ARTÍCULO

# On learning intentionally

**Alejandro Guerra-Hernández**

Departamento de Inteligencia Artificial
Facultad de Física e Inteligencia Artificial
Universidad Veracruzana
Sebastián Camacho No. 5, Xalapa, Ver., México 91000.
aguerra@uv.mx

**Amal El Fallah-Seghrouchni**

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

**Henry Soldano**

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

**Abstract**

Any attempt to include learning competences in agent technologies, involves the problem of integrating such competences in the rationality of the agents. This work discusses the design of learning intentional agents in a multiagent system (MAS). Learning intentionally means that these agents learn to adopt their intentions under practical reasoning considerations, specified in terms of Beliefs, Desires, and Intentions (BDI). Specifically, they are intended to update the formula expressing when a plan is executable (context of plans), if some failure is detected. At the MAS level, a form of distributed social learning, based on collaborative goal adoption, is built around the concept of social awareness. Examples illustrate the role of induction of logical decision trees, an inductive logic programming method, in the design of these BDI learning agents.

**Key words:** Learning, Intentionality, Practical Reasoning, BDI Agents, Multiagent Systems.

## 1 Introduction

This work discusses the design of BDI learning agents in a multiagent system (MAS). Since the BDI model of agency [9, 4, 10, 21] is well known, it is not discussed here in detail. Instead, the focus of interest is on: how does practical reasoning [3] led to the induction of logical decision trees [1], as the learning method adopted for these agents; and how does social awareness induce a hierarchy of MAS levels, where distributed learning is pos-

sible using a protocol, based on collaborative goal adoption [5].

To guide the discussion, the generic learning agent architecture (Fig. 1), proposed by S. Russell and P. Norvig [20], is adopted. The performance element is responsible for deciding what to do. It can be seen as an agent without learning competences. The learning element is intended to provide modifications to the performance element to improve the behavior of the agent. The critic provides feedback about the performance of the agent, while the problem generator suggests cases or situations which constitute informative experiences for the learning element.

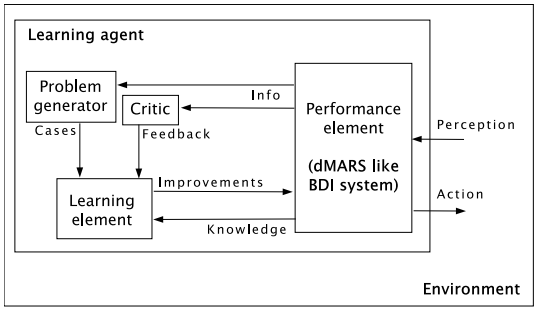


**Figure 1. Generic learning architecture and a BDI interpreter as the performance element.**

In what follows, it is assumed that the performance element of these agents is a BDI system implemented *ex-nihilio* [12, 13], following dMARS specification [14] (Fig. 2). So, if nothing else is suggested, dMARS syntax and semantics are assumed.

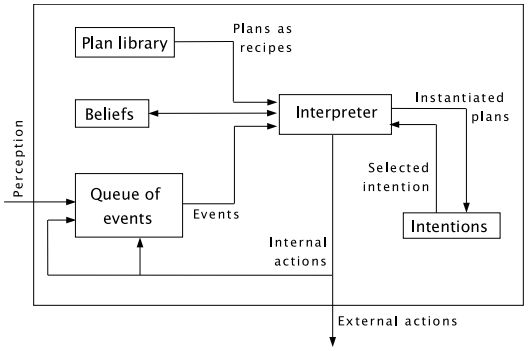


**Figure 2. A BDI architecture following dMARS specification.**

The design of the learning element, and consequently the adoption of a particular learning method, is usually affected by some major issues: Which component of the performance element will be improved by learning? Which representation is used by this component? What kind of feedback, training examples, and prior knowledge are available? The answer to these questions is highly affected by practical reasoning considerations. First, section 2 explains the differences between practical and epistemic reasoning, and their implementation in BDI agents. Then, it is discussed the adoption of the context of plans, a formula expressing when a plan is successfully executable, as target concept. Section 3 details different design issues, as the representation used for the target concept, the feedback available for the learning element, and the use of background knowledge. Then Tilde [1], a learning method to induce logical decision trees under the design constraints explained before, is briefly introduced. Section 4 discusses a hierarchy of MAS, based on the degree of awareness a learning agent has about other agents in the system. This hierarchy is used to define the kind of distributed learning our agents will perform. Section 5 exemplifies how do our agents learn at the first two levels of the hierarchy, covering the centralized and distributed learning cases. Section 6 discusses conclusions and future work.

# 2 Practical and epistemic reasoning in BDI agents

Consider that the robots shown in figure 3 are BDI agents. Now, suppose that agent `r2` is thinking about what object to take. Since it is deciding what to do, `r2` is performing practical reasoning. Suppose `r1` tries to decide what object `r2` will take. This is epistemic reasoning , i.e., deciding what is believed. Practical and epistemic reasoning are present explicitly in the BDI interpreter. They are activated by the `achieve` and `test` goals respectively (See the plan body in figure 3).

Practical reasoning lead to the adoption of intentions. For example, if the event `(achieve (p-sanded board))` is perceived by an agent, it will look for the plans which trigger unifies this event (relevant plans). Then, the agent will select a relevant plan which context is a logical consequence of its current beliefs (applicable plan) to form an intention. By default, the selector functions take the the first plan, or intention, founded, but meta plans [9], and utility functions [21] are

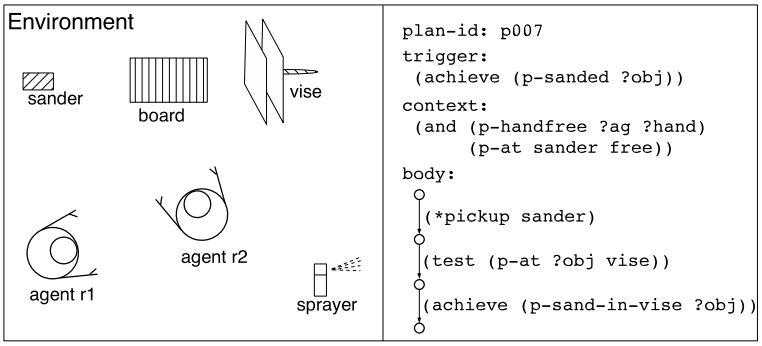


**Figure 3. A simple scenario of BDI agents and a plan.**

also possible. Observe that, although the agent seems to be performing a kind of epistemic reasoning to verify if a plan is applicable, in fact the agent is verifying if it has practical reasons to adopt the plan as an intention.

Even when the beliefs 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 (filter of admissibility) [3]. In doing so, beliefs 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 the reasonableness of arbitrary choices in Buridan cases[1], e.g., it is *practical reasonable* for agent `r2` 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. Sometimes this is called wishful thinking, and it is not *epistemic reasonable* for agent `r1` to belief that agent `r2` will adopt a particular plan, in this way.

What is relevant here, is that the context of plans express practical reasons to act in some way and not in another, validated by the background frame of beliefs and prior intentions. They are the target concept, if BDI agents are going to learn about their practical reasons to adopt a plan as an intention.

# 3 Design issues

Once the context of the plans is identified as the component of the performance element to be improved by learning, some issues of design may be discussed. They include representation of the target concept, the feedback available, the use of prior knowledge, and the learning method adopted, induction of logical decision trees.

## 3.1 Representations

Representations in the BDI interpreter are based on two first-order formulae: Belief formulae and situation formulae. While beliefs are grounded belief formulae, like Prolog facts, belief formulae are not necessarily grounded. Every belief formula is also a situation formula, but situation formulae also include conjunctions and/or disjunctions of belief formulae. For example, the context of plan `p007` (Fig. 3) is a situation formula.

The context of plans are represented as situation formulae. This representation has two immediate consequences for candidate learning methods: First, given the first-order representation of formulae, propositional learning methods are discarded. Second, the fact that the context of plans is represented as situation formulae, demands disjunctive hypothesis, e.g., decision trees.

## 3.2 Feedback

Getting feedback from the BDI interpreter is almost direct, since it already detects and processes success and failure of the execution of intentions. Again, there are different methods to detect failures [9, 21]. In this case, the primitive actions of the agents, verify if they achieve their intended effects, e.g., if belief $\phi$ is true after action $\alpha$. If a step of a plan returns `nil`, because the effect of the action was out of its domain, the failure is detected. Then, a set of internal actions, adding

[1] According to the philosopher Jean Buridan (cited in [3]), they are situations equally desirable.

and deleting beliefs accordingly to the result obtained, is executed.

It is possible to define a special internal action to generate a log file of training examples for the learning task. By special, it is meant that such internal action is not definded in dMARS. Items to built these examples include: the beliefs characterizing the situation when the plan was selected to form an intention; the identifications of the agent and the plan in question; and the feedback itself, as the label of success or failure obtained after the execution of the intention.

### 3.3 Background knowledge

Although, usually agents do not use the knowledge they have while learning, this is not a good idea provided that BDI agents have very rich prior information. There are two possible sources of background knowledge for them. First, the plan library may be seen as prior information, since 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 interpreter keeps track of predicates, functions, and their signatures, used in the definitions of the plans in the library of each agent. This can be used by the agent to specify the language in which the target concept is going to be expressed.

### 3.4 Top-down Induction of Logical Decision Trees

Because of the representation of the context of plans as situation formulae, decision trees are adopted as target representation. Top-down induction of decision trees is a widely used and efficient machine learning (ML) technique. As introduced in the ID3 algorithm [19] 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. Training examples are represented as a fixed set of attribute-value pairs.

The structure of decision trees satisfy the constraints imposed by the disjunctive nature of situation formulae, but the propositional representation of the examples do not satisfy the first-order requirements of these formulae. Another limitation is that ID3-like algorithms can not use information beyond the training examples, i.e., other things the agent believes, as their plans. Inductive Logic Programming (ILP) [16] can overcome the limitations of classic ML inductive methods.

Logical decision trees upgrade the attribute-value representation of classic decision trees, to a first-order representation using the ILP setting known as learning from interpretations [1]. In this setting, each training example $e$ is represented by a set of Prolog facts. Background knowledge can be given in the form of a Prolog program $B$. The real example is represented by the interpretation formed by 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 the example, as it is the case for 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 labeled examples $E$, each consisting of a a set of definite clauses $e$ labeled 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 labeled 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 [16]. The local assumption is relevant if we want the agents configuring their learning settings themselves.

ACE [2] is a learning from interpretations system to induce logical decision trees, i.e., decision trees where every internal node is a first-order conjunction of literals. It uses the same heuristics that ID3-like algorithms (gain-ratio and post-pruning heuristics), but computations of the tests are based on the classical refinement operator under $\Theta$-subsumption [18], which requires the specification of a language $L$ stating which kind of tests are allowed in the decision tree. Section 5 exemplifies these concepts in detail.

# 4 Social Awareness

Awareness of other agents in the system seems to be indicative of a MAS hierarchy of increasing complexity. In a certain way, this hierarchy corresponds to the scale of intentionality proposed by D. Dennett [6]. Levels in this hierarchy are as follows:

1. At the first level, agents act and learn from direct interaction with the environment. They are not explicitly aware of other agents in the MAS. However, the changes produced by other agents in the environment may be perceived by the learning agent. For example, in the scenario of figure 3, `r1` can be specialized in painting objects, while `r2` sands them. It is possible to program the painter robot, without awareness of the sander, i.e., all `r1` has to know is that once an object is sanded, it can be painted. The true isolated learning case with one agent, may be seen as a special case of this level.

2. At the second level, agents act and learn from direct interaction with other agents using message exchange. 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.

3. At the third level, agents act and learn from the observation of the actions performed by other agents in the system. It involves a different kind of awareness from that of the second level. 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. Observe that this seems to involve epistemic reasoning, e.g., adopting the belief that that `r2` will sand the object.

The purpose of our agents while learning, is to update the situation formula (context) expressing when a plan is executable, when the execution of the plan fails. If an agent can update this context after its own experience, no communication is needed and learning is performed at the first level of the hierarchy. Otherwise, the agent will try to learn from the experience of other agents, starting a kind of collaborative goal adoption [5] process, where the BDI agents in the MAS that have the same plan, cooperate with the goal of the learner agent, because they are also interested in the results of his learning process, e.g., inducing the context of the shared plan. The group of agents interested in a learning goal, may be seen as an emerging social structure, where the roles of learner and supervisors are dynamically assigned.

# 5 BDI learning agents

This section explains in detail how do our BDI agents learn at the first two levels of the hierarchy introduced above. The scene shown in figure 3, and the plan `p007`, are used in all the examples.

## 5.1 Centralized learning (level 1)

Consider that the agent `r1` has selected the plan `p007` to form an intention 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 did the plan fail, even if there were practical reasons to adopt it as an intention.

In order to execute the learning process, the agent post the event `(achive (p-learn r1 p007))` when detecting the failure of the plan. There is a relevant plan for this kind of events:

```
(define-plan 'plan-learning
  :trigger '(achieve (p-learn ?ag ?plan))
  :context '(p-start-learning ?ag ?plan)
  :body '((*generate-setting ?ag ?plan)
          (*run-ace)
          (*show-results)))
```

When the success or failure of an intention is detected, the agent `r1` keeps track of these executions as learning examples. If enough learning examples (six in the example) have been collected by the agent, the predicate `p-start-learning` is true, and `plan-learning` becomes executable.

The action `*generate-setting` configures the learning task of the agent, generating a file identified by the extension `.kb` (knowledge base) which contains the learning examples; a file identified by

the extension `.bg` (background) which contains the prior knowledge or background theory; and a file identified by the extension `.s` (setting) which contains the parameters of induction and the language bias $L$, i.e., the predicates used to build the target concept. The `plan-id` names these files, that are generated automatically by the agent as follows.

Each learning example is coded as a model of the learning from interpretations paradigm. A 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 that the agent had when the plan was selected to form an intention. Partial models are memorized by the agent when the plan is selected as relevant and applicable. The label is added in the execution phase of the BDI interpreter. In this example, the knowledge base is stored in the file `p007.kb`, and includes the following models:

```
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,left).
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_freehand(r1,left).
p_at(board,free).
end(model(6)).
```

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 Prolog-like format, encoding 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 printed while learning (talking); the minimal number of cases in a leaf; the format of the output (C4.5, an ID3-like format, and as a logic program); and the classes 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 inducing the tree (language bias $L$). 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 in the definition 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 also asks the learning algorithm to 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 to define 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 that the variable must be instantiated after the examples in the knowledge base. Different formulations for $L$ are possible: all predicates known, the predicates used in the set of plans that deal with the given event, etc.

Then 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 itself the definition of the plan. The strategy adopted to incorporate the results obtained, depends on the domain of application, i.e., in some cases 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:  [success] [1.0/1.0]
```

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. The result suggests to adopt the new context:

```
(and (p-handfree ?ag ?hand)
     (p-at ?obj free)
     (not (p-state ?ag painted)))
```

## 5.2 Decentralized learning (level 2)

Now, consider why should a BDI agent learning isolated (level 1) try to exchange messages to learn (level 2)? There are two situations where an agent should consider to communicate while learning: when the agent is no able to start the execution of its learning process, i.e., it does not have enough examples; or when the agent is not able to find out an hypothesis to explain the failure of the plan in question. In both cases the learning agent may ask for more evidence to other agents in the MAS.

Recall that training examples in this setting are not represented as fixed vectors of attribute-value pairs, but as models, i.e., labeled sets of definite clauses. Vertical fragmentation is not a problem in such representation, because models are more flexible than the attribute-value representation. For example, consider the following two models:

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

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

From the learning from interpretations perspective, these training examples are just models of the same target concept, `success`, even if they are not homogeneous, e,g., model 2 has information about the board, while model 1 does not. In propositional representations this implies that board information is in a different data source (vertical fragmentation). This is very important. Since our BDI learning agents face only horizontal fragmentation, the exchange of training examples is enough to achieve learning at the second level.

A group of agents is said to have the same *competence* for a given event, if they share the same plan to deal with it. Competence in our architecture is defined in terms of sets of plans that deal with a particular event. Once competence is defined, two ways of sending messages asking for help in a learning process are possible: i) The learner agent broadcasts its help message, including the `id` of the plan it is trying to update, then other agents in the MAS may accept and process the message, if and only if the plan in question is on their competence; and ii) Competence is used to build a directory for each agent, where each plan in their library is associated with the list of all agents in the MAS sharing the plan. This is possible because our architecture have two special variables storing the `id` of agents defined in the system, and the global plan library of the system, i.e., the union of all plans used by all the agents in the MAS. The latter was adopted implementing a function called `mas-competence`. This function builds the directory for each agent. For example:

```
> (agent-mas-competence r1)
((P001 (R2 R3)) (P002 (R2 R3))
 (P003 (R2 R3)) (P004 NIL)
 ...)
```

So, when agent `r1` is learning about plan `p003`, it believes that agents `r2` and `r3` have the same

plan. On the contrary, plan p004 is not shared with anybody else. If two agents have the same plan, they can be engaged in a process of distributed data gathering, i.e., they can share the examples they have collected to learn individually, as follows. Every agent has a plan to ask for examples:

```
(define-plan 'plan-ask-help-learning
  :trigger '(achieve (p-learn ?ag ?plan))
  :context '(not(p-start learning ?ag ?plan))
  :body '((*ask-help ?ag ?plan)))
```

The trigger of this plan is also (achieve (p-learn ?ag ?plan)), but it is executed only when the agent can not start its learning process, i.e., it does not have enough training examples. The external action *ask-help sends a message to the agents sharing the ?plan, communicating the following goal: (achieve (p-help ?ag ?plan)). This results in the goal being posted in the event queue of the receiver. Agents answer to a help request, because they have a plan to process this event:

```
(define-plan 'plan-answer-help-learning
  :trigger '(achieve (p-help ?ag ?plan))
  :context 'true
  :body '((*answer-help ?ag ?plan)))
```

The answer-help plan is always executed, i.e., its plan context is true, and the external action *answer-help sends back the examples found to the learner agent. If no examples are found, an empty list is send back. This can be avoided if the plan is modified to test if at least one example has been found, but sending back the empty list lets the learning agent to know if its request has been processed. In order to implement *answer-help the syntax of the system must be slightly modified. Achieve goals syntax must validate expressions of the form (achieve (p-add-examples ?ag ?examples)). The message send by the external action *answer-help is of this general form. As usual, the learner agent include the examples of these messages using a plan for it:

```
(define-plan 'plan-include-examples
  :trigger '(achieve
             (p-add-examples ?ag ?examples))
  :contex 'true
  :body '((*include-examples
           ?ag ?examples)))
```

This plan is always executed, resulting in the examples being stored in the log of the learner agent. Figure 5 shows graphically these interactions. Agent r2 detects a failure of plan p007, but it has not enough examples to start learning. The agent sends a p-help event (dotted arrows) to agents r1 and r3 because it believes, they have the same competence, i.e., they share the plan p007. These agents send back (solid arrows) their examples about p007. Once agent r2 has enough examples it can start learning hypothesis h. Models obtained in this way are:

```
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)).
```

The learning agent r2 obtains the following logical decision tree:

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

This result is the same that the one obtained by the isolated agent at the first level, but there are important differences. Since the first execution by the agent r2 of plan p007 lead to a failure (model 1), this agent would not be able to collect successful training examples for this plan (plans that failed are not reconsidered again in dMARS). It

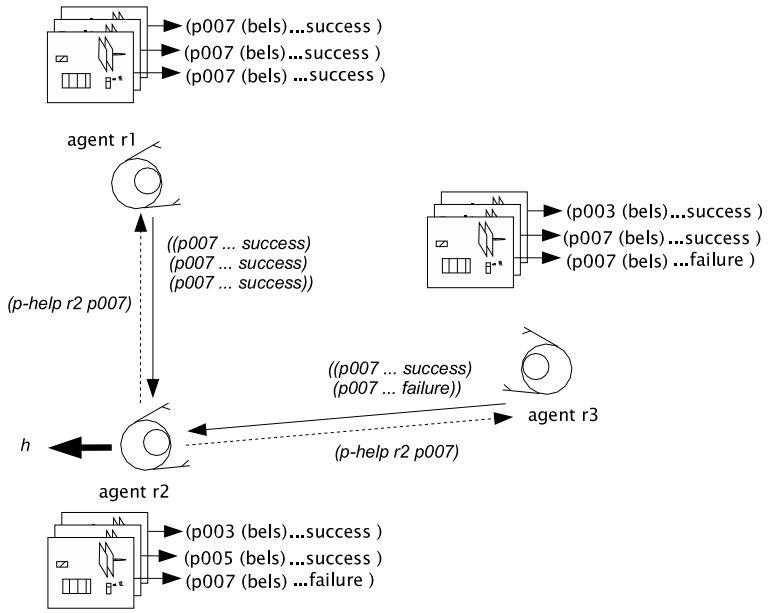


**Figure 5. MAS learning interactions**

means that outside the MAS, agent `r2` is not able to learn about plan `p007`. The failure example of agent `r3` is needed to induce the tree, and this means that any agent would not be able to learn about `p007` at the first level. Observe that training examples are collected by the agents themselves, hence obtaining examples is harder than in classic supervised learning. ILP methods help the agent since fewer examples may be needed to learn, provided that the background theory is relevant. Interactions at the second level of MAS may help to collect examples faster, taking benefit of the experience of other agents.

If the definition of the plan is modified accordingly to the decision tree induced, all agents using the plan share the new definition. This justifies the cooperative goal adoption strategy: sharing examples is easy, while learning is not. Since the learner agent will share the hypotesis finded, cooperation became advantageous.

# 6 Results and future work

We belief that the achievement of learning agents, particularly as technological solutions, involves seriously considering the way learning is integrated in the rationality of the agents. Learning intentionally was the way to integrate learning competences in BDI agents, but the choices of design described here, e.g., what component of the agent is to be improved while learning? etc., possibly apply to other kind of agents,

In our approach, theory of practical reasoning [3] by M. Bratman, provided the choices of design. Up to our knowledge, there is only another approach to BDI learning agents, based on case-based reasoning [17]. The approach has been better formalized by J.M. Corchado et al. [11] and both systems has been tested in realistic problems. However, there are some gaps in the mapping between practical reasoning and CBR implementations, e.g., what is the relationship between the frame of flat beliefs (true or false) and prior intentions, and the methods used to retrieve solutions in CBR? We decided to adopt a position closer to intentionality, as defined by Bratman.

The architecture proposed is currently implemented in Lisp and it includes three extra BDI features: i) Some MAS simulation facilities; ii) An interface to the DTP theorem [8] prover to perform epistemic reasoning; and iii) The learning competences. The examples show that BDI agents situated in a MAS, increase their chances of learning if they can share learning examples. Our research contributes from a MAS learning perspective, to extend the well known BDI model of rational agency, beyond its limitations, i.e., lack of learning competences and MAS functionality.

Future work includes the redefinition of the architecture in terms of AgentSpeak(L), which has a

simpler syntax for the context of plans (conjunctive form) and a simpler semantic model. Then alternative representations for the target concept can be considered. Native Lisp methods for epistemic reasoning and learning will be provided. Message exchange will be supported by an agent communication language, e.g., KQML. Particularly, we are interested on incremental learning methods that enable us to formulate more interesting protocols to exchange learning examples at level 2, and how do convergence may be guaranteed then.

## References


[1] Blockeel, H., De Raedt, L. Top-down induction of first-order logical decision trees. Artificial Intelligence, 101(1–2):285–297, 1998.

[2] Blockeel et al., H. Executing query packs in ILP. In: *Inductive Logic Programming, 10th International Conference ILP2000*, London, U.K. Lecture Notes in Artificial Intelligence, Vol. 1866, pp. 60–77. Springer Verlag, Heidelberg, Germany, 2000.

[3] Bratman, M.E. Intention, Plans, and Practical Reasoning. Harvard University Press, Cambridge MA., USA, 1987.

[4] Bratman et al., M.E. Plans and resource-bounded practical reasoning. Computational Intelligence, 4:349–355, 1988.

[5] Castelfranchi, C. Modelling Social Action for AI Agents. Artificial Intelligence, 103(1):157–182, 1998.

[6] Dennett, D.C. The Intentional Stance. MIT Press, Cambridge MA., USA, 1987.

[7] Charniak, E., McDermott D. Introduction to Artificial Intelligence. Addison-Wesley, USA, 1985.

[8] Geddis, D.F. Caching and First-Order inference in model elimination theorem provers. PhD Thesis. Stanford University, 1995.

[9] Georgeff, M. P., Lansky, A. L. Reactive reasoning and planning. In: *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pp. 677–682, Seattle, WA., USA, 1987.

[10] Georgeff et al., M. 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).* Lecture Notes in Artificial Intelligence, Vol. 1555, pp. 1–10. Springer Verlag, Heidelberg, Germany, 1999.

[11] Corchado et al., J.M. Development of CBR-BDI Agents: A Tourist Guide Application. In: *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Proceedings.* Lecture Notes in Computer Science, Vol. 3155, pp. 547–559 Springer Verlag, Heildelberg, Germany, 2004.

[12] Guerra Hernández, A., El Fallah-Seghrouchni, A., Soldano, H. BDI Multi-agent Learning Based on First-Order Induction of Logical Decision Trees. In: *Intelligent Agent Technology: Research and Development*, 2nd Asia-Pacific Conference on IAT, World Scientific, pp. 160-169, Maebashi, Japan, 2001.

[13] Guerra Hernández, A., El Fallah-Seghrouchni, A., Soldano, H. Learning on BDI Multiagent Systems. In: *CLIMA IV Computational Logics on Multiagent Systems. Revised and Selected Papers.* Lecture Notes in Computer Science, Vol. 3259, pp. 218–233. Springer Verlag, Heidelberg, Germany, 2004.

[14] D'Inverno et al., M. A Formal Specification of dMARS. In: *Intelligent Agents IV.* Lecture Notes in Artificial Intelligence, Vol. 1365, pp. 155–176. Springer-Verlag, Heidelberg, Germany, 1997.

[15] Mitchell, T.M. Machine Learning. Mc Graw-Hill International Editions, Singapore, 1997.

[16] Muggleton, S., de Raed, L. Inductive Logic Programming: Theory and Methods. Journal of Logic Programming, 19:629–679, 1994.

[17] Olivia et al., C. Case-Based BDI Agents: An Effective Approach for Intelligent Search on the web. In: *Proceeding AAAI-99, Spring Symposium on Intelligent Agents in Cyberspace.* Stanford University, USA, 1999.

[18] Plotkin, G.D. A note on inductive generalization. Machine Intelligence, 5:153-163, 1970. Edinburgh Univ. Press, Edinburgh.

[19] Quinlan, J.R. Induction of Decision Trees. Machine Learning 1:81–106, 1986.

[20] Russell, S.J., Norvig, P. Artificial Intelligence, a modern approach. Prentice-Hall, New Jersey NJ, USA, 1995.

[21] Wooldridge, M. Reasoning about Rational Agents. The MIT Press, Cambridge, MA., USA., 2000.