Knowledge and Experience Reuse through Communication among Competent (Peer) Agents

Francisco J. Martín, Enric Plaza, and Josep Lluís Arcos
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: {martin,enric,arcos}@iiia.csic.es
WWW: http://www.iiia.csic.es/Projects/FedLearn/CoopCBR.html

Abstract

This article addresses an extension of the knowledge modelling approaches, namely to multi-agent systems where communication and coordination are necessary. We propose the notion of competent agent and define the basic capabilities of these agents for the extension to be effective. An agent is *competent* when it is able to reason about its own competence and that of the other agents with which it cooperates in a given domain. In our framework, an agent has competence models of itself and of its acquaintances from which it can decide, for a specific problem to be solved, the type of cooperative activity it can request and from which agent. In this paper we focus on societies of peer agents, i. e. agents that are able to solve the same type of task but that may have different degrees of competence for specific problem ranges.

1 Introduction

Cooperative multi-agent systems (MAS) that are being designed around the *team* metaphor tend to describe a MAS by the roles of the agents in the cooperative activity they are engaged in while solving problems. This approach tends to emphasize the view of agents as specialists in a team and lends it to develop frameworks with a fixed division of labour—typically, each agent is responsible for a subtask in a static task/agent assignment. Our research work in agents that both cooperate and learn [22] has led us to focus on MAS where agents need to dynamically choose, for a specific problem, which agent to cooperate with for each (sub)task.

In this paper we will focus on the needs for knowledge sharing and reuse in cooperative MAS with a flexible and dynamic task/agent assignment. We will

propose the notion of competent agent, and we will develop some agent cooperation strategies based on two assumptions: i) each agent has a competence model of the other agents in the cooperative MAS, and ii) each agent is able of dynamically decide with which agent to cooperate in order to solve a given task. Moreover, we will illustrate our approach with CoDiT, a MAS wherein agents use case-based reasoning (CBR) to recommend therapy for diabetic patients. This application domain allows us to exemplify the sharing and reuse of problem solving knowledge and experiential knowledge (in the form of cases in CBR) among agents that reside anywhere in a IP network.

The structure of the paper is as follows. Section 2 introduces the framework of competent agents and CoDiT, our MAS diabetes therapy application. Then, Section 3 presents the representation language the agents use to reasoning and the agent communication scheme the agents use to share and reuse knowledge and experience. Next, Section 4 presents two cooperation modes for MAS performing CBR and how this is related to the representation and communication languages. This section also explains some strategies (inside of a cooperation mode) for using competence models and choosing with whom to cooperate. Section 5 analyzes some related approaches, and finally, the paper closes with our current conclusions.

2 Competent Agents

A *competent agent* is an agent that: i) is reflectively aware of its own competence, for a range of tasks, in solving problems , and ii) is aware of the competence of the other agents in a cooperative MAS. In our approach, we propose to design a competent agent endowing it of competence models of itself and competence models of acquaintance agents in the MAS. Moreover, a competence model of an acquaintance agent has to exist for any specific task (upon which cooperation will take effect) and only makes sense inside a given cooperation mode.

Competent Agent A competent agent is a tuple $A_i\langle \mathcal{M}, \mathcal{A}, \mathcal{C}, \mathcal{S}\rangle$ with $\mathcal{M} = M_1, \ldots, M_n$ cooperation modes, $\mathcal{A} = A_1, \ldots, A_m$ acquaintance agents, competence models $\mathcal{C} = C_1, \ldots, C_k$, and reflective competence models (or competence self-models) $\mathcal{S}$.

Competence Model A competence model $\mathbf{C} \in \mathcal{C}$ of agent A_i is a tuple $\mathbf{C} = \langle A, T, P, M\rangle$ where $A \in \mathcal{A}$, T is a task of A, P is a problem specification, and $M \in \mathcal{M}$ is a cooperation mode. Moreover, a competence self-model $\mathbf{S} \in \mathcal{S}$ is a tuple $\mathbf{S} = \langle T, P\rangle$.

Clearly, the competence model depends on the agent A to which a request is done and also on the task T requested to that agent. Moreover, the competence depends also on the *type* of request asked to A concerning T: this aspect is modelled by the notion of *cooperation mode*. Two cooperation modes for CBR

are proposed in §4, DistCBR and CoICBR: in DistCBR experience in the form of cases of A is shared by A_i but the problem solving knowledge used to solve T belongs to A; conversely, in CoICBR experience in the form of cases of A is shared by A_i and the problem solving knowledge used to solve T belongs to A_i. Section 4 explains how these to cooperation modes are based two types of requests, namely *task delegation* and *mobile problem solving methods*.

Problem Specification Finally, a problem specification $P = (D, E)$ (of A_i about A for T) consists basically of two models: a problem description D and an agent performance evaluation E. A problem description D characterizes a range of problems for which agent A is competent, and commonly a competence assessment problem solving method can compare the current problem with D and yield back a degree of confidence for A's competence. On the other hand, model E holds information about A's performance concerning mundane matters such as response delay, unavailability frequency, etc. Since these aspects are quite domain-dependent, we will not do into more details here; however, concerning the knowledge acquisition work involved in acquiring those competence models, it is worth to say that we are using inductive learning techniques as those of [7].

Regarding the notion of *peer agents*, let's say that we are interested here not in specialist agents but in agents able to solve, in principle, all the tasks an subtasks of a problem, but the degree of validity may vary along different ranges of problems. This setting may look strange at first, but it is tantamount to assume that agents do not have perfect knowledge: in effect, what sense would it make for a agent to cooperate inside a MAS if the agent already knows all it needs? Another way to view our domain of interest is considering that we are interested into MAS whose agents are able to learn from experience. In what follows, then, we shall assume we are dealing with MAS that fulfil the following assumptions.

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

Peer Agents The involved agents are capable of solving the task in 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 call this framework *federated peer learning* [22], an although this article will say few about learning, it is relevant to understand the example system in the next subsection where agents using Case-based Reasoning learn from experience in a medical domain.

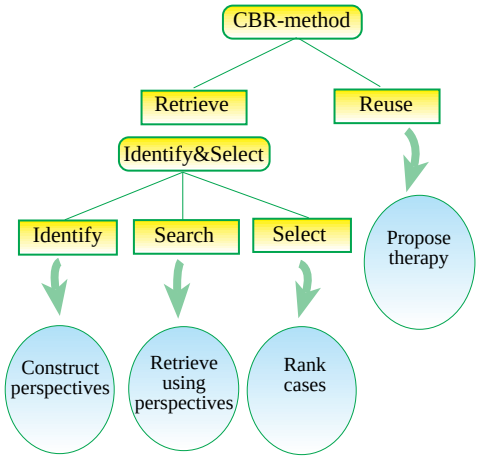

Figure 1: Task decomposition of the case-based reasoning method for diabetes therapy. The retrieve task will be augmented with communication capabilities for searching cases solved by other agents.

2.1 Case-based Reasoning in Diabetes

We are developing CoDiT, a multi-agent system for therapy recommendation in diabetic patients lying in the framework of the SMASH project[1]. This multi-agent system consists of a group of agents that perform case-based reasoning (CBR) and are able to communicate and cooperate for the purpose of recommending a therapy. Each agent has a case base with data of the patients of a specific M.D.; moreover, legal and deontological reasons prevents that patient data could be centralized since only the patient's doctor is entitled to have that data. Thus, this scenario fits the MAS approach since resources are distributed but some doctor (or her agent) could also be interested in the case of a patient that is unknown to her but stored in some other doctors case base. The doctors we are cooperating with have developed a common ontology to represent the patients data—thus we can focus on the cooperation and communication for the therapy task. Finally, remark that our diabetes therapy CBR agents are *peer agents* in the MAS, since each agent is capable of solving the whole task alone (recommending a therapy) using the resources available in its case base. However, it is intuitively appealing the scenario in which they are able to exchange patient data (maintaining anonymity for legal and deontological reasons) in order to improve their performance.

The basic CBR task decomposition is that shown in Fig. 1: retrieve and reuse. There is also an automatic retain task (not shown in the figure) that incorporates a solved problem into the agent's episodic memory. Summarily, the retrieve task has a decomposition method with three subtasks: identify, search and select. The

[1]Systems of Multiagents for Medical Services in Hospitals, more information is available at http://www.iiia.csic.es/Projects/smash/

identify task has a method that constructs a perspective[2] on the patient; then task search retrieves from the case-base those cases that are subsumed by that perspective (i.e. those cases that satisfy the model built by the perspective). Next, the select task has a method that constructs a preference model of the retrieved cases from domain-specific knowledge. Finally, reuse is a task that takes the most preferred case and adapts its solution (therapy) to the current patient; the adaptation method uses domain-specific knowledge and if a most preferred case cannot be adapted then tries to adapt the next-preferred case.

Section 4 explains how the methods used in the retrieve task can incorporate communication and cooperation with other agents in order to find relevant cases known for other agents. Moreover, section 4 will introduce some strategy, based on competence models, needed to determine with which of the peer agents is better to cooperate for solving a specific problem.

3 Representation and Communication

Current knowledge representation languages have been thought to develop isolated applications and usually cannot provide such applications with the communication abilities required by agent-based systems. For instance, that is the case of the knowledge representation language Noos [4]. Noos is an object-centered language based on feature terms and developed in our institute which was specially designed to integrate problem solving and learning techniques. Nonetheless, although Noos can furnish applications with the reasoning abilities required by intelligent agents, it lacks communication abilities except for a graphical user interface. One of our aims is to promote Noos to an agent language and benefit from the research effort deployed in it. Thereby, we are developing Plural, a seamless extension of Noos with agent-oriented capabilities [15]. Plural can be thought as an extension of a knowledge representation language with an agent communication language and an agent coordination language. Both of these languages have been provided at the knowledge representation level by means of a set of constructs with the same nature as the rest of Noos constructs. In the following subsections we succinctly describe the Noos representation language and present the communication abilities with which Plural provides Noos.

3.1 The Noos Language

Noos is a reflective object-centered representation language designed to support knowledge modelling of problem solving and learning [6]. 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 [29] or components of expertise [28][3].

[2] A perspective constructs from the patient model a specific model of the relevant aspects for the task at hand [5].

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

The Noos representation language is based on four components: *domain knowledge*, *problem solving knowledge*, *episodic memory*, and *metalevel*.

Domain knowledge specifies a set of *concepts*, a set of *relations* among concepts, and problem data that are relevant for an application. Concepts and relations define the *domain ontology* of an application. For instance, the domain ontology of CoDiT defines concepts such as `insulin-dependency` or `pregnant`. Problem data, described using the domain ontology, define specific situations (specific problems) that have to be solved. For instance, in the diabetes application, problem data represents information from specific patients.

Problems to be solved in a domain are modeled as *tasks*. For instance, the main task in the diabetes application is to recommend therapies for diabetic patients. In Noos, a problem solving *method* (PSM) models a way to solve problems. Methods can be elementary or can be decomposed into subtasks. These new (sub)tasks may be achieved by other methods. A method defines a specific combination of the results of the subtasks in order to solve the task it performs. For a given subtask there may be multiple alternative methods that may be capable of solving that subtask in different situations. This recursive decomposition of a task into subtasks by means of a method is called the task/method decomposition. For instance, Figure 1 shows the task/method decomposition of the case-based reasoning method for diabetes therapy. The `CBR-method` is decomposed into two subtasks, namely `retrieve` and `reuse`. For each subtask one or several alternative methods are specified—e.g. subtask `retrieve` has the `identify&select` method that is decomposed in (sub)tasks namely `identify`, `search`, and `select`.

Noos provides an initial set of existing methods, called *built-in methods*, that are those of a general-purpose language plus some constructs enabling introspection. New methods can be defined by refinement of these built-in methods.

Problem solving in Noos is considered as the construction of an *episodic model*. In this sense the Noos approach to problem solving is close to that of CommonKADS [29] and to the TASK language [21]. Our view of "problem solving as modelling" is that problem solving is the process of constructing an episodic model. This model is obtained from transformations of problem data performed using problem solving knowledge. A clear and explicit separation between tasks, methods, and domain knowledge permits the dynamical link between a given problem, tasks, and methods as well as the dynamical choice of a method suited to achieve a task in that problem context: a 'task' uses a 'method' on a 'episode' (described using domain knowledge and problem data). An episodic model is the set of knowledge pieces used for solving a specific problem. Once a problem is solved Noos automatically memorizes (stores and indexes) the episodic model that has been built. Episodic memory is the (accessible and retrievable) collection of the episodic models of the problems that a system has solved and constitutes its *experiential knowledge*. The memorization of episodic models is a basic building block for integrating learning into a KM framework.

Metalevel (or reflective) knowledge is knowledge *about* domain knowledge,

problem solving knowledge, and episodic memory. Metalevel knowledge includes *preferences* to model decision making about sets of alternatives present in domain knowledge and problem solving knowledge. For instance, metalevel knowledge can be used to model criteria for preferring some methods over other methods for a task in a specific situation. Specifically, competence models are represented in Noos in the metalevel.

On the other hand, the metalevel has models about how problems are solved in the system—or in other words, knowledge about episodic models. Knowledge about episodes models knowledge such as the method that has succeeded in achieving a specific task, the methods that have failed in achieving a specific task, the tasks that have been engaged in the evaluation of a method, and the preference orders built while choosing among alternatives. At the conclusions section we will describe a future research line where the experience accumulated in delegating tasks to other acquaintances may be used to improve the competence models that a Noos agent has about its acquaintances.

Communication and coordination capabilities provided by Plural enable Noos to the exchange of tasks (*task delegation*) and the exchange of methods (*mobile problem solving methods*) in multi-agent scenarios. Both capabilities will be explained in the following subsection.

3.2 Communication and Coordination in Plural

Plural provides Noos with a communication scheme based on *message-passing* as communication protocol and *point-to-point* as addressing scheme. Namely, the means by which Plural allows agents to share knowledge takes the form of messages routed from an agent to another. However, Plural agents do not communicate directly with one another, instead, the rely on interagents. An interagent is an autonomous software agent which manages (intermediates) the communication and coordination between an agent and the agent society wherein it is situated [15, 17]. Each agent has attached its own interagent which constitutes the sole and exclusive means through which it interacts with other agents. An interagent gives a permanent identity to its owner and establishes what messages can be forwarded, to whom and when. An interagent supports a range of programmable communication and coordination facilities. On top of such facilities, Plural provides Noos with some constructs which offer, at the knowledge representation level, the coordination level required to the exchange of tasks and methods allowing agents to cooperate in non-trivial ways. These constructs allow the exchange of knowledge to be performed at the knowledge representation level transparently to the agent communication and coordination language.

Up to now, two new constructs have been provided—defforeign and defmobile. The defforeign construct allows a Plural agent to remotely invoke methods owned by other Plural agents[4] (foreign evaluation). The defmobile construct, on the other hand, allows a Plural agent to send methods to be remotely evaluated to

[4]A method can remotely be invoked by other acquaintances only if its owner agent have previously made it accessible for other agents.

other agents (mobile evaluation)[5]. Thus, using such constructs Plural agents are provided with the following capabilities.

Foreign Evaluation This capability allows a Plural agent to remotely invoke methods, like in the *remote procedure call* (RPC) model, owned by any of its acquaintances. When an agent invokes a method owned by an acquaintance it also provides the necessary parameters to evaluate it. Foreign evaluation, like the RPC model, is based on the notions of *data mobility* and *control mobility* among agents. Data is exchanged among agents in the form of parameters and results of foreign evaluations. Through foreign evaluation a thread of control started at one agent can continue at other agent and thereafter come back.

Mobile Evaluation Plural agents are capable of transporting Noos methods (code) from one agent to another to be remotely evaluated. Transporting a method implies that the whole task/method decomposition of a method is transferred from an agent to another. Mobile evaluation is based on the notions of *data mobility* and *code mobility*. Data is exchanged among agents in the form of parameters and results of mobile evaluations. The code that composes a mobile evaluation is packaged up and conveyed from one agent to another.

By means of these capabilities, a Plural agent can delegate a task to any of its acquaintances (task delegation) or can send a method to solve remotely a task on its behalf to other agents (mobile problem solving methods).

Task Delegation Plural allows an agent to delegate another agent to carry out a task on its behalf. Thus, we talk about an *originator* agent meaning the agent who originates a task and about a *helper* agent meaning the agent to who a task is delegated. The foreign evaluation capability of Plural is what allows an agent to delegate a task to an acquaintance. In order to perform a foreign evaluation the *originator* has to indicate the helper agent's PSM that has to be applied to carried out the forwarded task. Task delegation enables an agent to share and reuse other agents' problem solving knowledge and experiential knowledge.

Mobile Problem Solving Methods Plural allows an agent to shift a task and the problem solving method to achieve that task to another agent[16]. The mobile evaluation capability of Plural is what allows a task and a method of that task to be dispatched from one agent (the originator) to a remote agent (the helper) on behalf of the originator. Using mobile PSMs an agent can share and reuse other acquaintances' experiential knowledge.

[5]The foreign evaluation and mobile evaluation capabilities of Plural are based on the notions of *foreign refinements*, *mobile refinements* and *alien references*. *Foreign refinements* allow an agent to remotely refine Noos terms owned by an acquaintance, *mobile refinements* allow an agent to refine Noos terms to be sent to an acquaintance, and *alien references* are the basic mechanism that allows Noos terms to be remotely referenced in a multi-agent setting. Readers interested in further details are referred to [15].

A Plural agent is an autonomous application developed in Noos which hold its own domain knowledge, problem solving knowledge, experience, and metalevel. In Plural an individual agent is built in the following four modules: a Noos interpreter; an interagent; a working memory; and a episodic memory. Several Plural agents can share the same interpreter but neither the same interagent nor the same working memory nor the same episodic memory. In order to share the knowledge stored in these memories, Plural agents have to deploy any of the cooperation modes explained below.

4 Cooperative Case-Based Reasoning

Communication and coordination constitute the basis for cooperation among Plural agents. Cooperation allows an agent achieve tasks which surpass its problem solving capabilities. Thus, cooperation can be thought of as an extension of agents' knowledge. In Plural, agents cooperate in order to share other agents' knowledge (problem solving, domain, episodic or metalevel knowledge). Cooperation in Plural is carried out based on the following conventions:

Willingness. We assume that all Plural agents in a specific domain application always are willing to cooperate with other Plural agents.

Faithfulness. A Plural agent will try to fulfil any commitment that it takes.

Sincerity. Plural agents will always tell what they believe.

Politeness. A Plural agent always acknowledges requests from other agents, that is, it does not ignore them. Thus, every time a Plural agent calls another Plural agent either it can establish a conversation or get a *busy* signal. Either agent will be able to terminate the conversation, but only after an agreement between both.

When a competent agent faces a (sub)task T which is inside its problem solving capabilities then it can perform such task alone. But in case that task T surpasses (is beyond) its problem-solving capabilities then it can cooperate with other agent(s) to try carry out that task. Therefore, that agent should decide:

- when cooperate, depending on whether a specific task is under its problem-solving capabilities or not;
- with whom cooperate, namely which agent/s is/are able to succeed in performing such task.
- how cooperate, that is to say which *cooperation mode* is suitable for successfully gathering necessary knowledge together to perform that task.

A competent agent $A_i\langle \mathcal{M}, \mathcal{A}, \mathcal{C}, \mathcal{S}\rangle$ can decide when cooperate thanks to its own competence self-models $\mathcal{S}$ which indicate its own suitability to achieve the task in hand. In case an agent does not feel competent to perform a specific

task, that is, its competence self-model indicates that such task is beyond its problem solving capabilities then such agent should find out the most competent agent to achieve that task by means of other agents' competence model $\mathcal{C}$.

A *cooperation mode* establishes the conventions that a couple of agents follow when exchanging knowledge. Thus, a cooperation mode expresses the actions that two particular agents must carry out in order to gather together the different elements required to perform a specific task. That is to say, a cooperation mode allows an agent (the originator) to reunite the set of knowledge pieces needed to construct an episodic model for a given problem with the help of an acquaintance (the helper). Different cooperation modes allow Plural agents to gather necessary knowledge together to perform specific tasks in different ways. For instance, several cooperation modes can be established depending on the following aspects:

- which agent originates the task; that is to say, if the task in hand is originated at one or another agent side.
- which agent owns the processor and the rest of computational resources needed to solve task T; in short, which computer machine is used.
- which agent owns the problem solving method used to solve that task;
- which agent owns the experience; in case a problem solving method uses experience acquired by an agent.

For instance, agent A who solves a specific task T in isolation accomplishes the following features:

Table 1: Agent A solve (sub)task T isolatedly

	Agent A	Agent B
Where task T is originated	$\surd$	
Who owns the Problem Solving Method	$\surd$	
Who owns the computational resources	$\surd$	
Who owns the experience	$\surd$	

Thus, a cooperation mode establishes how two agents must behave to accomplish a particular task. However, when a competent agent can opt for more than acquaintance to cooperate solving a specific (sub)task, then different *cooperation strategies* can be established for each cooperation mode depending on different criteria followed by the agent to solve such (sub)task. For instance, depending on how the set of helper agents chosen to cooperate is constructed and how this set is sorted to be traversed in search of a competent agent. In this way, it could be told that a cooperation mode settles how two agents cooperate whereas a cooperation strategy settles how more than two agents do.

In this way, the term *cooperative case-based reasoning* groups together the set of cooperation modes and cooperation strategies that can be deployed by a

collectivity of CBR agents wherein each CBR agent has its own base of preceding cases CB previously solved by itself.

Therefore, cooperation among CBR agents can be thought as an extension of agents' set of precedents, that is to say an expansion of the individual memory of a CBR agent to the memories of a collectivity of CBR agents. For instance, in CoDiT the retrieve task incorporates cooperation with other agents in order to find relevant cases known for other agents—i.e. to find the patient record most relevant to the current problem. Thus, the competent agents that constitute CoDiT are provided with a competence model on top of which two cooperation strategies each using a cooperation mode have been developed as explained in §4.1 and §4.2. Thereby, from now on we concentrate exclusively on the retrieve task and consider that the reuse task is always performed by the originator.

We propose two cooperation modes between CBR agents: Distributed Case-Based Reasoning (DistCBR) and Collective Case-Based Reasoning (ColCBR). Intuitively, both DistCBR and ColCBR are based on solving the retrieve task reusing the experiential knowledge (in form of cases) of other CBR agents.

DistCBR. An agent (the originator) delegates the retrieve task to another agent (the helper) indicating the helper's CBR method to solve such task.

ColCBR. An agent (the originator) forwards the retrieve task and the PSM of that task to an acquaintance (the helper). That is to say, the originator, in addition to the task, also conveys the PSM to solve that task.

Table 2: DistCBR for agents A and B and agent A's (sub)task T

	Agent A	Agent B
Where task T is originated	√	
Who owns the Problem Solving Method		√
Who owns the computational resources		√
Who owns the experience		√

Table 3: ColCBR for agents A and B and agent A's (sub)task T

	Agent A	Agent B
Where task T is originated	√	
Who owns the Problem Solving Method	√	
Who owns the computational resources		√
Who owns the experience		√

The difference between both cooperation modes basically resides in which agent owns the problem solving method to find the most relevant case (Tables 2 and 3 try to make clear this difference). In both cooperation modes helper's experiential knowledge is shared and then reused by the originator. However, whilst the DistCBR cooperation mode also allows helper's problem

solving knowledge to be shared and reused by the originator, using the ColCBR cooperation mode the PSM sent by the originator is shared by the helper to retrieve the most relevant case(s) that will be later reused by the originator. From the standpoint of implementing these cooperation modes, we can say that DistCBR is supported by the task delegation capability of Plural whereas ColCBR is supported thanks to the mobile problem solving methods capability of Plural. On the other hand, from an authority point of view, it can be said that using DistCBR the originator delegates authority to the helper to solve the task in hand. On the contrary, using ColCBR the originator maintains the authority, since it has fully control over the PSM applied, merely it uses the experiential knowledge of the helper.

4.1 Distributed CBR

The DistCBR cooperation mode enables an agent to share experiential knowledge acquired by an acquaintance by means of particular problem solving methods provided by this.

The DistCBR cooperation mode allows a CBR agent (the originator) to extend its base of precedents $CB_{originator}$ with the base of precedents CB_{helper} of an acquaintance (the helper). However, the way in which the base of precedents CB_{helper} is accessed—i.e. how precedents are identified and selected—depends exclusively on the helper's problem solving methods.

The following actions are performed by two CBR agents whilst cooperating using the DistCBR cooperation mode:

1. The originator asks the helper to solve (delegates) the retrieve task indicating which helper's problem solving method must be applied to solve such task.

2. On receipt of the task, the helper retrieves the most relevant precedent(s) using its corresponding retrieval method (as indicated by the originator).

3. Thereafter, the helper refers the available precedent(s) back to the originator which will have been inferred using its own (helper's) PSM.

In the DistCBR cooperation mode, a competent agent $A_i\langle\{\mathsf{DistCBR}\}, \mathcal{A}, \mathcal{C}, \mathcal{S}\rangle$ solving a specific (sub)task T could deploy different strategies to perform task T depending on which subset of acquaintances $\mathcal{A}$ agent A_i asks for help and in which order. For instance, CoDiT has a *cooperation strategy* that uses the competence models to decide to which agent and when it shifts the retrieve task. Thus, the retrieve task of a CBR agent shown in Fig. 1 is transformed to that of Fig. 2.

There are now two possible ways to solve the retrieve task: i) using method identify&select as before (it's the method owned by the agent that works on the episodic memory of that agent) or ii) using the DistRetrieval method that can delegate the retrieve task to other agents. There is a metalevel task (with respect to the retrieve task) that decides the base-level method to use; this task,

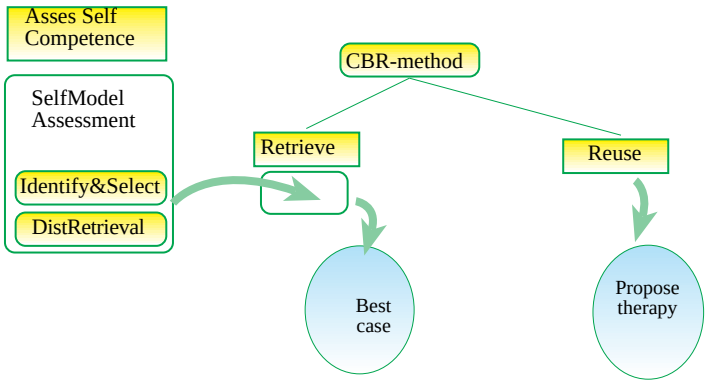

Figure 2: Task decomposition of the DistCBR cooperation mode for diabetes therapy. There is a metalevel task assess-self-model that decides whether to solve the retrieve task using the agent own method identify&select or the DistRetrieval method that uses task delegation to other agents.

called assess-self-model, has a (metalevel) method that uses the self-competence model to decide whether the agent is competent for the problem in hand or needs to ask other agents for help. As a result, this self-competence-assessment method decides that whether identify&select or DistRetrieval is the method to be used for the retrieve task.

The second part of the cooperation strategy is embodied by the DistRetrieval: choosing with which other competent agents to cooperate in order to retrieve a case that is similar to the current problem. The current strategy has several phases:

Phase 1 Search for relevant patient records from most competent agents.

1. Assess the competence degree of the competent agents using the competence models.
2. Select the agent A_j with higher competence degree and delegate to it the retrieve task; in this way A_j will use its own identify&select$_j$ method to retrieve the most relevant patient case from its case base CB_j.
3. If the helper agent cannot solve the task (no good enough case can be found in its case-base) try next best competent agent A_{j+1}; otherwise Phase 2 starts.

Phase 2 Assess result of helper agent. Firstly, notice that the criteria for deciding what is a "relevant" patient case is different for competent agent—this stems from the fact that each one uses a different method identify&select$_j$ for the retrieve task. Since the originating agent may have a very strict relevance criterion it is wise to assess *a posteriori* the result recommended by a helper agent.

1. Assess the most relevant case recommended by the helper agent against the originator agent relevance criteria.
2. If the case is good enough, the agent proceeds to the reuse task (see below).
3. Otherwise the case is cached (as the best case an agent A_j can give for the current problem).
4. Since, according to the originator the solution given by A_j is considered a failure, Phase 1 should be resumed and the next most competent agent is to be queried. However, if no competent agent remains then Phase 3 is entered.

Phase 3 Relaxation of relevance criteria. This phase arises when no competent agent can provide a case in a set of case-bases that is good enough for the strict relevance criteria of the originator agent. However, relevance criteria used for retrieval are just *a priori* assessments, and it may be possible that some of the cached cases (the "best cases" from the competent helper agents) may be good enough to solve the current problem on the reuse task (see below).

1. First, using some less relevance criteria the "best cases" cached from the helper agents are ranked.
2. The highest ranking cached case is selected and the reuse task will adapt its solution, if possible, to the current problem.

Reuse The solution of the best retrieved case has to be adapted to the current problem. In CoDiT domain knowledge is used to adapt, if possible, the recommended therapy to the current patient. However case relevance does not assure solution reuse, and when the solution cannot be adapted the retrieved case is also considered to be a failure and the agent resumes previous phases in search of a more adequate episode.

In this cooperation strategy, the originator delegates the authority solving the task retrieve whenever its self-competence model indicates that it is not so competent for the problem in hand.

4.2 Collective CBR

The CoICBR cooperation mode allows a couple of CBR agents to share experiential knowledge. By means of the CoICBR cooperation mode a CBR agent (the originator) can make use of the base of precedents CB_{helper} of an acquaintance (the helper). The way in which the base of precedents CB_{helper} is accessed, in this case, depends on the originator's problem solving method sent.

The CoICBR cooperation mode implies the following actions to be carried out between two CBR agents:

1. The originator sends the retrieve task to be solved and a originator's retrieval method to be applied to solve such task together to the helper.

2. On receipt of the task and the PSM, the helper retrieves the most relevant precedent(s) using the PSM received.

3. Thereafter, the helper refers the available precedent(s) back to the originator which will have been inferred using the originator's PSM method.

CoDiT has also been provided with a cooperation strategy based on the CoICBR cooperation mode to solve the retrieve task. A competent agent $A_i\langle\{\text{CoICBR}\}, \mathcal{A}, \mathcal{C}, \mathcal{S}\rangle$ deploying this cooperation strategy also uses competence models to decide when cooperate and with which agents.

The first part of this cooperation strategy coincides with the cooperation strategy explained above with the exception of the second way to solve the retrieve task. Since now the self-competence-assessment method can choose between the identify&select and CoIRetrieval methods. The CoIRetrieval method is a mobile problem solving method which shifts the retrieve task and the identify&select method of the competent agent to an acquaintance with the aim of using its precedents. Thus, in this case the retrieval task of a CBR agent is alike that shown in Fig. 2 except for the DistRetrieval method that has been changed for the CoIRetrieval method.

The second part of this cooperation strategy is much more straightforward than in the strategy explained formerly. On the one hand, Phase 1 has been modified slightly as explained below, Phase 2 and 3 have no sense in this strategy since a posterior assessment is not necessary to be carried out using the CoICBR cooperation mode, given that the identify&select method used by all involved competent agents will be the same. On the other hand, the reuse task remains just as above. Phase 1 have been changed as follows:

Phase 1 Search for relevant patient records from most competent agents.

1. Assess the competence degree of the competent agents using the competence models.
2. Select the agent A_j with higher competence degree and forward the retrieve task and the identify&select method to agent A_j in order to retrieve the most relevant patient case from A_j's case base CB_j.
3. If the helper agent A_j cannot solve the task (no good enough case can be found in CB_j) try next best competent agent A_{j+1}; otherwise the reuse task starts.

Reuse The solution of the best retrieved case has to be adapted to the current problem. As above when the solution cannot be adapted the retrieved case is also considered to be a failure and the agent resumes previous phases in search of a more adequate episode.

This cooperation strategy guarantees that when the originator shifts a task to an acquaintance it carries on maintaining the authority solving the retrieve task.

5 Related Work

In regard to *agent models* (representations of one-self and other agents in a multi-agent scenario), it can be told that the identification, design, and implementation of strategies for cooperation based on agent models have been key research issues since the early years of the field of Distributed Artificial Intelligence (DAI). Indeed, the Actors model of concurrent computation already introduces a simple representation of others. An actor during its *behaviour definition* is provided with a list of identifiers (*acquaintance list*) indicating the name and location of each acquaintance [3]. In other early work Cammarata et al. introduce different strategies of cooperation based on more complex agent models for the collision avoidance in the air traffic control (ATC) domain [9]. They present four distinct ATC systems, where each involved agent models destinations, speeds, fuel levels and so on of others what allows agent groups to solve shared tasks effectively.

The contract net protocol [26, 12] facilitates distributed control of cooperative task distribution (task-sharing) among the nodes of a distributed problem solver. In this approach, when a particular node lacks necessary knowledge to perform a task it announces (broadcasts) that task to the rest of nodes. Then, each node evaluates its own suitability to carry out that task and bids a figurative price (since each node is endowed with a strength parameter as the hypothetical capital for granting contracts or charging services) for that task. Afterwards, the originator chooses the most suitable among all bids submitted. A shortcoming of the contract net protocol is the communication overhead caused by broadcasting. Recent approaches—like addressee learning based on CBR [19]—have tackled this problem.

In ARCHON[30, 11], a development environment for building communities of cooperating heterogeneous systems, self and acquaintance models are used to store the domain-specific information that defines the individual behaviour of each member of the community. Both models include, at different details, agent's skills, current status, goals and so on. Other approaches such as GRATE [14] or CooperA[27] also provide an agent with representations of the capabilities, current status, and goals of other agents and itself allowing it to decide which tasks are performed locally and which are delegated.

A *twin-base* agent modeling approach have been proposed in the framework of the Virtual Secretary Project (*ViSe2*) [2]. The twin-base is composed of a *task-base* and *cooperator-base*: the task-base provides direct mappings between tasks and suitable expert agents to perform such tasks, whereas the *cooperator-base* collects stable information of the others and acts as an auxiliary base to the task-base. In addition, a capability revision process keeps the mapping information consistent [10].

As stated in the introduction, our approach focuses on cooperating peer agents which are not merely specialist at specific subtasks but are capable to solve the overall task in hand. This makes a difference from the aforementioned approaches since they rely on agents as specialists in a team what tends to a fixed division of labour, whereas we are addressing more flexible and dynamic

task/agent assignments.

In respect of *communication and coordination issues* with which Plural provides Noos, for several years, agent-based software engineering has faced the matter of enabling heterogeneous programs written by different people, at different times, in different languages and with different interfaces to communicate and interoperate [13]. Researchers in the ARPA Knowledge Sharing Effort have proposed agent communication languages (ACLs) as the means to allow the exchange of knowledge among software agents in order to facilitate their interoperation [13]. Generally, an ACL is composed of three main elements: an open-ended vocabulary appropriate to a common application area, an inner language (KIF—Knowledge Interchange Format) to encode the information content communicated among agents, and an outer language (KQML—Knowledge Query and Manipulation Language) to express the intentions of agents [18]. In our approach, two communication protocols have been devised (Plural-to-interagent and interagent-to-interagent) whose communication languages are based on KQML. Therefore, both agent-to-interagent messages and interagent-to-interagent messages are expressed as KQML performatives. Nevertheless, since we are dealing with homogeneous agents a translation phase of the inner language is not necessary, and given that we assume that all Plural agents taking part in a specific domain application share the same ontology, Plural agents do not need to allude to a concrete ontology to properly exchange knowledge.

Nowadays, KQML has become the communication language par excellence in agent-based systems. However, when several computational entities interact by exchanging messages a higher level of interaction concerned with the conventions that they share during the exchange should be addressed [8]. This level of interaction is not supported by KQML, whereas coordination languages—like COOL[8]—allow such conventions to be explicitly expressed. Making shared conventions explicit allows interdependencies among agents' activities to be managed.

Our approach is based on interagents [15, 17] which—likewise KQML facilitators[20]—are inspired by the *efficient secretary* metaphor already introduced in the Actors model of concurrent computation (as receptionists) [3]. Nevertheless, *interagents* (unlike KQML facilitators) offer the coordination level required by agents to cooperate in non-trivial ways. On the other hand, unlike KQML facilitators interagents have no knowledge (models) about the reasoning capabilities of their owners [25].

With regard to other approaches *placing CBR in multi-agent settings* [24], the CBR-TEAM [23] system allow cooperative retrieval and composition of a case whose subcases are distributed across several (specialist) agents in a MAS. A *Negotiation Retrieval* (NR) algorithm based on a distributed constraint optimization process is introduced to smooth inconsistencies by cooperatively retrieve cases while negotiating compromises to solve conflicts. Schematically, a complex query is presented to several agents. Then each agent is responsible for retrieving information related to a part of the query. Afterwards, agents negotiate to piece together an acceptable response. Again, agents' specialization makes a difference with our approach.

6 Conclusions

In this article, we have introduced a competent agent as an agent which is provided with a set of cooperation modes, a set of acquaintances, a competence model for each acquaintance, and a reflective competence model. Competence models indicate, for a given cooperation mode and a given (sub)task, the description of the problems for which a particular agent is competent as well as a performance evaluation of that agent. Using competence models a competent agent can decide when cooperate and with which acquaintance(s).

In this respect, our research has focused on *competent homogeneous peer learning* multi-agent systems. In short, systems wherein all involved competent agents use the same representation language and are capable of the task in hand—which is solved using knowledge acquired by learning from individual experience and specifically using case-based reasoning. We are investigating so-called *cooperative case-based reasoning*, i.e. different cooperation modes and cooperation strategies in agent societies of CBR agents wherein each agent owns an individual base of preceding cases CB. A cooperation mode establishes the conventions that a couple of agents follow when exchanging knowledge. For a specific cooperation mode and for a specific (sub)task different cooperation strategies can be established depending on how the set of competent agents chosen to cooperate is constructed and how is sorted to be traversed in search of a satisfactory solution for the task in hand.

Particularly, we are developing CoDiT a multi-agent system for therapy recommendation in diabetic patients, which lies in the framework of the SMASH project[1]. In this system each agent stores a case base of a particular doctor's patient records. Nonetheless, due to legal and deontological reasons patient data cannot be centralized. An ontology for this domain have been developed by the doctors involved in the SMASH project, what have allowed us to concentrate on communication and cooperation issues. Up to now, we have mainly focused on how the retrieve task of these agents can be upgraded with different cooperation strategies based on competence models. Thus, we have devised two cooperation modes—DistCBR and ColCBR—which enables an agent (the originator) to retrieve precedents owned by other agent (the helper). Using DistCBR the originator uses the helper's retrieval problem solving method to find relevant cases, whereas, using ColCBR the retrieval problem solving method employed belongs to the originator. That is, what basically makes a difference between both cooperation modes is which agent lets the problem solving knowledge to identify, search and select the most relevant precedent(s). In short, in both cooperation modes helper's experiential knowledge is shared and then reused by the originator. However, while the DistCBR cooperation mode also allows helper's problem solving knowledge to be shared and reused by the originator, using the ColCBR cooperation mode the PSM sent by the originator is shared by the helper to retrieve the most relevant case(s) that will be later reused by the originator. Furthermore, we have shown a cooperation strategy based on competence models for each one of these cooperation modes.

CoDiT is being developed using Plural [15], an extension of the Noos knowl-

edge representation language [4]. Noos have been designed to support knowledge modelling of problem solving and learning. Plural provides Noos with an agent communication language and an agent coordination language supported by autonomous software agents called interagents. An interagent intermediates the communication and coordination between an agent and the agent society wherein this is situated. On top of communication and coordination services provided by interagents Plural incorporates foreign evaluation and mobile evaluation of problem solving methods which, respectively, allow a Plural agent to delegate tasks and send mobile problem solving methods to other agents in order to solve problems on its behalf [16]. The task delegation capability of Plural underpins the DistCBR cooperation mode, whereas the mobile problem solving methods capability of Plural supports the ColCBR cooperation mode.

Finally, in our approach even if all agents involved in CoDiT at the beginning are provided with the same domain knowledge about diabetes domain, in the long term they will have disparate experiences, coming from their disparate memories and separate existence—since each agent will probably have been involved in solving different problems. Precisely, this *plurality* in agent's past experiences is what can give rise to prove the clear leverage of cooperation in order to share and reuse other agents' experiential knowledge.

References

[1] The SMASH project. http://www.iiia.csic.es/Projects/smash/.

[2] The Virtual Secretary project. http://www.cs.uit.no/DOS/Virt_Sec/.

[3] Gul Agha. *Actors, A Model of Concurrent Computation in Distributed Systems*. The MIT Press, 1986.

[4] Josep Lluís Arcos. *The Noos representation language*. PhD thesis, Universitat Politècnica de Catalunya, 1997.

[5] Josep Lluís Arcos and Ramon Lopez de Mantaras. Perspectives: A declarative bias mechanism for case retrieval. In *Proc. ICCBR-97*, volume 1266 of *Lecture Notes in Artificial Intelligence*, pages 279–290. Springer Verlag, 1997.

[6] 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.

[7] Eva Armengol and Enric Plaza. Induction of feature terms with indie. In *Proc. ECML-97*, volume 1224 of *Lecture Notes in Artificial Intelligence*, pages 33–48. Springer Verlag, 1997.

[8] Mihai Barbuceanu and Mark S. Fox. Cool: A language for describing coordination in multi agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

[9] Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 767–770, 1983. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 102–105, Morgan Kaufmann, 1988.).

[10] Wen Cao, Gheng-Gang Bian, and Gunnar Hartvigsen. Achieving efficient cooperation in a multi-agent system: the twin-base modeling. In *Proceedings of Cooperative Information Agents -DAI meets Database Systems*, number 1202 in Lecture Notes in Artificial Intelligence, pages 210–221. Springer, 1997.

[11] David Cockburn and Nick R. Jennings. Archon: A distributed artificial intelligence system for industrial applications. In *Foundations of Distributed Artificial Intelligence*. John Wiley and Sons, 1996.

[12] Randall Davis and Reid Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, January 1983.

[13] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM, Special Issue on Intelligent Agents*, 37(7):48–53, July 1994.

[14] Nick Jennings. *Cooperation in Industrial Multi-Agent Systems*. World Scientific, 1994.

[15] Francisco J. Martin, Enric Plaza, and Josep L. Arcos. Interagents: Providing knowledge representation languages with agent-oriented capabilities. 1998. Submitted.

[16] Francisco J. Martin, Enric Plaza, and Josep L. Arcos. Mobile problem solving methods in multi-agent systems. 1998. Submitted.

[17] Francisco J. Martin, Enric Plaza, Juan A. Rodriguez-Aguilar, and Jordi Sabater. Jim, a java interagent for multi-agent systems. 1998. Submitted.

[18] James Mayfield, Yannis Labrou, and Tim Finin. Evaluation of kqml as an agent communication language. In Michael Wooldridge and Jörg Müller, editors, *Intelligent Agents II*, pages 347–360. Springer Verlag, 1996.

[19] Takuya Ohko, Kazuo Hiraki, and Yuichiro Anzai. Addresse learning and message interception for communication. In Gerhard Weiss, editor, *Distributed Artificial Intelligence Meets Machine Learning. Learning in Multi-agent Environments*, number 1221 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1997.

[20] R. S. Patil, R. E. Fikes, P. F. Patel-Schneider, D. McKay, T. Finin, T. R. Gruber, and R. Neches. The darpa knowledge sharing effort: Progress report. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

[21] C. Pierret-Golbreich and E. Hugonnard. Organization and use of generic models based on the TASK language. In *EKAW'94*, 1994.

[22] Enric Plaza, Josep Lluís Arcos, and Francisco Martín. Cooperative case-based reasoning. In *Distributed Artificial Intelligence meets Machine Learning*, volume 1221 of *Lecture Notes in Artificial Intelligence*, pages 180–201. Springer Verlag, 1997.

[23] M. V. Nagendra Prasad, Victor R. Lesser, and Susan E. Lander. Retrieval and reasoning in distributed case bases. *Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries*, 7:74–87, 1995. Also as UMASS CS Technical Report 95-27.

[24] M. V. Nagendra Prasad and Enric Plaza. Corporate memories as distributed case libraries. In *Corporate Memory & Enterprise Modeling track in KAW'96, Tenth Knowledge Acquisition for Knowledge-Based Systems*, 1996.

[25] Narinder Singh and Mark Gisi. Coordinating distributed objects with declarative interfaces. In *Coordination Languages and Models*, number 1061 in Lecture Notes in Computer Science, pages 368–385. Springer, 1996.

[26] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[27] Lorenzo Sommaruga, Nikos M. Avouris, and Marc H. Van Liedekerke. The evolution of the coopera platform. In *Foundations of Distributed Artificial Intelligence*. John Wiley and Sons, 1996.

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

[29] 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.

[30] T. Wittig. *ARCHON: An Architecture for Multi-Agent Systems*. Ellis Horwood, 1992.