Exploratory and Experimental Learning ? For Teachers and Researchers too !

Ruddy Lelouche

Version sent on 25 October 2005, updated on 23 April 2006

EXPLORATORY AND EXPERIMENTAL LEARNING... FOR TEACHERS AND RESEARCHERS TOO!

Ruddy Lelouche

Département d'Informatique et de Génie Logiciel
Université Laval, Québec, CANADA
Ruddy.Lelouche@ift.ulaval.ca

Laboratoire d'Informatique, de Robotique
et de Microélectronique (LIRMM), Montpellier, FRANCE
Ruddy.Lelouche@lirmm.fr

ABSTRACT

The term "exploratory learning" usually denotes some type of activity used by students, with or without a teacher's supervision, to facilitate the learning and mastery of a predefined and relatively circumscribed domain. That exploration may be more or less guided, according to the domain structure and the teacher's pedagogical goals and strategies. But there is also another way to use exploratory and even experimental learning, which may be used inside or outside an educational setting: rather than students, the actors would be researchers and research-minded teachers, and their goal would be to discover and structure a new, unexplored or ill-structured domain. Using two existing computer systems as examples, this paper attempts to show how this is possible, and compares the proposed exploratory and experimental learning by teachers and researchers with the more traditional exploratory learning by students.

KEYWORDS

Constructivist approach, structured exploration, researcher learning, user-centred learning, unstructured domain.

1. INTRODUCTION: TWO APPROACHES TO LEARNING

There are two main approaches to learning, especially for problem-solving domains. One is based on the how-to-do question; it uses essentially imitation, and consists in working out examples similar to those presented in a manual or by a teacher; in our opinion, this leads to a surface understanding of the method or procedure to be learned. The other approach is based on what and why questions; it uses a more constructivist approach: the learner first studies the procedure or method to be learned, and then verifies her understanding of the procedure by working out a variety of examples, some very dissimilar to the ones known; in our opinion, that leads to a deeper understanding and a better learning. The former approach reflects what students have a tendency to do more and more often, because it is less time-consuming, and more efficient for quickly preparing exams or quizzes. The latter is what teachers (in the broadest sense: from assistants or coaches to university professors) would like students to do for a more permanent learning. In that perspective, "Think then code!" is Koffman and Wolfgang's [2005] motto to present their well-known textbook on programming.

This paper is concerned essentially with the second approach to learning. More specifically, the question it addresses is: since both approaches rely on working out examples, what makes the difference between an imitative exploration and a constructivist one? The answer is what we shall call *structured exploration*.

Section 2 deals with the traditional students' understanding and learning processes, using as an example the ALGORITHMIK system for acquiring elementary programming skills. Section 3 deals with the discovery and construction of knowledge (another form of learning) by teachers and researchers, and it uses as an example PILÉFACE, a system to tackle the pragmatics of language exchanges. Section 4 bundles up these two facets to present a more general view of exploratory and experimental learning by students, teachers and researchers.

2. STUDENTS' LEARNING: KNOWLEDGE ACQUISITION

In this section, we first briefly present the experimental setup of ALGORITHMIK, and then show how this setup facilitates students learning and what this learning consists of.

Proc. of the *IADIS International Conference Cognition and Exploratory Learning in Digital Age (CELDA 2005)*, Porto, Portugal, 14 -16 December 2005, p. 167-174. (ISBN 972-8924-05-4 © 2005 IADIS)

2.1 The experimental setup of ALGORITHMIK

The ALGORITHMIK system [Dion, 1988; Lelouche, 1999] is an intelligent tutoring system aimed at learning algorithmics and programming, designed for college students in an introductory programming class. Its main objective is to help the student to learn to use in an appropriate fashion basic control structures (IF THEN, IF THEN ELSE, WHILE DO, ITERATE *n* TIMES) and procedure calls. To that effect, ALGORITHMIK uses a micro-world called "Karel the robot" [Pattis, 1981], in which the student must program a robot to make it achieve various tasks that require using the control structures under study. As a major advantage, this micro-world requires no data structure, thus separating two difficulty types commonly felt by beginners: data structures and control structures. Besides, this environment favours a top-down and user-centred design, thus leading the student into applying, from the start, sound software engineering principles: at any point while developing her program, she can freely name any intermediate subtask (a cognitive act) for later development, thus keeping her focus at the abstraction level most meaningful at that stage of the problem-solving process.

2.1.1 Tutor roles

To help the student solve her problem, the system includes an intelligent tutor. However, that one intervenes only at the student's request. The student keeps absolute control over the progress of the learning session: if she wants, she may develop, test and correct her program without having to call or to be interrupted by the tutor. That tutor has in fact three circumscribed roles:

1º Problem selection. The first role of the tutor is to choose the next *problem* to be submitted to the student. The available problems are categorised according to their assumed difficulty: decision structures, iterative structures, or all control structures. When starting a session, the student may choose a category, then her first problem in that category; afterwards, she is guided into working on more difficult problems. Alternatively, she may relinquish the use of the tutor altogether and devise her own problem. In that case however, since the system "knows" nothing about the task to be performed by the robot, the student is left alone (no tutoring).

2º Advice. During her programming activity, the student who is stuck may ask the tutor a hint (*Hint* command) so that she may complete the program part being worked on. Depending on the adopted programming model, that hint will be higher-level (corresponding to higher level goals) when the student begins the development, and closer and closer to the final code as the student gets nearer the leaves of the functional decomposition tree. If the tutor-given hint is insufficient, the student may further ask for a certain part of the code (*Help out* command), which will allow her to continue developing her program.

3º Verdict. At any time during the session, but especially when her program is nearly completed, the student may ask the tutor to verify her solution, whether it is final or only partial (*Check program* command). In this case, the tutor will give one of the four following verdicts:

(1) your program is correct;
(2) the part of your program presently coded is correct;
(3) your program is incorrect (at the student's request, the tutor may reveal the error location, then its description);
(4) your program seems to be correct, but I do not understand the strategy used (this verdict, only possible for a completed program, is not aimed at helping the student, but at possibly improving later the knowledge base of the solving strategies known to the system, i.e. the system cognitive effectiveness).

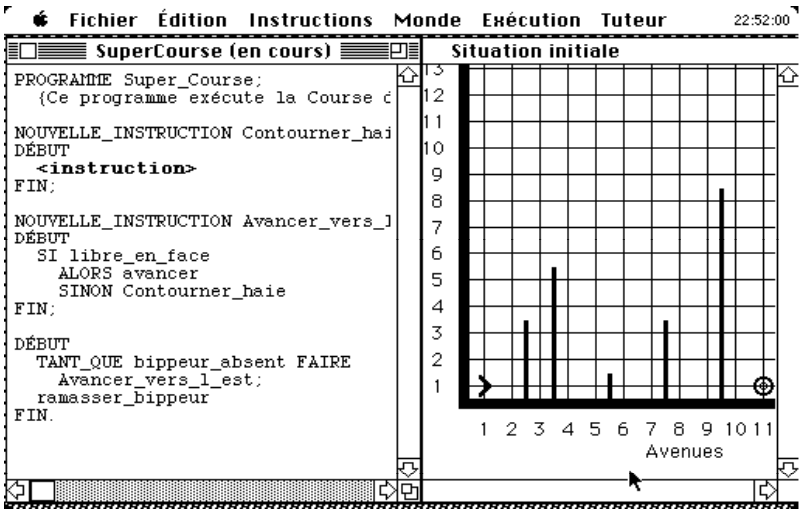

Figure 1. — ALGORITHMIK environment interface.

2.1.2 Graphical components

Besides the tutor, ALGORITHMIK also provides the student with three interactive components: a structure editor for code entry, a graphic situation editor for setting and modifying Karel's environment, and a graphic simulator for executing and tracing the program (see figure 1).

2.1.3 Using the system

Let us assume that the student decides to use the *tutor* and to work on the first of the "all control structures" category, the super-steeplechase problem [Lelouche, 1999]. She is then presented with the following task description (here translated into English):

> Karel must run a super-steeplechase. The hurdles have a variable height and a zero width. The end of the run is marked by a beeper. Karel begins its run facing east; at the end of the run, it must also face east and is supposed to have picked up the beeper ending the run.

and with a display like that laid out on figure 1, showing a program window in which she can work on her program using the *structure editor*, and an example of an initial situation.

If the student is not happy with the proposed situation, she may modify it using the *graphic situation editor*, which allows her to place the robot at the desired intersection and with the desired orientation, and to place bippers and wall sections or move them around as she wishes. The structure editor and the situation editor are convenient interactive graphic tools to use, especially to vary the program situation parameters through experimentation. The *execution simulator*, which can be activated using the *Execution* menu (figure 2), is used to visualise, on the active situation, the execution of the active program window. In addition, the instruction currently being executed is simultaneously highlighted in the program text; thus, at all times, the cognitive relationship between the active program instruction and the result of its execution is visible. Besides, at any time during execution, the student can change the execution mode (*Fast speed*, *Slow speed*, or *Stepwise*), or can activate the *Pause* and *Abort execution* commands (see figure 2).

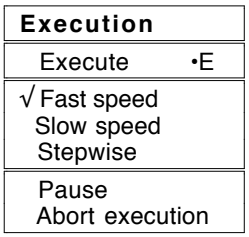

Figure 2. — ALGORITHMIK Execution *menu.*

2.2 Learning by students

The ALGORITHMIK tutoring strategy is based on the constructivist paradigm [Boder & Cavallo, 1990; Harel & Papert, 1990; Papert, 1980]: the learning process is not a passive one where the student stores new information, but an active one where the student modifies and reconstructs her vision of the world. Under these conditions, a programming tutor is not to give out ready-made solutions. Instead it should ease the student's apprenticeship of program development, admittedly following some method, but nevertheless allowing the student to *experiment*, using a trial and error approach. It is thus necessary to guard against offering feedback too quickly and instead to give the student enough time to make mistakes, discover them and repair them. In this context, the tutor is there essentially for helping out the student (if the need arises), or for signalling her unnoticed errors.

Moreover, one learns through developing one's own viewpoint [Papert, 1980] much more than through adopting someone else's. Thus a tutoring system should allow the student:

- to express what she thinks and what she wants to do,
- to formulate hypotheses, and if necessary correct these hypotheses,
- to adopt several viewpoints and try various solution methods.

In that perspective, immediate feedback prevents exploration and negates the constructivist approach to learning. Certainly a directive style, with immediate unconditional feedback, because of its "bullying" the student, gives the fastest results, as verified by the ACT-R team itself [Anderson & al., 1995]. But that does not mean that the student has better integrated the programming abilities on which she has been trained, and, above all, she still will not have learned how to debug a program that proves erroneous, especially if someone else's!

In that context, the ALGORITHMIK facilities help the student follow the program execution, both on the situation field and in the program code. In addition, by slowing down or even stopping the execution at any program point that she may find unclear, she is able to gather some information, possibly to modify the local

situation (Karel's location and orientation, bippers, and wall sections), in order to understand better how the program really works, and thus to correct her errors eventually. The execution simulator capabilities are particularly useful for problems invented by the student or by the teacher, as part of exploration and experimentation, where the help of the tutor is not available.

The ALGORITHMIK system has later been extended to incorporate various data about the characteristics of the programs being worked on by the learners and about the learners' actions. The system thus modified, called TIRPA (*Tuteur Intelligent pour la Résolution de problèmes en Algorithmique*), was used to make *experiments with actual students*. Jacques Malouin, a student of ours, conducted these experiments, and their results are reported in his Ph.D. dissertation [Malouin, 1993]. However, detailing them is outside the scope of this paper.

The ALGORITHMIK-TIRPA software thus appears to be an interesting alternative to more complex tools (like using a more realistic robot). Indeed, it is easy to learn and to master, because only what is important is displayed and included in the controls available to the learner. These simple controls allow her to concentrate on the algorithm cognitive design and development, rather than play around with more sophisticated tools.

3. TEACHERS' LEARNING: KNOWLEDGE DISCOVERY AND BUILDING

We now study a quite different learning domain: second or foreign language acquisition. In that domain, increasing emphasis has been placed on the use of a given speech act in a given situation [Austin, 1962; Searle, 1969] and on the importance and variety of sociolinguistic factors affecting message and form [Coste & al., 1976]. Nowadays, teachers refer not only to the correctness of the linguistic form (linguistic code), but also to the appropriateness of the message it conveys in a given social context (social code). Learners are now being encouraged to take these factors into account as they respond to and express themselves in second language; mastery of these factors constitute a major hurdle for foreign language learners.

3.1 Rationale and experimental setup of PILÉFACE

PILÉFACE (*Programme Intelligent pour les Langues Étrangères Facilitant l'Approche Communicative de l'Enseignement* — Intelligent Program for Foreign Languages Fostering a Teaching Communicative Approach) is a computer program initially aimed at providing language learners with the necessary knowledge to select or recognise the language forms considered to be the most appropriate for receptive and expressive use, and to fulfil a language function in a given communicative situation. We summarise what problems the system was to address and what difficulties had to be tackled in its design; we then show a couple of examples of its use.

3.1.1 Communicative situation variables and rules in language acquisition

When transmitting a message to somebody else, we generally have a *communication intention* or goal [Austin,1962; Searle, 1969, 1979]. A communicative goal is realised through a *message* which, in the oral setting, is an utterance. In general, various utterances may convey the intended meaning, but with a different effect. However only a few of them (and sometimes only one) are usually appropriate, i.e. adapted to the context in which the exchange takes place, or *communicative situation*. That situation can be characterised in terms of variables, as well as rules binding them, which we call the *communicative situation variables and rules*.

In the classroom, the teacher's primary role is to *transmit language knowledge to the students*. Amongst various pedagogical acts, the teacher may *formulate usage rules* to be used in a given communicative situation. However, (s)he does this only occasionally, on demand.

When using a computer, the teacher's aim is to *"transmit language knowledge" to the computer system*, in this case a knowledge-based system, so that the system can subsequently transmit that knowledge to the targeted computer user, i.e. the language learner. For the design of such a system, this yields the following duality:

- *for the language teacher*, the computer is a particular kind of learner, namely one who needs to be provided with variables and rules in an exhaustive way;
- *for the knowledge engineer*, the system must exhibit two kinds of competence, in this case one in the French language and another in teaching French.

The latter refers to the double competence traditionally found in any intelligent tutoring system [Wenger, 1987]. Thus we needed to provide our system with two kinds of knowledge: the native speaker's linguistic knowledge, and the language teacher's pedagogical knowledge [Lelouche & Huot, 1998].

3.1.2 The system modelling approach

In order to tune better the representation model simulating the native speaker, we decided to limit our work first to the *face-to-face greeting intention*: it is one of the first communication intentions expressed by an individual, and it conditions the climate of any subsequent exchange. We thus had to build a computer program "expert" in the various linguistic realisations, or utterances, expressing the face-to-face greeting intention, and that program would then be able to recognise and to produce linguistic realisations expressing that intention.

When trying to provide a computer program with some everyday life or *common sense* knowledge, a general modelling problem is to distinguish between explicit and implicit knowledge. Explicit knowledge is defined and described explicitly in readily available material such as books, courses, movies, etc. When referring to human communication, implicit knowledge is not formally described in any written or visual material, but is nevertheless easily manipulated or processed by native speakers. And that is what makes knowledge transfer from the native speaker to the computer so difficult! In the greeting function case, amidst explicit knowledge, we find *greeting forms* like "Bonjour", "Salut", "Good morning" or "Hello" (we can find them in dictionaries or in language textbooks), or *terms of address* like the first name, the last name, or a title (they are quite presented and explained in [Braun, 1988], for example). However, the *rules* guiding a native speaker when (s)he combines these various forms to produce a linguistic realisation (here an utterance) fitting into a given situation are not formulated explicitly; these rules are thus implicit knowledge, as is most pragmatic knowledge.

In order to get a better understanding of these rules and to make them explicit, we took an approach which can be computerised [Hayes-Roth & al., 1983], and adapted it to build the PILÉFACE system [Lelouche & Huot, 1998]. That approach involves five stages: (1) identifying and structuring the communicative situation variables (that task is hard and still under progress, since previous authors like Preston [1986] had identified some fifty factors influencing the linguistic form, but without making any reference to which factors have a higher priority than others, or which are consequences of others, or — even less — in which way), (2) identifying the rules connecting these variables, (3) defining a hierarchy for these rules, (4) defining a hierarchy of variables, and finally (5) validating the resulting model.

3.1.3 Examples from the PILÉFACE system

Recall that PILÉFACE [Lelouche & Huot, 1998], as well as the learner, must be able to generate linguistic forms, and also to recognise them. In either case, using communicative situation variables and rules (see section 3.1.1), the system first performs some "non linguistic" analysis of the communicative situation at hand. This is carried out by two inference engines [Lelouche, 1994], called *Extracteur* and *Formaliseur* respectively, which yields what we call the *exchange style*. The latter describes linguistic characteristics of a production (a particular form) supposed to meet the pragmatic constraints resulting from the given situation; it consists of five variables, called *level of the exchange style*, *personalization index*, *tonality*, *insistence*, and "*tu/vous*". Using that exchange style, the third engine of the reasoning chain differs, depending on whether the system is to *generate* suitable productions or to *diagnose* the situational appropriateness of a student's production. We now examine each case.

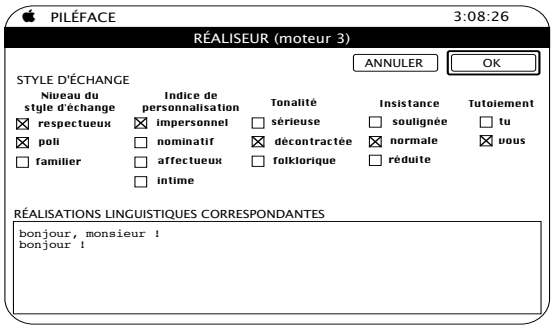

*Figure 3. — PILÉFACE generation example (*Réaliseur *development interface).*

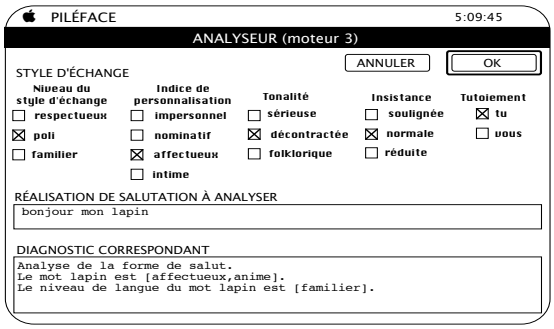

*Figure 4. — PILÉFACE analysis example (*Analyseur *development interface).*

In the generation case, the problem is the following. Given a particular communicative situation and a particular communication intention, one (i.e. the student, or the computer) is to produce one or several appropriate linguistic forms. We call *Réaliseur* the third engine in this case, i.e. the system module responsible

for generating linguistic forms on the basis of the *exchange style* provided it. The *Réaliseur* interface shown in figure 3 displays the five "linguistic" variables making up the current exchange style ("style d'échange") and their possible values, as well as the values retained through check boxes. Using the information shown, the *Réaliseur* generated the possible productions "Bonjour, monsieur!" and "Bonjour", which are rather conservative forms, because of the exchange style of the situation.

In the case of production analysis, the system must be able to analyse a given linguistic form (e.g. submitted by a language learner) in order to diagnose whether it is or is not an acceptable expression of the given intention in the given situation. We call *Analyseur* the third engine in this case. Using a different situation example, figure 4 gives an idea of the *Analyseur* capabilities. The communicative situation is supposed to have led to the shown exchange style (checked values), and the user submitted the greeting realisation "Bonjour, mon lapin!" (Hello, my dove! Actually, "lapin" means "rabbit", but it is unlikely that this term would be used in English as a term of address; this is why we suggested "Hello, my dove!" as an approximate English equivalent). Giving its diagnosis to comment on this realisation, the *Analyseur* found inappropriate the use of "lapin", considering it to be unduly "familier" for an exchange style level defined simply as "poli".

It is worth stressing that the screen snapshots shown in figures 3 and 4 *are not the ones presented to the student* (such an interface would be more like the one shown on figure 5, and would not refer to any communicative or intermediate variable like the ones describing the exchange style). Indeed, they are *interfaces presented to the developers* to check the internal working of the *Réaliseur* and of the *Analyseur*. Similar interfaces exist to demonstrate and assess the capabilities of the first two inference engines [Lelouche, 1994].

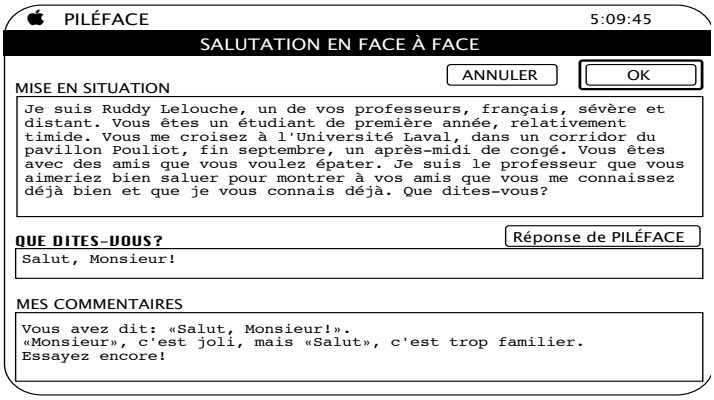

Figure 5. — Student–PILÉFACE interface example.

3.2 Learning by teachers and researchers

The main interest of these interfaces is that they deal with only one module, were it the *Extracteur*, the *Formaliseur*, the *Réaliseur*, or the *Analyseur*, which allows the developers to concentrate on one module at a time. That is why we chose this system as exemplary. Indeed, in every case, such an interface allows the developers (good French speakers and language teachers and researchers) to experiment with the input parameters of the module being tested, e.g. by varying one input parameter at a time, and to observe the corresponding changes in the output parameters of that module (in the case of figures 3 and 4, those are respectively the linguistic forms generated and the diagnostic of the form given as additional input).

Actually, such interfaces can be, and have been, used and exploited in more powerful ways. Indeed, in the validation stage mentioned in section 3.1.2, one is to *explore* the model currently implemented by submitting it to a number of tests: for every test, one is to imagine a different communicative situation (experimentation), and to provide the system with the values of the variables which suitably characterise that situation. By "running the program", one can then assess whether the values of the variables which have been derived according to the model are still consistent with the situation. For every set of tests, the nature of the variables in question and the precision of their values depend on the refinement state of the model when that particular set of tests is performed. That refinement state naturally applies to the variables provided to the system as input data, and also to the output variables derived from them.

The validation stage always yields a positive result, i.e. a meaningful research outcome. Indeed, either the system "passes" all tests or it does not. If it does, that gives us (teachers and researchers) a chance to model more complex or more detailed situations, i.e. *to refine our model*, and therefore the system cognitive capabilities (ability to express and to deal with such situations). Otherwise, some tests uncover situations where the system-provided results are inconsistent with what a native speaker would do or understand; in that case, it means that some rules are not properly specified, and/or that some variables are not appropriately modelling the intended communicative situation; whatever the outcome, that gives us a chance *to correct our model*. In either case, this is a win-win situation, and we always learn!

4. THE EXPLORATORY AND EXPERIMENTAL LEARNING PROCESS

With two different systems as examples, we have shown how exploratory learning can take place. The main difference between the two systems, is indeed between the two learning domains they address. In the first system, the learning domain, elementary programming, is well known and well circumscribed. That allows the developers, mainly teachers and their possible computer helpers, to offer the students a well-designed system, completely tested, in which the student can use exploration (if working on her own problems) and even guided exploration (if using the embedded programming tutor) as a learning activity. In the second system, the targeted domain, pragmatic aspects of language exchanges, has been extensively described in language research, but never structured, and has not been submitted to the computer assessment and validation. As a result, computer technology, although initially aimed at guiding students working on various communicative situations, turned out in fact to be mainly used by teachers and researchers to design, develop and check the system "in progress". Besides, the system is "condemned" to remain under construction, because that is the state of the domain itself!

Table 1. — Comparing learning by students and by researchers when using computer technology.

	For the student	***For the researcher***
Type of learning domain	Well known and circumscribed domain (e.g. elementary know and know-how domains, like programming, physics, mathematics, engineering).	Possibly well described, but largely unstructured or ill-structured domain (e.g. most know-how-to-be domains, like psychology, interpersonal relationships, or... language usage rules).
Characterisation of such a domain	Concepts and relations are well established, and have been organised in a structured curriculum.	Concepts are at least partly fuzzy, and their precise relationships are largely unknown.
Role of exploration	Acquisition of known knowledge.	Discovery and building of unknown knowledge.
Type of questions addressed through exploration	What if? How to? Check me!	What? Why? What if?
User's motivation and expectation	To get results that lead to a good grade and/or a congratulation message.	To get results that lead to a good representation of the domain to be structured and modelled.
Type of cognitive task performed by using the technology	Guided exploration or experimentation (discovery guided by the teacher through technology)	*Structured exploration*: Exploration for discovering the domain, Experimentation for assessing the current model, Exploration for refining the current model.
Final role of computer technology	Helping the student to go through a given (piece of) curriculum (incidental role).	Enabling the researcher to structure and model the domain (vital role).

The similarities and differences between student learning and researcher learning are more developed in table 1, although in a concise form. In this table, we do not mention the teacher, because, depending on his/her goals and expertise, s/he can either mimic the role of a student (to test some pedagogical acts, or some problems to be worked on by students, for example, even in a known domain), or play the role of a researcher, in particular in an unstructured domain like the rules governing language exchanges. Indeed, in a domain which has never been structured before, computer technology can be used as an exploratory and discovery tool in itself. Such was the case in the 70's for the first expert systems like Mycin, Dendral or Prospector (see [Hayes-Roth & al., 1983] for a synthesis). Certainly, pragmalinguistics and in particular language variation have been extensively studied and described (Preston [1986] cites several authors), but Preston's fifty categories of language variation have never been linked together by way of priority or causality rules. The use of exploration and experimentation with a knowledge-based system has indeed allowed us [Lelouche & Huot, 1998] to start uncovering such priorities, causalities, and rules (see section 3.1.2). This discovery has been made possible thanks to a systematic *structured exploration* approach, a 3-stage process summarised in table 1.

5. CONCLUSION

In this paper, using two existing computer systems as examples, we have shown how exploration and experimentation can be used in an educational setting. In the traditional setting, the learning domain is well known, the expected user is a student, and the technology, suitably prepared by the domain specialists and the teachers, is used to guide her through her learning. In a more research-oriented setting, the learning domain is widely unstructured, the expected user may be a student in the long run but is immediately a researcher or a research-minded teacher, and technology is used to enable that user to explore the domain, discover its elements and structure, and model it as appropriately as possible.

We believe that the second type of setting is presently quite underused, and that technology in general, and exploratory and experimental learning techniques in particular, could be used more often for such endeavours as discovering and structuring a new or unexplored domain. Indeed are not learning and discovery permanently ongoing processes, if not life goals?

REFERENCES

Anderson J. R., A. T. Corbett, K. Koedinger & R. Pelletier (1995) "Cognitive tutors: Lessons learned". *The Journal of Learning Sciences*, 4,167-207.

Austin J.L. (1962) *How to do Things with Words.* Clarendon Press (Oxford, England).

Boder A. & D. Cavallo (1990) "An epistemological approach to intelligent tutoring systems". *Intelligent Tutoring Media*, Vol. 1, No. 1, p. 23.

Braun F. (1988) *Terms of Address — Problems of Pattern and Usage in Various Languages and Cultures*. Mouton de Gruyter (Berlin).

Coste D., Courtillon J., Ferenczi V., Martins-Baltar M., Papo E. & Roulet E. (1976) *Un Niveau–Seuil*. Conseil de l'Europe (Strasbourg, France).

Dion P. (1988) *Conception et implantation d'un système de tutorat pour l'enseignement de l'algorithmique.* Master's thesis. Université Laval (Québec, Canada).

Harel I. & S. Papert (1990) "Software design as a learning environment". *Interactive Learning Environments* (E. Soloway, ed.). Ablex Publ. (Norwood, N.J.).

Hayes-Roth F., Waterman D. A. & Lenat D. B., eds. (1983) *Building Expert Systems.* Addison-Wesley (Reading, Mass.).

Koffman E. & P. Wolfgang (2005) *Objects, Abstraction, Data Structures and Design using Java*. J. Wiley & sons (Hoboken, N.J., USA).

Lelouche R. (1999) "Can a student-controlled environment and the model-tracing methodology go together?". *Advanced Research in Computers and Communications in Education, Vol. 1* (G. Cummings, T. Okamoto, L. Gomez, eds.). IOS Press (Amsterdam, Netherlands), p.938-945.

Lelouche R. (1994) "Dealing with pragmatic and implicit information in an ICALL system: the PILÉFACE example". *Journal of Artificial Intelligence and Education*, Vol. 5, No. 4, p. 501-532.

Lelouche R. & D. Huot (1998) "Influence of communicative situation variables on linguistic form". *CALL (Computer-Assisted Language Learning) Journal*, special issue on CALL of French, vol. 11 N°5, p. 523-541.

Malouin J. (1993) *Du système expert au tuteur intelligent: application d'un modèle d'apprentissage au développement d'un système d'enseignement de l'algorithmique.* Ph.D. dissertatiion,. Université Laval (Québec, Canada).

Papert S. (1980) *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books (New York).

Pattis R. (1981) *Karel the Robot. a Gentle Introduction to the Art of Programming*. J. Wiley (New York, U.S.A.).

Preston D. R. (1986) "Fifty some-odd categories of language variation". *International Journal of the Sociology of Language*, Vol. 57, p. 9-47.

Searle J. (1979). *Expression and Meaning: Studies in the Theory of Speech Acts.* Cambridge University Press (Oxford, England).

Searle J. (1969) *Speech Acts.* Cambridge University Press (Oxford, England).

Wenger É. (1987) *Artificial Intelligence and Tutoring Systems*. Morgan Kaufmann (Los Altos, CA).