Distributed Learning in Intentional BDI Multi-Agent Systems

Alejandro Guerra Hernández
Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Sebastián Camacho No 5, Xalapa, Ver.,
México 91000

Amal El Fallah-Seghrouchni
Université Paris 6
Laboratoire d'Informatique de Paris 6
UMR 7606 - CNRS, 8 rue du Capitaine Scott
Paris 75015, France

Henry Soldano
Université Paris 13
Laboratoire d'Informatique de Paris Nord
UMR 7030 - CNRS, Institut Galilée,
Av. Jean-Baptiste Clément, Villetaneuese 93430, France

Abstract

Despite the relevance of the belief-desire-intention (BDI) model of rational agency, little work has been done to deal with its two main limitations: the lack of learning competences and explicit multi-agent functionality. Our work deals with the problem of designing BDI learning agents situated in a Multi-Agent System (MAS). From the MAS learning perspective, we have proposed an extended BDI architecture, where agents are able to perform induction of first-order logical decision trees. These agents learn about their practical reasons to adopt a plan as an intention. Particularly, induction is used to update these reasons (the context of plans), if a plan fails when executed, after it had been selected to form an intention. Here, we emphasize the way MAS concepts, as cooperative goal adoption, enable distributed forms of learning, e.g., distributed data gathering. Consistency between learning and the theory of practical reasoning is guaranteed, i.e., learning is just another compentence of the agents, performed under BDI rationality.

1. Introduction

This paper focuses on the main limitations of the belief-desire-intention (BDI) model of rational agency: the lack of learning compentences and explicit multi-agent functionality. Particularly, it describes how BDI agents can learn exploiting the presence of other agents in a Multi-Agent System (MAS), through the exchange of training examples. From a MAS learning perspective, we have proposed [10] a BDI learning architecture, inspired on the definition of a learning agent by S. Russell and P. Norvig [18]. Achieving BDI learning agents is not evident, because the nature of the model. While this kind of agents perform practical reasoning, directed towards action, they use first-order representations. Learning methods directed towards action are very popular in MAS learning, but use less expressive representations, propositional ones. Learning methods with more expressive representations are usually conceived as isolated learning systems, directed towards epistemic reasoning, e.g., knowldege discovery and data mining. This may explain why in the abundant MAS learning literature, only C. Olivia et al. [15] has considered the problem of BDI learning agents, despite the relevance of the model.

The approach we use to solve this problem, consists in using Inductive Logic Programming (ILP) [14] methods, particularly Induction of Logical Decision Trees [3], to enable the BDI agents to learn why they should adopt a particular plan instance as an intention. The practical reasons to adopt a plan are coded in the plan context. Isolated learning is considered an special case in a hierarchy of MAS learning systems, one where the learning agents are not aware of other agents in the system. This level is reported in detail in [9]. Here, we emphasize learning at the second level of the learning MAS hierarchy, where learning agents are aware of other agents in the system, and message exchange is possible to act, hence to learn.

The paper is organized as follows: Section 2 introduces the basic concepts of the BDI model of agency, necessary to explain our approach. Section 3 describes briefly our BDI learning architecture, and a hierarchy of learning MAS based on awareness. Section 4 introduces an example of a BDI learning agent at level two of the hierarchy of learning MAS; and section 5 presents conclusions and future work.

2. BDI agency

Software agents are usually characterized as computer systems that exhibit flexible autonomous behavior [20], which means that these systems are capable of independent, autonomous action in order to meet its design objectives. BDI models of agency approach this kind of behavior through two related theories about the philosophical concept of intentionality: Intentional Systems, defined by D. Dennett [6] as entities which appear to be subject of beliefs, desires and other propositional attitudes; and the Practical Reasoning theory, proposed by M. Bratman [2] as a common sense psychological framework to understand ourselves and others, based on beliefs, desires, and intentions conceived as partial plans. These two related notions of intentionality provide us with the tools to describe agents at the right level of abstraction, adopting the intentional stance, i.e., in terms of belief, desires and intentions (BDI), and to design agents in a compatible way with such intentional description, i.e., agents as practical reasoning systems. Different aspects of intentionality and practical reasoning have been formally studied, resulting in the so called BDI logics [17]. For a road map of the evolution of these formalisms, see [19, 20]. Implementations of these models of agency make use of refinement techniques, e.g., using specifications in Z language [12] like dMARS specification [11].

BDI concepts will be introduced using a very simple scenario proposed by Charniak and McDermott [7]. This scenario (Fig. 1) is originally composed by 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! to introduce the case of incompatible goals, since once painted the robot is not operational 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 [5], e.g., sharing goals, conflict for resources, etc. This scenario, and the plan showed with it, will be used in the examples in the rest of the paper.

2.1. The BDI model

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

Beliefs. They are grounded literals of first-order logic, representing information about the world, updated by the perception of the environment, and by the execution of intentions. The scenario shown in Fig. 1 is represented like: `(p-state r1 ok)`, `(p-handfree r1 left)`, `(p-at sander free)`, etc. Where `free` is a constant meaning that the object is not at the vise or an agent has it. Everything else is self explicative. Syntax looks lispy because we are implementing on Lisp, more on this at the end of the section.

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

Desires. Also known as goals, correspond to the tasks allocated to the agent and are usually considered as logically consistent among them. Two kinds of desires are usually adopted: To achieve a desire, expressed as a belief formula; and to test a situation formula, that is a belief formula or a disjunction and/or a conjunction of them. While achieve goals involve practical reasoning, test goals involve epistemic reasoning.

Event queue. Perception is mapped to events stored in a queue. Events are of three kinds: The acquisition or removal of a belief; the reception of a message; and the acquisition of a desire. Events are implemented as structures keeping track of historical information, e.g., plans used to deal with them, success or failure, etc. The reception and emission of messages is used to implement MAS learning competence of our BDI agents. Agent communication languages can easily be included in our architecture, since Lisp packages exist for them, e.g., FIPA ACL and KQML.

Plans. BDI agents have a predefined library of plans. Each plan has several components, the most relevant for us are shown in the simplified plan on Fig. 1. The trigger works as the invocation condition of a plan, it specifies the event a plan is supposed to deal with. If the agent detects an event like `(achieve (p-sanded board))` in the event queue, it will consider plan `p007` as relevant. The context specifies, as a situation formula, if the plan is applicable, i.e. when the plan should be considered to form an intention. 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 as a tree composed by exter-

nal actions (procedures, identified by a symbol starting by '*'); internal actions (add or del a belief); and 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, a plan might 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.

Intentions. They are courses of action an agent has commited to carry out. Each intention is implemented as a stack of plan instances. A plan instance is composed by a plan and the substitutions that makes it relevant and applicable. When updating intentions, two cases are possible: If the event being processed is external, i.e., no plan has generated it, then an empty stack is created and the plan instance selected is pushed on it. If the event is internal, it means that a previous plan has generated it, then the plan instance is pushed on the existing stacks containing this previous plan.

Figure 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 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 a plan instance for it.
4. Push this plan instance 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 it is a subgoal, post it to the event queue.

Different algorithms for the BDI interpreter, correspond to different commitment strategies of the agent, e.g., single-minded and open-minded commitments.

2.2. Some issues on implementation

Our BDI architecture, based on dMARS specification [11], is implemented on Lisp. We decide to do our own implementation because there is evidence [13] that adapting existing software to produce a learning agent, is not obvious at all, and sometimes it requires depth changes in the original design. Our implementation was intended to be extended with learning competences from the beginning. The specification of dMARS is pretty much influenced by Lisp, since its predecesor PRS was implemented using this language. Lisp was a natural choice.

The BDI architecture provides implemented functions to definine primitive actions available to the agents in the system; to define plans in terms of primitive actions; to define agents and assign them competences in terms of a plan library; to bootstrap each agent with initial events; and to execute the agents under different commitment strategies. These can be considered as standard BDI features, together with syntax verification tools supporting the definition of agents, and built-in functions to test if a formula is a logical consequences of a set of beliefs. 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, running as parallel processes in the same Lisp image. This image constitutes the environment shared by the agents; and an interface to use DTP theorem prover [8] 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.

3. BDI learning architecture

The BDI architecture introduced in the previous section will be seen as the performance component of the BDI

learning agents. Consider again that these agents perform practical reasoning to behave, while traditional AI systems, may be seen as performing epistemic reasoning directed towards beliefs or knowledge. From the role of beliefs in the theory of practical reasoning, e.g., the filter of admissibility, it is clear that even when they justify the behavior of the agent, the adoption of new intentions is constrained by the beliefs, together with prior intentions. Beliefs are playing a different role that the one they have in epistemic reasoning. Particularly, practical reasons to act sometimes differ from epistemic reasons. The context of plans may be seen as encoding practical reasons to act in some way and not in another. This context, together with the background frame of beliefs and prior intentions, support the rational behavior of intentional agents, as suggested by the theory of practical rationality. Since the plan library is defined apriori, we decided to extend the BDI architecture enabling the agents to learn about the context of plans if they failed when executed, i.e., updating the practical reasons to adopt a plan as part of intentions.

Representations in our BDI architecture are based on two first-order formulae: Belief formulae and Situation formulae. Belief formulae are first-order literals, possibly not grounded, i.e., including free variables. Situation formulae are conjunctions and/or disjunctions of belief formulae. These representation issues have two immediate results for considering candidate learning methods. First, using first-order representation for belief and situation formulae, discards the consideration of propositional learning methods. Second, the fact that the context of plans is represented as situation formulae, requires that the target representation of the learning method enables disjunctive hypothesis, i.e., decision trees. Getting feedback from our BDI architecture is almost direct, since it already processes success and failure of the execution of plan instances. Plans are used as background knowledge, and the language used to define each agent helps to configure the learning setting autonomously.

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 D. Dennett [6]. We intend to cover learning at levels 1 and 2 of this hierarchy. Levels in this hierarchy are as follows:

Level 0. Only one agent is there, the true isolated learning case, it can be seen as a special case of level 1.

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. Anyway, 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 the painter agent 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 agent 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 other agents actions. 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, so that, for instance the painter robot is able to perceive that the sander robot is going to sand the table.

3.1. Topdo wn Induction of Logical Decision Trees

From the issues discussed above, we decided to use decision trees as the target representation for the candidate learning method. Logical decision trees are based on the ILP paradigm known as learning from interpretations [3], enabling first-order representations for decision trees. 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, including knowledge that is true for any e. 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 training examples, as is the case of attribute-value representations, 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 set of definite clauses labeled with a value y in the domain of Y; iii) A language $L \subseteq Prolog$; 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) $\forall y' \neq y : H \wedge e \wedge B \not\models label(y')$.

ACE [4] is a learning from interpretations system, operating on logical decision trees, that is, decision trees where every test is a first-order conjunction of literals; and a variable introduced in some node (that does not occur in higher nodes) can not occur in the right subtree. ACE uses the same heuristics that C4.5 [16] (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. Agents using our

BDI learning architecture can configure themselves the setting required by ACE.

4. Distributed learning in a MAS of level 2

At level two of the hierarchy of learning MAS, agents are supposed to be aware of the presence of other agents in the system, and so, their learning processes may exploit this awareness, e.g., sharing information through message passing. Level two relies on communication, but in order to exploit it, the design of the BDI agents at this level must determine [1]: i) What is the purpose of an individual BDI agent to communicate? ii) When should an individual BDI agent communicate? iii) How does an individual BDI agent communicate?

In order to answer these questions, learning at level two is characterized as a distributed learning setting, where the learner and the data sources are BDI agents. This characterization enable us to argue that learning at level two may be seen as a parallel distributed learning setting, with only horizontal data fragmentation, i.e., training examples are distribued among the agents in the MAS, but all of them are complete examples. From the intentional stance, it is possible to use the concepts of competence and cooperative interaction situations, particularly cooperative goal adoption to define why, when, and how communicate to learn.

4.1. Learning at level 2: Why, When, and How.

The purpose of our agents while learning, is to update the situation formula (context) expressing when a plan is executable, if the execution of such a plan turns out to fail under the current context. If an agent can update this context after its own experience, no communication is needed. Otherwise, the agent will try to learn from the experience of other agents in the context of a cooperative interaction situation. Approaching the distributed learning problem in this way, introduce an interaction, where agents have compatible goals and competences, but insufficient resources, i.e., training examples. This interaction situation involves a kind of cooperative goal adoption [5], where the BDI agents in the MAS that have the same plan, cooperate with the goal of the learner agent, because they are co-interested in the results of the learning process, e.g., inducing the context of the shared plan. The group of agents interested in a learning goal, may be seen as a social structure, as it is the case for the dependence network, emerging from the goals.

Now, consider why should a BDI agent learning isolated (level 1), try to learn at level 2. There are two situations under which an agent should consider to communicate while learning. 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 for training examples to other agents in the MAS. The agent is no able to find out an hypothesis to explain the failure of the plan in question. It 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 also 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. For instance, 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 homogeneuos, 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). Since BDI learning agents face only horizontal fragmentation, the exchange of training examples is enough to achieve learning at level 2. In what follows, it is shown how three agents learn in the Charniak scenario (Fig. 1).

4.2. BDI architecture and agents at level 2

First, it is needed to introduce the concept of competence in the BDI architecture, usually defined in terms of the events an agent can deal with. To keep compatibility with other BDI architectures, our agents store this formulation of competence in the slot `goal-domain`, updated every time a plan is added to the plan library of the agents. A group of agents is said to have the same competence for a given event, if they share the same plan to deal with the event. It is also possible to test if a goal is in the competence of an agent, etc. Observe that we decided to define competence in terms of plans, because the event they deal with, is included in the plan definition.

What is relevant here is that once competence is defined, two ways of sending messages asking for help in a learning process are possible: i) The learner agent broadcast its message including the `id` of the plan it is trying to update, then the other agents in the MAS accept and process the message, if and only if the plan in question is on their competences; and ii) Competence is used to build a directory for

each agent, associating with each plan in the agent library, the id of other agents in the MAS that share 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.

We adopted the second option, implementing a function called mas-competence. Once all agents in the MAS have been defined, this function built for each agent, a directory associating to each plan in their library, the agents that share the plan. For example:

```
> (agent-mas-competence r1)
((P001 (R2 R3)) (P002 (R2 R3))
 (P003 (R2 R3)) (P009 NIL)
 ...)
```

In this example, when agent r1 is learning about plan p003, it believes that agents r2 and r3 have the same plan. By the contrary, plan p009 is not shared with any agent. 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, while continue learning individually.

Every BDI learning agent has a plan to ask for examples:

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

The trigger of this plan is (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, with 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
'(achieve (p-help ?ag ?plan))
'true
'((*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 learner agent, the examples found. 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, before executing *answer-help, 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
'(achieve (p-add-examples ?ag ?examples))
'true
'((*include-examples ?ag ?examples)))
```

This plan is always executed, *include-examples simply push the ?examples in the logfile of the agent ?ag.

Figure 3. MAS learning scenario. 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 competences, i.e., the same plan p007. These agents send back (solid arrows) their examples about p007. Once agent r2 has enough examples it can start learning hypothesis h.

In this way, the models shown in table 1 were obtained by agent r2 in a MAS including also agents r1 and r3 (Fig. 3) Agent r2 is the learner agent, trying to induce a new context for p007 to sand and object (see Fig. 1), since the original context, e.g., having one hand free, and the object being free, lead the agent r2 to form an intention on p007, and to fail later while executing it.

Once the learner agent r2 has a number of examples greater than a threshold (5 in the example) the agent executes a modified non-interactive version of ACE, and sug-

```
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 1. Training examples as models in ACE, from an agent learning at a level 2 MAS.

gests the user to watch the file `p007.out`, containing the result of the learning process, to accordingly, modify the definition of the plan. Output for our example is:

```
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:  [succes] [1.0/1.0]
```

This output suggests that `p007` will be executed with `success` if `plan_context(Agent,p007)` $\wedge$ `not p_state(Agent,painted)`, i.e., when the agent has one hand free, the object is free, and the agent is not painted. Here the user plays the role of a supervisor, but automatic updating of the context is also possible (the tree is transformed to a set of rules, and the new context is the disjunction of the bodies for the rules with head success). 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 6 models and the time of induction was 0.01 seconds, running on a Linux RedHat 8.0 Pentium 4, at 1.6 GHz.

This learning process is similar to one executed by an isolated agent at level 1, but since the first execution by the agent `r2` of plan `p007` lead to a failure (model 1), it would not be able to collect success training examples for this plan (plans that failed to process an event are not reconsidered again in the original BDI architecture). 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, without it ACE is not able to induce this tree, and this means that no agent would be able to learn at level 1, about `p007`.

Also, observe that since BDI agents collect the training examples after their own experience, these examples are harder to obtain than in classic machine learning, where they are previously collected; or in reinforcement learning, where agents explore in a natural way the space of hypothesis while acting. The use of ILP methods helps the agent since fewer examples may be needed to learn, provided that the background theory is relevant. Situating the learning agent in a MAS at level 2, may help it to collect training examples faster, taking benefit of the experience of other agents.

The results of a leaning process are shared by the agents in the MAS because of the way they are defined in the BDI architecture. If the user or the agent modify the plan definition accordingly to the decision tree found, this change affects automatically all agents having this plan in its library.

5. Conclusions and future work

We have shown how ILP methods, particularly the induction of logical decision trees, can be used to extend a BDI architecture with learning skills for the agents implemented with. These skills were considered as being part of the practical rationality behind the behavior of BDI agents. The result is a BDI learning agent architecture implemented on Lisp, that includes three extra BDI features: i) Some MAS simulation facilities; and ii) An interface to DTP theorem prover; iii) Learning competences. The example 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.

Future work includes: i) Implementing more MAS facilities for the architecture, e.g., including an ACL; ii) Designing protocols for sharing information of the learning set in more complex situations, e.g., agents having the same competences, but different plans; and iii) Consider the relationship between learning and the multi modal logic theories of intentional agency, e.g., the learning process described here, maintains the strong-realism conditions, do it for other forms or realism? iv) At the University of Veracruz, this work is being applied to to control groups of Khepera robots, studying the performance of the architecture in real world environments.

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

[4] 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 on Artificial Intelligence, Vol. 1866, 60–77. Springer Verlag, Heidelberg, Germany (2000)

[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. Ph.D. Thesis. Stanford University, Stanford, CA., USA (1995)

[9] 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, Proceedings of the 2nd Asia-Pacific Conference on IAT, World Scientific, p. 160-169, Maebashi, Japan (2001)

[10] Guerra Hernández, A., El Fallah-Seghrouchni, A., Soldano, H.: Learning on BDI Multiagent Systems. In: CLIMA IV Computational Logics on Multiagent Systems. Fort Lauderdale, FL., USA (2004)

[11] 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

[12] Lightfoot, D., Formal Specification Using Z. The Macmillan Press LTD, Macmillan Computer Science Series, London, UK (1991)

[13] Metral, M.: A generic learning interface architecture. Massachusetts Institute of Technology. Master's thesis. Cambridge, MA., USA (1992)

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

[15] 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)

[16] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA., USA (1993)

[17] Rao A.S., Georgeff, M.P.: Decision procedures of BDI logics. Journal of Logic and Computation, 8(3):293–344 (1998)

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

[19] 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)

[20] Wooldridge, M.: Reasoning about Rational Agents. MIT Press, Cambridge MA., USA (2000)