Using Cognitive Agents for building Pedagogical Strategies
In a Multi-strategic Intelligent Tutoring System

Claude Frasson
Université de Montréal
Département d'informatique et de recherche opérationnelle
2920 Chemin de la Tour
Montréal, Québec, Canada H3T 1J4
E-mail : frasson@iro.umontreal.ca
Tel : 1-514-343-7019 Fax : 1-514-343-5834

Abstract: acquiring and modeling knowledge in Intelligent Tutoring Systems (ITS) design is more complex than in most other knowledge based systems since ITS incorporate different kinds of expertise (knowledge on the subject matter, knowledge on the learner’s knowledge, pedagogical expertise, etc.) that evolve continuously. The pedagogical expertise, in particular, tends to be distributed among several components such as virtual learners and a tutor, which play different roles to enhance learning in collaborative mode. In addition, all these knowledge need to be improved in order to be adapted to the learner and the different learning situations. To foster such an improvement we introduced the principle of cognitive agents. A cognitive agent is a form of intelligent agent which is reactive, instructable, adaptive and cognitive. The property of instructability is particularly important since by allowing the instructional designer to feed the agent with new knowledge during its execution, it helps to solve the knowledge acquisition problem.

This paper describes the use of cognitive agents for implementing pedagogical strategies and more generally for detecting which strategy is more suited to a given learner. Our approach leads to the definition of a multi-strategic ITS, based on cognitive agents, that are able to switch among various strategies. In this framework we use cognitive agents to model the expertise of the various pedagogical strategies. We illustrate the paper with an example drawn from a small ITS in the domain of radiology which has been implemented using a cognitive agent platform.

keywords: cognitive agents, intelligent agents, multi-strategic intelligent tutoring systems, pedagogical strategies, knowledge acquisition.

Introduction
Intelligent Tutoring Systems (ITS) are complex systems that involve several different types of expertise: knowledge on the subject matter, knowledge on the learner’s knowledge, pedagogical expertise, etc. When building an ITS, even a small prototype, it appears that collecting this expertise is a time-consuming process that often requires many attempts before reaching a suitable system. A second problem is managing this knowledge so that they be adapted to the learning process. In other kinds of knowledge based systems, the classical approach for solving these problems is to separate the knowledge acquisition stage from the real use of the system. Our opinion is that this approach is unsuitable in our field. An ITS should always be considered to be in a constant evolution.

1 This paper resumes two papers produced at AIED-97 and ITS-98
Acquiring and structuring knowledge has an impact on most of the modules involved in an ITS, in which everything, from the curriculum to the pedagogical module, is affected. In order to implement modules which need complex and evolutionary expertise, we introduced the principle of actor [Frasson et al, 1996] [Mengelle and Frasson, 1996], [Frasson, Mengelle and Aimeur, 1997] a new form of intelligent agent that has cognitive properties. When we consider the pedagogical module, the interest for these cognitive agents becomes higher. In fact the evolution of learning approaches leads to a distributed principle of tutoring, with active implication of the learner in the knowledge acquisition process. This distributed approach is complex to realize, because the right combination of participants for an efficient learning is imprecise and depends on the learner too. It requires a more structured and organized framework and needs to be adapted to the learner. For that, it is necessary to learn more from the impacts of pedagogical actions. This double objective can be achieved with cognitive agents.

This paper shows how the characteristics of a cognitive agent are appropriate for handling the pedagogical expertise, providing both a flexible structure and a way to improve learning. We also describe the architecture of a multi-strategic ITS involving several cognitive agents and which is implemented in the context of the SAFARI project [Gecsei and Frasson, 1994]. We show an example of such an implementation for a pedagogical strategy called learning by disturbing [Aïmeur and Frasson, 1996]. An illustration of a cognitive agent platform is given in section 4 with an example drawn from a small ITS in the domain of radiology.

The need for a structured and improved pedagogical expertise

In the first versions of ITS, pedagogical expertise was centralized in a module that regulated the interactions between the learner and the tutor. The control of the training was assumed by the tutor, not the learner. More recent ITS developments consider a co-operative approach between the learner and the system [Chan and Baskin, 1990] using a co-learner or a learning companion who simulates the behavior of a second learner who would learn together with the human learner. Various alternatives to this co-operative approach were then conceived, leading more recently [Palthepu et al, 1991] to an inverted model of ITS called "learning by teaching" in which the learner could teach the learning companion by giving explanations. Another pedagogical strategy, called learning by disturbing, was recently introduced and experimented [Aimeur and Frasson, 1996]. Here the computer is simulated by two agents: a tutor and a troublemaker. The troublemaker is a particular companion who has a specific behavior: it will deliberately mislead the student to systematically test his self-confidence and his knowledge. The learner debates the solution with the troublemaker in a process controlled by a tutor. Should there be an impasse the tutor can intervene by giving a hint, the correct solution, etc.

The consequence of this evolution is that learning appears as a distributed process between several active components. If we consider that pedagogical expertise is complex, sometimes imprecise and difficult to evaluate, there is a need for a structured framework able to control the individual and social behavior of the different learners (real and virtual) and their interactions with the tutor.

To handle this situation we have defined the concept of cognitive agents which are intelligent agents with specific capabilities. Like other intelligent agents, cognitive agents are
autonomous entities that can operate without human control [Castefranchi, 1995] and interact with other cognitive agents using a communication language. To ensure this social ability, each actor needs some perception capabilities allowing it to react according to the activity of the others. Cognitive agents support two kinds of reactions: immediate responses to stimuli without reasoning (like in reactive agents), and controlled reactions that require planning, prediction and diagnosis capabilities [Morignot and Hayes-Roth, 1995].

The main characteristics that distinguish our cognitive agents from other intelligent agents lies in their evolution abilities. At the beginning of the paper, we noted the importance of knowledge acquisition and knowledge improvement in ITS. To address this problem we first decided to make cognitive agents instructable [Huffman, 1994]. We consider two kinds of instruction. Classical instruction which allows the designer to run the system and observe the agents' behavior, then to modify the agents' expertise in a cyclical process. In addition to this kind of instruction we want to allow the designer to feed agents with new knowledge during their execution. This dynamic instruction forms the main property of instructable agents [Huffman and Laird, 1995].

We also study how to provide the agents with self-improvement abilities. In particular, like adaptive agents, they have to adapt their perception and their decisions (reasoning methods, control strategy, ...) to the current objectives, available information, and performance criteria. In addition, in order to dynamically improve the behavior of the ITS, agents need to be cognitive. The cognitive aspect of an agent relies upon its capability to learn by experience and discover new facts or improve its knowledge for a better use. To sum up, a cognitive agent is an intelligent agent which is reactive, instructable, adaptive and cognitive capabilities [Frasson et al, 1996].

### Specification of cognitive agents

The architecture of a cognitive agent (figure 1) is general and can be applied to various types of expertise but its design is particularly adapted to the requirements mentioned above for the pedagogical expertise. They are realized by two main layers: the knowledge layer which includes the representations of all pedagogical knowledge types, and the cognitive layer that supports the abilities of the agent to learn.

The knowledge layer contains three modules: perception, action and control. The perception module recognizes typical situations in which it can intervene. Its role is to detect the opportunity of an intervention. The action module contains actions to be activated to react in a given situation\(^2\). The control module contains knowledge for answering the questions: should I decide to intervene? how? on what?... Its role is to decide according to an analysis of the situation. A combination of control and action corresponds to reactive instructional planning [Vassileva, 1995]. This layer requires both procedural and declarative knowledge.

\(^2\) Similar to a reaction to a stimuli
The *procedural knowledge* is represented by a set of *typical situations* (which act as *triggers*) and of *tasks*.

- The *typical situations* describe opportunities for the agent to intervene and are included in the perception module. A *typical situation* is a condition formulation based on the agent's environment (other agents' past actions, availability of information, etc.) which is permanently evaluated. When such a condition is verified it leads to the execution of the specific *task* associated with it (cf. following example of a troublemaker’s typical situation)

**Figure 1: Conceptual architecture of a cognitive agent**

TypicalSituation ReactionToLearnerAction  // Name of the typical situation
ALIAS TroublemakerChooseReaction  // Task to trigger
FOCUS { (Troublemaker at control tasks) , (Blackboard) }  // To evaluate the condition, this typical situation requires only
// to consider the blackboard and the troublemaker's previous behavior
EVALUATOR:
// This piece of code checks whether a diagnosis of the student's action
// is available and that the troublemaker has not already chosen a
// specific reaction
{ VAR diagnosis, notFirstReaction;
diagnosis = lookbb "RECENT_DIAGNOSIS";
if (diagnosis == nil ) then { return false; }
firstReaction = lookact for { "TroublemakerChooseReaction" };
if ( notFirstReaction == nil ) then { return true;} else { return false;}}

- The *tasks* support the agent's activity. We consider two main categories of tasks: *action tasks* (for instance, if we consider again the troublemaker action tasks will be: *display a message, mislead the learner*), and *control tasks* that support decision and call other tasks (for instance, *decide to mislead the learner*). The following example illustrates a control task of the troublemaker.

ControlTask TroublemakerChooseReaction ()
// This control task allows the troublemaker to choose its reaction
{ VAR level,which;
// Get the diagnosis of the student’s action in the blackboard
which = lookbb "RECENT_DIAGNOSIS";
// Request a service of another agent: the LearnerModelManager
level = (req "ReturnStudentLevel" with { which } to {"LearnerModelManager"});
// If the LearnerModelManager concludes the level of the student is good,
// the troublemaker chooses to mislead him, else it gives good advice.
if (level >= 3) then calltask "TroublemakerMislead" with { which }; else calltask "TroublemakerHelp" with { which };}

Some *declarative knowledge* is also generally useful to ensure the various tasks of the agent. This knowledge is described by a set of objects linked by binary relationships. The ontology
of the objects is defined at the same time as the cognitive agents. This is because the objects depend so heavily on the application. The objects are similar to data structures in that they are composed of several fields, each which may be either a basic data type (i.e. lists, string, integer, etc.) or references to other objects. These objects are used to store and transfer information. In particular the information is then used to help the cognitive agent plan its activities. For instance, the troublemaker uses various objects to record the student’s level, his motivation, etc.

The cognitive layer is essentially composed of cognitive tasks that support the self-improvement abilities of the agents (for instance, learn to decide when being wrong). It incorporates machine learning techniques to acquire more experience from the analysis of the student’s performance regarding different learning situations. It categorizes the different learners using conceptual clustering techniques and builds a case base of situations. Improvement consists in using case-based reasoning to adapt the situation to a new case. This layer is not absolutely required to ensure the actor’s basic functioning; that’s why we focus on the knowledge layer which supports pedagogical expertise.

Example of the learning by disturbing strategy
A strategy contains the following elements: a list of cognitive agents, the goal of the strategy, its conditions of applicability, the results of its application (post-conditions). To illustrate the previous architecture, we come back to the learning by disturbing strategy. Here, the learner is placed in a environment with two cognitive agents: a tutor and a troublemaker (simulated student). The learner and the troublemaker work together on a given problem under the supervision of the tutor. We have implemented a prototype applying this strategy to a medical domain: the diagnostic reasoning in mammography analysis (figure 2).

The goal of the learners team is to identify the features circled on the mammography by associating them with a finding.

The control of the session is shared between the cognitive agents and a session manager (this aspect is indicated on figure 3). The cognitive agents take all the pedagogical decisions while the session manager controls all that is directly related to the domain such as the interface of the problem and the interpretation of the learner’s actions. As the learner interacts with the interface, the cognitive agents must be informed of his progress in order to adopt a behavior (select a task to apply).
For example, the tutor can congratulate the learner after a successful step in the problem resolution, it can give him a hint, etc. Each time the learner acts, the sessions manager composes and send to the cognitive agents a diagnosis message describing the action and the knowledge elements that are evidenced by it. This information is then processed by the cognitive agents. Let us consider the state of the example described on figure 2; the diagnosis of the student’s action is available, so the ReactionToLearnerAction typical situation is triggered and the TroublemakerChooseReaction control task becomes active. Here, the troublemaker (see Pierre’s window) considers that the learner has a good knowledge level and calls the TroublemakerMislead action task in order to try to mislead him.

Integration of cognitive agents in a multi-strategic ITS

In the previous section, we considered only the learning by disturbing strategy. Using this strategy is not however suitable for all the various kinds of students. That’s why a modern ITS should incorporate several learning strategies. Each strategy has specific advantages and it appears useful to use adequately the strategy that will strengthen the acquisition process for a given learner. The selection of a strategy depends on several factors:

- the knowledge level of the learner: For instance, working with a companion would suit a learner whose knowledge level is low and who requires assistance or who needs to be motivated by cooperation. On the other hand, in a learning by disturbing approach the learner needs to defend his point of view. He must know how to retrieve information that has been previously stored. This strategy reinforces good learners but discourages poor ones.
• the domain: some strategies are more adapted to a domain than others. For instance to teach concepts one can use a directive approach while when teaching problem solving the learner may benefit from a learning by doing approach with a companion.
• the motivation: motivation of the learner may evolve with time. Varying the strategy used over time can maintain the learner’s interest, in particular it can be relevant to alternate between individual learning to group learning,
• the affective characteristics: one has to take into consideration different aspects of the learner such as self-confidence, learning style, emotional state (all of these also vary over time),

To dynamically consider all these aspects we need to define a multi-strategic ITS. This main principle led us to define the architecture depicted on figure 3.
The curriculum is an organized representation of the subject matter to be taught. At the very least, it is composed of objectives that the student is working towards (such as learning a particular concept or acquiring a specific skill) and of activities that the system can select, present and manage to help the student achieve those objectives. The activities are characterized by the objective(s) that they support and by the role that they play (i.e. lesson, demonstration, exercise, etc.). The session manager chooses what to teach (objectives of the curriculum to be reached) and which pedagogical strategy is to be used at a given time, based on the description of the applicability of the strategy and on the learner model. To choose the best strategy, the session manager uses a knowledge base which expertise stems from the above criteria (domain, motivation,...). Once selected, the pedagogical strategy is activated based on the cooperation of several pedagogical agents.

![Figure 3: Architecture of a Multi-strategic Intelligent Tutoring System](image)

The strategy editor is a knowledge elicitation tool able to help the expert in pedagogy to detail the characteristics of the agents and to define new strategies by combining new or existing agents (this editor is presently implemented in Java).

We have chosen to model the majority of these components as cognitive agents because they are designed to progressively acquire the necessary expertise. In particular, in the above architecture the pedagogical agents and session manager are cognitive agents since they contain complex pedagogical expertise subject to improvement. For instance, the session manager must take into account numerous
factors in the process of selecting a strategy and the complexity of this expertise prohibits a successful model on the first attempt. The capacity that the cognitive agents have to evolve enables us to start with a small body of expertise, coming first from the strategy editor and which is steadily enriched.

The previous architecture and the different properties of the cognitive agent paradigm leads to a development cycle able to contribute to the problem of pedagogical knowledge acquisition for the ITS. The cognitive level of agents allows to start with a small set of pedagogical knowledge and to progressively enrich their expertise by using a dynamic instruction process. The development cycle is the following: the expert in pedagogy uses the strategy editor to feed the cognitive agents, that are involved in the strategy he wants to develop, with an initial expertise (classical instruction). The strategy is experimented on a concrete example with a learner. Then, the session manager is able to test the efficiency of the strategy thanks to the post-conditions and the learner model and proposes to the expert in pedagogy a new set of selection rules for the strategies. In case of learning (the cognitive agent has changed its behavior) the session manager will inform the expert who will decide upon the validity of the suggestion. He will be able to eventually enrich the expertise using the strategy editor.

Example of learning in cognitive agents

In this section we reproduce part of paper [Mengelle, DeLean and Frasson, 1998] which deals with an experiment on how cognitive agents can learn. This learning process we used in this example is quite simple. A human expert, that will be next named programmer, runs agents on several problems. Then a learning algorithm will try to find what are the differences between the traces in cases of failure or success.

The programmer will interpret this result in order to improve the agents’ expertise (adding of modifying rules). The process will re-start until results become satisfactory. To summarize this situation: agents are in charge of the diagnosis and the reparation will be ensured in a distinct stage using a cooperation between the agent and the human programmer.

To avoid complexity, we decided to illustrate this process with an example which is outside of the ITS field. Two agents are trying to build words. The alphabet consists of 4 letters: red square, red circle, blue square and blue circle. The tracer chooses the form while the painter decides the color. They are not allowed to communicate and look only at the current state of the word to make their decision.

We define four basic rules:
1. A red letter must follow a square
2. A blue letter must follow two consecutive red letters
3. A circle must follow a blue letter
4. A square must follow two consecutive circles.

The tracer knows only about rules 1 and 2, while the painter knows rules 3 and 4.

It is therefore possible for our agents to come to a dead end. For instance, if the two last letters are red and the last one is a square, the painter can’t paint. The tracer has likewise problems. The agents aim to reach 20-letter words.
We are here mainly interested in the learning phase. The learning algorithm we used is a class regression tree. We feed it with a bank of examples. It returns a classification tree which will allow human experts to infer some new rules in order to improve the agents’ performance (see figure 4). Each example describes a previous sequence of interactions between the two agents in order to build a word and the result of this activity (assessment of the agents’ goals, here: to build 20-letters words respecting the basic rules). The basic description we have about agents’ activities is the trace of the activation of the various tasks. But, this trace first requires some processing before learning. We need to give some hints to focus learning. Here, the programmer may have the intuition that the number of blue circles is a key factor for success. Hence, it may be useful to examine the various traces of agents’ activities and count the number of occurrences of the tasks Paint-Blue and Draw-Circle. The indicators a programmer may want to focus on are quite independent from this example; they may concern for example the number of occurrences of a task, the number of occurrences of a task after another one, the presence of a specific pattern inside the trace… The approach we adopt to solve this problem is to supply the human expert with a set of heuristics that have been already used on other examples to extract pertinent indicators.

We promote a two-step process (see figure 4). Because a trace of tasks’ activation can be complex, the first step allows filtering it. We give the programmer various tools that allow him to hide some agents, some tasks or to consider only certain period of time. In the above example, the view we defined considers only two tasks for each one of the two agents. The second step concerns the extraction of
pertinent indicators from this filtered trace. Figure 4 presents some of the heuristics we defined; roughly, they compute the number of times a task occurred. In this example, we consider 48 indicators (see figure 4); one of them is the number of times the task Draw-Circle follows the task Paint-Blue (i.e. circle following a blue letter in the word).

<table>
<thead>
<tr>
<th>Original Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Original Tree Diagram]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tree 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Tree 1 Diagram]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tree 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Tree 2 Diagram]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tree 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Tree 3 Diagram]</td>
</tr>
</tbody>
</table>

Figure 5: Classification trees that result from learning (after about 2000 attempts)

At the beginning the choice is at random and about half of the traces are failures. Applying the learning algorithm tells us that the most discriminating factor (i.e. head of the original tree on figure 5) between failure and success is choosing a blue square after a red circle (this must happen at least 5 times to succeed). We consider that only one of the two agents can learn at each time; so, we make the Painter learn a new rule: “after a red circle, choose the color blue if possible” (this heuristic is added to the Painter expertise by the programmer).

With this new rule, we run the agents another 2000 times and we notice that results significantly decrease (a success each 8 attempts). The programmer interprets the new tree (tree 1) and adds a new heuristic in the tracer: “after a blue circle, choose a square if possible”.

This time, performance is improved and agents build 2 good words each 3 tries. Tree 2 leads to generalize the previous heuristic of the Tracer: "after a circle, choose a square if possible". The performance remains however unchanged. A new application of the learning algorithm (tree 3) doesn’t tell us any new information.
This basic example told us that a thing as simple as the trace of agents’ activity can provide a programmer interesting features in order to improve the expertise of a society of agents (here, the ratio of success grows from $1/2$ to $2/3$). In the beginning of the paper, we addressed the problem of the complexity of ITS expertise; we think that tools such as this learning algorithm can be useful in order to help human designers to improve the expertise of a society of cognitive agents.

Conclusions

The cognitive agent paradigm has been presented in order to facilitate the acquisition and modeling of knowledge, a problem which is still a crucial step in the design of ITS. We have also highlighted the importance of multi-strategic ITS. In such systems the necessity to acquire knowledge about multiple pedagogical strategies separately from the domain expertise acquisition poses an even greater problem. We try to solve the problem of knowledge evolution with the introduction of a cognitive layer allowing cognitive agents to progressively enrich their pedagogical expertise. A cognitive agent platform has been implemented in C++ and we used it to implement the radiology ITS described in the paper. We are currently implementing an authoring environment in Java ( prototypes of editors and observation tools are available for a Smalltalk version of the cognitive agent platform). Our research interests mainly focus now on improvement of the individual and the social behavior of the cognitive agents.

Acknowledgements:

The medical prototype has been developed at the University of Montreal by Daniel Leibu and Hugo Dufort. The radiological expertise was provided by the cognitive science team at McGill University based on the doctoral work of Roger Azevedo. Charles De Léan has implemented the C++ version of the cognitive agent platform we used for the example. We thank the Ministère de l’Industrie, Commerce, Science et Technologie du Gouvernement du Québec for his support of the project.

References


