Pluggable Advisors as Epiphyte Systems

François Pachet [1], Sylvain Giroux [2], Gilbert Paquette [3]

[1]LAFORIA, Université Pierre&Marie Curie, 4, place Jussieu, 75252 Paris Cedex 05, France - E-mail: pachet@laforia.ibp.fr

[2]IREMIA, de La Réunion, 15, avenue René Cassin, 97489 St-Denis, La Réunion, France - E-mail: giroux@iremia.fr

[3]LICEF, Télé-Université, 1001, rue Sherbrooke est, H2X3M4 Montréal, Canada - E-mail: gilbert_paquette@teluq.uquebec.ca

Summary

Most "over the shoulder" advisor systems (OSAS) are developed as self-contained systems, whose components are designed from scratch. The advisor architecture we present here departs from this approach. An OSAS is seen as an independent, non disturbing, neutral system (hence the adjective epiphyte), that observes the activity of a host system, according to a given viewpoint. Our architecture provides a model for specifying viewpoints on an application's activity, and the desired structure of the OSAS. Viewpoints are represented by a hierarchical structure called a task tree. Once specified, the task tree is used to generate automatically the OSAS, that feeds on the application. As ivy grows on a tree, our OSAS grow on the application and adapts to its evolution..

Keywords: advisors, authoring tools, reusable components, multi-agent systems.

1. Introduction

The context of our work is the formalization of pedagogical expertise, and the development of architectures and tools for the design of tutorial systems in the context of remote teaching (as practiced by the Télé-Université, Montréal). Pedagogical expertise as a whole is too vague a notion to consider realistic generic frameworks that would capture its essential characteristics. Therefore, we address here only a subset of this expertise, i.e. knowledge necessary to "give relevant advice" to students according to their apparent behavior. Our general context of study is "over the shoulder" advisor systems (hereafter referred to as OSAS).

The design of OSAS has been an extensive object of study. [Jonassen & Grabinger 91], [Wilson & Jonassen 90] propose modules that produce contextual help for authoring tools; [Paquette 92] describes an intelligent domain-dependent help for tutorial systems in the context of experiments in physics, to name but a few. However, most advisor systems are designed and developed either as self-contained systems, or as components of top-down designed, integrated pedagogical environments. As, a consequence, the design of OSAS is "buried into" the design of large pedagogical systems, which makes transposing their architectures in different contexts a difficult task. Our approach in the construction of OSAS is to consider them as independent software components and not as parts of a pre-determined self-contained system. Separating the design of advisor components from the design of learning environment (authoring tools, tutorial systems) has enormous benefits. From a conceptual point of view, it provides a division of the knowledge acquisition task that helps to reduce the complexity of learning environment design. From a knowledge engineering point of view, this approach increases software reusability. Our ultimate goal is to consider OSAS as extensions of *existing applications*, which were not necessarily designed to be parts of a tutorial system.

More precisely, we consider an OSAS as made of two components: 1) the *host system* and 2) the *advisor system* per se. The host system is any application involving interactions with a user or a set of users. The host system interactively performs some sort of "reasonable", well-defined task, which defines the subject-matter to be taught, by *interaction* with a user. The advisor system performs an independent task, which consists in producing pieces of advice to the user, by analyzing its observations of the interaction between the user and the host system. It is a distinct system with its own identity and its own structure, and is connected to the host system, so as to receive all the necessary information. The *structure* of the advisor system and its *connection* to the host system is precisely the subject matter of this paper.

This paper proposes a general framework, as well as an implementation and an environment for building OSAS. The paper is organized as follows. In section 2 we characterize host and advisor systems and stress on two dimensions of advisor system that we consider essential. In Section 3 we propose our solution, the epiphyte systems, based on a botanical analogy. Section 4 describes an implementation of the architecture and its complete environment called EpiTalk. Section 5 is an overview of current applications. We conclude by describing current and future extensions of the architecture based on the current results.

2. Characterization of host systems and advisors

2.1 Nature of advice

Let us first specify what we mean by "advice" in tutorial systems. The notion of advice may encompass a great variety of counseling actions. Advice may be issued according to two main modes: "opportunistic" or "on demand". In both cases, the advice does not interfere with the normal user's interaction. The common characteristic of advice may be identified as any "neutral action", i.e. - from the computer point of view - an action that does not perturb the functioning of the system's behavior. A typical neutral action is the display of text in a specialized "advice" window. A typical counter-example is a directive action such as: closing a window, opening new ones, or triggering arbitrary actions. Here are some typical examples of advice that we want our OSAS to be able to generate:

1 - You should try to expand the window you are currently looking at (because useful information may then be visible to you).
2 - You could try to use the spreadsheet option to sort your results (you have been playing with the same drawing tools for half a hour, with the same parameters, and I don't think it is a good idea).
3 - You should try to identify two variables, consider the other ones as constants, and do some experiments using the simulator tool (You seem to be lost in the complexity of finding 5-variable invariants).
4 - In the beginning of your work with the system you started an experiment session, then shifted to a drawing session then shifted back to a new experiment session. Consider exploiting the "define new attribute" tool.
5 - You only use the three first items of any list presented to you. Do you know how to use the scroll bars ?
6 - You never use the menus associated with the right button of the mouse. One of them allows you to compare two elements of a list. Try this in the case "x = y".
7 - The first item in the stack is usually not interesting. Shift directly to the second one.
8 - The button which you are pointing to activates an irreversible operation. Are you sure you want do delete the results of your previous experiments ?

As we see on these examples, some pieces of advice are totally context-free, and may be seen as demons: they are triggered as soon as a given configuration of parameters is reached (this is typically the case of advice #8 or help balloons in the Macintosh finder). Other are context-dependent and take the user's history into account. This is the case of those based on an analysis of regularities in the history of the user (such as advice #5 and #6). Other are not only context-dependent, but depend on a specific point of view the advisor has on the user's activity (this is the case of advice #3). A precise definition of what is advice is difficult because of the great variety of levels of abstraction. However, we distinguish in this paper two important characteristics of advice:

1) advice is given according to a specific *viewpoint* the advisor system has on the activity of the student

Advice may focus on various aspects of an application. For instance, we can issue a piece of advice based on analysis of the result of the user's activity (e.g. a courseware in the case of authoring tools), or based on the user observed reasoning process. In the first case, the advisor is interested in the *coherence* between various parts of the user's production, (e.g. between the objectives of a course designed with a didactic engineering workbench [Paquette & al 94]). In the second case, the advisor focuses its attention on the various steps followed by the user in its problem solving process, e.g. according to a model of the specific scientific methodology in the discovery of a chemical law. This notion of viewpoint represents each particular orientation of the advisor system, including the subject of its attention and study.

2) there are several *hierarchical levels of abstraction* of advice

As showed by the examples, advice does not necessarily address the same conceptual level, within a given viewpoint. This variety of levels corresponds to a variety of reasoning processes and is hierarchical by nature. Some pieces of advice are issued according to local and ephemeral information. Others need more information, only accessible from a global perspective on the user's activity. Others manipulate abstract information (such as "you started a plotting session, then shifted to a simulation session") which is itself the result of lower levels inferences.

2.2 Nature of Host systems

As mentioned above, we aim at building advisor systems as extensions of the application about which counseling is given. This separation favors the vision of OSAS as *pluggable* components. To achieve such a separation, a clear distinction is set between the application and the advisor. We call the *host* system, the application about which advice is given. The activity of the host system is perceived by the advisor system according to a certain analysis scheme.

In our remote-teaching context, we need to design advisor systems either for one student using a set of tools to perform a given task (Fig. 1) or for many students and teachers interacting with each other through computerized tools (Fig. 2). In any case what is important is not the nature of the actor, but the nature of the interactions. This leads us to adopt a systemic point of view: we consider the user(s) as *parts* of the host system, and define host systems as the union of a set of *tools* AND a set of *users*. A user is any agent interacting with a computer application. In our context of remote-teaching, users are typically students and teachers. Tools are software components of an application, such as windows, plotters, elements of the multi-media environment. Because we consider the host system as the union {tools} U {users}, *interactions* of users with tools are considered as *intra-actions within* the host system. This systemic definition of a host system leads naturally to the definition of an advisor system as a system that *observes* its host (from the outside) in order to produce advice (to the users, considered as parts of the host system). Host systems considered as observable dynamic systems have four major characteristics with respect to intra-action nature, intra-action ordering, distribution and dynamicity.

1. *Intra-actions* within the host is the only available food of advisor systems
Ideally, an advisor system would need to know the detailed internal mental states and activities of users to produce relevant advice. Unfortunately the only thing the system can observe is the intra-actions *within the host system*, either between one user and tools or between users through tools. Our formalism has to enable the specification of 1) which parts of the host system are interesting to observe for a given viewpoint, and 2) which precise intra-actions are relevant for a given level of advice.

2. The *sequence* of intra-actions is usually not constrained in host systems
Since we do not want to impose constraints on the way users use tools, no assumptions can be made *a priori* on the sequence of intra-actions. To take account of this non-deterministic aspect of users activity, advisors should be able to manage unexpected sequences of interactions.

3. The host system is *distributed*
Users, tools, and the advisor system constitute an inherently distributed system. Users are distributed as they work on different machines. Tools also may be distributed either physically as in a virtual classroom [Paquette & al 93] or conceptually, as in multi-windowing systems. Finally the advisor's reasoning *as a process* is independent of the host system activity and may be performed on a remote machine for reasons of efficiency.

4. *Dynamicity* of host systems

During its lifetime, the host system evolves dynamically through the introduction of new tools (e.g. opening a window), and the disappearance of old ones (e.g. closing a window). The advisor system has to follow the evolution of its host in order to get the right information at the right time. As its host evolves, the advisor must plug itself in new parts of the host and release its grips from obsolete parts. Our formalism has to allow for the expression of these changes, so that the advisor system may adapt itself to new configurations of the host system.

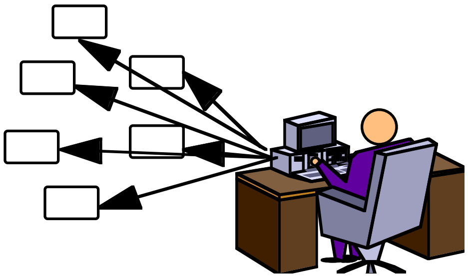

Fig. 1 one user interacting with a set of tools.

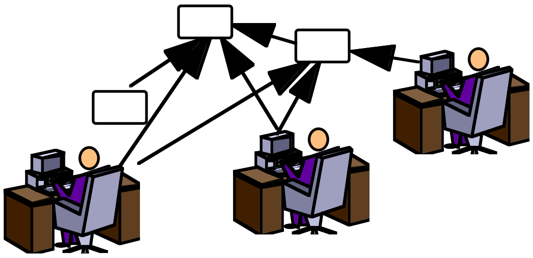

Fig. 2. Users interacting through a set of tools

3. Epiphyte systems

Nature has a remarkable knack for creative distributed problem-solving. As it is often the case in Artificial Intelligence, the complex problem we are addressing has already been solved in the real world by Nature, with a special kind of organization, well-known by botanists: *epiphytes* plants. Epiphytes plants grow on host plants without causing them any damage (as opposed to *parasites,* which cause damage to their host, and - worst in the hierarchy - *predators*, who kill their host). Ivy, and most orchid flowers are typical examples of epiphytes plants. They live a life of their own, but need the presence of an existing living organism to grow on, with which they entertain a special kind of symbiosis. By analogy, our solution is to consider plugabble advisors as "epiphyte" systems, i.e. as systems growing onto other systems without perturbing them whatsoever. Based on this metaphor, we propose an architecture for advisor systems that provides a solution to the four problems mentioned above: 1) advisors have a hierarchical structure that allows the variety of levels of advice to be expressed, 2) advisors have a life of their own, with respect to sequentiality: their survival is independent of any hypothesis made about the sequence of actions within their host system, 3) advisors can adapt to distributed environments, and finally 4) they can adapt to the temporal evolution of the host system.

This botanic metaphor is the starting point of our architecture, named EpiTalk. EpiTalk provides a precise framework to design advisor systems in a way that reflects the characteristics of advice (viewpoints-dependency and hierarchical nature), while respecting the four above-mentioned characteristics of host systems. In the next five sections we describe how EpiTalk allows advisor systems to be specified as epiphyte systems whose structure is deduced from this botanical metaphor.

3.1 Viewpoints as task trees

We naturally consider epiphyte systems as multi-agent systems [Gasser 91]. Our motivations for using multi-agent architectures differ from previous approaches ([Girard & al 92], [Futtersack 90]) in the sense that we focus on the social organization and intra-structure of the multi-agent system rather than on the communication paradigms. As we will see, the structure of our multi-agent advisor system is deeply rooted in the interpretation of the notion of viewpoint, which we will describe now.

Our multi-agent architecture is based on a particular representation of view points. In Epitalk, a viewpoint is materialized by a hierarchical structure called a *task tree*. This tree contains nodes which represent abstract *tasks*. Each node may have 0 to n descendants. The relation between a node and its descendants expresses composition. The word task historically refers to the notion of *pedagogical task* introduced by [Paquette & al 94]. Here, it is to be understood as a part-of (in the hierarchical sense of the term) a given viewpoint on the activity of the system. The notion of task is intentionally vague:

tasks may either refer to generic tasks - in the sense of [Chandrasekaran 87] - to be performed by the student, or to parts of the student's duty, or any other hierarchical structure describing the host system's activity. Fig. 3 (on the left) shows an example of a task tree drawn from a tutorial system helping students discovering laws in physics [Paquette 91]. In this system the task tree represents the hierarchy of pedagogical task to be performed by the student. Terminal tasks correspond to actual tools provided by the system. Intermediate tasks represent abstract tasks such as "making experiments", "analyzing results" or "finding examples and counter-examples".

The task tree is at the heart of the advisor system. The task-tree is used as a *template* to generate the multi-agent system. Each task of the task-tree contains information which is used to generate the multi-agent advisor system, and to adapt it to the evolution of the host. As we will now see, the task tree will 1) dictate the structure of the multi-agent system 2) contain specifications for linking agents to the host systems intra-actions and 3) contain specifications for adapting the advisor system to the host evolution.

By definition, the task tree contains three kinds of nodes: task nodes, star nodes and leaves or terminal tasks. Task nodes are non-terminal nodes, which may have any number of descendants. Terminal nodes do not have descendants. Star nodes are particular task nodes, which may have any number of "instanciations". These nodes are used to represent the ubiquitous notion of "0 to n" occurrences of a task. The various instanciations are generated dynamically as the user navigates in the system.

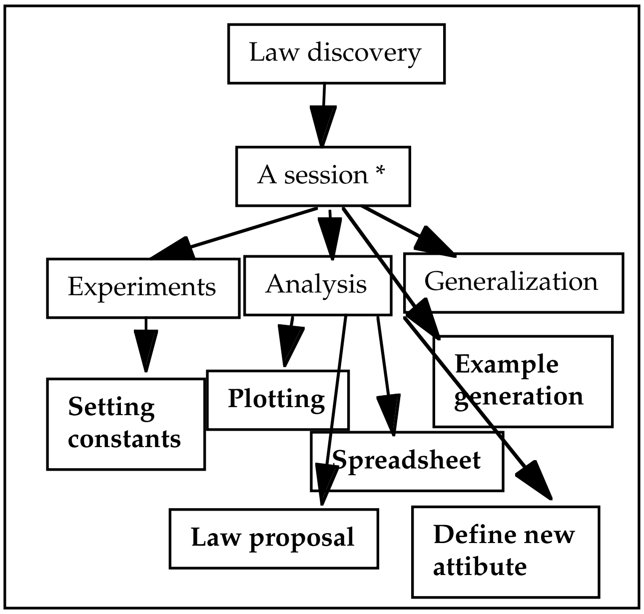

Figure 3. An example of a task graph.

3.2 Structure of the multi-agent system

The task tree as defined above provides a natural solution for the structure of the multi-agent system: we define the structure of the multi-agent system as *isomorphic* to the chosen viewpoint, i.e. task tree. Practically this means that each task of the task tree will generate one agent in the multi-agent system. We distinguish between terminal agents (corresponding to terminal tasks) and non-terminal agents (corresponding to non terminal task nodes). Star nodes are expanded during the evolution of the host system, to generate 0 to n agents (see below § 3.3.2), according the actual dynamic evolution of the system. This isomorphism yields the structural organization of the multi-agent system. The hierarchical nature of advice is therefore represented by the hierarchical organization of agents of the advisor system, which is itself isomorphic to the hierarchy defined in the task tree.

3.2.1. Plugging an Advisor

To adapt the structure of the advisor to the structure of the host system, we need to introduce a link between the host system and the epiphyte system. This link - called *epiphyte link* - is based on the principle that *only terminal agents are connected to the host system*. They are responsible for observing the host, and collecting intra-actions. Each terminal node in the task tree contains specifications of which parts of the host system has to be observed, and which particular intra-actions are relevant for this level of advice. In our object-oriented setting, this specification is a set of *classes* that have to be spied on and, for each of these classes, the list of "messages of interest".

3.2.2. Evolution of the host system

The evolution of the host sytem is materialized by the creation or deletion of "relevant" objects. The evolution of the host system is taken care of by two kinds of specifications in the task nodes. The adaptation of the epiphyte system to the host evolution is done either by connecting existing advisors to the newly created object (resp. un-connecting advisors from deleted objects), or creating new advisors for the newly created object (resp. deleting advisors who are no longer in use). These specifications are contained in the corresponding task nodes. Similar to the epiphyte links, in our object-oriented setting, these specifications will correspond to message selectors.

3.3 Inter-agent communications

Now that the structure of the multi-agent system is specified by a task tree, we have to say how agents communicate between them. We introduce a communication scheme based on a bottom-up traversal of the tree. Each time an agent receives information it 1) processes the information (to issue local advice or to update a model of the activity being observed) and 2) transmits to its direct father any information he considers relevant. This scheme is applied recursively for any agent of the tree, terminal or non-terminal, until the root agent is reached. Terminal agents receive information directly from the host system, whereas non-terminal agents receive information from agents below them in the hierarchy. This bottom-up interpretation of the agent tree naturally reflects the various levels of abstraction: as information sifts up in the tree it is processed by agents that interpret it according to their own local vision of the host system. When an intra-action is detected in the host system, it is sent to the terminal agents that are interested in this action. These terminal agents process the information and transmit information to their father, eventually reaching the root agent. Each terminal agent has a local vision of the system. Intermediate agents represent intermediate aspects of the system's activity according to the given viewpoint. Only the top agent of the tree has a global vision of the system.

This hierarchical structure of agents together with the communication scheme is the backbone - and the only one ! - of the advisor system, for a given viewpoint. Several viewpoints for a given host system will correspond to several task trees, which will generate several different multi-agent advisor systems. Now this backbone in itself serves as a repository for representing knowledge used to produce advice. This is the subject of the next section.

3.4 Agent Structure and Organization of Knowledge

The internal structure of agents is based on an anthropomorphic metaphor: each agent has a set of *anatomical parts.* Each anatomical part represents a particular type of activity the agent is supposed to perform. For example, updating a memory of the recorded activity is performed by a part called `AgentMemory`. Extracting regularities will be performed by a specific part called `RegularityExtractor`. Performing local inferences will be performed by a part called `RuleBaseActivator`, and so forth. More generally, <u>the script of an agent is the same for all agents in the tree</u>. When an agent receives information it:

- transmits the information to all its anatomical parts and
- transmits an abstract résumé of its activity to its father in the agent tree.

The behavior of a particular agent is therefore entirely specified by the set of its anatomical parts. One of the main benefits of this solution is to allow the construction of libraries of reusable anatomical parts. The next section gives some details about a particular implementation of EpiTalk which includes an implementation of these anatomical parts.

4. EpiTalk-80

EpiTalk is an abstract architecture, that can be implemented in various ways. EpiTalk-80 is an implementation of our architecture in Smalltalk-80, together with a complete environment for specifying epiphyte systems. No hypothesis is made on the nature of the host system. Thanks to a novel and generic spying mechanism, any Smalltalk application can be used as an host system for EpiTalk. EpiTalk-80 includes a sophisticated interface to specify epiphyte descriptions, generate the corresponding multi-agent system and launch the OSAS.

4.1 Specification of epiphyte descriptions

In EpiTalk-80, epiphyte links are specified by indicating, for each terminal agent, which classes he is interested in, and for each of these classes, which message he wants to analyze. This specification is associated to each terminal task. When the multi-agent system is generated, the corresponding terminal agent will find all the necessary information in the terminal task to generate "spies" (described below) which will track the corresponding messages, and warn the agent when one of these message is sent.

4.2 Realization in Smalltalk-80

We distinguish three components in the implementation of EpiTalk-80: 1) the multi-agent architecture, based on the ReActalk multi-agent system [Giroux 93], 2) The inference architecture based on the NéOpus embedded rule-based system [Pachet 92], and 3) the spying mechanism. We only outline two interesting features of this combination here, and refer to the literature for more details.

4.2.1. Factoring out common rules

In NéOpus, rule bases are activated in a forward-chaining fashion and the result of inferences is to modify the agent's local model of the activity in the host system. A powerful feature of this rule-based mechanism is a mechanism for factoring out common rules of a set of rule bases, called Rule Base Inheritance [Pachet 92b]. This mechanism allows the construction of rule bases as subclasses of existing rule bases. In our context, this mechanism is used to refine or specialize existing rule bases to adapt the reasoning to particular situations. For example, advice #1 and #5 are typically common to a large number of tools. Instead of specifying this knowledge for each advisor agent, we define a general-purpose rule base containing such general advice, and refine it for each particular advisor.

4.2.2. The spying mechanism

Based on so-called *organizational reflection* [Giroux 93], a novel and original spying mechanism was developed for EpiTalk to link advisor agents to the various components of the applications. The main idea behind the spying mechanism is to intercept messages sent to spied objects by substituting physically the spied object with an object that represents a *specialized spy*. Each time a spy receives a message, it alerts interested terminal advisors. The spy knows which advisors may be interested by the message, from the specification of the epiphyte description. Then the spy sends the message to its original receiver so that the system behaves as if no interception occurred. This mechanism allows the epiphyte multi-agent system to observe the host system without perturbing it, and to adapt dynamically to the host system's evolution.

5. Applications

The epiphyte architecture presented here is used in a number of applications. The first one is the design of a generic didactic workbench (the DEW system [Paquette & al. 94]). In this system, the (only) user is a teacher, tools are various windows in which the user specifies the characteristics of his course, according to various dimensions. An advisor system was built on top of the existing application, to produce advice to the user (here a teacher) to help him formulate his teaching strategies, course contents and interface. The task tree in this epiphyte system corresponds exactly to the pedagogical task graph, which decomposes the activity of the user into several *projects*, themselves having several *objectives*, and *teaching units*, and so forth. The task graph contains around 15 types of nodes. The expertise, whose extraction is still in progress, will contain around 300 rules, dispatched and organized according to the various levels of the task graph.

An other successful application of epiphyte systems was started to add an advisor system to an existing set of tools built from the law discovery tutorial system [Paquette 91]. These tools aim at helping students formulate physical laws using a simulator, test them and generalize them. This application is still in progress but has already showed promising results [Paquette & al 94b].

A third substantial application was made to turn the Smalltalk-80 environment itself into a tutorial system for beginners, taking full advantages of the reflective aspects of Smalltalk. Improved browsers and debuggers are built on top of the existing tools without modifying the source code, which adapt to the beginner's behavior, and suggest useful advice (such as advice number #1, #5 and #7). Finally the epiphyte architecture presented here is applied to other non-learning environment, such as debugging in actor languages, and dynamic typing of smalltalk programs.

6. Conclusion

We presented a novel architecture for the construction of *over the shoulder advisor systems*. This architecture is based on a botanical metaphor, and considers advisors systems as epiphyte systems that feed on any distributed application, without damaging it. The architecture proposes a representation of the epiphyte system as a multi-agent system articulated around a task graph, which is the materialization of a particular viewpoint on the host system's activity. The main advantages of this architecture are 1) to provide a framework to organize knowledge used to produce advice which takes into account the hierarchical nature of advisors, and 2) take into account the multiplicity of viewpoints advisors may have on a given host system. Secondly, our architecture promotes software reusability in two ways: by allowing any application to be turned into a potential host system, and by allowing the construction of a libraries of reusable anatomical parts for advisor agents.

7. References

[Chandrasekaran 87] Chandrasekaran B. *Towards a functional architecture for intelligence based on generic information processing tasks*. Proc. of the 10th IJCAI, Milan, Vol. 2, 1987, pp. 1183-1192.

[Futtersack 90] Futtersack, M. *QUIZ: A Multi-Agent Architecture for Intelligent Tutors.* Ph.D. thesis (in French), University of Paris 6, France, Dec. 1990.

[Gasser 91] Gasser, L. *Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics*. Artificial Intelligence, 47, pp. 107-138, 1991.

[Girard & al. 92] Girard, J. Gauthier, G. Levesque, S. *A Multi-Agent Architecture.* Proc. of ITS '92, Montréal, Canada, pp. 172-182. Lecture notes in Computer Science 608, Springer-Verlag, 1992.

[Giroux 93] Giroux, S. *Systems and agents : a necessary unity*. Ph.D. thesis (in French), University of Montréal, Québec, Canada, August 1993.

[Jonassen & Grabinger 91] Jonassen, D.H. Grabinger, S. *Instructional Design and Development Advisor: Intelligent Job Aid, Help System, and Intelligent Authoring System.* Division of Instructional Technology, university of Colorado, Denver, 1991.

[Pachet 92a] Pachet, F. *Rule Base Inheritance*. Conference "Object-based representations", La Grande Motte, France, June 1992.

[Pachet 92b] Pachet, F. *Knowledge Representation with Objects and Rules: the NéOpus System*. Ph.D. thesis (in French), University Paris 6, France, Sept. 1992.

[Paquette 91]. Paquette, G. *Metaknowledge in learning environments*. Ph.D. thesis (in French), Université du Maine, France, October 1991.

[Paquette 92]. Paquette, G. *Metaknowledge in the LOUTI development system*. Proc. of CSCSI-92, Canadian Society for Computational Study of Intelligence, Vancouver, Canada, May 1992.

[Paquette & al 93]. Paquette, G. Bergeron, G. Bourdeau, J. *The Virtual Classroom revisited.* Conference TeleTeaching, Trondheim, Norway, August 1993.

[Paquette & al 94] Paquette, G., Crevier, F. Aubin, C. Frasson, C.. *Design of a Knowledge-based Didactic and Generic Workbench*. Same conference.

[Paquette & al 94b] Paquette, G. Giroux, S. Crevier, F. *Méthodes et outils de développement de systèmes conseillers dans les environnements de formation*, Internal research report, LICEF, Télé-Université, Montréal, Dec. 1993.

[Wilson & Jonassen 90] Wilson B.G. Jonassen, D.H. *Automated instructional systems design: A review of prototype systems.* Journal of Artificial Intelligence in Education, 2 (2), pp. 17-30, 1990.