Published in the *Proceedings of the ECAI'96 Workshop on Learning in Distributed AI Systems*

Cooperation Modes among Case-Based Reasoning Agents

Enric Plaza, Josep Lluís Arcos, and Francisco Martín
IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
Vox: +34-3-5809570, Fax: +34-3-5809661
{enric,arcos,martin}@iiia.csic.es
http://www.iiia.csic.es

Abstract

We are investigating possible modes of cooperation among homogeneous agents with learning capabilities. In this paper we will be focused on agents that learn and solve problems using Case-based Reasoning (CBR), and we will present two modes of cooperation among them: Distributed Case-based Reasoning (DistCBR) and Collective Case-based Reasoning (ColCBR). We illustrate these modes with an application where different CBR agents able to recommend chromatography techniques for protein purification cooperate. The approach taken is to extend Noos, the representation language being used by the CBR agents, to allow communication and mobile methods.

1 Introduction

We are investigating possible modes of cooperation among homogeneous agents with learning capabilities. Specifically, in this paper we are interested in a cooperative setting that assumes coordination among agents fulfilling the following conditions:

Homogeneous Agents The representation languages of the involved agents are the same. Consequently, communication among agents do not require a translation phase.

Peer Agents The involved agents are capable of solving the task at hand. In other words, cooperating agents are not merely specialists at specific subtasks. Instead, they are capable to solve the overall task by themselves (most of time, at least). This condition implies a peer to peer communication form.

Learning Agents The agents solve the task based knowledge acquired by learning from their individual, usually divergent, experience in solving problems and cooperating with other agents in solving problems.

We will call these conditions of agent cooperation a *federated peer learning* (FPL) framework. The FPL framework define a class of cooperative settings where learning can prove to have a clear leverage. In fact, we are focusing on the issue of how learning agents, that may have either the same method or several different methods for solving a given task and that moreover may can achieve a cooperative problem solving behavior that improves the individual behavior. The problem solving behavior of the agents will be biased by their individual learning based on their separate experience—since different sets of problems will actually occur in different locations. Consequently, even agents in principle similar can diverge as result of the individual learning

experience, and cooperation may profit from these biasing by improving the overall performance of the involved agents.

In the FPL framework, we will focus in this paper on two modes of cooperation among case-based reasoning (CBR) agents. A CBR agent uses a form of *lazy learning* where past experiences are "generalized" (so to speak) by means of a similarity estimate between the current problem C and the precedent cases CB solved by the agent. The similarity-based reasoning (or analogical reasoning) involved follows the basic heuristic stating that *the more similar a case C is to a precedent $P \in CB$ the more similar the solution of C is to the solution of P*. While in eager forms of learning—like inductive techniques—the general descriptions for classes of solutions are built beforehand, lazy learning works in a on-demand, case-by-case basis. Learning in CBR can be seen as enlarging by means of a similarity estimate—thus, generalizing—a precedent case P until it includes the current case C [7]. We will show that the lazy nature of learning in CBR is very amenable to take advantage of cooperation.

The approach taken to communicate CBR systems is to extend Noos, a representation language developed at our Institute for integrating learning and problem solving that has been used to build several CBR systems [4]. The extension of Noos, *Plural* Noos, allows communication and mobile (or "migrating") methods among agents that use Noos as representation language. In particular, we will show two modes of cooperation among CBR agents: Distributed Case-based Reasoning (DistCBR) and Collective Case-based Reasoning (ColCBR). Intuitively, in DistCBR cooperation mode an agent A_i *delegates its authority* to another peer agent A_j to solve a problem —for instance when A_i is unable to solve it adequately. In contrast, ColCBR cooperation mode *maintains the authority* of the originating agent: an agent A_i can transmit a mobile method to another agent A_j to be executed there. That is to say, A_i *uses the experience* accumulated by other peer agents while maintaining the control on *how* the problem is solved.

Before explaining both DistCBR and ColCBR modes of cooperation in more detail, we will first introduce the task domain in which we are working.

1.1 The Task of Protein Purification

We have developed CHROMA, system implemented in Noos that recommends chromatography techniques to purify proteins from tissues and cultures [5]. CHROMA includes two learning methods (a case-based method and an inductive method) and two problem solving methods (a CBR method and a classification method that uses the induced knowledge). Moreover, a metalevel method is able to prefer, for a particular problem, which problem solving method is more likely to succeed. Currently, we are simplifying the system for the cooperative CBR experiments and we will assume that CBR agents for protein purification will only embody one CBR method (see last section for future work on the more complex situation).

Why choose this task domain? The protein purification task is amenable to cooperative solutions since there are thousands of proteins and chromatography techniques are in current use in hundreds of industrial chemical labs that have their own bias as to the kinds of problems they regularly solve and the problems they seldom attack—but that can be regularly solved at another location. Moreover, different locations may have different methods for case-based reasoning that rely on a knowledge modelling analysis of their particular problems and their local expertise and biases.

The structure of the paper is as follows: first the Noos representation language is introduced and then the *Plural* Noos extension is summarized. Next, Distributed Case-based Reasoning (DistCBR) and Collective Case-based Reasoning (ColCBR) are discussed and their support by *Plural* Noos is explained. Finally, some discussion about the generality of the approach and future work closes the paper.

2 Representation and Communication

The approach taken to develop cooperative CBR is to extend Noos, a representation language for integrating learning and problem solving that has been used to develop several CBR systems. In this section we first present some basic notions of the language, and later the *Plural* extension that supports communication and cooperation among CBR agents using Noos.

2.1 The Noos Representation Language

Noos is a reflective object-centered representation language designed to support knowledge modelling of problem solving and learning [3, 4]. Noos is based on the task/method decomposition principle and the analysis of knowledge requirements for methods —and it is related to knowledge modelling frameworks like KADS [12] or components of expertise [11][1]. A *method* models a way to solve a task. A method can be elementary or can be decomposed in subtasks. These new (sub)tasks can be achieved by corresponding methods in the same way. For a given task there may be multiple alternative methods (alternative ways to solve the task). For instance, a `CBR` method[1] is decomposed into the `retrieve`, `select` and `reuse` subtasks and there are several possible methods to achieve each subtask. Decision-taking in Noos is modelled by a preference language that allows the specification of the conditions in which an alternative is better than others. Reasoning about preferences permits an agent to select a method from a set of alternatives or to choose to cooperate with an agent from a set of associate agents —as will be shown later.

The integration of learning and problem solving methods in Noos has two aspects. First, whenever some knowledge required by a problem solving method is not directly available there is an opportunity for learning. Secondly, learning methods are methods with introspection capabilities that can be analyzed also by means of a task/method decomposition. The basis for integrating learning methods is the *episodic memory*. The episodic memory stores the decisions taken during the inference—like successful methods engaged to tasks, results obtained by achieved tasks, and methods that have failed to achieve tasks. Noos provides two ways to perform introspection: using metalevel methods or using a set of retrieval methods provided by the language. Retrieval methods allow Noos to inspect and analyse previous specific situations in the episodic memory. For instance, case-based reasoning methods require to access stored cases, select one of them according to some criteria, and finally reuse the solution. The reuse task reinstantiates the solution to the current problem or constructs a new solution according the precedent solution and the current problem[2].

An example of a CBR method used by CHROMA is the `analogy-by-determination` method. This method has a `retrieve` subtask with a `retrieve-by-determination` method that uses `protein` as determination[10]. This method retrieves from the episodic memory the solved experiments that satisfy the determination—purifying the same protein as the current experiment. The next subtask selects the most relevant precedent case according to domain knowledge criteria—like the kind of sample from which the protein is purified from. Finally, the last subtask `reuse` reinstantiates the purification plan of the most relevant precedent to the current problem. The knowledge required in this domain includes knowledge about proteins, chromatography techniques and purification plans.

Noos is a representation language based on *descriptions*. A description is formed by a collection of *features*. The values of features are constants or other descriptions. This approach is close to the *ψ-term* formalism [2]. Domain knowledge is represented in Noos by descriptions of the *concepts* in that domain. Descriptions have a correspondence to labeled graphs representation as shown in the description of an experiment in the chromatography domain of Figure 1.

Methods are also represented as descriptions. The features of a method description represent the *subtasks* in which that method is decomposed. Methods are defined by refinement from a set of

[1] For related approaches see the Knowledge Engineering Methods and Languages web page at <URL:ftp://swi.psy.uva.nl/pub/keml/keml.html>

[2] In this paper we are focusing only in CBR learning methods—other learning methods like inductive methods [5] and analytical methods have also been integrated in this way.

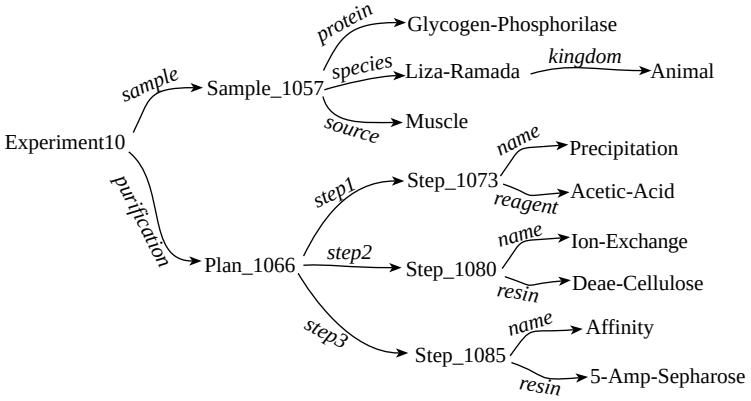

Figure 1: A case description in CHROMA.

built-in methods. The set of built-in methods in Noos are those of a general-purpose language plus some constructs enabling introspection. The uniform representation of methods as descriptions is what allows *Plural* Noos to transmit over the network both entity descriptions and methods in the same way.

Inference in Noos is engaged by queries. For instance, solving the chromatography problem `experiment10` is engaged by querying the feature `purification` as follows: `(>> purification of experiment10)`. The purification task is solved by the corresponding method associated with the `purification` feature of the problem. In the CHROMA system this method is the `analogy-by-determination` method explained before.

2.2 CBR in Protein Purification

We will introduce the CBR method used in our example domain of protein purification. We have to remark that Noos is not a CBR shell with a built-in, fixed way of performing case-based reasoning. Noos allows the *configuration* of a CBR system after a knowledge model analysis of the domain has been performed. Such a configuration is done with the component blocks provided by Noos—like generic retrieval methods—that are refined (or biased) in order to incorporate the domain knowledge we have modelled. In CHROMA the domain knowledge is used to characterize which features are more important when judging the similarity between a current problem and a precedent case. Noos allows to express such a knowledge by means of retrieval methods and preference methods. This abstraction permits to ignore implementation details like the indexing algorithms and, most importantly, will permit the communication of such methods among CBR agents. In this way a CBR agent can profit by lazy learning not only from the cases in its own Case-Base but also those cases known by other agents.

The configuration of the specific CBR method used in CHROMA is the following.

Goal-driven Retrieval The `retrieval` method is a generic method that selects from memory all cases obeying a constraint declared as `pattern`. Intuitively, it retrieves all cases subsumed by (all cases that match) the pattern. Domain knowledge in CHROMA state that we are interested only in cases where the `protein` feature has the same value as our current problem—and the rest of cases should be dismissed as irrelevant. This form of retrieval is called goal-driven retrieval (since the protein is the goal in our process) and can be represented by a general method called `retrieve-by-determination`.

Domain Selection Criteria A second component is a `preference` method that allows to impose a partial order among retrieved cases. In CHROMA there are three basic preferences:

Preference n. 1 Domain knowledge in CHROMA state that usually the most important criterion for similarity is having the same value in the `source` feature as in the current

problem. This preference method imposes a partial order from the retrieved cases with that value to the retrieved cases that do not.

Preference n. 2 Another preference method is regarding the `species` feature—i.e. the species of the sample tissue or culture (`source`) from which the protein is purified. This preference discriminates the retrieved cases that are incomparable with preference n. 1.

Preference n. 3 The final component is also a preference regarding the `kingdom` taxon of the source, and it is applied to all retrieved cases that are not preferred among them by the preceding preference methods.

In our extension of CHROMA to distributed agents, each lab will supplement these general preferences with other specific preference criteria due to the kinds of problems they regularly solve and their local expertise. For instance, for a given tissues the specie criterion could be more relevant than the source criterion. Thus, each CBR agent will possibly contain selection criteria adapted to its own experience.

Reuse Finally, the last `reuse` method reinstantiates the purification plan of the most relevant precedent according to the previous domain preferences.

Learning in CBR is lazy: a CBR system imposes a partial order among (a relevant subset of) the past examples based on the current problem. The solution of a problem is determined by the solution of the case(s) that is maximal in the partial ordering established by preferences. Thus, solutions proposed by the system are function of the individual experience of the CBR system plus the domain knowledge given by the system designers during the knowledge modelling stage. Later in the paper we show how lazy learning plus method configuration can be used to support cooperation modes that improve the performance of a collectivity of CBR agents.

2.3 Agent Communication with *Plural* Noos

Plural provides a seamless extension of Noos that supports a distributed scoping and reference for all the basic constructs in Noos. An *agent* in *Plural* is a particular Noos application with a known network address, and the *acquaintances* of an agent are those agents with known address—as in the actors model. The CBR cooperation modes will use three *Plural* Noos capabilities: network references, remote evaluation, and mobile methods. Using *Plural* Noos, arbitrary Noos descriptions (of entities and methods) can be transmitted over the network from one agent to another. In particular, cases and CBR methods can be transmitted from a CBR agent to another.

Network references extend Noos references to agents over the net. For instance, a reference like `(>> feature1 of problem2)` is extended to a network reference on a specific agent as `(>> feature1 of problem2 at agent3)`. Syntactically, a reference to a feature in an `agent-i` like `(>> feature of entity)` once transmitted to a new agent `agent-j` becomes a network reference equivalent to `(>> feature of entity at agent-i)` in `agent-j`; and an identifier of an entity in `agent-i` like `entity55` once transmitted to a new agent becomes a network reference like `entity55@agent-i` in `agent-j`. Network references are transmitted over the network: if `entity55@agent-i` is a value of the feature `my-friend` of `entity99` in `agent-j` and a new agent has the reference `(>> my-friend of entity99 at agent-j)` it will get the original network reference `entity55@agent-i`.

Remote evaluation allows an agent to use a method owned by another agent —as in remote procedure call (RPC). Specifically, remote evaluation allows an agent `agent-i` to ask an `agent-j` to execute a specific method `method-k@agent-j` for a given `problem-n` of `agent-i`, as in the expression `(noos-eval (method-k@agent-j problem-n) at agent-j)`. In this process, `agent-j` receives the network reference `problem-n@agent-i` and applies `method-k` to it. During evaluation, further references in `agent-j` of features of `problem-n@agent-i` are interpreted as network references and automatically engage a transmission to `agent-i` asking the value for that feature. `Agent-i` is responsible for inferring that value and transmit it as answer to `agent-j` (in fact a network reference to that value).

Network references avoid the problem of maintaining state when objects with state are copied over the network. State is local to agents, and when a description is referenced by another agent, a network reference is transmitted[3]. In essence, if we view a Noos description as a labeled graph, transmission of a description starts at the graph root and "copies" the graph in the destination in the following way: if a graph node is constant (like a number or a string) a fresh copy is produced, otherwise a network reference to that node is created. Since Noos performs lazy evaluation, not all the nodes in a graph are transmitted when the root is referenced, but only those needed by remote references. Path equality (sharing) and circularities in the graph are preserved.

Remote evaluation requires the remote agent to own a particular method that can then be used by the originating agent. For some cooperation modes it is necessary to support so-called *mobile code* or *migrating programs*. *Mobile methods* are supported by *Plural* Noos by the capability of transmitting method descriptions. A mobile method description is first defined in an originating agent `agent-i`. Then, the *Plural* Noos construct `jump` can be used to bind the mobile method with the appropriate references and to instantiate the method at the destination agent. In the following example a `mobile-method-k`—already defined in `agent-i`—jumps to `agent-j` containing references to local information (`description`) and remote information (`requirements`).

```
(jump (define (mobile-method-k)
         (description (>> description of problem-13)))
         (requirements (>> of mm-requirements at agent-j)))
    to agent-j)
```

When a method jumps to a remote agent, the whole task/method decomposition of the mobile method is "copied" in the following sense: the name of the built-in of which the method is a refinement is transmitted, as well as its subtasks. Recursively, the methods defined in the originating agent for those subtasks are also "copied". The references of the mobile method in the originating agent are transformed to network references in the way we explained before. In the example shown, reference `(>> description of problem-13)` in `description` is transformed at `agent-j` to the network reference `(>> description of problem-13@agent-i)`.

3 Modes of Cooperation for CBR Agents

Since learning is lazy in CBR systems, cooperation involves expanding the set of precedents to be used in similarity-based reasoning from the individual memory of a CBR agent to the memories of a collectivity of CBR agents. We argue that there are to general ways to do so: Distributed Case-based Reasoning (DistCBR) and Collective Case-based Reasoning (ColCBR). Intuitively, both DistCBR and ColCBR are based on solving a problem by reusing with the knowledge learned by other CBR agents. Given an agent (the *originator*) trying to solve a given problem, the difference between both modes is regarding which similarity-based reasoning method is used: that of the originator or that of the CBR agent that is helping the originator.

In other words, the difference is the following:

DistCBR is based on an agent transmitting the problem and the task to be achieved to another agent, and the CBR method used is that of the receiving agent. In this sense, the CBR process is *distributed* since every agent works using its own method of solving problems.

ColCBR is based on an agent transmitting also the method that is to be used to solve that problem to another agent (and that method will use the knowledge learnt by the receiving agent). In other terms, the originator is using the memory of the other agents as an extension of its own—as a *collective memory*—by means of being able to impose to other agents the use of the CBR method of the originator.

[3] Our approach is similar to that of the distributed object-oriented language Obliq[6] regarding the fact that network references are local to a site (here, an agent). A major difference is that *Plural* transmit *terms* over the net while Obliq transmit *closures*.

From the standpoint of implementing those cooperation modes, we can say that DistCBR is supported by the remote evaluation capability and ColCBR is supported by remote programming (or mobile code) capability of *Plural* Noos.

3.1 Distributed Case-based Reasoning

The DistCBR cooperation mode is, intuitively, a class of cooperation protocols where a CBR agent A_{orig} is able to ask to one or several other CBR agents $\{A_1 ... A_n\}$ to solve a problem on its behalf. The cooperation mode definition leaves to specific protocols designed for given task domains the specification of which criteria an agent A_{orig} uses to ask another to solve a problem, how to choose which agents to ask and in which order. ColCBR is based on the *Plural* Noos capability of *remote evaluation*. A specific protocol for the protein purification task is given below. This is not a shortcoming or underspecification of our framework: since these issues and decisions are domain-dependent they are to be established by a knowledge modelling analysis of the task domain that later implemented by Noos methods. The only difference is that these *Plural* Noos methods will have references to —and will engage communication with— other agents.

DistCBR involves two main cooperation tasks: a) A_{orig} sends the (identification of the) current case C_{curr} to an agent A_j, and b) asking A_j to solve the purification task on the case C_{curr}. As result, agent A_{orig} receives a solution inferred by A_j based on its own $CBR-method_j$ and its case-base CB_j —or a failure token. Upon a failure of the agent A_j, A_{orig} can iterate the cooperation tasks with the next agent of its preference.

An agent in DistCBR CHROMA has a set of acquaintances $\{A_1 ... A_n\}$ that are agents having at least a CBR method for solving protein purification problems and a case-base of such problems already solved. A_{orig} can prefer to ask first to an agent A_i that has previously solved for it a problem regarding the same protein (goal preference)[4]. In general, each CBR agent may have a different protocol for deciding which agent to ask to solve the current problem. In order to start a DistCBR cooperation, the originating agent only needs to know the name (identifier) of the CBR method used by each acquaintance for the task `purification`—by convention we will assume all agents use the same public name `protein-purif-method`[5]. The cooperation tasks of DistCBR are achieved in the chromatography domain by the following process:

```
(noos-eval (protein-purif-method@agent-j case-33) at agent-j)
```

This syntax is equivalent to the following process:

1. transmit the identifier of the originating agent `agent-i` and the identifier of the current case `case-33` to `agent-j`,
2. then ask `agent-j` to perform the process:
   `(noos-eval (protein-purif-method case-33@agent-i))`
3. finally transmit back to `agent-i` the result.

This process can be iterated on other acquaintances until a solution can be obtained for an agent that has an appropriate case precedent for the current problem.

The *Plural* Noos extension of Noos can be seen as performing this process: communication, task translation and then transmission back of the results.

3.2 Collective Case-based Reasoning

The ColCBR cooperation mode is, intuitively, a class of cooperation protocols where a CBR agent A_i is able to send a specific CBR method $CBR-method_i$ of its choosing to one or several CBR

[4] This is the same preference that the stand-alone CHROMA system applies in the retrieval task (prefer a case with the same protein as the current problem).

[5] These method names can be easily acquired asking the acquaintances `(>> method of (task purification of purification-problem at agent-j))` but we have no room for the discussion here.

agents $\{A_1 ... A_n\}$ that are capable of using that method with their case-base to solve the task at hand. ColCBR is based on the *Plural* Noos capability of *mobile methods*: an originating agent A_i can define a method $CBR - method_i$, bind it to the current problem C_{curr}, and migrate it to another agent A_j that has previously solved for it a problem regarding the same protein (goal preference). The mobile CBR method, upon transmission to A_j, can perform the CBR subtasks (`retrieve`, `select`, `reuse`) using the case-base CB_j. When the mobile CBR method finishes the result (or a failure token) is sent back to A_i. The originating agent A_i can then decide if it is necessary to send the mobile CBR method to a new acquaintance and start a new iteration.

In the chromatography domain, the cooperation tasks of ColCBR are achieved as follows. First, a CBR method for protein purification `cbr-pp-mobile-method` is defined in originating `agent-i`; then the method is bounded to the current problem `case-33` and sent to `agent-j` by the expression:

```
(jump (define (cbr-pp-mobile-method)
         (case case-33))
    to agent-j)
```

This is equivalent to the following process:

1. The identifier of `cbr-pp-mobile-method` is sent to `agent-j`,
2. Since the method is defined in `agent-i`, `agent-j` requests the subtasks of `cbr-pp-mobile-method`; as result `agent-j` will receive the methods for those subtasks and (the identifier of) `case-33`.
3. Recursively, the methods of the subtasks will be transmitted and their subtasks methods will be requested, until all the task/method decomposition is transmitted to `agent-j`.
4. Finally, `cbr-pp-mobile-method` is executed by `agent-j` and the result is returned to the originating `agent-i`.

In general, the originating agent in ColCBR can have *several* mobile methods for a task. In ColCBR an agent could have several mobile CBR methods with a preference ordering among them from the more constrained CBR method to the less constrained. In this way, the agent can assure that it can retrieve the precedent cases from the distributed case-base that comply to the most relevant requirements for the task, and only when no precedent is found, a second mobile CBR agents searches for a less relevant precedent case in the distributed case-base.

4 Discussion and Future Work

We have presented two simple yet powerful cooperative modes of case-based reasoning and learning. Even assuming that all the participating agents start with the same CBR method, the individuality of the learning agents (the separate existence of agents having different memories given by disparate past experience) implies a distinct content (resulting from learning) for each agent. In the DistCBR cooperation mode an originating agent *delegates authority* to another peer agent to solve the problem. In contrast, ColCBR *maintains the authority* of the originating agent, since it decides which CBR method to apply and merely *uses the experience* accumulated by other peer agents.

In the protocols developed for the chromatography domain, since an agent only cooperates with another agent when the originator is not able to solve a problem (according to the domain knowledge constraints), the result of cooperation is always better than no cooperation, and communication is engaged only when need be. However, these protocols are domain dependent and are the result of a knowledge modelling process. The cooperation modes are, we argue, general for agents that capable of lazy learning.

The lazy nature of learning in CBR helps in the reuse and exploitation of the experience of different agents in a cooperative setting. Since the implicit generalization of similarity-based

reasoning is performed on a case-by-case basis, and the cooperation is also made on a case-by-case basis, both can be integrated seamlessly. Eager learning, as induction, perform learning over sets of cases and built new knowledge structures capable of solving new problems—and some of them discard the particular cases after induction. In this setting the Distributed Mode seems applicable, since every agent uses the induced knowledge structures to solve a particular problem. However, the Collective Mode seems problematic—inapplicable in fact if the agents discard the particular cases. This mode is based on the idea of extending the memory of an agent to the memory of the rest of agents by forming a collective memory. However, the distribution of agents and experience can meaningfully exploit the collective memory in a lazy, on-demand way. An eager use of collective memory, for instance, would be for an agent to perform induction over *all* cases known to all associate agents. This option implies a communication overhead and in the long run amounts to a centralized view of learning where every agent is aware of all the accumulated experience of every other agent.

We are exploring a cooperation mode for inductive learners that involve specialized agents and migrating methods. In this framework, called MILC[6], a classification task is distributed among a collectivity of agents specialized in parts of a taxonomy. Every agent, for a given problem, is required to determine whether it is in its area of expertise or not. In the negative situation the agent wraps the problem into a migrating method that is then transported to an associate agent. The agent receiving the migrating method can be a central authority (like a team leader) that knows the specialization of the associate agents. If there is no central authority the method migrates from one agent to another until the specialist is found. Note that in this MILC framework every inductive learning agent will deal only with a subset of all cases solved by the collectivity of agents.

Related work is KQML and CBR-TEAM. The communication capabilities of *Plural* Noos are compatible to the basic constructs of KQML [8]. Since we are dealing with homogeneous peer agents the rather general features of KQML (like ontologies and representations translation) are not needed, there is no need for *Plural* to use the KQML equivalent constructs[7]. It remains future work to see if Noos agents communicating with agents using other representation languages like Loom or KIF could actually use KQML constructs. The CBR-TEAM system uses negotiated case retrieval as a form of cooperative CBR among heterogeneous agents (subtask specialists) [9]. The overall task is a distributed constraint optimization process over the shared interface parameters (parameters optimized by more than one agent).

Although the CBR cooperation modes we propose are quite general descriptions, there are more options that those explained in this paper and that are envisioned as future work. For instance, we plan use the full CHROMA application which integrates induction and CBR. In this setting, DistCBR would use the metalevel method of CHROMA that selects the appropriate problem-solving method; while ColCBR the originating agent would be able to send to other agents the method of its choosing.

A variant of the DistCBR and ColCBR cooperation modes consists of asking k acquaintances to solve the problem instead of asking one by one until a solution is achieved. This variant requires a new task on the originating agent that performs some selection of the solution or consensus aggregation function. Both selection and consensus require A_{orig} having a model of the reliability of the agents involved —the model can be based on some learning method based on the previous results of those agents. However, the selection and consensus processes do not pertain to the cooperation mode as such, but to the knowledge modelling analysis of the task domain. For instance, in our domain more than one chromatography plan can effectively purify a protein, so it is possible to recommend more than one correct solution (although a solution ranking is of course highly desirable).

[6] Migratory Inductive Learners for Classification tasks

[7] An example of equivalence is the following. KQML has an `ask-all` construct. Finding all solutions of a task in Noos syntax is written `(*>> task of problem at agent)`.

Acknowledgements

The research reported on this paper has been developed at the IIIA inside the ANALOG Project funded by Spanish CICYT grant 122/93, a CSIC fellowship, and a CIRIT fellowship.

References

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994. <URL:http://www.iiia.csic.es/People/enric/AICom_ToC.html>.

[2] Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. *J. Logic Programming*, 16:195–234, 1993.

[3] Josep Lluís Arcos and Enric Plaza. Integration of learning into a knowledge modelling framework. In Luc Steels, Guss Schreiber, and Walter Van de Velde, editors, *A Future for Knowledge Acquisition*, number 867 in Lecture Notes in Artificial Intelligence, pages 355–373. Springer-Verlag, 1994.

[4] Josep Lluís Arcos and Enric Plaza. Inference and reflaction in the object-centered representation language Noos. *Journal of Future Generation Computer Systems*, 1996. To appear. Available online at <URL:http://www.iiia.csic.es/Projects/analog/analog-project.html>.

[5] Eva Armengol and Enric Plaza. Integrating induction in a case-based reasoner. In J. P. Haton, M. Keane, and M. Manago, editors, *Advances in Case-Based Reasoning*, number 984 in Lecture Notes in Artificial Intelligence, pages 3–17. Springer-Verlag, 1994.

[6] Luca Cardelli. Obliq, a language with distributed scope. Technical report, DEC, Systems Research Center, 1995.

[7] D. Dubois, F. Esteva, P. Garcia, L. Godo, and H. Prade. Similarity-based consequence relations. In Ch. Froidebaux and J. Kohlas, editors, *Symbolic and Qualitative Approaches to Reasoning and Uncertainty*, number 946 in Lecture Notes in Artificial Intelligence, pages 171–179. Springer-Verlag, 1995.

[8] Tim Finin, Jay Weber, and et al. Specification of the kqml agent-communication language. Technical report, The DARPA Knowledge Sharing Initiative, 1994. <URL:http://retriever.cs.umbc.edu/kqml/kqmlspec/spec.html>.

[9] M V Nagendra Prassad, Victor R Lesser, and Susan Lander. Retrieval and reasoning in distributed case bases. Technical report, UMass Computer Science Department, 1995.

[10] S. Russell. *The use of knowledge in Analogy and Induction*. Morgan Kaufmann, 1990.

[11] Luc Steels. Components of expertise. *AI Magazine*, 11(2):28–49, 1990.

[12] Bob Wielinga, Walter van de Velde, Guss Schreiber, and H. Akkermans. Towards a unification of knowledge modelling approaches. In J. M. David, J. P. Krivine, and R. Simmons, editors, *Second generation Expert Systems*, pages 299–335. Springer Verlag, 1993.