], Stuart Russell and Peter Norvig [24] have conceptually structured a generic

learning agent architecture into four components: i) A learning component responsible for making improvements by executing a learning algorithm; ii) A performance component responsible of taking actions; iii) A critic component responsible for providing feedback; and iv) A problem generator responsible for suggesting actions that will lead to informative experiences. The design of the learning component, and consequently the choice of a particular learning method, is usually affected by five major issues. They are considered here assuming that our BDI architecture, adapted after dMARS specification [15] as described up to here, corresponds to the performance component of a BDI learning architecture.

3.1 Which elements of the performance component are to be improved by learning?

BDI agents perform practical reasoning [2] to behave, i.e., reasoning directed towards action, while no-agent AI systems, may be seen as performing epistemic reasoning, i.e., reasoning directed towards beliefs. From the role of beliefs in the theory of practical reasoning, e.g., the asymmetry thesis, the standard and filter of admissibility, it is clear that even when they justify the behavior of the agent, they do it as a part of a background frame that, together with prior intentions, constrain the adoption of new intentions. In doing so, they are playing a different role that the one they play in epistemic reasoning. Particularly, practical reasons to act sometimes differ from epistemic reasons. This is the case for reasonableness of arbitrary choices in Buridan cases[3], e.g., it is *practical reasonable* to choose any plan in the set of relevant applicable plans to form an intention, even if there is no epistemic reason, no reason purely based on the beliefs of the agent, behind this choice. The context of plans may be seen as encoding practical reasons to act in some way and not in another, that together with the background frame of beliefs and prior intentions, support the rational behavior of intentional agents. Then we decided to extend the BDI architecture, enabling the agents to learn about the context of their plans, i.e., when plans are executable. Properly, our agents are not learning their plans [9], but when to use them.

3.2 What representation is used for these elements?

As mentioned, representations in our BDI architecture are based on two first-order formulae: Belief formulae and situation formulae. Beliefs are grounded belief formulae, like Prolog facts. Belief formulae are used to define plans. Every belief formula is also a situation formula, but situation formulae also include conjunctions and/or disjunctions of belief formulae. The context of plans are represented as situation formulae. These representation issues have two immediate consequences when considering candidate learning methods: First, given the representation of belief and situation formulae, propositional learning methods are discarded. Second, the fact that the context of plans is represented as situation formulae, demands that the target representation of the learning method, enables disjunctive hypothesis, e.g., decision trees.

[3] According to the philosopher Jean Buridan, they are situations equally desirable.

3.3 What feedback is available to learn?

Getting feedback from our BDI architecture is almost direct, since it already detects and processes success and failure of the execution of plan instances. This is done by executing a set of internal actions, up to now, add and delete beliefs. These internal actions are predefined for each plan in the library. The architecture is then extended with a special internal action, that generates a log file of training examples for the learning task. Items to built these examples include: the beliefs characterizing the moment when the plan was selected, the label of success or failure after the execution of the plan, and the plan-id.

3.4 What prior information is available to learn?

There are already two sources of prior information in our BDI architecture. First, the plan library of the agents can be seen as prior information, in the sense that plans state expected effects which, from the perspective of the agent, must hold in the environment, i.e., the event e will be satisfied if the plan p is executed, and this is the case if the context of p is a logical consequence of the beliefs of the agent. Second, our BDI architecture also keeps track of predicates, functions, and their signatures, used to define the plans in the library of each agent. These elements can be used to specify the language for the target concept of the learning process.

3.5 Is it a centralized or distributed learning case?

We believe that awareness seems to be indicative of a learning MAS hierarchy of increasing complexity. In a certain way, this hierarchy of learning environments corresponds to the scale of intentionality of Daniel Dennett [7]. We intend to perform learning at levels 1 and 2 of this hierarchy. Level 0, i.e., only one agent is there, the true isolated learning case, can be seen as a special case of level 1. Levels in this hierarchy are as follows:

Level 1. At this level, agents act and learn from direct interaction with the environment, without being explicitly aware of other agents in the MAS. However, the changes other agents produce in the environment can be perceived by the learning agent. Consider again the robot scenario with two robots: one specialized in painting objects, the other in sanding objects. It is possible to program the painter robot, without awareness of the other robot in the environment, i.e., all it has to learn is that once an object is sanded, it can be painted.

Level 2. At this level, agents act and learn from direct interaction with other agents, using exchange of messages. For the example above, the sander robot can inform the painter robot, that an object is already sanded. Also, the painter agent can ask the sander robot for this information. Exchange of training examples in learning processes is also considered at level 2.

Level 3. At this level, agents learn from the observation of the actions performed by other agents in the system. It involves a different kind of awareness from that of level 2. Agents are not only aware of the presence of other agents, but are also aware of their competences, hence the painter robot is able to perceive that the sander robot is going to sand the table.

3.6 Top-down Induction of Logical Decision Trees

From the representation of the context of plans, as discussed above, we decided to use decision trees as target representation. Top-down induction of decision trees (TDIDT) is a widely used and efficient machine learning technique. As introduced in the ID3 algorithm [22] it approximates discrete value-target functions. Learned functions are represented as trees, corresponding to a disjunction of conjunctions of constrains on the attribute values of the instances. Each path from the decision tree root to a leaf, corresponds to a conjunction of attribute tests, and the tree itself is the disjunction of these conjunctions, i.e., the kind of representation we need for the plan context. However, training examples are represented as a fixed set of attribute-value pairs, i.e., a propositional representation, which does not fix our requirements. Another limitation of ID3-like algorithms, is that they can not use information beyond the training examples, i.e., other things the agent believes, as their plans. ILP [20] can overcome these two main limitations of classic ML inductive methods.

Logical decision trees upgrade the attribute-value representation to a first-order representation, using the ILP paradigm known as learning from interpretations [4]. In this setting, each training example e is represented by a set of facts that encode all the properties of e. Background knowledge can be given in the form of a Prolog program B. The interpretation that represents the example is the set of all ground facts that are entailed by $e \wedge B$, i.e., its minimal Herbrand model. Observe that instead of using a fixed-length vector to represent e, as the case of attribute-value pairs representation, a set of facts is used. This makes the representation much more flexible. Learning from interpretations can be defined as follows. Given: i) A target variable Y; ii) A set of labelled examples E, each consisting of a a set of definite clauses e labelled with a value y in the domain of Y; iii) A language L; iv) A background theory B. Find a hypothesis $H \in L$ such that for all examples labelled with y: i) $H \wedge e \wedge B \models label(y)$; and ii) $\forall y' \neq y : H \wedge e \wedge B \not\models label(y')$.

Learning from interpretations exploits the local assumption, i.e., all the information that is relevant for a single example is localized in two ways. Information contained in the examples is separated from the information in background knowledge. Information in one example is separated from information in other examples. The learning from interpretations setting can be seen as situated somewhere between the attribute-value and learning from entailment [20] settings. It allows extending attribute-value representation towards ILP, without sacrifying efficiency.

ACE [5] is a learning from interpretations system, building logical decision trees, that is, decision trees where every internal node is a first-order conjunc-

tion of literals. It uses the same heuristics that ID3 algorithms (gain-ratio and post-pruning heuristics), but computations of the tests are based on the classical refinement operator under Θ-subsumption, which requires the specification of a language L stating which kind of tests are allowed in the decision tree. This is exemplified in the next section, showing how the agents in our extended architecture, guided by autonomy and intentionality, determine when they should learn, configure its learning set, and execute ACE.

4 Learning at level 1 (centralized learning)

Consider that the agent identified as `r1` in figure 1, has selected the plan `p007` to deal with the event `(achieve (p-sanded board))`. Then during the execution phase of the interpreter, this plan will either succeed or fail. If the plan fails, we want the agent trying to learn why the plan failed considering that the agent had practical reasons, expressed in the context of the plan, to adopt it as an intention. So the agent should reconsider after its experience, the situation formula expressing the context of the plan. In order to execute the learning process, the agent needs to generate a set of three files consisting of the training examples, the background theory, and the parameters for ACE, including the specification of the target language L, the desired format output, etc. The `plan-id` is used to identify these files. Files are as follows: i) The knowledge base, identified by the extension `.kb`, which contains the examples labelled with the class they belong to; ii) The background theory, identified by the extension `.bg`; and iii) the language bias, identified by the extension `.s`. These files are generated automatically by the agent, as follows.

When the success or failure of its intention is detected, the agent `r1` tracks these executions in a log file identified as `p007.kb` to indicate to ACE that it contains the examples associated to this plan. Models for the example are shown in table 1. Each model starts with a label that indicates the `success` or `failure` of the plan execution. Then a predicate `plan` is added to establish that the model is an instance of the execution of a particular plan by a particular agent. The model also contains the beliefs of the agent when the plan was selected to create the plan instance. Partial models are memorized by the agent when the plan is selected as relevant and applicable. The label is added in the execution phase. The knowledege base for the examples is stored in the file `p007.kb`.

The background theory contains information about the plan being learned. The symbols for the variables and constants are taken from the plan definition. A function of our system translates the original definition of plan `p007` to this format. It encodes the plan context of `p007`:

```
plan_context(Ag,p007) :- p_handfree(Ag,Hand), p_at(Obj,free).
```

Then the configuration file is generated. Following the example, this information is stored in a file called `p007.s`. The first part of this file is common to all configurations. It specifies the information ACE prints while learning (talking); the minimal number of cases to learn; the format of the output (either a logical

```
begin(model(1)).
success.
plan(r1,p007).
p_state(r1,ok).
p_handfree(r1,left).
p_at(board,free).
end(model(1)).
```

```
begin(model(2)).
success.
plan(r1,p007).
p_state(r1,ok).
p_handfree(r1,right).
p_at(board,free).
end(model(2)).
```

```
begin(model(3)).
failure.
plan(r1,p007).
p_state(r1,painted).
p_handfree(r1,left).
p_handfree(r1,right).
p_at(board,free).
end(model(3)).
```

```
begin(model(4)).
failure.
plan(r1,p007)
p_state(r1,painted).
p_handfree(r1,right).
p_at(board,free).
p_at(sander,vise).
end(model(4)).
```

```
begin(model(5)).
success.
plan(r1,p007)
p_state(r1,ok).
p_handfree(r1,left).
p_at(board,free).
end(model(5)).
```

```
begin(model(6))
success.
plan(r1,p007)
p_state(r1,ok).
p_handfree(r1,left).
p_at(board,free).
end(model(6))
```

Table 1. Training examples as models at level 1, examples are generated by a single agent.

decision tree or a logic program); and the classes used for the target concept, i.e., either success or failure.

```
talking(0).
load(models).
minimal_cases(1).
output_options([c45,lp]).
classes([success, failure]).
```

The second part of the configuration file specifies the predicates to be considered while generating tests for the nodes of the tree. The way our agent generates this file relies on the agent definition. Every time a plan is defined, the interpreter keeps track of the predicates used to define it, and their signature. In this example, three predicates have been used to define the agent: *(p_state/2, p_freehand/2, p_at/2)*. So the agent asks the learning algorithm to consider these predicates with variables as arguments:

```
rmode(p_state(Ag,State)).
rmode(p_freehand(Ag,Hand)).
rmode(p_at(Obj,Place)).
```

Then the agent asks the learning algorithm to also consider these predicates with arguments instantiated after the examples:

```
rmode(p_state(+Ag,#)).
```

```
rmode(p_freehand(+Ag,#)).
rmode(p_at(+Obj,#)).
```

Finally the predicates used in the background theory are considered too. At least the two following forms are common to all configurations:

```
rmode(plan_context(Ag,Plan)).
rmode(plan_context(+Ag,#)).
```

The `rmode` command is used by ACE to determine the language bias L. The '#' sign may be seen as a variable place holder, that takes its constant values from the examples in the knowledge base. The '+' prefix means the variable must be instantiated after the examples in the knowledge base.

Once the number of examples is greater than a threshold (5 in the example) the agent executes a modified non-interactive version of ACE, and suggests the user to watch the `p007.out` file, containing the result of the learning process, to accordingly modify the definition of the plan. It is also possible that the agent modifies the definition of the plan itself. The strategy adopted to incorporate the results of learning, depends on the domain of application, i.e., sometimes the supervision of the user is preferable.

Output for our example is:

```
Compact notation of pruned tree:
plan_context(Ag,p007) ?
+--yes: p_state(Ag,painted) ?
|       +--yes: [failure] [2.0/2.0]
|       +--no:  [success] [3.0/3.0]
+--no:  [succes] [1.0/1.0]

Equivalent logic program:
n1:-plan_context(Ag,p007).
n2:-plan_context(Ag,p007),p_state(Ag,painted).
class([failure]):-plan_context(Ag,p007),p_state(Ag,painted).
class([succes]):-not(n1).
class([succes]):-plan_context(Ag,p007),not(n2).
```

Fractions of the $[i/j]$ form indicate that there were i examples in the class, and that j of them were well classified by the test proposed. This example used six models and the time of induction was 0.01 seconds, running on a Linux RedHat 8.0 Pentium 4, at 1.6 GHz.

5 A MAS of BDI learning agents (level 2)

The example of the previous section corresponds to level 1 in our hierarchy of learning MAS. At level 2, agents are supposed to learn while they are aware of other agents. Communication is very important when learning in a MAS, but the

design of the agent should determine when, what, and why should an individual agent communicate [1].

There are two situations under which a BDI agent should consider communicating while learning. First, the agent is no able to start the execution of its learning process, i.e., it does not have enough examples to run ACE. In this case the agent can ask other agents in the MAS for training examples. Second, the agent is unable to find out a hypothesis to explain the failure of the plan in question, i.e., after the execution of the learning process, the tree produced by ACE has only the node `[failure]`, or the hypothesis found is the original plan context being learned. This means that the examples used by the BDI agent to learn, were insufficient to find out why the plan has failed. In this case the agent may ask other agents in the MAS for more evidence, before executing ACE again.

The results of a learning process are shared by the agents in the MAS due to the way they are defined in the BDI architecture. If the agent has found a hypothesis for the failure of its plan, it will communicate this result to the user, asking for modifications of the plan definition accordingly to the decision tree found. If the user modifies the plan definition, this change automatically affects all agents having this plan in the library. Observe that this does not imply that all plans are shared by the agents, therefore heterogeneous MAS are possible in our architecture.

The concept of competence is used to address communications. It is defined as the set of all the trigger events an agent can deal with, i.e., the union of all triggers in the agent's plan library. Then two ways of sending messages are possible. The agent broadcasts its message including the trigger and the `plan-id` of the plan to be learned, The other agents accept and process the message if the trigger event is on their competences. Alternatively, competence is used to build a directory for each agent, where each trigger event in the competence of an agent, is associated to the `id` of other agents in the system that deal with the same trigger event.

Competence and plans determine what to communicate. If two agents have the same plan for the same event, they can be engaged in a process of distributed data gathering, i.e. they can share the examples they have collected. In this case agents are involved in collecting data, but each agent learns locally. The models obtained from three agents in the scenario of figure 1 are shown in table 2. Agent `r2` is the learner agent.

This learning process results in the same decision tree obtained in the previous section, but since the first execution done by the learning agent `r2` of plan `p007` lead to a failure (model 1), it would not be able to collect success training examples for this plan (the BDI interpreter blocks plans that failed for a given event, to avoid the agent trying to execute them again). It means that outside the MAS, agent `r2` is not able to learn for this plan. Also the failure example of agent `r3` is important, since without it ACE is not able to induce a tree.

```
begin(model(1)).
failure.
p_state(r2,painted).
p_freehand(r2,right).
p_freehand(r2,left).
p_at(board,free).
plan(r2,p007).
end(model(1)).
```

```
begin(model(2)).
success.
p_freehand(r1,right).
p_at(board,free).
plan(r1,p007).
p_state(r1,ok).
end(model(2)).
```

```
begin(model(3)).
success.
p_state(r1,ok).
p_at(board,free).
p_handfree(r1,left).
plan(r1,p007).
end(model(3)).
```

```
begin(model(4)).
success.
p_state(r3,ok).
p_freehand(r3,left).
p_at(board,free).
plan(r3,p007).
end(model(4)).
```

```
begin(model(5)).
success.
p_state(r1,ok).
p_freehand(r1,left).
p_at(board,free).
plan(r1,p007).
end(model(5)).
```

```
begin(model(6)).
failure.
p_state(r3,painted).
p_freehand(r3,right).
p_freehand(r3,left).
p_at(board,free).
plan(r3,p007).
end(model(6)).
```

Table 2. Models at level 2, examples are generated by different agents.

6 Conclusion

We have shown how ILP methods, particularly the induction of logical decision trees, can be used to extend a BDI architecture to endow agents with learning skills. These skills were designed to be compatible with the practical rationality behind the behavior of BDI agents. The result is a BDI learning agent architecture implemented on Lisp. The architecture also includes two non-standard BDI features, several options for MAS simulation, and an interface to the DTP theorem prover. The example introduced in the previous section, shows that BDI agents situated in a MAS, increase their chances of learning if they can share training examples. Our research contributes from a MAS learning perspective, to extend the well known and studied BDI model of rational agency, beyond its limitations, i.e., lack of learning competences and MAS functionality.

As mentioned here in, at the moment of submission, only Cindy Olivia et al. [21] are focused on the same problem. They present a mono-agent Case-Based BDI framework applied to intelligent search on the web. Even when agents in this framework are based on BDI representations, they perform case-based reasoning (CBR) to act, instead of practical reasoning. CBR is a learning method directed towards action, which makes it very attractive for learning agents, nevertheless much more work is needed to understand the relationship between CBR and the theory of practical reasoning, i.e., what the meaning of similarity functions in terms of practical reasoning is. Observe that this work is ssituated at level 0 (true isolated agents) of the hierarchy of MAS learning systems proposed here.

Future work includes implementing more MAS features for the architecture, e.g., including an ACL. More interestingly, it is possible to design protocols for sharing information of the learning set in more complex situations, e.g., agents

having the same competences, but different plans. This is particularly true if ACE is modified to learn incrementaly with each example it receives. We must also consider the relationship between learning and the multi modal logic theories of intentional agency, e.g., If the learning processes described here, maintain the strong-realism conditions, and other forms or realism.

Acknowledgments

We thank David Kinny and Pablo Noriega for their help in our research. The first author is supported by Mexican scholarships from Conacyt, contract 70354, and Promep, contract UVER-53.

References

1. Bradzil, P., et.al.: Learning in Distributed Systems and Multi-Agent Environments. In: Kodratoff (ed.): Machine Learning - EWSL-91, European Working Session on Learning. Lecture Notes in Computer Science, Vol. 482. Springer-Verlag, Heidelberg, Germany (1991)
2. Bratman, M.: Intention, Plans, and Practical Reasoning. Harvard University Press, Cambridge MA., USA (1987)
3. Bratman, M., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational Intelligence. 4:349–355 (1988)
4. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial Intelligence, 101(1–2):285–297 (1998)
5. Blockeel et al., H.: Executing query packs in ILP. In: Cussens, J. and Frish, A. (eds.): Inductive Logic Programming, 10th International Conference, ILP2000, London, U.K. Lecture Notes in Artificial Intelligence, Vol. 1866, pages 60–77. Springer Verlag, Heidelberg, Germany (2000)
6. Castelfranchi, C.: Modelling Social Action for AI Agents. Artificial Intelligence, 103(1):157–182 (1998)
7. Dennett, D.C.: The Intentional Stance. MIT Press, Cambridge MA., USA (1987)
8. Charniak, E., McDermott D.: Introduction to Artificial Intelligence. Addison-Wesley, USA (1985)
9. García, F.: Apprentissage et Planification. In: Proceedings of JICAA'97 USA (1997) 15–26
10. Geddis, D.F.: Caching and First-Order inference in model elimination theorem provers. Ph.D. Thesis. Stanford University, Stanford, CA., USA (1995)
11. Georgeff M.P., Lansky A.L.: Reactive Reasoning and Planning. In: Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pages 667–682, Seattle WA., USA (1987)
12. Georgeff, M.P., Rao A.S.: A Profile of the Australian AI Institute. IEEE Expert, 11(6):89–92, December (1996)
13. Georgeff, M.P. et.al.: The Belief-Desire-Intention Model of Agency. In: Müller, J., Singh M.P., and Rao, A.S. (eds.): Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98). Lecture Notes in Artificial Intelligence, Vol. 1555, pages 1–10. Springer Verlag, Hedelgerg, Germany (1999)

14. Huber, M.: A BDI-theoretic mobile agent architecture. In: Proceedings of the Third Conference on Autonomous Agents (Agents99). Seattle, WA., USA (1999) 236–243
15. D'Inverno, M., Kinny, D., Luck, M., Wooldridge M.: A Formal Specification of dMARS. In: Intelligent Agents IV. Lecture Notes in Artificial Intelligence, Vol. 1365. Springer-Verlag, Berlin Heidelberg New York (1997) 155–176
16. Kinny, D. and Georgeff, M.P., Commitment and effectiveness of situated agents. In: Proceeding of the Twelfth International Conference on Artificial Intelligence IJCAI-91, pages 82–88, Sidney, Australia (1991)
17. Lightfoot, D., Formal Specification Using Z. The Macmillan Press LTD, Macmillan Computer Science Series, London, UK (1991)
18. Metral, M.: A generic learning interface architecture. Massachusetts Institute of Technology. Master's thesis. Cambridge, MA., USA (1992)
19. Mitchell, T.M.: Machine Learning, Mc Graw-Hill International Editions, Singapore (1997)
20. Muggleton, S., de Raed, L.: Inductive Logic Programming: Theory and Methods. Journal of Logic Programming, 19:629–679 (1994)
21. Olivia, C., et.al.: Case-Based BDI agents: An Effective Approach to Intelligent Search on the WWW. In: AAAI Symposium on Intelligent Agents. Stanford University, Stanford CA., USA (1999)
22. Quinlan, J.R.: Induction of Decision Trees. Machine Learning 1:81–106 (1986)
23. Rao A.S., Georgeff, M.P.: Decision procedures of BDI logics. Journal of Logic and Computation, 8(3):293–344 (1998)
24. Russell, S.J., Norvig, P.: Artificial Intelligence, a modern approach. Prentice-Hall, New Jersey NJ, USA (1995)
25. Singh, M., Rao, A.S., Georgeff, M.P.: Formal Methods in DAI: Logic-based representations and reasoning. In: Weiss, G. (ed.): Multiagent Systems, a modern approach to Distributed Artificial Intelligence. MIT Press, Cambridge MA., USA (1999)
26. Stone, P., Veloso, M.: Multiagent Systems: A Survey from a Machine Learning Perspective. Autonomous Robotics, 8(3):345-383 (2000)
27. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An introduction. MIT Press, Cambridge, MA., USA (1998)
28. Weiss, G., Sen, S.: Adaptation and Learning in Multiagent Systems. Lecture Notes in Artificial Intelligence, Vol. 1042. Springer-Verlag, Berlin Heidelberg New York (1996)
29. Wooldridge, M.: Reasoning about Rational Agents. MIT Press, Cambridge MA., USA (2000)