Teaching and Learning with Intelligent Agents: Actors

Thierry MENGENELLE, Charles DE LÉAN, Claude FRASSON

Université de Montréal
Département d'informatique et de recherche opérationnelle
2920 Chemin de la Tour
Montréal (Québec) - H3T 1J4 - Canada
E-mail : {mengelle, delean, frasson}@iro.umontreal.ca
Tel : 1-514-343-6159 - Fax : 1-514-343-5834

Abstract
Various research projects suggest to use intelligent agents inside Intelligent Tutoring Systems (ITS). In the framework of the Safari project, we propose a new kind of agent (actor) to model the pedagogical expertise of ITS. We use actors to implement multi-strategic ITS. They allow to teach the student a domain using powerful cooperative pedagogical strategies. Such systems involve complex expertise that has to evolve continuously. To improve the knowledge acquisition process and to foster revision of expertise, we give the actors some learning abilities. The paper first describes how to use pedagogical actors for implementing ITS. This involves three main points: description of actors (properties regarding other intelligent agents, architecture, language, and advanced features such as personality or learning), protocol of communication with the other module of the ITS (including the pedagogical material) and definition of pedagogical material (representation of the domain). We take several examples to discuss the various advantages of actors according to the knowledge engineering point of view. The second part of the paper focuses on the learning abilities of actors. We discuss different learning approaches and illustrate one of them with an example involving two simple actors.

Topic
1. Pedagogical agents
2. Architecture for ITS
3. Teaching and learning strategies
4. Authoring shells and tools
1. Introduction

From the very beginning, artificial intelligence was a key element of the research on Intelligent Tutoring Systems (ITS). Most of the first ITS were built around expert systems. The drawbacks of classical expert systems (knowledge acquisition, revision of expertise,…) were even more crucial for ITS, due to their multiple expertise (domain, pedagogy, interface, …). For a few years now, another field of artificial intelligence has influenced the research on ITS: agents and multi-agent systems. Most conferences now propose workshops on pedagogical agents. Many reasons promote the emergence of this new field of research…

From a software-engineering point of view, agents have advantages of modularity and flexibility. Several works on artificial intelligence study the learning possibilities of agents, for instance the instructable agents [Huffman and Laird, 95]. Some aspects of agents also allow to improve knowledge acquisition, like those presented in the Disciple approach [Tecuci, 96]. Most recent projects on groupware reuse the agent paradigm. The emergence of Internet also promotes the use of agents as personal assistants (helping search, information filtering, …).

Regarding the field of ITS, we find various applications of agents: structuring the expertise of ITS [Costa and Perkusich, 97] [Morin and Lelouche, 97] [Ritter, 97], coaching [Lester et al, 97], animated characters [André et al, 97] [Elliot, 97], etc. Our research deals with these various aspects, but we mainly focus on agents for structuring the pedagogical expertise. The pedagogical strategies we are interested in are indeed fairly complex. They require the cooperation among several pedagogical actors: tutor, companion, troublemaker [Frasson et al, 96]. Regardless of the basic properties of intelligent agents, actors have also some advanced features. They are intended to model personalities and, as we will see in the paper, they exhibit some learning capabilities in order to improve themselves. We first present the context of this study: the Safari project. The first main part of this paper deals with the use of actors to model the pedagogical expertise of an ITS: we describe the basic characteristics of actors, the protocol they use
to communicate with the domain elements (exercises, simulation, problems, …) and the structure we define for these elements. Then, the second part studies one of the advanced features of actors: learning.

2. Using pedagogical agents in an ITS: The Safari example

In a Safari ITS, the session management module uses a global curriculum and an advanced learner model [Nkambou et al, 96] in order to determine the next pedagogical activity (called resource). Safari promotes the use of what we named adidactic resources. These resources are structured to store the subject of the activity (the domain) and let another part of the ITS make all the pedagogical decisions. The pedagogical module is viewed as a plug-in. It is independent of the resource and may be reused. Furthermore, this architecture allows using various pedagogical modules, each one related to a specific pedagogical strategy. The session management module chooses the right pedagogical module according to the learner model.

Figure 1: Using actors in the Safari ITS architecture
The pedagogical strategies, we are interested in are cooperative strategies such as *learning with a companion* [Chan and Baskin, 90] or *learning by disturbing* [Aimeur and Frasson, 96]. In these approaches, learning appears as a distributed process between several active components: humans and/or simulated modules. We implement each simulated module (for instance: a tutor, a companion) with a new form of agent named *actor* [Frasson et al, 96] [Mengelle and Frasson, 96]. The MATHEMA system [Costa and Perkusich, 97] also promotes the use of a society of artificial agents; the work of [Morin and Lelouche, 97] stems from this same approach. Besides the granularity of agents (in our approach an agent is not a tutoring function but a pedagogical actor which can run different functions), the main difference lies in that actors have been designed to evolve easily. That’s why we are interested with related work about learning algorithm and instructable agents [Huffman and Laird, 95]. This is important, because when building an ITS, it appears that collecting the pedagogical expertise is a time-consuming process that requires many attempts before reaching a suitable system. This expertise is even more complex when addressing cooperative strategies; collecting it requires fine studies of interactions like these reported by [Dillembourg et al, 97].

Due to its modularity and the learning possibilities we foresee, the actor approach is particularly suitable for building components involving a complex expertise which may vary over time (for instance, pedagogical modules on figure 1). Managing a tutoring session requires taking into account various parameters such as the available time, the student’s favorite strategies, the kind of content, the various pedagogical material presently available, etc. The complexity of this expertise leads us to also encode the session management module of with actors. However, in this paper, we focus on the dialog protocol between actors and resources. We first briefly discuss the basic properties of these intelligent agents; for details the reader may refer [Frasson et al, 96] or [Mengelle and Frasson, 96].
3. The actor paradigm

3.1. General features

In previous papers, we defined an actor as an intelligent agent which is reactive, instructable, adaptive and cognitive [Frasson et al, 96]. A simplest definition actor is to consider only its two main properties: reacting to the activity of the others and learning abilities.

The first property stems from the combination of three approaches: reactive agents, deliberative agents and interacting agents [Müller, 96]. To ensure these three kinds of reactions, we define a modular architecture depicted in [Frasson et al, 96] and summarized in the bottom-right part of figure 4. Reactivity results from a direct association between perception and action modules (the detection of a typical situation can directly trigger an action task). The third module (control module) supports deliberation and the abilities for social behavior. This module involves several control tasks that are in charge of planning, decisions and invoking services of other actors. Regarding the last point, we propose two general protocols for communication/cooperation among agents: direct request of a specific service of a given actor\(^1\) and broadcasting of a request to the whole society using the blackboard structure\(^2\).

The second property (learning) that forms the end of the paper is ensured via a cognitive module.

3.2. Language for building actors

The actors' language is an interpreted language, which allows describing an actor as a set of typical situations that trigger some tasks. To describe the language, we take the

---

\(^1\) In figure 4, the arrow between ‘pedagogical actor 1’ and ‘dialog actor’ illustrates this communication.

\(^2\) In the same figure, the ‘dialog actor’ broadcasts a message to the whole society.
example of the definition of a task that the Companion actor triggers when the learner asks for help³.

When this task is triggered, it first dialogs with the resource in order to determine the kind of help the actor may provide to answer the student’s request. The complete dialog protocol between actors and resources will be discussed in the next section. Figure 2 shows that this communication is done through an exchange of structures via the blackboard⁴. In order to keep the example simple, we consider here a simplified version of what is really exchanged; this structure consists of only two fields: one contains a hint, the other contains the complete answer in text form.

Figure 2: The actor language: example of a task definition

³ This example is related to the learning with a companion strategy which involves three actors: the human learner and two computerized agents (a teacher and a companion).
⁴ The blackboard is accessed through 3 main primitives: putbb something as identifier, lookbb identifier and getbb identifier which also removes the keyed data from the blackboard.
To perform a task, an actor generally calls several other tasks. Here, we consider that the Companion calls the task `shouldGiveAnswer` (primitive `calltask`) in order to decide whether to give the complete response or a simple hint. Because some of the tasks can take a few seconds to execute, we want to allow executing them while waiting for the learner. The new task will be executed concurrently and the calling task will get an id of the task. With that id, the calling task may check if the task is over (with the primitive `existreply`), or simply wait after the result with the `getreply` primitive.

A task may request a service of another actor through the `req` primitive whose grammar is very similar as the one of `calltask`. If the requested actor accepts to give the service (it can refuse), it will start a task and give its result. In the example, the Companion requests the Teacher to give a hint (`GiveHint`) when the learner refuses its help. The other way an actor can use to ask services of other actors is to broadcast a request in the blackboard.

![Diagram](image)

Figure 3: Linking actors to other modules

Actors can also communicate with other modules using a set of primitives that are stored in a global library. A programmer will update this library each time a new service will
become available (see figure 3). The task we take as example provides examples of such primitives:

- **getstudentinfo** access the learner model (in this case the self-confidence of the learner will be used by the `shouldGiveAnswer` task).
- **ask**, **say** and **setexpression** (change the facial expression) concern the actors’ interface.

### 4. Managing the dialog between actors and resources

![Figure 4: Managing the dialog between a resource and a society of pedagogical actors](image-url)

- **Session Management Module**
  - A resource to be loaded

- **Communication Protocol**
  - **Resource ➔ Dialog actor**
    - SendDescription (<resource-type>, <local-domain-base>)
    - Inform (<diagnosis>, <possible-reactions>)

  - **Dialog actor ➔ Resource**
    - Load-Resource (<resource-id>)
    - ApplyDecision (<reaction>)

- **Services of the other modules...**
  - `SendDescription` with `Resource`
  - `Inform` with `Diagnosis` and `Possible-reactions`
  - `Load-Resource` with `Resource-id`
  - `ApplyDecision` with `Reaction`

- **Available services for other actors**
  - `<diagnosis>` = (<situation-id>, <partial learner model>)
    - `<situation-id>` = 'true', 'false', 'partial', 'empty', ...
    - `<partial learner model>` = set of (<knowledge-id>, <status>)
      - `<status>` = 'ok', 'not ok' or <error-id>

- **<reaction>** = (<reaction-id>, <list of parameters>)
  - `<reaction-id>`
  - `<list of parameters>`
  - `<positive-hint>`
    - Display a general hint about `<knowledge-id>`.
  - `<negative-hint>`
    - Display a message which aims to mislead the student about `<knowledge-id>`.
  - `<error-id>`
    - Display the answer a learner with the `<partial learner model>` would probably make.

These are only few examples and the list will grow over the time...
Resources can have different internal structures, but in order to dialog with pedagogical actors, they have to respect a specific protocol. This protocol, which is detailed on figure 4, involves four kinds of messages. First of all, regarding the decision of the session management module, a pedagogical actor calls the loading of a given resource. The resource answers by giving some static information (for instance its type (e.g. MCQ, HTML page, ...)). Then after each interaction with the learner, the resource informs actors about what presumably happened (diagnosis of student action) and what are the possible reactions. The diagnosis describes the global situation (is it a good answer, a partial answer...) and specifies a partial learner model which is supposed to explain this situation. We defined a very simple structure for this model: the status of each knowledge involved in the pedagogical activity (basic status are: mastered, not mastered or identification of a frequent mistake). The possible reactions describe what the resource is able to do at this point (for instance it may: reformulate the question, give a correct hint about <a-knowledge-id>, try to mislead the learned with a frequent error about <a-knowledge-id>, display the correct answer, etc). Then, the actors use their own pedagogical expertise to choose one of the possible reactions and then inform the resource (ApplyDecision).

The main principle which underlines the previous protocol is that the resource is in charge of the diagnosis of student knowledge and informs the actors about what they can do next. So, clearly, the global quality of the ITS also depends on the expertise of the resources. But one of the main benefit of this approach lies in that a designer can easily build an ITS using very simple and unsophisticated resources; then make it evolve by integrating more complex and powerful resources when needed (for instance, when a part of a course appears to be complex). To illustrate this point, next paragraph will take the example of two kinds of resources.
5. Modeling the domain: the resources

Figure 5 presents an adidactic resource which has been specifically designed for an ITS in the field of radiology. Here, we take the example of the *learning by disturbing* strategy [Aimeur and Frasson, 96] where the computer simulates two agents. The *troublemaker* is a companion who 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 the *tutor*.

The learner determines a possible identification of the feature that he has circled on the mammography. The resource diagnoses that this is a correct solution and sends this diagnosis to the actors (cf. protocol). The dialog actor (*resource manager*) receives this diagnosis and put it in the blackboard. The troublemaker then decides to mislead the
learner and changes its facial expression (see Pierre’s window on figure 5). In the present state, the expression an actor will adopt results from a sequence of tasks activation (e.g. in the code described in figure 2, the actor *decides* to be happy or to be annoyed). The emotional state that the actor will show results from its activity. In our case, we think that the emotional state of the agent should also explicitly influence its activity. We aim to add actors explicit personality. The influence of personality on agents’ activity and on emotional state forms another advanced feature of actors. To deal with this point we are presently interested in the virtual theater project [Doyle and Hayes-Roth, 96] [Rousseau and Hayes-Roth, 97], the animated characters PPP-Persona [André et al, 97] and Herman-the-bug [Lester et al, 97], the work of [Elliot, 97] on affectivity… Detailing this aspect is however outside the scope of the paper.

The adidactic resource described above was time-consuming to implement because of the complex and domain-related algorithms that is uses for diagnosis. It appears however that most of the elements of a domain doesn’t require such specific diagnosis. To illustrate this point, we take the example of a classical tool for knowledge assessment: MCQ. This well-known tool for student’s evaluation allows a very straightforward diagnosis, and hence fits our problematic of quickly building efficient resources.

<table>
<thead>
<tr>
<th>Question-ID</th>
<th>&lt;xxxx&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related knowledge</td>
<td>&lt;capital-id&gt;, &lt;communism-id&gt;</td>
</tr>
<tr>
<td>Question</td>
<td>Choose the towns that are or have been capital of communist countries: 1 – Rome 2 – Moscow 3 – Stalingrad 4 – Bucarest</td>
</tr>
<tr>
<td>Refinement of the question</td>
<td>Null</td>
</tr>
<tr>
<td>Refinement of the answers</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Answer</td>
<td>Knowledge ok</td>
</tr>
<tr>
<td>1</td>
<td>&lt;capital-id&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;capital-id&gt;, &lt;communism-id&gt;</td>
</tr>
<tr>
<td>(...)</td>
<td></td>
</tr>
<tr>
<td>Hints on knowledge</td>
<td>True/False</td>
</tr>
<tr>
<td>&lt;capital-id&gt;</td>
<td>true</td>
</tr>
<tr>
<td>&lt;communism-id&gt;</td>
<td>true</td>
</tr>
<tr>
<td>&lt;communism-id&gt;</td>
<td>false</td>
</tr>
</tbody>
</table>

Figure 6: A generic structure to implement actors’ manageable MCQ
The figure 6 describes a very simple structure allowing a designer to easily implement actors manageable MCQ. Each MCQ consists of four main parts:

- an **identification part** with an identifier and a list of the piece of knowledge that are useful to answer the question (here, the Capital and Communism paradigms),
- a **subject part** which specifies the text of the question and, eventually, some other texts to display additional information about the question or the possible answers,
- a part for **answer processing** that allows the diagnosis of the student’s knowledge,
- and a **set of hints** related to the various knowledge elements that are involved in this question (wrong hints allows the troublemaker to mislead the learner).

To ensure answer processing, we take a simple approach. The designed links each possible answer with a set of mastered knowledge and with a set of not-mastered knowledge. For instance, in the above example, the fact that the student chooses Rome as a capital of a communist country may reflect a good master of the *Capital* paradigm and some problem about *Communism*. Considering this structure, when the student chooses an answer, the resource can establish a diagnosis in a very straightforward fashion.

To illustrate this approach, let us assume the learner answer the first choice. The resource sends the actors its diagnosis (the student masters `<capital-id>` and has problems on `<communism-id>`) and the possible reactions it can ensure (detail of the answer 1 or 3, positive hint about `<capital-id>` or `<communism-id>`, wrong hint on `<communism-id>`). Actors choose one of these possible reactions and inform the resource of their choice. With this simple example, one can see that if some of the elements of the previous structure are not specified, the resource will offer fewer potentialities but the whole system will still work.

---

5 The real structure we used to implement MCQ is a little more complex and allows a fine classification of kinds of errors. In this paper, however, we avoid some details and take a n over-simple example.
6 In the real structure we go beyond to this duality mastered/not mastered with the introduction of the notion of frequent errors.
We designed editors allowing a designer to easily fill in this structure in order to implement MCQ. Initially, we designed the structure to allow a designer to implement MCQ that already exist in his mind. It appears however that this structure can became a useful tool to help a teacher to design pedagogical efficient MCQ. When designing an exercise, the author first specifies the various elements of knowledge that are useful to solve it. Thus, in the above example, we may easily suggest the author to define a new answer that would reflect some problems on \textit{<capital-id>} while traducing a good mastering of \textit{<communism-id>} (e.g. Stalingrad). So, we aim to implement a coaching system to make the designer’s work more efficient.

Regardless of the definition of resources, the other point on which we want to make the designer’s work easier concerns the definition of new pedagogical strategies. Because the expertise of the strategies we foresee is quite complex, we promote an incremental development. With respect to this issue of evolving systems, we decided to add the pedagogical actors some learning abilities. The last part of the paper deals with this point.

6. Learning in pedagogical actors

6.1. Basic learning features of actors

Improving the expertise of an actor requires two distinct tasks: diagnosis of a problem then revision of the actor expertise in order to solve it. The diagnosis stage aims to check that the behavior of the actor remains in accordance with its goals. Each actor considers two distinct global goals: its individual goal (e.g. the aim of the Troublemaker is to mislead the learner) and the collective goal of the society (the goal of the whole learning by disturbing strategy is to help the student to learn and to improve his self-confidence). In most cases, these goals can be divided in several subgoals. The aim of the diagnosis stage is hence to check every subgoal and to notify the possible problems. Once a subgoal failure is detected, the revision stage aims to modify the expertise of the actor in order to improve its future behavior.
Each of these two tasks (diagnosis and revision) can be ensured whether by the human expert, whether by the actor or using a cooperation between these two partners. In the instructable agent approach [Huffman, 94] [Huffman and Laird, 95], the actor is in charge of diagnosis and dialogs with the human to improve its expertise. We also promote another kind of instructability where the human can interrupt the actor in order to provide it with some new knowledge. This process of dynamic instruction is described in [Mengelle et al, 97]. Besides instructability, we also study situations where actors can do both diagnosis and reparation, using some learning algorithm. We design an environment which offers all these forms of learning. The choice of a given learning approach will depend on various parameters such as: the availability of the human expert, or the expected performance… Another key factor will influence the kind of learning: will the reparation occur during the actor’s activity (like in classical instructable systems) or after it (during a specifically dedicated learning stage)? To summarize, a learning approach is characterized by the entity (actor, human, or both entities) that will be in charge of each stage (diagnosis or revision) and by the time where the revision will take place (during the activity of actors or after it).

Before describing a concrete example, let us examine some tools an actor may use to ensure the diagnosis or revision stages. When we want actors to ensure diagnosis, we specify some functions that allow them to assess their goals (individual and collective goals). One of the main sources of knowledge these functions may use is the learner model, which generally reflects the influence of the actors’ activity on the student. When dealing with reparation, we aim to supply actors with different kinds of learning algorithms. Next paragraph illustrates the use of a classification learning algorithm in order to improve the expertise of two simple actors.
6.2. A first experiment

This learning process we used in this example is quite simple. A human expert, that will be next named programmer, runs actors 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 actors’ expertise (adding of modifying rules). The process will re-start until results become satisfactory. To summarize this situation: actors are in charge of the diagnosis and the reparation will be ensured in a distinct stage using a cooperation between the actor and the human programmer.

To avoid complexity, we decided to illustrate this process with an example which is outside of the ITS field. Two actors 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 actors 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 actors 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 actors’ performance (see figure 7). Each example describes a previous sequence of interactions between the two actors in order to build a word and the result of this activity (assessment of the actors’ goals, here: to build 20-letters words respecting the basic rules). The basic description we have about actors’ 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 actors’ 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 7). 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 actors, 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 actors. The second step concerns the extraction of pertinent indicators from this filtered trace. Figure 7 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 7); 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).

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 8) 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 actors 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 actors 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”.
Teaching and Learning with Intelligent Agents: Actors.

Figure 8: Classification trees that result from learning (after about 2000 attempts)

This time, performance is improved and actors 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 actors’ activity can provide a programmer interesting features in order to improve the expertise of a society of actors (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 pedagogical actors.

Conclusion

We have first introduced the actor paradigm to allow Safari ITS to foster learning using various powerful cooperative strategies such as learning by disturbing. After having described the basic features of actors, we detailed the communication protocol that actors use to dialog with the resources. We have particularly discussed the advantages of this approach using the knowledge engineering point of view. Actors can manage complex and powerful resources but we also promote the use of generic and quickly implemented resources such as MCQ. In the last part of the paper, we have discussed another interesting feature of actors: learning. Several approaches have been presented. The example that we described shows that learning from the trace of actors’ activities can help a designer to improve the expertise of an ITS. Our work on learning is still ongoing. We are also presently dealing with the personality and affectivity issues. An editor and a coaching system for building pedagogical efficient MCQ are also under development.

Acknowledgements

This research has been supported by the MICST (Ministère de l’Industrie, Commerce, Science et Technologie du Gouvernement du Québec) under the Synergie program. We also thank the CRSNG for his financial support. We also would like to thank two M.Sc. students. Daniel Leibu worked on actors and resource design. With Roger Azevedo, they designed and implemented the ITS in the field of radiology.
References


