Published in G Weiss (Ed.), *Distributed Artificial Intelligence meets Machine Learning*, Lecture Notes in Artificial Intelligence, Springer Verlag, 1997.

Cooperative Case-Based Reasoning

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
Email: {enric,arcos,martin}@iiia.csic.es
WWW: http://www.iiia.csic.es/Projects/FedLearn/CoopCBR.html

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. Noos is knowledge modeling framework designed to integrate learning methods and based on the task/method decomposition principle. The extension we present, *Plural* Noos, allows communication and cooperation among agents implemented in Noos by means of three basic constructs: alien references, foreign method evaluation, and mobile methods.

1 Introduction

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). Before presenting our approach it is relevant to state how we view the relation between cooperation processes and learning processes in the framework of multiagent systems (MAS).

1.1 On Cooperation and Learning

In a multiagent environment, where agents have learning capabilities, the distinction between learning and cooperation is sometimes blurred. Does communication involve learning (e.g. learning by being told)? Is any overall improvement of a multiagent system performance some kind of learning? An answer to these

and related questions will require some more years of theoretical and experimental work in MAS learning. So instead of trying to answer these questions now we will point out the relationship between cooperation and learning.

First of all we may ask two negative questions: Why is there at all a need to cooperate? And why is there a need to learn? The answer to the second question is rather obvious: some agent needs to learn whenever it lacks some knowledge to perform some task— "perfect" knowledge has no room for learning. Learning has to do with improvement according to some criteria—i. e. amending those lacks for the task at hand. We can think the answer to the first question along the same line of thought: an agent needs to cooperate with other agents because it lacks some knowledge or some capability to perform a task. An agent with "perfect" knowledge and "complete" capabilities for a given task has no need to require the cooperation of other agents.

The parallelism of learning and cooperation stems from the fact that both are ways to deal with a agent's real shortcomings and lacks. We can summarize this parallelism as follows:

Learning Why is there a need to learn?
- Improving individual performance
- Improving precision (or quality of solutions)
- Improving efficiency (or speed of finding solutions)
- Improving the scope of solvable problems

Cooperation Why is there a need to cooperate?
- Improving individual performance
- Improving quality of solutions
- Improving efficiency in achieving solutions
- Achieving tasks that could not be solved in isolation

We are interested in investigating the interplay of learning and cooperation in this view. The approach presented in this paper explores a simple interplay of both: agents require the help of other agents when they are not capable of resolving a problem. In the future we hope to explore more complex interplays, for instance, when an agent can decide not to improve itself (not to learn) in situations when there is already a proficient agent in the MAS because it can simply require the help of this cooperative partner. The next subsection explains in more more details the approach we take.

1.2 Federated Peer Learning

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 an 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 [10]. 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.

1.3 The Task of Protein Purification

We have developed CHROMA, a 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 § 5 for future work on more complex situations).

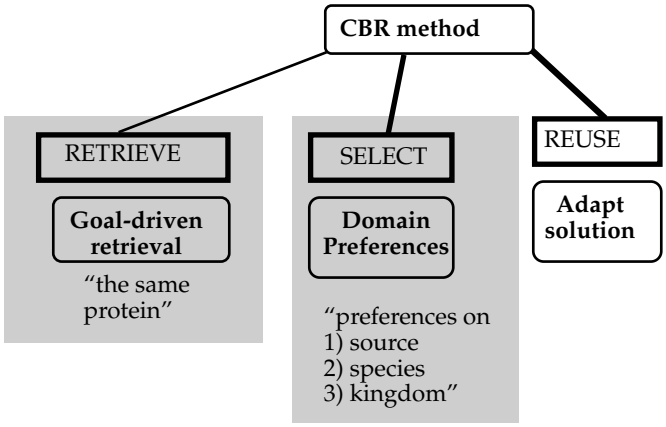

Fig. 1. The case-based reasoning method in CHROMA. The shaded part will be modified to adapt this method to a multiagent system (see Figure 7).

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 modeling 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 modeling 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 modeling frameworks like KADS [15] or components of expertise [14][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 modeled 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

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

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 case-based reasoning 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[13]. 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 an object-centered representation language based on *feature terms*. *Feature terms* are record-like data structures embodying a collection of *features*. Intuitively, a feature term is a syntactic expression that denotes sets of elements in some appropriate domain of interpretation. In this way feature terms can be viewed also as partial descriptions. The values of features are constants or other feature terms. Our approach is close to the *ψ-term* [2,8] and *extensible records* [7,9] formalisms.

The difference between feature terms and first order terms is the following: a first order term, e. g. $f(x, g(x, y), z)$, can be formally described as a tree and a fixed tree traversal order—in other words, variables are identified by position. The intuition behind a feature term is that it can be described as a labeled graph—in other words, variables are identified by name (regardless of order or position). This difference allows to represent partial knowledge.

Formally, we describe the Noos signature Σ as the tuple $\langle \mathcal{S}, \mathcal{M}, \mathcal{F}, \leq \rangle$ such that:

- $\mathcal{S}$ is a set of *sort symbols* including $\bot, \top$;
- $\mathcal{M}$ is a set of *method symbols*;
- $\mathcal{F}$ is a set of *feature symbols*;
- $\leq$ is a decidable partial order on $\mathcal{S}$ such that $\bot$ is the least element and $\top$ is the greatest element.

Given the signature Σ and a set $\mathcal{V}$ of variables, we define a feature term ψ as an expression of the form:

$$\psi \quad ::= \quad X : s\ [f_1 \doteq \Psi_1 \cdots f_n \doteq \Psi_n]$$

where X is a variable in $\mathcal{V}$, s is a sort in $\mathcal{S}$, $f_1, \cdots, f_n$ are features in $\mathcal{F}$, $n \geq 0$, and each Ψ_i is either a feature term, a set of feature terms or a method application $\#m$.

[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.

Domain knowledge is represented in Noos by a collection of feature terms describing the concepts and their relations for a given domain. Feature terms have a correspondence to labeled graphs representation as shown in the description of an experiment in the chromatography domain of Figure 2.

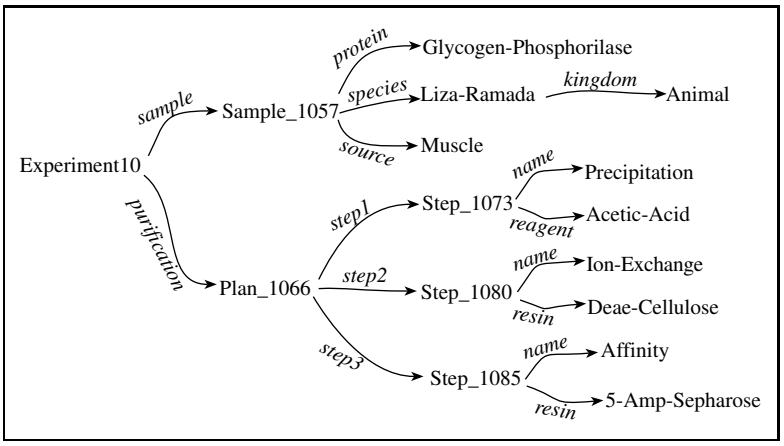

Fig. 2. A case description in CHROMA.

Methods are also represented as feature terms. The features of a method description represent the *subtasks* in which that method is decomposed. Methods are defined by refinement from a set of built-in methods. That is to say, a method is a feature term

$$\psi_m \quad ::= \quad X : m\ [f_1 \doteq \Psi_1 \cdots f_n \doteq \Psi_n]$$

as above except that now m is a sort in $\mathcal{M}$, i. e. it is a refinement of a built-in method.

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 feature terms is what allows *Plural* Noos to transmit over the network both domain knowledge and methods in the same way.

Inference in Noos is on demand and 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 below.

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 modeled. 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 modeling 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.

3 Agent Communication with *Plural* Noos

Plural provides a seamless extension of Noos that supports distributed scope and reference for all the basic constructs in Noos. A *Plural* Noos agent is a particular Noos application with a known address and with several *acquaintances*. An agent address is composed of one IP address, a port number and one identifier. The last identifier is needed since more than one agent can coexist within the same *Plural* Noos process. The acquaintances of an agent are those agents whose address is known by the agent —as in the actors model. Each *Plural* agent can have different acquaintances. If an agent A_i belongs to the acquaintances of an agent A_j, then $A_j$also belongs to the acquaintances of A_i.

A *Plural* Noos agent can be involved in solving only one problem at a time. Each problem solving process has a different identifier. When a *Plural* agent is solving a problem only accepts requests related to the same process identifier. In this way, possible deadlocks are avoided. Other deadlocks caused by circularities inside the same problem solving process are detected by the *Plural* Noos implementation. When an agent A_i requires a service from another agent A_j, and this one is already busy solving another different problem, A_i receives a *busy* message. Then A_i decides to wait some time to request the service to A_j again or ask it to another member of its acquaintances.

All *Plural* Noos agents taking part in an specific domain application share the same signature Σ. That is to say, the feature symbols, the sort symbols, and the method symbols are shared among all *Plural* agents involved in an application. So *Plural* Noos allows arbitrary Noos terms to be exchanged among one agent and its acquaintances. In particular, cases and CBR methods are terms that can be transmitted from a CBR agent to another. The CBR cooperation modes which this paper describes will use three *Plural* Noos capabilities: alien references, foreign evaluation, and mobile methods.

3.1 Alien References

Alien references extend Noos references to agents over the net. For instance, when the term identifier `experiment10` in `agent-i` is transmitted to `agent-j`, it is handled as an alien reference and it becomes naturalized as `experiment10@agent-i`

by the agent-j. In the same way, a reference to a feature in agent-i, as (>> purification of experiment10), once transmitted to agent-j becomes an alien reference, (>> purification of experiment10@agent-i), in agent-j. Notice that feature symbols are shared among *Plural* Noos agents. If the value referenced by an alien reference in agent-j is needed then a transmission is automatically engaged asking for the value to agent-i. Agent-i is responsible for inferring that value and transmit it as answer to the agent-j request. Alien references avoid the problem of maintaining state when terms with state are transmitted over the network. State is local to agents and when an agent makes reference to a term which belongs to another agent, a alien reference is established [3]. Alien references are transmitted over the network: if experiment10@agent-i is a value of the feature purification of entity experiment21 in agent-j and a new agent agent-k has the reference (>> purification of experiment21@agent-j) eventually agent-k will get the alien reference experiment10@agent-i.

Alien references make up the basic mechanism that underpins the exchange of terms among *Plural* Noos agents over the network. In essence, as Noos terms can be seen as labeled graphs, and since Noos performs a lazy evaluation, not all the nodes in a graph are transmitted when the root is referenced by a remote agent. Instead, the transmission of a term from an agent agent-i to another agent agent-j starts by sending the graph root (an identifier, the sort, and the name of the root features). If the graph node sent to agent-j is a constant (a number, a string or a sort) a local reference is established by agent-j. Otherwise, agent-j establishes an alien reference to that node. When agent-j requires the value of any of the features of that node, a new transmission is engaged asking for it to agent-i. Then agent-i inferres its value and sends it to agent-j. Path equality (sharing) and circularities in the graph are preserved.

The next example describes how the term experiment10 (see Figure 2) in agent-i is transmitted to agent-j. In the first step the term identifier experiment10 and the names of its features sample and purification are sent to agent-j. Since experiment10 is not a constant, an alien reference will be established in agent-j, as showed in Figure 3. Then, if the value of the feature sample is required by agent-j, it will be automatically requested to agent-i. Next, agent-i will resolve that reference to sample_1057. This term identifier and the names of its features protein, species and source will be sent back to agent-j, and a new alien reference sample_1057@agent-i will be established (see Figure 4) in agent-j, since sample_1057 is not a constant. Figure 5 shows the state achieved once the values of features protein, species and source have been required by agent-j. Values Glycogen-Phosphorilase, Liza-Ramada and Muscle are all of them sorts, and since sorts are shared, a reference to the local sorts has been established in agent-j, when they have been received. Finally, as Figure 6 shows, when the value of feature kingdom of Liza-Ramada is re-

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

quired by `agent-j`, the value `Animal` is inferred in `agent-j`, without need to ask `agent-i`, since `Liza-Ramada` is local to `agent-i`.

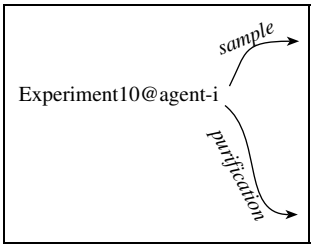

Fig. 3. An alien reference to `experiment10` at `agent-i` is established in `agent-j`.

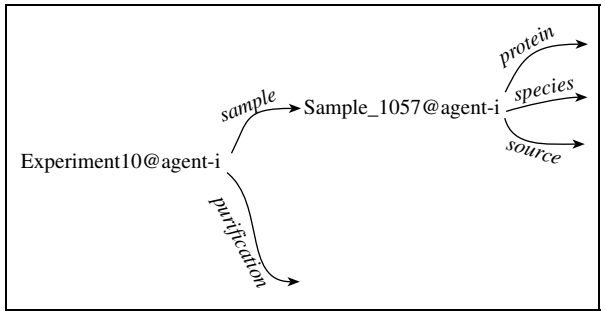

Fig. 4. An alien reference to `Sample_1057` at `agent-i` is established in `agent-j`.

3.2 Foreign Evaluation

The *Plural* Noos foreign evaluation capability allows an agent to use a method owned by another agent —as in remote procedure call (RPC). Specifically, foreign evaluation allows an agent `agent-i` to ask another agent `agent-j` to execute a specific method using the parameters given by `agent-i`, as in the next expression of `agent-i`.

```
(define (foreign-eval)
  (method (define (protein-purif-method)
            (case experiment10)))
  (at agent-j))
```

In this expression, an agent `agent-i` asks to another agent `agent-j` to evaluate the method `protein-purif-method` using as `case` the `experiment10`. Then

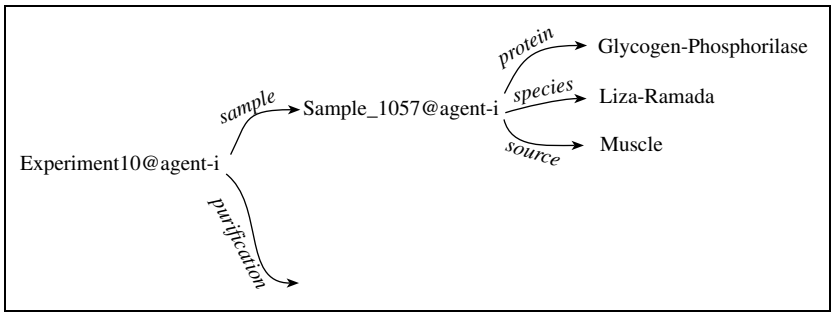

Fig. 5. `Glycogen-Phosphorilase`, `Liza-Ramada` and `Muscle` are references to sorts. Since sorts are shared by all agents they do not require alien references.

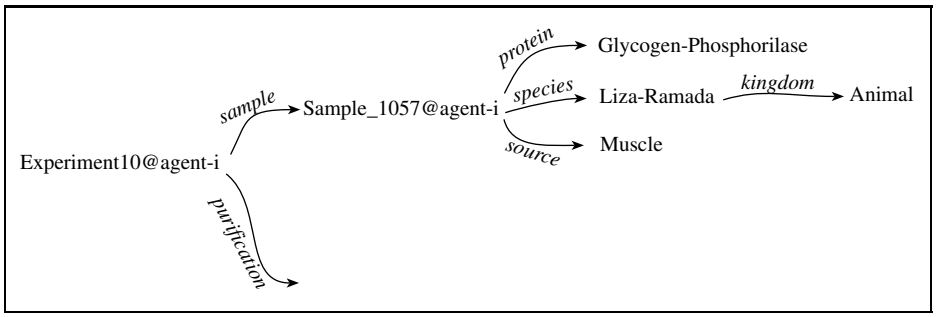

Fig. 6. The value of feature `kingdom` is inferred from the local sort.

`agent-j` will start the evaluation of its own `protein-purif-method` method, annotating that this evaluation is being performed for the remote agent `agent-i`. When the `case` feature of this method is required during the evaluation, it will be automatically asked to `agent-i`. Then the `experiment10` term will be sent from `agent-i` to `agent-j`, such as was explained in the last subsection. This value will be an alien reference in `agent-j` and will become naturalized as `experiment10@agent-i`. During the evaluation, further references in `agent-j` to features of `experiment10@agent-i` are interpreted as alien references as well. And its values will be transmitted from `agent-i` as they are needed. Once `agent-j` finishes the evaluation of method `protein-purif-method` the result got will be sent back to `agent-i`, as answer to the evaluation of the `foreign-eval` method.

3.3 Mobile Methods

For some cooperation modes it is necessary to support so-called mobile (or migrating) methods. In *Plural* Noos a mobile method defined in an agent `agent-i` can be transmitted to any member of its acquaintances. When an agent `agent-i` sends a mobile method to `agent-j`, this process involves also transmitting the whole task/method decomposition to `agent-j`—i. e. the subtasks of that method, and the methods for those subtasks. The process of sending a mobile method, called `jump`, consists of

1. sending the the name of the built-in of which the method is a refinement
2. the names of its features (i. e. the method's subtasks)
3. Recursively, the methods defined for those subtasks

While foreign evaluation requires the remote agent to own a particular method which can be used by the originating agent, the mobile methods capability of *Plural* Noos does not require it.

Mobile methods are supported by the *Plural* Noos capability of transmitting method terms. A mobile method term is first defined in an originating agent `agent-i`:

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

When a method jumps to a remote agent, the whole task/method decomposition of the mobile method is transmitted in a lazy way similar to that explained in § 3.1. Nevertheless, there is a main difference between the `jump` process and the transmission of a feature term resulting from an alien reference. In the `jump` process, when a reference is made to a feature of a mobile method (i. e. a subtask), *Plural* Noos requests to the originating agent the *method* corresponding to that feature name. In this way, the whole task/method decomposition of the mobile method will be transmitted, on demand, from the originating agent to the target agent.

4 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 two 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.

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

Regarding the chromatography domain, the CBR method for CHROMA shown in Figure 1 is modified as shown in Figure 7. Since the shaded part in both figures is the part that an originating agent wants to ask other agents to perform over their own case-bases, we introduce a new method, `protein-purify-method`, that simply gathers together both tasks, `retrieve` and `select`, that have to be distributed over other agents. In this way, DistCBR will be implemented using `protein-purify-method` by foreign evaluation and ColCBR will be implemented using `protein-purify-method` as a mobile method.

4.1 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. DistCBR is based on the

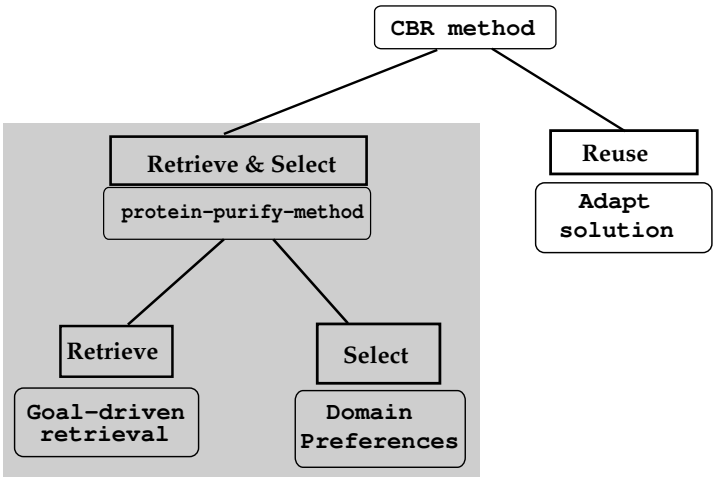

Fig. 7. The case-based reasoning method for DistCBR and ColCBR in CHROMA. The shaded part is changed from that of Figure 1 and are the subtasks performed by other agents on request of the originating agent.

Plural Noos capability of *foreign 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 modeling 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

[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).

the task `purification`—by convention we will assume all agents use the same public name `protein-purif-method`[5].

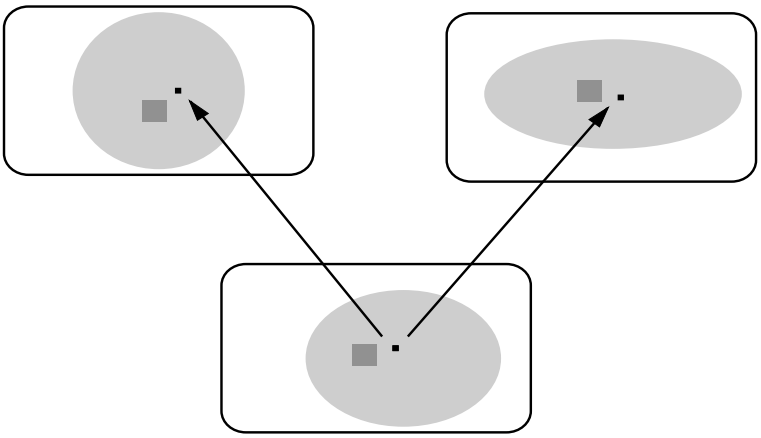

Fig. 8. In Distributed CBR each agent uses its own `retrieve-&-select` method on the current problem. The shaded areas represent a similarity degree centered around the current problem (the black dot). The most similar case in each agent's memory is depicted as a shaded box. The shaded areas are in general different because the criteria that specify what is "similar" may vary from one agent to another. Compare to Figure 9 that shows the effect of using a unique mobile CBR method in ColCBR.

The cooperation tasks of DistCBR are achieved in the implementation by requiring the `foreign-eval` of a M_k (say `protein-purif-method-k`), for each RM_k in the collection of methods for the `retrieve-&-select` task. The *Plural* Noos syntax is as follows:

```
(define (foreign-eval)
  (method (define (protein-purif-method-k)
            (case case-33)))
  (at agent-j))
```

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 current implementation of DistCBR CROMA has two strategies to select the acquaintances to which an agent asks help. The first one, as mentioned, simply asks other acquaintances in some specific order until one of them can solve the problem requested using its own method. The results obtained in its

[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.

strategy for DistCBR cooperation mode crucially depends on to ordering in which an agent selects an acquaintance, and a more complex handling of it is discussed at § 5. A second strategy, that we call *conservative*, allows more control to the originating agent at the cost of more communication. The conservative strategy of DistCBR involves the originating agent asking to solve the problem to all its acquaintances and obtaining the best cases according to them. Then, the originating agent can select with of them is best according to its own criteria, for instance according to the Preferences in § 2.2.

4.2 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 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:

```
(define (jump)
  (method (define (cbr-pp-mobile-method)
            (case case-33))
  (at 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.

In the current implementation of ColCBR CHROMA the agents follow *conservative* strategy in asking for help to other agents the rationale of which is to assure a result as close as possible to the original CBR method for a standalone system. In particular, ColCBR CHROMA *conservative* strategy tries to find the best precedent case known by a federation of agents (its acquaintances)—where "best" is interpreted in the sense of the preferences explained in § 2.2.

In order to do so, an agent with this strategy has a collection of methods $\{M_1, M_1^m, \ldots, M_4, M_4^m\}$ for the task `retrieve-&-select`. The first two M_1 and M_1^m are methods that considers the preferences in § 2.2 as restrictions: in this way it can retrieve only the precedent cases satisfying all these conditions. Method M_1 retrieves cases from the case-base of the originating method. If this method fails—i. e. there is no such a case in memory—Noos backtracks taking the second option, namely M_1^m, that is a mobile method version of method M_1. M_1^m is sent one by one to all the acquaintance—if the mobile method sent to an agent fails, *Plural* Noos sends the mobile method to the next acquaintance. If one of them returns such a case the `retrieve-&-select` task is finished. Otherwise it means that all agents have failed—none of them have a precedent case satisfying all the constraints in § 2.2. In this situation, Noos selects the next method, namely M_2. Now both M_2 and M_2^m are a less restricted version of M_1 and M_1^m where the less important constraint in § 2.2 (Preference 3) is dropped. Using M_2 and M_2^m now DistCBR CHROMA can retrieve a precedent case from its memory or one of its acquaintance receiving the mobile method M_2^m can retrieve a precedent case from its memory. Again, if any mobile method retrieves a case complying to the constraints the process stops, otherwise proceeds with M_3 and M_3^m (that only requires as constraint Preference 1 in § 2.2) and with M_4 and M_4^m (that only performs Goal-driven Retrieval but enforces no preference).

It is easy to see that this strategy assures that the originating agent finds the most preferred case according to the established preferences from any case base of an acquaintance agent. Figure 8 shows intuitively the effect of a mobile CBR method: the same retrieval and selection method is used in each agent, the only difference being the case that is retrieved in each agent according to its past experience.

5 Future Work on Cooperative Case-based Reasoning

The conservative strategy of last section is not obliged by neither by the ColCBR mode of cooperation nor by the *Plural* Noos language. It is perfectly possible and

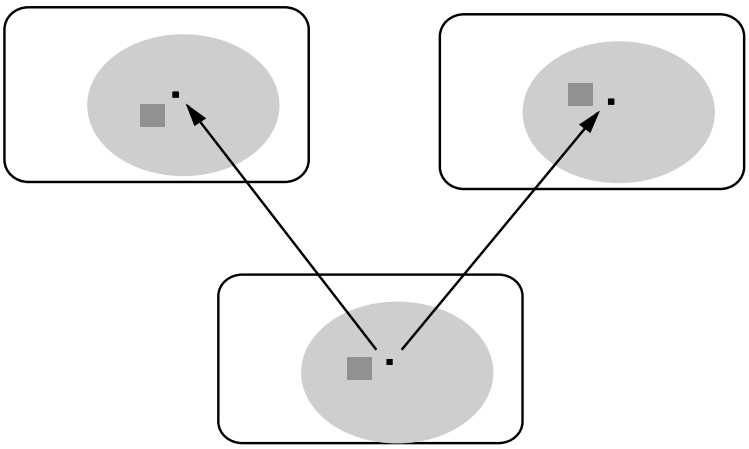

Fig. 9. In Collective CBR mobile methods assure that the similarity considered will be the same in all agents. The shaded area represents a similarity degree centered around the current problem (the black dot). The most similar case in each agent's memory is depicted as a shaded box. The shaded area is equal in the originating agent and in the two agents that receive a mobile CBR method, while in DistCBR (see Figure 8) they are different.

rational that the originating agent sends a mobile CBR method that embodies the preferences in § 2.2 to the acquaintance agents. In this strategy, the first acquaintance agent that has case satisfying *some* of the preferences in § 2.2 will be retrieved. In this strategy, the order in which we take the acquaintance agents to solicit them to solve a problem becomes crucial. There are two approaches to solve this issue: instituting authority and learning competence models. *Instituting authority* involves selecting a priori the class of problems for which each agent is competent on and giving him authority to solve them. This selection can be typically established by the designer of a multiagent system (MAS) or by the institution(s) that grant the cooperation of one of its agent into a MAS. The second approach involves agents learning a model a *competence model* of other agents in a MAS—i. e. each agent has to determine (learn) an individual model of which problems other agents in a MAS are competent to solve. This approach is high in our research agenda on federated learning.

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 modeling 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 Discussion

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 modeling 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.

Related work is KQML and CBR-TEAM. The communication capabilities of *Plural* Noos are compatible to the basic constructs of KQML [11]. 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[6]. 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) [12]. The overall task is a distributed constraint optimization process over the shared interface parameters (parameters optimized by more than one agent).

In this paper we have focused on modes of cooperation among agents able to perform some lazy learning, but we focused the learning process on learning about the task domain—chromatography techniques in our application. However, as a result we are quite aware that learning has also to play a major role regarding the cooperation process itself. We plan to study this issue by the agents being capable to learn *competence models* of other agents. We think this approach can be useful for any MAS where the authority of an agent is not predetermined by the system designer. In fact, we can think about Federated Peer Learning as a framework in which the authority of each participating agent is dynamically allocated by other participant agents assessing their scope and degree of competence.

Acknowledgements

The research reported on this paper has been developed at the IIIA in the framework of the ANALOG Project (CICYT grant TIC 122/93),the SMASH Project (CICYT grant TIC 96-1038), a CSIC fellowship, and DGR-CIRIT fellowship FI-DT/96-8472.

Updated information will be posted on the WWW at URL `http://www.iiia.csic.es/Projects/FedLearn/CoopCBR.html` and `http://www.iiia.csic.es/Projects/FedLearn/plural.html`.

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. Available online at <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.

[6] 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)`.

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 reflection in the object-centered representation language Noos. *Journal of Future Generation Computer Systems*, 12:173–188, 1996.
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. Luca Cardelli and John Mitchell. Operarions on records. In Carl A. Gunter and John C. Mitchell, editors, *Theoretical aspects of object-oriented programming: types, semantics and language design*, Foundations of computing series, pages 295–350. MIT Press, 1994.
8. B. Carpenter. *The Logic of typed Feature Structures.* Tracts in theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1992.
9. Laurent Dami. *Software Composition: Towards an Integration of Functional and Object-Oriented Approaches.* PhD thesis, University of Geneva, 1994.
10. 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.
11. Tim Finin, Jay Weber, and et al. Specificastion of the kqml agent-communication language. Technical report, The DARPA Knowledge Sharing Initiative, 1994. <http://retriever.cs.umbc.edu/kqml/kqmlspec/spec.html>.
12. M V Nagendra Prassad, Victor R Lesser, and Susan Lander. Retrieval and reasoning in distributed case bases. Technical report, UMass Computer Science Department, 1995.
13. S. Russell. *The use of knowledge in Analogy and Induction.* Morgan Kaufmann, 1990.
14. Luc Steels. Components of expertise. *AI Magazine*, 11(2):28–49, 1990.
15. 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.