Learning by Knowledge Exchange in Logical Agents

Stefania Costantini
Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
Email: stefcost@di.univaq.it

Arianna Tocchio
Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
Email: tocchio@di.univaq.it

***Abstract*— In this paper we introduce a form of cooperation among agents based on exchanging sets of rules. In principle, the approach extends to agent societies a feature which is proper of human societies, i.e., the cultural transmission of abilities. However, acquiring knowledge from untrustworthy agents should be avoided, and the new knowledge should be evaluated according to its usefulness. After discussing the general principles of our approach, we present a prototypical implementation.**

I. Introduction

Adaptive autonomous agents are capable of adapting their behavior according to changes in the environment. Then, adaptive agents must take profit of past experiences using some learning approach. As it is widely acknowledged, the effects of learning should include at least one of the following:

- The range of behaviors is expanded: the agent can do more.
- The accuracy on tasks is improved: the agent can do things better.
- The speed is improved: the agent can do things faster.

According to [16] [2], three learning techniques are usable to develop adaptive autonomous agents: reinforcement learning, models learning and classifier systems.

In reinforcement learning, the mechanism consists in assigning rewards (weights) to actions that contribute to the resolution of a problem. This approach has been used for instance to improve coordination between autonomous agents [17]. In models learning, agents try to find causal relations between their actions and the events occurring in the environment. In general, they use either probabilistic models or logical models. The original ideas by [16] have then been widely developed in various approaches. Induction (and also inductive logic programming) is often considered as a particular form of models learning. Classification is the most common form of automatic learning. In the agent context, an agent can try to classify applicable rules by setting priorities and then updating these priorities according to the results achieved [12]. Memory-based reasoning (MBR) is based on the idea that if a given action took place in the past in a given situation s and gave good results, it will be useful in a new situation s' similar to s. Incremental learning techniques have been recently introduced for improving MBR in dynamic applications where data arrive continuously [9].

In a multi-agent setting however, other forms of learning can be introduced that, though related to the classical ones, are specifically tailored to multi-agent systems (MAS) topics and issues. For instance in [10], in order to recall practical solutions to coordination problems, agents learn coordinated procedures from execution traces and store them into a case-base that is organized around expectations about other agents. Agents also learn better estimates for how likely individual actions are to succeed in order to improve the quality of decisions when planning, communicating, and adapting plans.

In this paper we discuss a learning approach useful to improve adaptive behavior in computational logic agents. We assume that whatever the formalism, these agents have a rule-based knowledge base. The approach is centered on the possibility of exchanging sets of rules between agents. These sets of rules can either define a procedure, or constitute a module for coping with some sort of situation, or be just a segment of a knowledge base. However, agents should then be able to evaluate how useful the new knowledge is. To this extent, we propose two techniques.

The first technique associates to the acquired knowledge a specific objective, meaning that the new rules should help the agent to reach that objective. After a while, the agent will evaluate whether (or to which extent) the objective has been reached. If the evaluation is unsatisfactory, the new knowledge can be discarded. There is a clear similarity with reinforcement learning, where here the action that is to be evaluated is the use of the new knowledge.

The second technique consists in acquiring the same knowledge from several other agents, and then comparing the results. The comparison is made based on a meta-specification, i.e., based for instance on efficiency, or on a measure of similarity of the results. The comparison will state which versions pass a given threshold, and which don't. The unsatisfactory ones will be discarded.

In a real application, a directory agent can be employed so as to inform agents of where to find the required knowledge. This directory agent may in principle be notified of the updates of the level of trust performed by agents that have

acquired a piece of knowledge from a certain agent, and thus compute and exhibit a value that represent the *reputation* of that agent. To this aim, the directory agent will employ suitable algorithms (for instance those presented in [19]) to assess the past behavior of agents, so as to allow avoidance of untrustworthy agents in future.

In Section 2 we discuss the proposed approach at some length. In Section 4 we present a prototypical implementation of the approach in the agent-oriented programming language DALI [3] [6], after shortly summarizing the main DALI features (Section 3). In DALI, all the above can take profit from the DALI communication architecture, that allows the agent to filter incoming and out-coming messages according to any kind of constraint, including trust [5]. Then for instance, new knowledge will be learned by trusted agents only; successful evaluation of the acquired knowledge can lead to an increase of the level of trust of the sending agent, while a decision to discard that knowledge can also result in a decrease of the level of trust. Moreover, the trial of different version of the same knowledge can be made in parallel, by exploiting the DALI children generation capability [7] that allows the agent to create sub-agents on specific tasks.

II. Learning by rule exchange

Learning may allow agents to survive and reach their goals in environments where a static knowledge is insufficient. The environmental context changes, cooperative or competitive agents can appear or disappear, ask for information, require resources, propose unknown goals and actions. Then, agents may try to improve their potentiality by interacting with other entities so as to perform unknown or difficult tasks.

One of the key features of MAS is the ability of "subcontracting" computations to agents that may possess the ability to perform them. More generally, agents can try to achieve a goal by means of cooperative distributed problem-solving. However, on the one hand not all tasks can be delegated and on the other hand agents may need or may want to acquire new abilities to cope with unknown situations. In our view, an improvement in the effectiveness of MAS may consist in introducing a key feature of human societies, i.e., cultural transmission of abilities. Without this possibility, agents are limited under two important respects:

- they are unable to expand the set of perceptions they can recognize, elaborate and react to;
- they are unable to expand their range of expertise.

Indeed, the flexibility and thus the "intelligence" of agents will increase if they become able not only to refine but also to enlarge their own capabilities. The need of acquiring new knowledge can be recognized by an agent at least in relation to the following situations:

1) There is an objective that the agent has been unable to reach: it has been unable to relate a plan (in the KGP perspective [13]) or intention (in a BDI perspective [18]) to that objective (or desire) and it has to acquire new knowledge (beliefs). As a particular case, there is a situation the agent is unable to cope with; for instance, there is an exogenuos event that the agent does not recognize.
2) There is some kind of computation that the agent is unable to perform.

Assuming that the agent establishes that it cannot resort to cooperation to get its task performed, it can still resort to cooperation in order to try to acquire the necessary piece of knowledge from another agent. The problems involved in this issue are at least the following: how to ask for what the agent needs; how to evaluate the actual usefulness of the new knowledge; and, how this kind of acquisition can be semantically justified in a logical agent.

In this context, we make the simplifying assumption that agents speak the same language, and thus we overlook the problem of ontologies that in an actual implementation would of course arise. We also assume that, whatever the underlying formalism, agents have a rule-based knowledge base. Two feasible ways of asking other agents can be:

- Ask by keyword, assuming that other agents have a way of matching the keyword with a piece of knowledge. Some kind of pattern-matching will have to be used by an agent in order to establish whether it can answer a request.
- Ask by predicate name.

An agent that would accept to give the requested knowledge, should answer by providing, together with the piece of knowledge, some kind of "control" information that should include at least:

- A specification of the way of using that knowledge, that specifies whether the rules apply automatically, e.g., in the case of reactive rules, or if there is either a predicate or a procedure to be invoked.
- In the former case, specify the format of the external event that triggers the rules; in the latter, specifying the invocation pattern of the predicate/procedure.

The details of the above are left to the specific implementation, related to the language/formalism in which the agents are expressed. Notice that it is not required that the involved agents be based on the same inference mechanism. However, they should be somehow "compatible", i.e., a prolog-based agent might acquire an Answer-Set program [21] and then use it, assuming that it is able to invoke an Answer-Set solver. Clearly, the exchanged piece of knowledge should include all the *relevant rules*, i.e., all the rules which are needed (directly or indirectly [8]) for actually exploiting that knowledge.

At this stage, the receiver agent has to face two problems:

(a) Establish whether the new knowledge is consistent, or at least compatible, with its knowledge base. This is a topic which has long been studied in belief revision [1]. However, we assume that the new knowledge is not directly incorporated to the existing knowledge base. On the contrary, in the first stage the new knowledge

is distinct from the existing well-established knowledge base, as it must be evaluated before being accepted.

(b) Establish whether the new knowledge is actually useful to the purposes for which it has been acquired. If so, it can possibly be asserted in the knowledge base. Otherwise, it can possibly be discarded.

Then, agents should be able to evaluate how useful the new knowledge is. Similarly to reinforcement learning, techniques must be identified so as to make this evaluation feasible with reasonable efficiency. Simple techniques to cope with this problem can be the following.

1) The new knowledge had been acquired in order to reach an objective: the agent can confirm/discharge the new knowledge according to its reaching/not reaching the objective. This evaluation can be related to additional parameters, like e.g. time, amount of resources needed, quality of results.
2) The new knowledge has been acquired for performing a computation: the agent can acquire the same knowledge by several sources, and compare the results. Results which are not "sufficiently good" (given some sort of evaluation) lead to the elimination of the related piece of knowledge. The others are used (compared/combined) to produce the accepted result.

A. Semantics of Learning by Rule Exchange

The semantics of Computational Logic agent languages may in principle be expressed as outlined in [3] for the DALI language. I.e., given program P_{Ag}, the semantics is based on the following.

1) An *initialization step* where P_{Ag} is transformed into a corresponding program P_0 by means of some sort of knowledge compilation (which can be understood as a rewriting of the program in an intermediate language).
2) A sequence of evolution steps, where reception of each event is understood as a transformation of P_i into P_{i+1}, where the transformation specifies how the event affects the agent program (e.g., it is recorded).

Then, one has a Program Evolution Sequence $PE = [P_0, ..., P_n]$ and a corresponding Semantic Evolution Sequence $[M_0, ..., M_n]$ where M_i is the semantic account of P_i (in [3] M_i is the model of P_i).

This semantic account can be adapted by transforming the initialization step into a more general knowledge compilation step, to be performed:

(i) At the initialization stage, as before.
(ii) Upon reception of new knowledge.
(iii) In consequence to the decision to accept/reject the new knowledge.

III. DALI in a Nutshell

DALI [3] [6] [20] is an Active Logic Programming language designed in the line of [14] for executable specification of logical agents. The Horn-clause language is a subset of DALI, which however includes the following agent-oriented features. The reactive and proactive behavior of the DALI agent is triggered by several kinds of events: external events, internal, present and past events. All the events and actions are timestamped, so as to record when they occurred.

An external event is a particular stimulus perceived by the agent from the environment. In fact, we define the set of external events perceived by the agent from time t_1 to time t_n as a set $E = \{e_1 : t_1, ..., e_n : t_n\}$ where $E \subseteq S$, and S is the set of the external stimuli that the agent can possibly perceive.

A single external event e_i is an atom indicated with a particular postfix in order to be distinguished from other DALI language events. More precisely:

Definition 1 (External Event): An external event is syntactically indicated by postfix E and it is defined as:
$ExtEvent ::= << Atom_E >> | seq << Atom_E >>$
where an *Atom* is a predicate symbol applied to a sequence of *terms* and a *term* is either a constant or a variable or a function symbol applied in turn to a sequence of terms.

External events allow an agent to react through a particular kind of rules, reactive rules, aimed at interacting with the external environment. When an event comes into the agent from its "external world", the agent can perceive it and decide to react. The reaction is defined by a reactive rule which has in its head that external event. The special token $:>$, used instead of $:-$, indicates that reactive rules performs forward reasoning.

Definition 2 (Reactive rule): A reactive rule has the form:
$ExtEvent_E :> Body$ or
$ExtEvent_{1E}, ..., ExtEvent_{nE} :> Body$

The agent remembers to have reacted by converting the external event into a *past event* (time-stamped). Operationally, if an incoming external event is recognized, i.e., corresponds to the head of a reactive rule, it is added into a list called EV and consumed according to the arrival order, unless priorities are specified.

The internal events define a kind of "individuality" of a DALI agent, making it proactive independently of the environment, of the user and of the other agents, and allowing it to manipulate and revise its knowledge. More precisely:

Definition 3 (Internal Event): An internal event is syntactically indicated by postfix I:
$InternalEvent ::= << Atom_I >>$

The internal event mechanism implies the definition of two rules. The first one contains the conditions (knowledge, past events, procedures, etc.) that must be true so that the reaction (in the second rule) may happen:
$IntEvent :- Conditions$
$IntEvent_I :> Body$

The goal defined in the first rule is automatically attempted with a default frequency customizable by means of directives in the initialization file. Whenever it succeeds, the internal event "has happened", and the reaction (second rule) is trig-

gered as if it were an external one. A DALI agent is able to build a plan in order to reach an objective, by using internal events of a particular kind, called *planning goals*.

Actions are the agent's way of affecting the environment, possibly in reaction to either an external or internal event. An action in DALI can be also a message sent by an agent to another one.

Definition 4 (Action): An action is syntactically indicated by postfix A:
$Action ::=$
$<< Atom_A >> | message_A << Atom, Atom >>$
Actions take place in the body of rules.

If an action has preconditions, they are defined by action rules, emphasized by a new token:

Definition 5 (Action rule): An action rule has the form: $Action :< Preconditions$.

Similarly to external and internal events, actions are recorded as past actions.

Past events represent the agent's "memory", that makes it capable to perform future activities while having experience of previous events, and of its own previous conclusions. Past events are kept for a certain default amount of time, that can be modified by the user through a suitable directive in the initialization file. A past event is syntactically indicated by the postfix P.

Procedurally, DALI is based on an Extended Resolution Procedure that interleaves different activities, and can be tuned by the user via directives.

The operational semantics of DALI is based on Dialogue Games Theory [4] [20]: the DALI Interpreter is modeled as a set of cooperating players. By means of this approach one is able to prove formal properies of the language in the form of properties that the game will necessarily fulfil.

A. DALI Communication Architecture

The DALI communication architecture consists of four levels. The first and last levels implement the DALI/FIPA communication protocol and a filter on communication, i.e. a set of rules that decide whether or not receive (*told* check level) or send a message (*tell* check level). The DALI communication filter is specified by means of meta-level rules defining the distinguished predicates *tell* and *told*. Whenever a message is received, with content part *primitive(Content,Sender)* the DALI interpreter automatically looks for a corresponding *told* rule. If such a rule is found, the interpreter attempts to prove $told(Sender, primitive(Content))$. If this goal succeeds, then the message is accepted, and $primitive(Content))$ is added to the set of the external events incoming into the receiver agent. Otherwise, the message is discarded. Symmetrically, the messages that an agent sends are subjected to a check via *tell* rules. The second level includes a meta-reasoning layer, that tries to understand message contents, possibly based on ontologies and/or on forms of commonsense reasoning. The third level consists of the DALI interpreter.

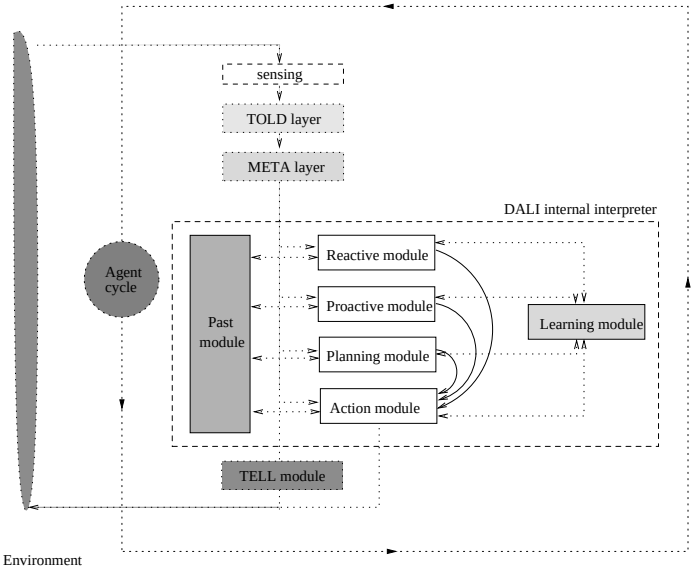

Fig. 1. DALI communication architecture

B. Children generation capability

We have introduced in the DALI framework the ability to generate children agents [7]. An important motivation for this improvement has been the need for our agents to face not-trivial planning problems by means of the invocation of a performant planner, such as for instance an Answer Set Solver. [11] [15] [21]. As a planning process can require a significant amount of time, the possibility for an agent to assign this time-expensive activity to its children can constitute a real advantage.

Another motivation for generating children is, more generally, that of splitting an agent goal into subgoals to be delegated to children. This possibly with the aim of obtaining different results by means of different strategies, and then comparing the various alternatives and choosing the best ones. The father provides the child with all the information useful to find the solution and, optionally, with an amount of time within which to resolve the assigned problem.

IV. Basic learning mechanisms

Agents that adopt forms of learning to improve their behavior can in perspective deal with more complex jobs, but expose themselves to some risks. The learned information could be either intentionally or accidentally wrong or simply not consistent with the agent specialization. Agents knowledge is generally divided into a set of facts and rules: the former represent the agent "beliefs" about itself and the world, while the latter determine the entity behavior. If learning one or more beliefs (plain facts) implies a certain degree of risk, adding rules coming from other agents to the knowledge base can

very dangerous. Thus, in our view it is necessary to elaborate different learning strategies for beliefs and rules, reserving to the latter case a more sophisticated acquisition process. In the following subsections we will first propose an approach to manage the exchange of facts and then we will discuss the more general problem of rules learning.

A. Beliefs learning

The belief base of DALI agents is composed of the facts which are present in the agent logic program, dynamically augmented by past events. Past events keep trace of what the agent has done/observed before: external and internal events, performed actions, reached internal conclusions and pursued planning activities.

Past events also represent external world knowledge. In fact, a DALI agent can ask for some facts by using the primitives $is\_a\_fact(Fact, Ag)$ and $query\_ref(Fact, Match\_number, Ag)$ where: $Fact$ is the desired information, $Match\_number$ represents the number of matches that the agent intends to receive and Ag is the name of the agent asking for the fact. While the first primitive allows the entity to require a ground fact, the second one supports the requests of non-ground information. Then, if for instance agent *dave* wants to know who is the lover of *susy*, it can send the following message to, e.g., *susy*'s friend *kate*:

$$message_A(kate, query\_ref(love(Y, susy), 1, dave)).$$

If the beliefs base of *kate* contains only the fact $love(susy, peter)$, no matching is directly found. This problem has been overcome in DALI via a meta-level support that, by using both ontologies and properties of relations, tries to "understand" message contents (namely, message contents are automatically subjected to a procedure $meta$ which is predefined though user-customizable). In this case, if the ontology of *kate* contains information on the symmetry of the predicate 'love', $loves(Y, susy)$ can be matched with $love(susy, peter)$ and the agent *kate* will return the result:

$$\begin{aligned} &send\_message\_to(dave,\\ &\quad inform(query\_ref(loves(Y, susy), 1),\\ &\quad values([loves(susy, peter)]), kate),\\ &\quad italian, []) \end{aligned}$$

Once received the desired information, the agent *dave* will update its beliefs by adding, as a past event, the fact:

$$past\_event(loves(susy, peter), 479379, kate).$$

where first value is the information about *susy* and *peter*, the second value is the acquisition time and the third value keeps track of the information source. The sender agent name is relevant: if trust in this agent reliability will be reduced under a certain threshold by negative cooperation experiences, all partial beliefs coming from it could be eliminated. At the same time, the *told filter* will get rid of at priori each communication act sent by the unreliable agent:

$$\begin{aligned} told(Sender, query\_ref(Fact, Match\_number)) :- \\ not(unreliable_P(Sender\_agent)). \end{aligned}$$

Another direct acquisition beliefs method in DALI agent is based on the *confirm* primitive. This method allows an agent to send a fact to another one. Also in this case, the fact will be added to the agent beliefs only if the message will overcome the told filter. For example, if the agent *dave* intends to send to *peter* the information $bought(car, red)$, it will send the message:

$$message_A(peter, confirm(bought(car, red), dave)).$$

and the *peter* beliefs will contain the past event: $past\_event(bought(car, red), 479379, dave)$.

A fact can be eliminated from the agent knowledge base by using the *disconfirm* primitive.

B. Rules learning

The rule-exchange approach to learning proposed in this paper is a first step into the complex world of learning rules. For instance, a DALI agent, when receiving a stimulus whose reaction is unknown, can ask other agents for acquiring rules capable of suggesting the right behavior to adopt. While retrieving and adding rules to the knowledge base is not difficult, the relevant problem of learning a correct information remains.

Intelligent agents can have different specializations for different contexts and a learning rules process cannot ignore this. Moreover, even agents having the same specializations can adopt behavioral rules which are mutually inconsistent. What could the solution be?

In the prototypical implementation that we present here, the learning rules process includes several steps starting from a verification of the source reliability. The solution is based on the introduction of a *mediator agent* that we call $yellow\_rules\_agent$, keeping track of the agents specialization and reliability. When an entity needs to learn something, it asks the $yellow\_rules\_agent$ for the names agents having the same specialization and being more reliable.

Once obtained this information, the agent may acquire the desired knowledge by some of them. If the agent will finally decide to incorporate the learned rules in its program because they work correctly, it will also send to $yellow\_rules\_agent$ a message indicating satisfaction. This will result in an increment of the reliability of the agent that has provided the rules. A negative experience will imply an unfavorable dispatch. In the present implementation agents return a numeric value indicating the "level of trust". We mean to add also the objective that the receiver agent meant to reach via the new knowledge, so as to conditionally rate agents with respect to this point. This in order to avoid a low reliability esteem for agents which are actually reliable in their own area of expertise. In fact, it may happen that an agent has a very specialized (and accurate) rule set which is mistakenly matched against some requests. In updating the level of trust the $yellow\_rules\_agent$ should

adopt a model that updates trust only when the information is sufficient, i.e., after a certain number of reports which are in accordance, sent by reliable agents.

The learned rules will be added to the agent knowledge base in the form of past events. A preliminary check will verify some properties such as, for example, the syntactic correctness or consistency. Rules that "survive" this check will be used by the agent in its activities, and their usefulness and efficiency will be recorded. After a certain time, according to results the acquired rules will be either definitely learned or eliminated. Below we describe in more detail all steps involved in our cooperative rules learning approach.

1) **New knowledge is needed.** A DALI agent behavior is described by: (i) a set of rules determining what reaction to apply in response to external stimuli; (ii) a set of rules useful to draw internal conclusions or to reach goals via planning strategies; (iii) a set of rules containing conditions for reacting external events or for performing actions; (iv) a set of horn-clauses. The need to acquire new knowledge arises whenever an agent receives a communication act whose content is unknown and the meta-level does not succeed in searching for a semantically equivalent content recognized by the entity. The communication act could be an external event, a proposed action, a request of information and so on. Having no internal means to cope with this situation, the agent activates the learning rules process. This process is risky enough, so the agent must try to search a suitable information source.
2) **Looking for information sources.** Our learning architecture allows a particular agent, $yellow\_rules\_agent$, to maintain the information useful to identify the desired rules source or sources. Each agent living in the environment is identified by the tuple:

   $source(A_i, S_i, KR_i, Q_i)$

   where the first parameter represents the agent identification and the second one is a string synthesizing the agent role in the environment. The third one is a list of rules keys that the agent A_i is willing to transfer to other agents. The fourth one is the reliability value, computed by $yellow\_rules\_agent$ according to positive and negative feedbacks. In fact, agents that receive rules from $agent_i$, at the end of the verification phase send a message to $yellow\_rules\_agent$ rating that knowledge. According to current and past values average, the $yellow\_rules\_agent$ computes $agent_i$ reliability by means of some kind of eveluation. For example, if the agent *dave* is a barman and is available to give to others the rules useful to serve a drink, the tuple *source* might be for instance: $source(dave, barman, [serve\_drink], 0.6)$.

   Whenever an agent A_k having the specialization S_k needs some rules, it will send to $yellow\_rules\_agent$ the message:

   $message_A(yellow\_rules\_agent, search\_sources(A_k, S_k, Key_k))$

   where Key_k is meant to indicate the desired rules. More precisely, this parameter allows $yellow\_rules\_agent$ to identify agents having the right information by finding a correspondence between Key_k and the elements of the rules keys list KR_i.
   If one or more agents fulfill the correspondence, the agent A_k will receive as a response the list of reliable agents corresponding to the expected specialization S_k and the key Key_k:

   $L_s = [(A_1, Q_1), ..., (A_n, Q_n)]$

   If no agents are available, the response will be the empty list.
   Having chosen the names of one or more agents to which one can ask missing rules according to the $yellow\_rules\_agent$ and personal reliability evaluation, the agent will then contact them.

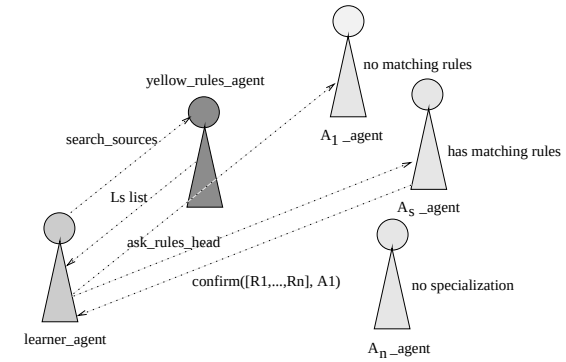

Fig. 2. A cooperative learning scenario

3) **Asking for missing rules.** In order to get the needed piece of knowledge, the agent can choose one of two techniques: the first one allows an agent to learn all required rules by specifying their heads. This implies that there be a strict correspondence between the heads of rules in two or more agents in order to be able to activate the learning process. But, agents often came from different platforms and technologies, so this correspondence could hardly be found. This limit can be overcome by adopting ontologies capable of matching rule heads which though looking syntactically different are semantically equivalent. If we consider the agent A_k having selected the couple (A_s, Q_s) and the head H_l, the following message will propose to the receiver agent the exchange of rules having the head H_l:

   $message_A(A_s, ask\_rules\_head(H_l, A_k))$

   The second technique allows an agent to ask for a specific key that can match with either the head or the

body of rules in the agent program. We may notice that in this manner the probability of finding corresponding rules will be higher, though the rules appropriateness and usefulness could be more in question. In this case, the message syntax will be:

$message_A(A_s, ask\_rules\_key(Key, A_k))$

Agents accepting the proposal to exchange rules that match with either the $Head$ or the Key will pack all retrieved rules and will send them back to the A_k entity.

$message_A(A_k, sent\_rules([R_1, ..., R_n], Ch, A_s))$

The parameter Ch represents the goal that must be invoked in order to activate the rules. In particular, if the agent *bob* receives from the agent *dave* the rules:

$[dangerE :> call\_policeA,$
$call\_police :< have\_a\_phone]$

the parameter Ch will correspond to $danger_E$.

As soon as these rules will be received by the learner agent, they will be unpacked and asserted as past events in its knowledge base with the suffix $learn_P(Rule, Time, Sender, Objective)$. Each rule will be re-asserted in a second version, where more information is associated to it, and in particular: the current time; the sender agent name; a parameter *Objective*, useful to remember what was the goal for which the request had been issued. For example, if the agent was in a dangerous situation when it requested the rule with the key *help*, it will memorize that the *Objective* of this learning rule process was to get safe. The objective introduction will allow the agent, after learning the rules, to check their effectiveness with respect to the associated goal.

4) **Add learned rules.** Rules added as past events are managed by a specific internal event, $gest\_learning(Rule)$, that implements the first filter level. This internal event filters one rule at the time. In order not to slow down the agent, this operation is performed in suitable time slots, i.e., when the agent is not performing complex tasks and the events queues have few items to be processed. Each rule is taken into account in order to be added to the knowledge base, and must fulfill two conditions (expressed in the first rule of the internal event): $learn\_if(Rule, Time, Sender)$ and $properties\_true(Rule)$. The first condition, $learn\_if(Rules, Time, Sender)$, is similar to a *told* one: the user can define in the same file of tell/told rules a set of constraints that the considered $Rules$, the $Time$ and the $Sender$ agent must respect:

$learn\_if(Rules, Time, Sender) :-$
$\quad constraint_1, ..., constraint_n.$

Constraints can avoid adding an incoming rule from an agent that was reliable for $yellow\_rules\_agent$, but is considered instead unreliable by the receiver agent under some different perspective. $Time$ can be used to either force or delay the assertion of the $Rules$. Other domain- or situation-dependent constraints can be expressed.

$gest\_learning(Rule) :-$
$\quad learn_P(Rule, Time, Sender,_)$,
$\quad learn\_if(Rule, Time, Sender)$,
$\quad properties\_true(Rule)$.
$gest\_learning_I(Rule) :>$
$\quad accept\_at\_present_A(Rule)$.

Fig. 3. First learning process filter

The second condition, $properties\_true(Rule)$, takes more specific properties of the $Rules$ into account, e.g.:

- the syntactic correctness according to prolog and DALI language;
- the absence of procedure calls without a corresponding procedure;
- the overlap of rules originating from different agents;
- the rule consistence with respect to previously learned clauses.

If certified by the internal event, the rules are added to the agent program with a label indicating that they are to be submitted to second filter. A particular label will emphasize that the rules have been learned provisionally. Final learning will take place only if the rules will overcome the second filter level, based on usefulness and efficiency. Some rules discarded by the first filter can remain for some time in the agent knowledge base, waiting for a successive integration. In fact, the learning process can generate new contexts where some previously false properties become true.
In order to avoid a rule that cannot be learned to be kept for too long in the agent memory, we have introduced a particular internal event that eliminates all past events $learn_P(Rule, Time, Sender, Objective)$ that has been kept for an amount of time that exceeds a threshold.

C. Exploiting basic mechanisms

Rules added to the agent program in order to be evaluated wait for the moment in which the agent will be in need of them. The estimate of their usefulness depends strictly on the kind of learned rules. Some, expressing a set of actions that the agent needs to perform, can be evaluated by examining the correspondence between the entity objective and the exhibited behavior. Some, useful to execute complex operations, can be evaluated by examining for instance the time spent in the calculation and the result quality. Here we propose two sample methods to estimate partially learned rules.

- **On Objectives** Once introduced in the agent program, each piece of knowledge is used by the entity during its life, keeping always track of its performance with respect to the corresponding objective. This testing phase can be performed in two modalities. The first one is based on the correspondence between the expressed *Objective* in $learn_P(Rule, Time, Sender, Objective)$ and the effective rule application result. For example, if *Objective*

$check\_objective(Rules, Value) : -$
$\quad learn_P(Rules, Time, Sender, Objective),$
$\quad expired\_time(Time, T),$
$\quad evaluate(Rules, Objective, Value).$
$check\_objective_I(Rules, Value) :>$
$\quad evaluate\_rule(Rules, Value).$
$evaluate\_rule(Rules, Value) : -$
$\quad accept\_definitely_A(Rules, Value).$
$accept\_definitely\_rule(Rules, Value) :<$
$\quad Value > Threshold.$
$evallreject\_rule_A(Rules, Value).$
$reject\_rule_A(Rules, Value) :<$
$\quad Value < Threshold.$

Fig. 4. Second learning process filter

expresses the agent safety, we would expect that the agent be safe or at least having made some progress in this direction. The utility and efficiency test is implemented by an internal event that, whenever a learned rule is invoked, checks from time to time its effect.
The evaluation is performed by the function $evaluate(Rules, Objective, Value)$ that considers:

- the degree of correspondence between the *Objective* and past events generated by the *Rules* application;
- given the state snapshot of the program execution, the degree of correspondence between the saved state and the declared *Objective*.

For each usage of $Rules$, the returned $Value$ increments/decrements the average calculated on the past evaluations. After some time, a negative result implies the rule elimination while a positive one determines a final learning. However, the agent maintains information on the *Rules* sources, so that also in future what is learned can be eliminated if, for example, the source becomes unreliable.

- **On comparison** If the learned rules are aimed at some kind of complex computation that returns a result, a suitable testing method can be adopted. The agent can generate children, and can assign each child a different set of rules acquired by different sources for the same calculation. The results, together with performance, time and resources spent, will be returned to the father that can decide which set of rules is better to adopt.

V. Conclusions

We have proposed a form of cooperation among agents that consists in improving each agent's skills by acquiring new knowledge form the others. The approach aims at extending to agent societies a feature which is proper of human societies, i.e., the cultural transmission of abilities. We have outlined the problems and advantages of this approach, and have discussed a prototype implementation in DALI. More experimental work is needed for proving the effectiveness of the approach, and for putting various methods of verification of the usefulness of learning at work. Indeed, the mechanisms for matching needs against rule sets of other agents (keywords or rule heads) is quite preliminary and must be checked in real applications, as it might result in low "precision", i.e., too many matches are found and "recall", i.e., too many matches are discarded.

References

[1] G. Antoniou (with contributions by M.-A. Williams). *Nonmonotonic Reasoning*, The MIT Press, Cambridge, Massachusetts, 1997, ISBN 0-262-01157-3.

[2] W. Brenner, R. Zarnekow and H. Wittig. *Intelligent Software Agents, Foundations and Applications*, Springer Verlag, Berlin, Germany, 1998.

[3] S. Costantini and A. Tocchio. *A Logic Programming Language for Multi-agent Systems*, In: Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424, Springer-Verlag, 2002.

[4] S. Costantini, A. Tocchio and A. Verticchio. *A Game-Theoretic Operational Semantics for the DALI Communication Architecture*, In: Proc. of WOA04, Pitagora Editrice Bologna, ISBN: 88-371-1533-4, Also available on-line, at the URL: http://woa04.unito.it/Pages/atti.html

[5] S. Costantini, A. Tocchio and A. Verticchio. *Communication and Trust in the DALI Logic Programming Agent-Oriented Language*, In: Proc. of the Italian Conference on Intelligent Systems AI*IA'04, 2004.

[6] S. Costantini and A. Tocchio. *The DALI Logic Programming Agent-Oriented Language*, In: Proceedings of the 9th European Conference, Jelia 2004, Lisbon, September 2004. LNAI 3229, Springer-Verlag, Germany, 2004.

[7] S. Costantini and A. Tocchio. *Enhancing Computational power: DALI child agents generation*, In: Electronic proceedings of CILC'05, Italian Conf. on Comp. Logic, Roma, 21-22 giugno 2005, URL http://www.disp.uniroma2.it/CILC2005/Programma.html.

[8] J. Dix. *A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties*, Fundamenta Informaticae 22(3), 1995.

[9] F. Enembreck and J.-P. Barths. *ELA - A new Approach for Learning Agents*, Journal of Autonomous Agents and Multi-Agent Systems, 3(10): 215-248, 2005.

[10] A. Garland and R. Alterman. *Autonomous Agents that Learn to Better Coordinate*, J. Autonoumous Agents and Multi-agent Systems, 2004.

[11] M. Gelfond and V. Lifschitz. *The Stable Model Semantics for Logic Programming*, In: Proc. of the Fifth Joint International Conference and Symposium. The MIT Press, 1988, 1070–1080.

[12] J. H. Holland. *Escaping brittleness: The possibility of general-purpose learning algorithms applied to parallel rule-based systems*, In: Machine Learning, an Artificial Intelligence Approach, Morgan-Kauffman, vol. 2, 1986.

[13] A. Kakas, P. Mancarella, K. Stathis, F. Sadri and F. Toni. *The KGP Model of Agency*, In: ECAI 04, Proc. of the 16th European Conf. on Artificial Intelligence, 2004.

[14] R. A. Kowalski. *How to be Artificially Intelligent - the Logical Way*, Draft, revised February 2004, Available on line, URL http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html.

[15] V. Lifschitz. *Answer Set Programming and Plan Generation.* Artif. Intelligence 138 (1-2), Elsevier Science Publishers, 2002, 39-54.

[16] P. Maes. *Modeling Adaptive Autonomuos Agents*, Artificial Life Journal, 1(1-2): 135-162, MIT Press, 1994.

[17] Oliveira E. *Agent, advanced features for negotiation and coordination*, In M. Luck (ed.), ACAI 2001, LNAI 2086: 173–186, Speringer-Verlag, 2001.

[18] A. S. Rao and M. Georgeff. *Modeling rational agents within a BDI-architecture*, In: Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91). Morgan Kaufmann, 1991: 473-484.

[19] J. Sabater and C. Sierra. *Reputation and social network analysis in multi-agent systems*, In: Proceedings of the First Int. Joint Conf. on Autonomous Agents and Multi-agent Systems: 475482. ACM Press, 2002.

[20] A. Tocchio. *Multi-Agent Systems in computational logic*, Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila, 2005.

[21] Material about Answer Set Programming (ASP) and web location of ASP solvers. http://tinfpc2.vub.ac.be/wasp/bin/view/Wasp/WebHome.