PEGASE: A generic and adaptable intelligent system for virtual reality learning environments

Cédric Buche, Cyril Bossard, Ronan Querrec, Pierre Chevaillier

PEGASE: A Generic and Adaptable Intelligent System for Virtual Reality Learning Environments

Cédric Buche, Cyril Bossard, Ronan Querrec and Pierre Chevaillier

European Center for Virtual Reality, Université Européenne de Bretagne,
Ecole Nationale d'Ingénieurs de Brest, Laboratoire d'Informatique des Systèmes Complexes,
Technopôle Brest-Iroise, 29283 Brest, Cedex 3, France

***Abstract*— The context of this research is the creation of human learning environments using virtual reality. We propose the integration of a generic and adaptable intelligent tutoring system (Pegase) into a virtual environment. The aim of this environment is to instruct the learner, and to assist the instructor. The proposed system is created using a multi-agent system. This system emits a set of knowledge (actions carried out by the learner, knowledge about the field, etc.) which Pegase uses to make informed decisions. Our study focuses on the representation of knowledge about the environment, and on the adaptable pedagogical agent providing instructive assistance.**

***Index Terms*— Virtual Learning Environment (VLE), multi-agent system, Intelligent Tutoring System (ITS), classifiers system.**

I. INTRODUCTION

Many fields of learning, like driving or professional training for firefighters, for instance, require learners to experience the setting in which they will work or operate. The learners must therefore acquire not only knowledge, but real, hands-on skills. Virtual environments (VE) immerse learners in such situations. Fig. 1 gives three examples of a road safety application (`AReViRoad`) [1], a SEVESO plant application [2] and `Gaspar` for logistics on aircraft carriers [3].

Fig. 1. From left to right: screenshots from the AReViRoad, Virthualis and Gaspar applications.

This work is designed to teach decision-making in VE. Tutoring systems to instruct learners and assist instructors already exist [4, 5], but are dedicated to a specific VE. In this paper, we propose an independent VE tutoring system called `Pegase`, in the field of procedural and collaborative work.

II. CONTEXT: ACQUISITION OF SKILLS USING VIRTUAL ENVIRONMENTS

Traditionally, most training programs aim to transmit knowledge. However, to facilitate the acquisition of knowledge, we must build on our prior knowledge and skills. In this context, we propose the use of Intelligent Tutoring Systems (ITS) in which this knowledge is used in conjunction with the training setting. In this case, knowledge can be manipulated, e.g., to automatically question the learner. Being competent does not only mean having acquired knowledge, but also *being able to use that knowledge*. In order to facilitate the acquisition of knowledge, we must provide the learner with the right setting. To this end, we suggest using interactive systems by which the learners can be immersed in VEs in which they can make trial attempts, take initiatives, make mistakes and try again in a similar situation (which may not be possible in reality). The simulation therefore provides an environment common to the learner, the instructor and to the skill to be acquired. It mediates the learning relationship (learner-skill) as well as the instructive relationship (instructor-learner). Thus, computer-generated simulations, combined with an ITS, create an opportunity to improve learners' skills by associating knowledge with the possibility of putting their skills into practice.

ITS have already been used without being associated with virtual reality. As [6] has shown, they usually conform to one of four models. The first, known as the domain model, contains a representation of the knowledge linked to the skill to be acquired. ITS also use a learner model which defines the learner's personal characteristics and ascertains the condition of the knowledge at a given moment. Using the domain and learner models, an ITS can evaluate the knowledge acquired by learners by comparing their activity with information about the field. However, the main objective of the ITS is to provide appropriate assistance to the learner or the instructor, depending on the setting (following activities or offering assistance). In this context, the pedagogical model can be used to make choices with regard to the training objective, with the aim of facilitating learning. Finally, an interface model is used to exchange information between the system and the user. Until now, this

Manuscript received on March 30,2010.
E-mail: cedric.buche@enib.fr

model has not been reified[1] in existing VEs designed for learning.

Within the context of our VE, we consider an ITS as a system which is part of the human Virtual Learning Environment (VLE). We propose to evaluate the extent to which ITS are integrated within existing VLE. We have grouped VLE into three categories:

1. VLE as conventional simulators

This first category includes those applications which include none of the four models, such as an application designed to assist in both maintenance and control of mobile cranes [7]. In this kind of VE, the system provides no explanations about the task to be performed, which would require a domain model. The environment is therefore unable to adapt to the learner, as this would require a learner model. Finally, the teaching method is the instructor's responsibility. This sort of system is not able to make decisions regarding instructive interventions, however, it can help learners to improve or modify pre-existing skills.

2. VLE with domain and/or learner models

This second category of VE is made up of applications which include a domain model and/or a learner model [8]. The most well-known example of this type of VE is `Steve`, a virtual character who assists in both teaching and learning procedural tasks [5]. Using the domain model, Steve can demonstrate and explain the procedure and above all, verify the learner's actions. However, `Steve` intervenes on demand. He is incapable of knowing when, how and why to intervene, which would require a pedagogical model. In a system such as this it is possible to acquire skills, but the participation of the instructor is still required for all pedagogical interventions.

3. VLE with domain, learner, and pedagogical models

This final category groups together the VEs presenting not only domain and learner models, but also a pedagogical model [9]. Let us examine the example of the educational agent, Hal, from the `Fiacre` system [4]. The application is designed to instruct individuals in learning to drive TGV high speed trains, using virtual reality (intervention on railways). As well as having all of `Steve`'s abilities, `Hal` assists the instructors in structuring the pedagogical discourse. In concrete terms, each anticipated behavior corresponds to a different instructive assistance (additional information, explanation of an object, etc.). The instructor must therefore list the possible errors for each piece of knowledge to be acquired. Furthermore, for each of these errors, the instructor must specify the way in which these pedagogical strategies should be conducted through instructive assistance, and furthermore must do so for each exercise. The main advantage of this kind of VLE lies in the assistance to the instructor in terms of the educational relationship linked to the learner, and in the didactic relationship linked to the skill to be learnt. However, the instructor must specify all of the knowledge to be acquired for each exercise.

Thus, most VLEs only include representation of the knowledge about one specific domain. Systems proposing a diagnostic component only rarely provide a mechanism for instructive assistance. Hal seems to us to be the most successful of these systems. However, the instructor must still make a list of the possible errors and specify the educational strategies for each exercise. Furthermore, the impact of the instructive assistance on the learner is not taken into consideration. In concrete terms, any proposed assistance which does not help the learner to make progress will be updated each time that specific situation occurs.

In order to resolve these shortcomings, we propose the integration of an intelligent tutoring system within a VE. This system must propose a flexible pedagogical model, i.e. a model in which instructive concepts can be easily added, modified or deleted. Furthermore, a model such as this must be *generic*, insofar as the pedagogical model must be exploitable independently of the task to be performed. Finally, the knowledge of the pedagogical model, along with its past experience, could be used to automatically suggest the appropriate interventions by taking into account both the learner and the context of the simulation: the system therefore becomes *adaptive*. Our model is called `Pegase` (PEdagogical Generic and Adaptive SystEm).

In the next section, we will describe the global architecture of `Pegase`. We will then go on to present our domain model (see section 4) and a description of our pedagogical model (see section 5), followed by a discussion of the advantages of our proposed models (see section 6). It must be noted that the proposal described here is applicable within the context *of the learning of procedural and collaborative tasks* and cannot be used in general learning situations.

III. PROPOSING AN INTELLIGENT TUTORING SYSTEM : PEGASE

Our proposal consists of reifying the four classic ITS models (domain, learner, pedagogical, interface), within a VE. We believe that errors can provide crucial information and thus decided to introduce a model called "error model". It is through the use of this new model that we will be able to generalize (something Hal could not do). Furthermore, we have added an "instructor model", in which the instructor specifies the knowledge about the exercise to be performed. The instructor defines the guidelines which describe the procedure(s) to be carried out and the role(s) played by the learner (and consequently those which must also be activated automatically).

These models must provide solutions to counter the shortcomings of the existing systems described above and must therefore display two important characteristics: genericity and adaptability. We thus suggest that it is possible to incorporate a generic and adaptive ITS from a VE by reifying the 6 ITS models. So that each model can share its information and conduct its analyses autonomously (independently of both the situation and of other models), an autonomous entity (known as an agent) is associated with each model.

The agents interact by exchanging messages containing data (see Fig. 2). This data can be extracted from the situation or inferred from the agent's internal reasoning using its knowledge (the model to which it is linked).

[1] Reification is a process through which concepts are explicitly represented by semantic representation (classes) to conceptual manipulation

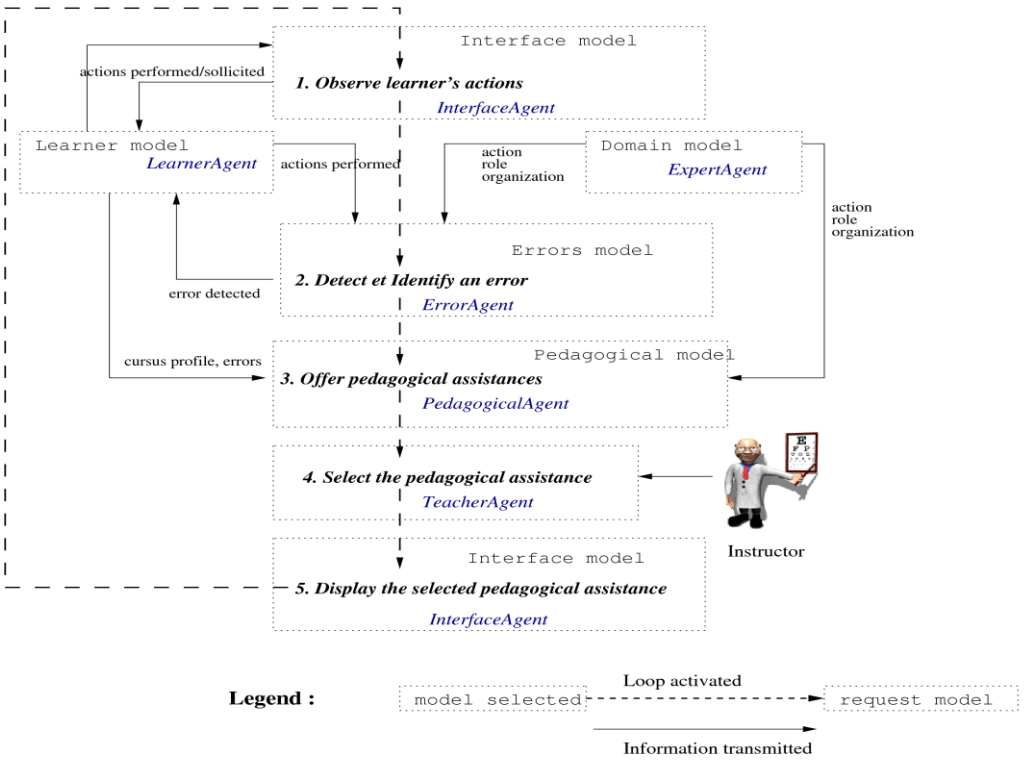

Fig. 2. The instructive process of our five-stage system.

Step 1. Observation:
Using the interface model, the system analyzes the learner's activity. The elements that are important for learning are supplied to the learner model. This information concerns the learner's actions, those elements which the learner can observe, and the learner's movements.

Step 2. Detecting and Identifying an Error:
The system analyzes the learner's actions (learner model) and compares them with the actions to be performed (domain model). This comparison is used in order to detect errors. If one is detected, an error identification mechanism is set up (using the error model).

Step 3. Proposing instructive assistance:
Using the learner model (characteristics, activities, errors, etc.), and the domain model (knowledge of the organizational structures), a mechanism simulating instructive reasoning recommends the instructive assistance for the given situation. It must be noted that this step is not optional; it occurs even if no error is detected.

Step 4. Choosing instructive assistance:
The instructor can choose one specific type of instructive assistance amongst those proposed.

Step 5. Representing instructive assistance:
The instructive assistance selected is presented in the VE.

To use the information from the VE, we must inform the environment in order to obtain controllable knowledge. This creates an informed VE (see section 4). The environment will then be *reified*. This knowledge is comple mented by additional information contained within the 6 ITS models. This data makes up a knowledge base for the pedagogical model which we call the *pedagogical situation*. This knowledge fuels the ITS's motor for making *instructive decisions* (see section 5). An example of the way in which the rules governing this motor are specified is presented in section 5.3.

IV. DOMAIN MODEL

To reify the concepts of the domain, we define the `Veha` (VE for Human Activity) metamodel. It describes the VE, not only in terms of geometric space, but by providing the semantics required for the artificial agents (ITS, autonomous characters) or humans (learners or instructors) to be able to construct for themselves a representation of the environment and act together to reach their goals. The `Veha` metamodel (M3) enables the construction of VE models (M2) and the corresponding concrete VEs (M1) (see table 1). `Veha` is based on `Uml` [2]. It extends `Uml` because `Uml` does not define the specific concepts of virtual reality.

TABLE 1: LAYERS OF MODELING (MI):THE POSITION OF VEHA WITHIN THE MOF FRAMEWORK, IN PARALLEL WITH UML.

M4	`Mof`[3] (`Uml` limitation)				
M3	`Uml` metamodel	`Veha` metamodel			
M2	`Uml` user model	V E1 model		…	
M1	user object	V E1a	V E1b	…	…

[2] `Uml` (Unified Modeling Language) an object modeling and specification language (http://www.omg.org/spec/UML/).

[3] `Mof` (Meta-Object Framework) a meta-model used to formally define Uml.

4.1 The Veha metamodel

The ITS needs to know which objects make up the VE, how to access it, its properties, its behavior, and how to interact with it. Three kinds of knowledge can be expressed using `Veha`:

1. Domain concepts: This entails the semantic description of the concepts relating to the field of activity concerned. It represents some of the knowledge that the learner must acquire (section 4.1.1).
2. The possibility of structuring and interacting with the environment: These concepts resemble those suggested in smart objects [10] which reify those properties required for interactions. The means available to the learner or to the ITS must be specified in order to modify the environment (section 4.1.2).
3. Entities' behavior: Within the framework of a VLE the environment's reactions to the learner's actions must be simulated. Entities' behavior also represents one of the elements of the knowledge to be transmitted and must be enforceable (section 4.1.3). In the following part of this section, we explain how `Veha` can be used to express these three kinds of knowledge.

4.1.1 Domain concepts

Knowledge of the domain is expressed both at the model (concept) level, and at the level of the occurrences of these concepts (tangible objects populating the environment). In `Veha` as in `Uml`, this knowledge is represented by classes (`Class`) and instances (`InstanceSpecification`).

In `Veha`, the notion of class is used to define a type of object (Fig. 3) from domain-specific ontology. The aim is to be able to apply semantics to each of the business concepts, whether or not they are tangibly represented in the VE (concepts v.s. concrete objects). All classes stem from the `Element` class. This class enables the identification of each of the elements of a business model from its name and the addition of a textual comment. This can be useful when providing the user with explanations regarding the significance of an object.

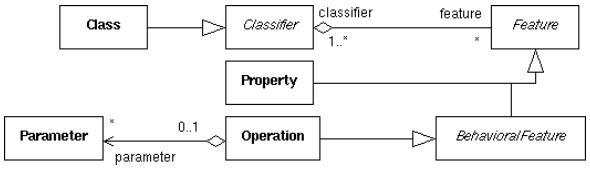

Fig. 3. Class diagram from the Veha metamodel: Features of a Classifier.

The structural properties (`Property`) and behavioral features (`BehavioralFeature`) of the classes are assigned to the `Classifier`[4] via the `Feature` class. The Property class represents the structural component of the `Classifier` (as much the attributes as the relationships with other business concepts). As in `Uml`, the `Operation` class is the only tangible sub-class of `BehavioralFeature`. It is used to express the effect that an object or a user can have on another object. It does this by defining the object's actual behavior rather than the method used to achieve that behavior. The way in which the behaviors associated with `Operation` are modeled is described using behavioral models (see section 4.1.3).

The `Veha`'s second key concept is the notion of `Class` and `Instance`, synonymous for the object. The `InstanceSpecification`, `Slot` and `AssociationInstance` classes represent the instantiation of `Class`, `Property` and `Association`, respectively. The term `InstanceSpecification` indicates that here, we represent an M1 level entity (see table 1) independently of the circumstances under which it is implemented.

The set of knowledge about the environment as specified in `Veha` can be accessed by the ITS and by the users (learners or instructors). The ITS can, for example, suggest to the learner a list of operations to be performed on one specific kind of object. Likewise, the instructor can modify the environment during the simulation by changing the attribute values of a tangible object.

4.1.2 The possibility of structuring and interacting with the environment

Most of the tangible objects within VEs are represented geometrically and are situated within the environment. The learner must be able to observe, recognize and manipulate these objects. The ITS also needs to be able to manipulate them within the context of the instructive assistance that it will implement (transparency, refocusing from the learner's point of view, etc.). Knowledge about the geometry of these objects must also be specified so that the ITS will be able to recontextualize its suggestions within the VE. These objects are entities and all have the properties of the instances `Veha`, i.e. `Class` as well as geometric and topological properties (see Fig. 4).

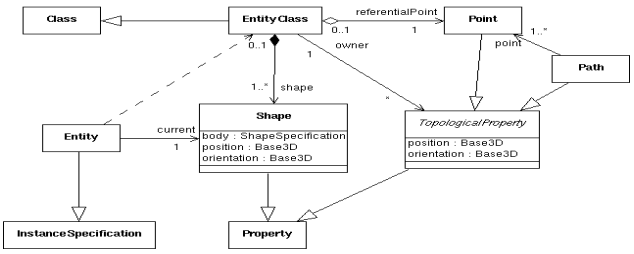

Fig. 4. Class diagram from the Veha metamodel: EntityClass.

Each entity is located at a global reference point. The `Shape` class is used to assign an instance of `EntityClass` to a graphical representation in the VE. It is possible to assign many forms to one class of entity. The ITS can use this knowledge to highlight an object or, on the contrary, to hide it. The `TopologicalProperty` class supports the notion of location (position and orientation) and is used to describe the topological properties of the elements within the VE. It is possible to assign informed points to an entity (`Point`) which can be used to create an interaction. This information is used by the ITS to turn the learner's attention to a specific object, for

[4] UML metamodel class which generalizes the concept of class.

example.

Any entity within the VE is an instance of the Entity class, which derives from `Kernel:InstanceSpecification`. The values of an entity's properties are defined by its slots. So these depend on the semantic, morphological, geometric and topological properties of the objects within the VE (supplied by `InstanceSpecification`).

4.1.3 Entities' behaviors

When the learner carries out an action in the environment, that environment must react in a realistic way for the learner to be able to understand the consequences of his actions. The learner therefore constructs a representation of the entities' behavior. For the ITS to be able to regulate this representation, the knowledge of entities' behaviors must also be specified, as for the two previous kinds of knowledge, and it must also be enforceable.

The role of the `Behavior` package is to model the possible behaviors of the entities within the VE; the objective being for the model to be interpreted in real-time by a behavioral controller, and to be introspected online. As for the structural aspects, introspection relies both on the behavioral model (M2) and on its "instantiation", i.e., the way it is carried out (M1). The two classes which support these notions are `Behavior` and `BehaviorExecution` (see Fig. 5). The `Veha` entities have reactive behaviors which are triggered by events that can be caused either by the learner or by another of the VE's entities.

Traditionally, behaviors are assigned pre-conditions and post-conditions concerning the entities and the environment. Behavioral modeling relies on state machines and the `Uml` activity model. Finally, it can also be based on functions written in programming language that can be consulted online (`OpaqueBehavior`). The first two methods are introspectable; the ITS can therefore describe or check the way the behavior is carried out.

The tutor can thus analyze, explain or check the context in which an entity's behavior is carried out by the learner. Better still, if a particular behavior has been specifically described (state machine or activity) it can also explain the way it will be carried out.

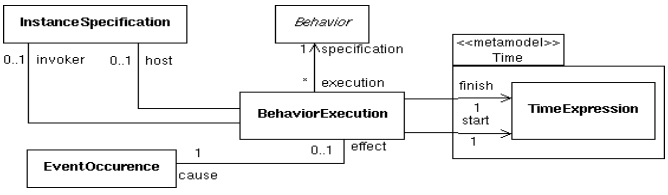

Fig. 5. Class diagram from the Veha metamodel:Behavior::Common package,the BehaviorExecution class.

4.2 Example of an environment in Veha

The `Veha` metamodel can automatically interpret a model described in `Uml`. Figure 6 shows the class diagram for an example of a VE in `Veha`. This example comes from an application created in `Veha`, but which has been greatly simplified for demonstration purposes. The application (`Gaspar`, [3]) is made up of around fifty classes and more than one thousand entities. This model shows the classes `Deflector` and `CatapultCabine` (left window). The catapult cabin shields the operators working on the catapult deck of an aircraft carrier. A pod can open (raise above the deck) or close (drop back down into the deck). The business model specifies all of the pod's properties (`height`, `speed`, etc.). The reactive behavior of a pod is specified by a state machine (top right-hand window). This state machine is sensitive to the signals `Open` and `Close`. Therefore, when the pod is Closed, if it receives the signal to Open, it changes to the Open state and performs the operation `Open()`. Within the context of this application, this operation is described in detail by an `OpaqueBehavior`, a C++ code which carries out the visual displacement of the pod depending on the speed attribute, and updates the height attribute.

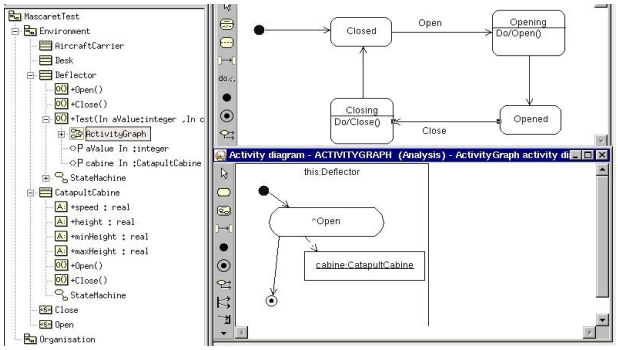

Fig. 6. Class diagram for a deflector and a catapult control pod.

In much the same way, deflectors also react sensitively. Due to the additional needs of this demonstration, we added a testing operation (`Test`). This operation takes its settings from a catapult control pod. The behavior of this operation is specified by an activity diagram (bottom right-hand window). There- fore, when a `Test` operation is evoked in an instance of the `Deflector` class, the operation sends the signal `Open` to the predefined pod.

This model is defined using *Objecteering* modeling software. It is then exported in an XMI file. The first proposal is to add an interpreter to the `Veha` metamodel within the *AReVi* virtual reality platform. The interpreter reads the XMI file and, for each class of `Uml` metamodel, creates an instance of the corresponding class in the `Veha` metamodel. Thus, for each business class defined in the XMI file, the interpreter creates a new instance of the Class class from the `Veha` metamodel. In the context of our example, an instance of the `Class` class is created for the `Deflector` class, and another created for the `CatapultCabine` class. The interpreter enables the reification of the business model and provides a set of methods facilitating the introspection of this model. It is therefore possible to ask the interpreter for the set of a class's properties, the signal which enables the passing from one state to another, and the operation which will then be conducted, all independently of any tangible object.

The VE is populated with entities, the instances of the `Entity` class of the `Veha` metamodel. From a technical standpoint, these instances are defined in an XML file. Using `Uml`, class instances can also be described and exported in the XMI file. However, no `Uml` modeler can make it simple to attribute a shape and a position to these instances. The geometric design of the VE is, in general, the result output by specialist modelers such as *3DS MAX* or *Blender*. We therefore suggest using an export plugin for 3DS Max which would generate the instance file read by the interpreter. Fig. 7 shows the visual result of the file defining the model (XMI) and the instance file (XML) in an application implemented using *AReVi*. The interpreter also provides the methods for interrogating and manipulating the entities. It is therefore possible to ask an entity for its property values, to carry out an operation, or to send it a signal in order to change its state.

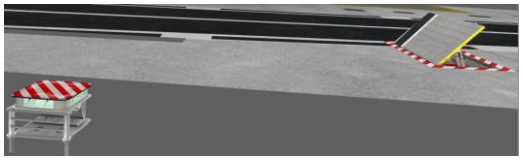

Fig. 7. Visualization of the instances of CatapultCabine and Deflector.

4.3 Procedure and Collaboration

Here we examine the acquisition of skills. The domain model not only contains knowledge about the environment in use, but also knowledge about the task which must be performed within that environment by the learners. Within the context of this research and the examples given in the introduction, activities are defined by the procedures describing the `Actions` to be performed by a number of entities, each with specifically defined roles. We use the same assumption as for the environment and propose the use of a metamodel based on `Uml` in order to define these activities. The procedures are therefore defined by activity diagrams. This kind of diagram uses the traditional possibilities for organizing its `Actions` (parallelism, sequence, junction, condition, etc.) As we are dealing with representing human activity, we consider that the sequence of activities takes place in an asynchronous manner.

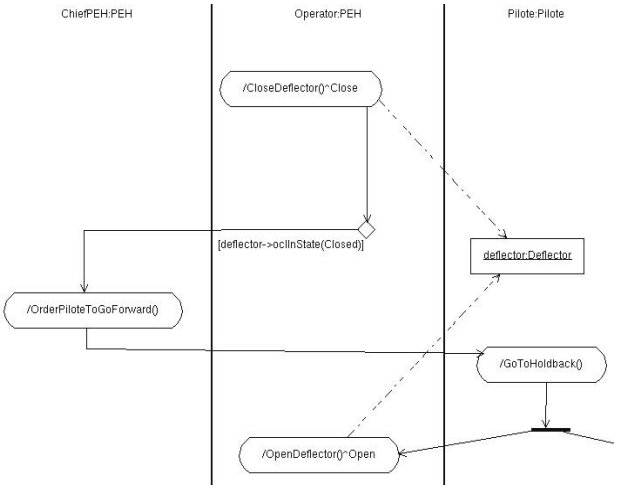

Fig. 8. Example of a procedure written using an activity diagram.

The organization roles are represented by activity corridors. The name of the corridor defines its role and its type, as well as the type of agent that is authorized to take this role. As in `Uml` 2.1, there are many different types of activity. This could be the execution of an agent's operations, a basic virtual action (playing an animation, reaching a given position, etc.) or sending a signal to a specific resource. The resources are drawn on by the environment's entities and represented by objects in `Uml`. The conditions are expressed in `Ocl` and stem from the roles and resources participating in the procedure. Figure 8 illustrates the example of a procedure expressed using an activity diagram. This procedure solicits the intervention of three roles (such as `Operator`) which must be played by characters of a pre-defined type (`PEH` for example). The characters which play these roles are those which are effectively instantiated in the environment. This procedure aims to make the airplane which is to be catapulted advance towards a given point by manipulating the deflector (a protective plate). The example of the procedure in Fig. 8 illustrates the complementary nature of the state machines used to define the reactive behavior of the objects in the environment and the activity diagrams defining a procedure. A procedure's action can be represented by sending an event to a given object to be manipulated, and the conditions of moving on to the following action can depend on the current state of the object.

We implemented agents' behaviors using knowledge about the procedures to select their actions. The learner plays one or more roles in the context of these procedures. The ITS also draws on this knowledge in order to choose which assistance to suggest. As for the environment, there are two levels of modeling available to the agents (including the ITS) and the users (instructors): the organizational structure and the organizational instances. The intelligent tutor is therefore able to recognize the sequence of actions independently of all organization. It can also follow the precise progress of the procedure being carried out in the team in which the learner plays one or more roles. It is therefore able to detect the learner's errors with respect to the order of the actions to be completed and compliance with the conditions defined in the procedure [11].

V. PEDAGOGICAL MODEL

Knowledge about the environment (the entities and about the task to be performed) are represented with the `Veha` model. Our ITS can thus manipulate them in order to construct its own knowledge, as shown in (section 5.1), and can simulate pedagogical reasoning 5.2). Finally, a tangible implementation of the ITS is proposed in section 5.3 (specification of the rules of simulated pedagogical reasoning).

5.1 Pedagogical Situation

It must be emphasized here that our work is done in the context of in situ learning. Within this theoretical framework, the contextual elements are paramount in the ITS's decision-making [12, 13]. In our case, we refer to context as the pedagogical situation which serves as a basis for decision-making. The aim is to define this sort of context from

a ”generic” standpoint, which would enable us to alter information without having to take into account the specific task being carried out. To do so, we must separate knowledge about the task to be performed (see section 5.1.1) from knowledge about the learner (see section 5.1.2).

5.1.1 Information concerning the task to be performed

We positioned our work in the context of training for procedural work. The aim of the ITS is to assist learners in their progression through the procedure. The skill to be acquired relates to the completion of the procedure in a dynamic environment.

First of all, we can consider the procedure as a sequence of actions defined by an expert. The elements to be considered are therefore subject to sequencing which cannot be questioned, and sometimes cannot be explained. Secondly, we think that memorization of the sequence of actions could be facilitated through understanding. In this context [14] suggests adding the notion of sub-objectives to the procedure. To meet this aim, i.e. the completion of the procedure, a set of causally linked sub-objectives must be conducted. The procedure must therefore be studied taking into account the distance to the procedure’s goal from a causal, rather than a chronological standpoint.

The above analysis highlights two ways of dealing with procedural learning: the study of business sequencing links which are strongly linked to the roles in the procedure, and the study of causal links between sub-objectives:

1. Sequencing Links

Sequencing links conduct the relationships between the actions using the strict description of the procedure. They are the direct consequence of the sequencing of actions as defined by the expert. We are interested in the information linked to the actions closest to the action requested by the learner. More precisely:

- the last correct action completed before that which the learner has just solicited;
- the action which has just been solicited by the learner;
- the correct actions to be carried out, taking into account the role(s) to be played (which are potentially different from the solicited action);
- the correct actions to be performed, when considering that all roles are played by the learner; and
- those actions following all the correct actions.

We chose the actions closest to that solicited in order to try to reduce the ”distance” between the goal (the end of the procedure) and the learner’s location in the procedure. Technically, this is done by carrying out plan recognition based on the `Veha` activity diagram shown in section 4.3. The pedagogical situation thus retains the knowledge linked to the actions that are chronologically close to that which is requested.

2. Causal links between sub-objectives

The procedure can be considered like a graph representing the sequence of causal sub-objectives. We therefore are looking at all of the actions linked to the one the learner is performing. In concrete terms, this means the actions requiring the effect of the correct desired action (usage conditions, state of a resource, etc.). A distinction must be made between these links, which correspond to individual logic, and sequencing, whose links correspond to the organization of a collective procedure. Technically, we are dealing with the links between post-conditions and pre-conditions mentioned in section 4.1.3.

It must be stressed that our objective here is to extract knowledge relating to the work to be carried out in order to assist pedagogical decision-making. Within this context, we look at the knowledge described in table 2. All the actions which have been identified up to this point (sequential and causal links) make up the pedagogical situation. More specifically, we are interested in the information related to the selected actions. At this point, we must specify the knowledge relating to the concept of action. From this perspective, the "action context” is made up of knowledge that is directly linked to the `Action` (description, resources, etc.), knowledge relating to the `Operation`, which is the target of the `Action`, as well as knowledge relating to the agent that has requested the action, since that agent is the central character. We therefore use action contexts in order to represent the knowledge associated with particular actions (a sub-group of the environment made up of the entities and agents considered relevant in the context of the action).

It is the responsibility of the pedagogical agent to construct this set of knowledge. The pedagogical agent retrieves or constructs the knowledge required about the task to be performed when it receives a message from the interface agent detailing an action which has been requested. This choice is debatable and indeed another possible solution is to update the knowledge when an error occurs. We chose to reconstruct the knowledge of the actions in order to retain the option to intervene, even if the learner’s actions are correct. This means that we can provide pedagogical assistance in order to reassure the learner about the decisions that they've made, or conversely to imply doubt if it looks like they are about to make a mistake (e.g., confirming false rules which contradict the choices the learner has made).

5.1.2 Information concerning the learner

The information about the learner comes from a number of sources, but all of it is collected by the learner model. This information relates both to static data (such as age) and dynamic data (such as elements of memory at a given time).

It should be noted that the learner’s errors are recorded and are analyzed. Our error model is based on the Cognitive Reliability and Error Analysis Method (CREAM). This approach proposed a classification scheme which makes a distinction between observations of errors (phenotypes) and its causes (genotypes). The causal links between phenotype and genotype are represented using a number of consequent-antecedent links. Finally, the pattern could be associated with a method of retrospective analysis (the search for causes). The most

probable cause-effect links is found using Dempster-Shafer's theory presented in [15].

Similarly, the contexts relating to the actions are also recorded. This information allows us to see whether or not learner has already used a particular resource, for example.

In concrete terms, we have just defined the input information and the relevant elements from which pedagogical decisions can be made.

TABLE 2: THE PEDAGOGICAL SITUATION: KNOWLEDGE ABOUT THE TASK TO BE PERFORMED.

Knowledge	Nature	Description
Context of the previous action	Sequential	The last correct action to have been performed. This action serves as a point of reference from which one can position oneself in the procedure.
Context of the requested action	Sequential	The requested action. This action could be correct or incorrect. The action has not necessarily been performed, in accordance with the pedagogical model.
Context of the correct action(s) without considering their roles	Sequential	In considering the last correct action, we can determine the actions to be performed within the context of the current procedure.
Context of the correct action(s)	Sequential	A sub-group of the previous item which does not take the roles played by the learner into account.
Context of the following action(s)	Sequential	For each correct action, we determine the actions which follow it according to the current procedure.
Context of related action(s)	Causal	In considering the actions to be performed following the last correct action, we retrieve the "causal" links between the actions independently of the procedure. We therefore obtain the actions which are related.

5.2 The Pedagogical Agent

The pedagogical situation (section 5.1) gives us the option of triggering pedagogical assistance relating to the elements detailed within it. It thus provides the possible outcomes of the pedagogical decision-making process. We now go on to define a model to simulate the behavioral decision-making of the pedagogical agent providing instructive assistance, i.e., a model linking knowledge and the proposed assistance. It must be noted that we are working within the context of learning procedural and collaborative tasks. We must therefore consider:

- The atypical nature of the knowledge involved (knowledge stemming from basic pedagogical methods to virtual reality);
- Adaptability (the agent's reasoning processes must self-adapt in order to take past experience into account);
- This reasoning must be specified prior to the event (initial specifications can therefore be made by an instructor).

The criteria which arise from these considerations are as follows: expressiveness, hierarchy, modularity, reactivity and adaptability.

After examining the existing families of behavioral architecture (connectionist, automata-based, rule-based), we opted for the rule-based families which best respond to the criteria outlined above. More precisely, we chose classifier systems [16]. This is a reactive and adaptive form of architecture, based on conditional rules.

We propose the use of a model based on a hierarchical classifier system. This system organizes knowledge while taking the abstraction of the data involved into account. It structures knowledge according to three levels, from rules based on abstract knowledge of educational methods (the pedagogical approach), to the rules based on concrete knowledge of virtual reality (pedagogical techniques), via an intermediary level (pedagogical attributes).

Each level of abstraction contains sets which group together a number of rules. One set represents a way of dealing with a particular approach, attitude or pedagogical technique. The rules are conditioned by the elements of the pedagogical situation, and favor the sets from the lower level. The system therefore uses a diffusion mechanism on all three levels which considers the rules matching the pedagogical situation. This gives rise to a list which then arranges the different suggestions for pedagogical assistance.

Fig. 9 illustrates the structure and the dynamics of the pedagogical model controlling the pedagogical agent's behavior. The information taken into account in the conditional part of the rules is retrieved by our ITS (pedagogical situation). These "inputs" are available at the three levels of data abstraction (approach, attitudes and pedagogical techniques). The rules whose conditional elements are satisfied in terms of input favor some of the sets of pedagogical rules from the lower level. The upper level (techniques), directly favors those pedagogical suggestions which can be applied within the environment. These suggestions are made to the instructor who chooses the one considered to be the most relevant.
Simulating pedagogical reasoning has two advantages:

1. As instructors are not always teachers, they too are being given pedagogical assistance.
2. Instructors are not simulation software experts, so the pedagogical agent will offer assistance to the learner, who will have the opportunity to make the most of the VE.

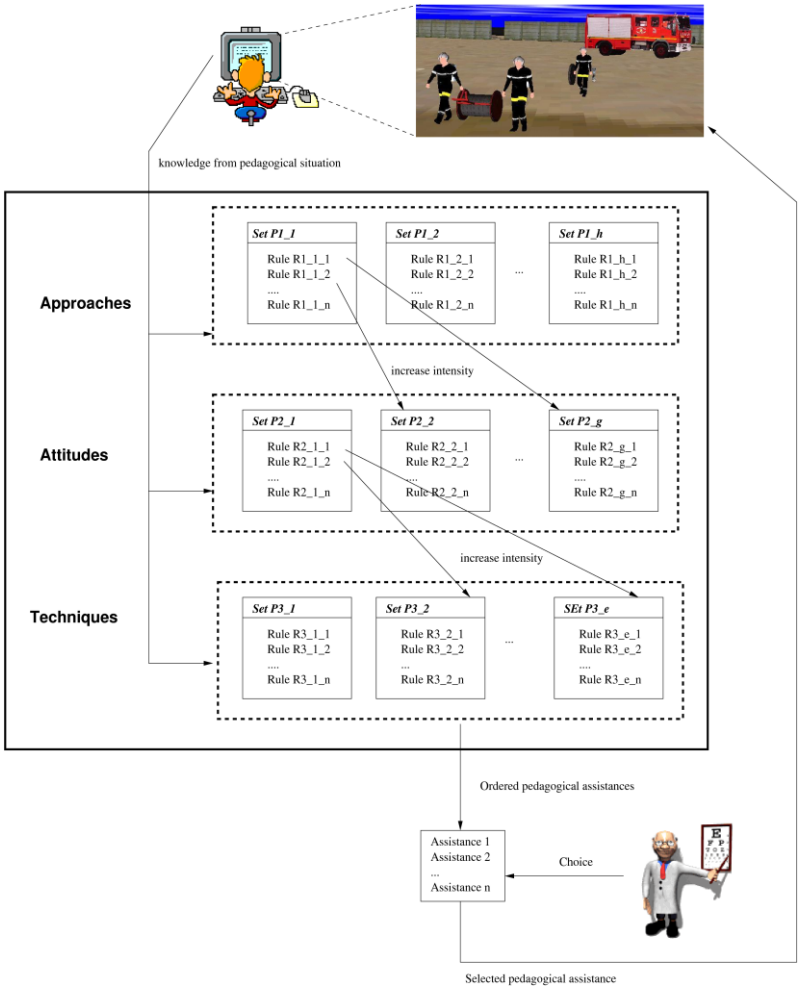

Fig. 9. Complete Representation of the Pedagogical Model.

5.3 Specifications of the Pedagogical Model

In order to implement the pedagogical model, the teacher must specify:

1. The sets of rules for the three levels of abstraction.
2. The pedagogical rules for each of the sets of rules.

Here, we will discuss information from the literature which can be used when specifying the pedagogical model.

5.3.1 Specifying the sets of pedagogical rules

We worked from the studies by [4] in order to define the sets of pedagogical rules. We obtained the following tables; 3, 4 and 5 corresponding to the three levels; approaches, attitudes and techniques, respectively. This information provides an opportunity to specify sets of rules at each of the three levels (see Fig.10)

5.3.2 Specifying the Pedagogical Rules

Once the sets of pedagogical rules are defined, the teacher must specify the associated rules.

A rule is represented by a sequence of characters. The effect and condition parts are based on the elements of the pedagogical situation

In the following example, we position ourselves at the *Pedagogical Methods* abstraction level, with a set of rules called Active. The first rule for this set is fulfilled if the learner is a novice (`Learner.Level==novice`), if they have per formed an organization error (`Learner.Error.type==procedural`) and if the action performed is different from the correct action (`!Task.RequestedAction in Task.CorrectActions`). In this case, the rule favors the `Explain` set from the following level.

```
if (Learner.Level == novice &&
    Learner.Error.type==procedural &&
    ! Task.RequestedAction in
      Task.CorrectActions)
then (Explain)
```

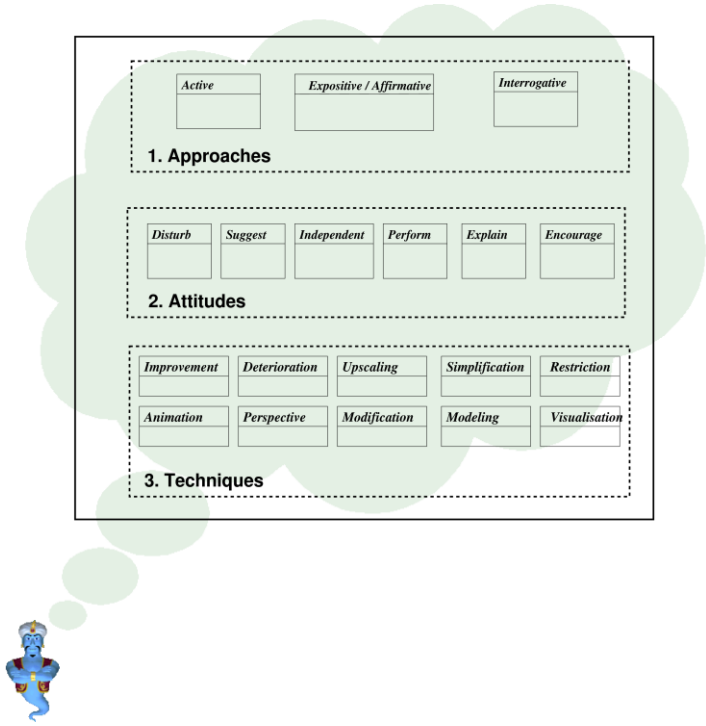

Fig. 10. Specifying the three levels of the pedagogical

TABLE 3: EXAMPLES OF SET DEfiNITIONS FOR THE "PEDAGOGICAL APPROACH" LEVEL OF ABSTRACTION BASED ON [4]

Pedagogical Approach	Description
Active / Constructivist	An active approach is learner-centered, considering them to be the main actors in the learning process. This approach suggests techniques through which they can produce, create and search. The knowledge required can be found in the environment.
Expositive / Affirmative	This is the most traditional approach which uses the display technique. It is based on a content-transfer approach. Knowledge is external.
Interrogative	This approach makes recommendations to the learners, guiding them towards the desired outcome. Learners may have the impression that they have discovered something new, but it is the instructor who will have guided the thought process. Knowledge is internal.

5.3.3 Use

A specific pedagogical model was created from the structure described above and from articles by [4]. These sets are described in Fig. 10, with each set containing an average of five rules. This pedagogical model was applied to two distinct VEs designed for learning collaborative procedures. No modification of the pedagogical model (sets and rules) is required for either of these applications, which although very different, are both based on the same kind of learning. However, we believe that these changes would only need to be made at the intermediate level. For other types of learning (for example for a scientific practice), these rules would probably need to be changed.

5.4 Artificial learning

Thanks to artificial learning, the weight of the rules for adapting to the instructors' preferences can be refined and their expertise imitated.

The learning algorithm is inspired by the Bucket Brigade [17, 18]. This system distributes remunerations to the rules which enabled them to be obtained. It is adapted to classifier systems [16] with a list of rules which, when followed one after the other, lead to an action. In our case,this sequence of events corresponds to the passing from one level to another. Remuneration is reflected by the instructor's choice: the pedagogical technique which they choose defines the rules in the third level which will be compensated. By back-chaining, the rules in levels one and two are also compensated. The weights of the rules which match the *pedagogical situation*, but which participate in activating a technique other than that chosen by the instructor will decrease. The algorithm shares out the remuneration, including a tax which means that the rules which rarely match are not put at a disadvantage, and that the strong rules are penalized in order to retain the adaptive nature of the system.

Therefore, as the exercises progress, the pedagogical agent must make suggestions which correspond more and more closely to the instructor's decisions. The pedagogical agent

could therefore temporarily take over and directly apply the assistance that it has chosen itself, should there be more than one learner at a time.

5.5 Use case: Gaspar

Gaspar is a virtual reality application developed to simulate human activities on an aircraft carrier. In Gaspar, a typical scene such as that shown in Fig. 11 is made up of around 1,000 entities, each with 3D representation (VRML), i.e. a total of 1 million facets. In this scene, there are around 50 agents, divided into 10 teams, each with an average of 5 roles. Each of these teams is responsible for an average of 5 procedures. The most complex procedure activates 9 roles and organizes 45 actions. In this scene, at each moment, around 50 behaviors are activated (both NPCs and entities). This sort of scene is implemented using AReVi[5] and is simulated in real-time (around 40 frames per second) on a desktop computer with 2GB of RAM, a 64 bit processor running at 1.3 GHz, and a GeForce card with 1GB of video memory.

TABLE 4: EXAMPLES OF SET DEfiNITIONS FOR THE "PEDAGOGICAL ATTITUDES" LEVEL OF ABSTRACTION BASED ON [4].

Pedagogical Attitudes	**Description**
Perform	Perform the task in the place of the learner. This strategy can be used by the instructor to show the learner the correct technique or move.
Disruption	Some instructors tease and disrupt the learners by giving them incorrect information or potentially incorrect solutions in order to test the learners' conviction of their ability to reason independently.
Suggest	Showing where the learners can find theoretical information or where to find information within the environment. These attitudes allow the instructor to show the learners that they can find the required information independently and therefore deal with the situation in a calm manner.
Independent learning	This attitude encourages the instructor to remain in the background as an observer rather than to intervene.
Explain	The explanations and information are also designed, quite simply, to explain the functioning of certain devices, rules of analysis, safety rules, etc.
Encourage	Encouraging the learners when they perform a task correctly.

TABLE 5: EXAMPLES OF SET DEfiNITIONS FOR THE "PEDAGOGICAL TECHNIQUES" LEVEL OF ABSTRACTION BASED ON [4].

Pedagogical Techniques	**Description**
Improvement	Addition of visual and audio symbols or animated films.
Deterioration	Unrealistic images. (points of reference erased, feed-back, deteriorated proprioceptive elements, altered colours, blurred background/surround, reduction of objects, iconization, etc.).
Upscaling	Exaggeration of reality (representing objects on a larger scale, or that are surreal, brighter or shinier, etc.).
Simplification	Simplification of the virtual scene (a crowd can be represented by people with simplified movements, simplified objects, simplified kinetic systems, wireframe images, etc.), schematic representations of certain devices.
Restriction	Limitation of certain movements or actions (limiting the area within which the learner can move around, etc.)
Animation	Animated sequence (automatic positioning, keys which turn automatically once in place, etc.).
Perspective	Altering the learner's normal viewpoint (view from behind, above, etc.).
Modification	Changes in appearance and texture (colours, flickering objects, etc.).
Modeling	The representation of abstract concepts, of physical phenomena invisible to the naked eye, types of errors, etc.
Visualisation	Hidden mechanisms (the inside of a motor, gears, etc.).

The decisional behavior of the ITS relies on a classifier system in which each rule presents a set of conditions required to activate an educational method, attitude or assistance. The main advantage here is that the rules are formulated in a general way, at the M2 level, and deal with the data from the concrete environment (M1 level). The ITS knows how to evaluate rules such as: "IF the entity is not in the state required to carry out the correct action and if the learner is novice THEN simplify the environment". Rules such as these can be expressed using the `Veha` metamodel, independently of the model of the virtual environment.

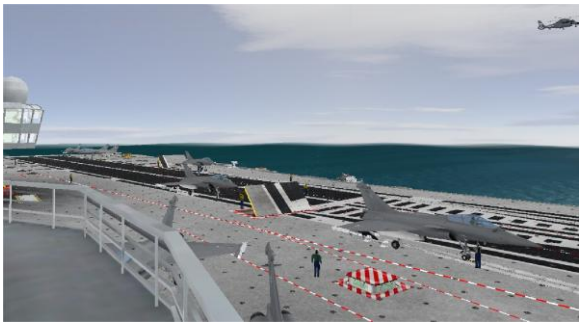

Fig. 11. View of a scene on an aircraft carrier in Gaspar.

[5] `http://sourceforge.net/projects/arevi/`

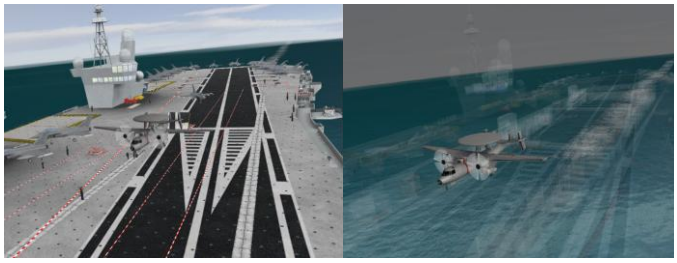

Fig. 12. The effect of applying the pedagogical assistance “simplify the environment” in the Gaspar application. (a) before; (b) after..

The veracity of these conditions is evaluated by the manipulation of the model, contextualized for specific environments using M1-level knowledge. For example in Gaspar, if the correct action is tensioning the hook in the context of the procedure catapulting the Hawkeye aircraft, the previous condition rule is automatically contextualized to ”*IF the Hawkeye aircraft is not in the state launch bar down required to carry out the operation tensioning the hook*”.

The classifier system builds up a list of proposals for educational assistance made up of the action elements of activated rules. The assistances are evaluated by the manipulation of the model, contextualized for specific environments using M1-level knowledge. For example, the assistance: "*simplify the environment*” translates to a corresponding solution proposed to the instructor "*make transparent all entities except the Hawkeye aircraft*” (see Fig. 12).

VI. DISCUSSION

Before concluding, we would like to discuss the benefits of our proposal. The study described in this article began by examining previous studies in this field and analyzing the uses of pre-existing ITS within VLE. We then went on to show that the Hal system is the most successful, and highlighted the elements which could be improved. Indeed, in this system, the pedagogical model depends partly on the exercise, and the errors and pedagogical strategies must be defined.

Furthermore, the instructor can only choose between two pedagogical methods (active or explanatory). We believe that it is possible to resolve the pedagogical model’s problems of genericity and modularity.

Without re-examining every element of our work, we can show how our proposal could solve some of the diffculties of existing models. The knowledge used for pedagogical reasoning does not depend on the specifics of the task to be performed. Therefore pedagogical rules do not, and indeed do not need to, consider specific information, (”if the learner can see airplane 2 then...”), but will rather use general knowledge independently of the exercise (”if the resources of the correct actions are visible, then...”). In much the same way, although the pedagogical assistance proposes tangible solutions to the instructor (”make the fireman flicker”), generic knowledge is also manipulated independently of the exercise (”make the characters involved in the following actions flicker”). Thus, the genericity of our proposal is one of its strongest characteristics, as illustrated by the inclusion of our ITS at the core of numerous applications: learning of collaborative procedures on aircraft carriers (`Gaspar`) [3] and for firefighters intervening in Seveso high risk areas (`SecuReVi`) [19]. In addition, the pedagogical model of our ITS has strong modularity, as it offers the option of adding, deleting or modifying each of its components that participates in pedagogical decision making (rules or sets of rules). Moreover, the artificial learning mechanism adapts the proposed pedagogical assistance to the learner-instructor pair. Therefore, our proposition provides solutions for the problems raised in the introduction. Finally, it must be emphasized that `Pegase` is directly based on the learner-instructor relationship.

However, we must not forget that there will undoubtedly be limitations linked to the use of our ITS in contexts of non-procedural learning. To be able to deal with this kind of training, we would have to rethink the elements which are so strongly linked to the notion of procedure, i.e. knowledge about the pedagogical situation.

REFERENCES

[1] D. Herviou and E. Maisel, “ARéViRoad : a virtual reality tool for traffic simulation,” in *Proceedings of Urban Transport*, 2006, pp. 297–306.

[2] L. Edward, D. Lourdeaux, D. Lenne, J. Barthes, and J. Burkhardt, “Modelling autonomous virtual agent behaviours in a virtual environment for risk,” *IJVR : International Journal of Virtual Reality*, vol. 7, no. 3, pp.13–22, September 2008.

[3] N. Marion, C. Septseault, A. Boudinot, and R. Querrec, “Gaspar : Aviation management on an aircraft carrier using virtual reality,” in *Cyberworlds 2007 proceedings*, October 2007, pp. 15–22.

[4] D. Lourdeaux, J. Burkhardt, F. Bernard, and P. Fuchs, “Relevance of an intelligent agent for virtual reality training,” *International Journal of Continuous Engineering and Life-long Learning*, vol. 12, no. 1/2/3/4, pp. 131–143, 2002.

[5] J. Rickel and W. L. Johnson, “Animated agents for procedural training in virtual reality : Perception, cognition, and motor control,” *Applied Artificial Intelligence*, vol. 13, 1999. [6] E. Wenger, *Artificial Intelligence and Tutoring Systems*. Los Altos, California: Morgan Kaufmann, 1987.

[7] P. Levesque, “Creation and use of 3d as-built models at EDF,” *in FIG Working Week 2003*, 2003.

[8] R. Hubal, “Embodied tutors for interaction skills simulation training,” *IJVR : International Journal of Virtual Reality*, vol. 7, no. 1, pp. 1–8,2008.

[9] K. Amokrane, D. Lourdeaux, and J. Burkhardt, “Hera: Learner tracking in a virtual environment,” *IJVR : International Journal of Virtual Reality*, vol. 7, no. 3, pp. 23–30, September 2008.

[10] M. Kallmann and D. Thalmann, “Modeling objects for interaction tasks,” in *Proceedings of Computer Animation and Simulation’98*, 1998, pp. 73–86.

[11] T. Trinh, C. Buche, R. Querrec, and J. Tisseau, “Modeling of errors realized by a human learner in virtual environment for training,” *International Journal of Computers, Communications and Control*, vol. IV, no. 1, pp.73–81, Mar. 2009.

[12] R. M. Turner, “Context-sensitive reasoning for autonomous agents and cooperative distributed problem solving,” in *Proceedings of the IJCAI Workshop on Using Knowledge in its Context*, 1993, pp. 141–151.

[13] J. Pomerol and P. Brézillon, “About some relationships between knowledge and context,” in *Modeling and Using Context: Third International and Interdisciplinary Conference, Context 2001*, V. Akman, P. Bouquet, R. Thomason, and R. A. Young, Eds. Berlin: Springer-Verlag, 2001, pp.461–464.

[14] J. F. Richard, *Les activités mentales : Comprendre, Raisonner, Trouver des solutions*. Paris: Armand Colin, 1990, iSBN 2-200-2187-0.

[15] N. El-Kechaï and C. Després, "A plan recognition process, based on a task model, for detecting learner's erroneous actions," in *Intelligent Tutoring Systems, 2006*, pp. 329–338.
[16] O. Sigaud and S. W. Wilson, "Learning classifier systems: A survey," *Journal of Soft Computing*, vol. 11, no. 11, pp. 1065–1078, 2007.
[17] J. H. Holland, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds. Los Altos, CA: Kaufmann, 1986, vol. 2, pp. 593–623.
[18] S. W. Wilson, "Hierarchical credit allocation in a classifier system," *in10th International Joint Conferences on Artificial Intelligence (IJCAI'87)*, Milan, Italy, 1987, pp. 217–220.
[19] R. Querrec, C. Buche, E. Maffre, and P. Chevaillier, "Multiagents systems for virtual environment for training. application to fire-fighting," *Special issue "Advanced Technology for Learning" of International Journal of Computers and Applications (IJCA)*, vol. 1, no. 1, pp. 25–34, juin 2004.

Cédric Buche (1979) is an assistant professor in computer science and works at the European Center for Virtual Reality (CERV). He is working on the use of behavior modeling agents applied to virtual environments for human learning. He is the leader of the Pegase project in MASCARET.

Cyril Bossard (1980) holds a Ph.D in Sport Sciences and works at the CERV. His research interests include decision-making in dynamic and collaborative situations and transfer of learning, applied to Virtual Environments.

Ronan Querrec (1973) is an assistant professor in Computer Science and works at the CERV. His research concerns virtual environments for training. On this theme, he is working on the MASCARET virtual environment meta-model project.

Pierre Chevaillier (1960) has been an associate professor in Computer Science at the Computer Science Laboratory for Complex Systems (LISyC) since 1996. He has been the head of the CERV since 2008. His research interests lie in virtual reality, knowledge representation and multi-agent systems. His research aims to develop virtual reality environments for human learning, based on adaptive tutoring systems (MASCARET project).