Use of Abstraction and Complexity Levels in Intelligent Educational Systems Design

Ruddy Lelouche
Département d'informatique, Université Laval
Québec G1K 7P4 Canada
Tel.: (418) 656-2131, ext. 2597 Fax: (418) 656-2324
e-mail: lelouche@ift.ulaval.ca

Jean-François Morin
Département d'informatique, Université Laval
Québec G1K 7P4 Canada
Tel.: (418) 843-5193 Fax: (418) 843-9245
e-mail: morin@ift.ulaval.ca

Abstract

We are interested in problem-solving domains, like engineering and most "exact science" disciplines. In these domains, the knowledge to be acquired by the student is twofold: the domain knowledge itself, but also and mainly the knowledge necessary to solve problems in that domain. As a result, an education-oriented system in such a domain must encompass three knowledge types: the domain knowledge and the problem-solving knowledge, constituting the knowledge to be acquired and mastered by the student, and the tutoring knowledge, used by the system to facilitate the student's learning process. In this paper, we show how these three knowledge types can be modelled, how they should interact with one another in order to fulfil the system educational purpose, and above all how the abstraction and complexity levels can shed a uniformizing light on the system operation and make it more user-friendly. We thus hope to bring some contribution to the general and important problem of finding a generic architecture to intelligent tutoring systems.

Introduction

Besides the complexity of teaching itself, the domains taught vary considerably in complexity, type of interactions, type of evaluation, relationships between theory and practice, etc. This way, teachable domains can be classified according to the type of knowledge to be acquired by the student: "know", "know-how", and "know-how-to-be". Examples of such knowledge types are respectively: anatomy or a language grammar, the skill to solve a mathematical or medical problem, and the capability to adapt to one's environment or to deal with personal relationships. Cost engineering knowledge is of the second type.

Moreover, almost all teachable domains vary in complexity, from simple basics to relatively complex problems to solve. Thus, a student should learn and master the basics of such a domain before he is taught wider notions. And when a human tutor detects errors or misunderstandings, he usually draws the student's attention on a small subset of the involved knowledge, so that he may correct his errors and/or misunderstandings. Such an interaction is focused on either a given set of the domain knowledge or the scope of knowledge involved by a given problem.

Problem-solving or know-how domains are the ones in which we are interested here. In such a domain, hereafter called a KH-domain, the knowledge to be acquired by the student is twofold: the domain knowledge itself, but also and mainly the knowledge necessary to solve problems in that domain. As a result, an education-oriented system in such a domain, which we here call a KH-ITS, must encompass three knowledge types: the domain knowledge and the problem-solving knowledge, constituting the knowledge to be acquired and mastered by the student, and the tutoring knowledge, used by the system to facilitate the student's learning process.

This paper has a double purpose:

- we present each of the three types of knowledge involved in a KH-ITS;
- for each type of knowledge, we show how abstraction and complexity levels appear and how it is possible to deal with them.

In order to do so, we present in section 1 our domain knowledge modelling and how we exemplify it in the case of cost engineering. Next, in section 2, we focus on the advantage of separating the problem-solving knowledge from the domain knowledge in a KH-ITS, and we present some problem-solving activities in cost engineering as an example. In section 3, we briefly describe some principles of tutoring knowledge modelling in a KH-ITS. In each of these three sections, we show how to use abstraction and complexity levels, exemplifying the principles in the cost engineering domain. Finally, section 4 presents the educational interests of using abstraction and complexity levels when modelling the three types of knowledge involved in a KH-ITS.

1 Domain knowledge

In order to describe the domain knowledge, we first present its characteristics in a general KH-ITS (§ 1.1). We then show how we model it in the cost engineering domain (§ 1.2), and how such an approach lets us introduce the notions of abstraction and complexity levels (§ 1.3).

1.1 General

The first type of knowledge involved in every ITS, the *domain knowledge* (DK), contains all theoretical and factual aspects of the knowledge to be taught to the student. Although its specific structure can be varied, it typically may include concepts, entities, and relations about the domain [Brodie & al., 1984], object classes and instances [Kim & Lochovsky, 1989], possible use restrictions, facts, rules, [Kowalski, 1979; Clocksin & Mellish, 1981], semantic or associative networks [Findler, 1979; Sowa, 1984], etc.

The main system activities centred on this knowledge type are:

- theoretical explanations about the various knowledge elements and their relationships in the teaching domain;
- providing the other modules of the ITS, i.e. problem-solving and tutoring, with the necessary background of domain knowledge that they need.

1.2 Application to cost engineering

In the particular domain of cost engineering, we chose to represent this type of knowledge with concepts, relations, and a special case of relations modelled as concepts, the factors. The model is presented with more details in [Lelouche & Morin, 1996].

Concepts can be basic entities like:

- *Investment:* It is an amount on which interest is computed; depending on the context or the type of problem, it may also be called capital, loan, or present value.
- *Interest:* It is the amount exceeding the initial investment, received by the lender from the borrower at the moment of its repayment.
- *Investment duration:* It is the total time during which the interest is computed over the investment; this duration is uninterrupted.
- *Future value:* It is the total amount to be repaid at the term of an investment; it is equal to the total accumulated interest added to the initial investment amount or present value.

With the notion of compounding, some other concepts can be defined:

- *Compounding:* If the investment duration is long enough, a partial interest may be periodically added to the principal; that amount itself then produces interest for the remaining of the investment duration; this principle is called interest compounding; when the interest is compounded before the term of the investment, the principal then augments accordingly, and becomes greater than the initial investment.
- *Compounding period:* It is the time elapsed between two consecutive interest compoundings; unless otherwise specified, the interest is compounded yearly.
- *Number of periods:* It is the number of interest compoundings during the investment duration.
- *Interest rate:* It is the amount of interest payable for one period depending on the principal:

$$\text{Interest rate} = \frac{\text{Interest amount for one period}}{\text{Capital amount}}$$

- *Annuity:* amount paid or received periodically (annually unless otherwise specified); although the periodic amounts may increase or decrease over time, we refer only to uniform (constant) annuities in this paper.

Concepts are linked to one another by various types of *relations*. The relations between concepts can be either usual knowledge-representation relations, like *subclass of*, *element of*, *sort of*, etc., or numerical relations represented by formulæ. Such a formula is given above. Another one is:

$$F = P \times (1 + i)^n \tag{1}$$

which, given the present value P of an investment over n periods at rate i, computes the corresponding future value F of that investment.

A formula such as (1) can be rewritten as:

$$F = P \times \Phi_{PF,i,n} \text{ where } \Phi_{PF,i,n} = (1 + i)^n \tag{2}$$

$$P = F \times \Phi_{FP,i,n} \text{ where } \Phi_{FP,i,n} = (1 + i)^{-n} \tag{3}$$

thus introducing the *factors* $\Phi_{PF,i,n}$ and $\Phi_{FP,i,n}$. Factors allow us to separate their definition (rightmost equalities above, a quantitative aspect) from their possible uses in the application domain (leftmost equalities, a qualitative aspect).

Similarly, the factor $\Phi_{AP,i,n}$ converts a series of identical annual amounts A into an unique present value P:

$$P = A \times \Phi_{AP,i,n} \quad \text{where} \quad \Phi_{AP,\,i,\,n} = \frac{(1 + i)^n - 1}{i(1 + i)^n} \tag{4}$$

Actually, $\Phi_{AP,i,n}$ is a sum of Φ_{FP} factors. The factor $\Phi_{PA,i,n}$ does the reverse process:

$$A = P \times \Phi_{PA,i,n} \quad \text{where} \quad \Phi_{PA,\,i,\,n} = \frac{i(1 + i)^n}{(1 + i)^n - 1} \tag{5}$$

There exist other factors converting gradient and geometrical series of amounts into a present or future value; such factors are also computed as a sum of $\Phi_{FP,i,n}$ factors.

Although the formulæ related to these factors essentially involve quantitative aspects, the similarities and differences between them, and the circumstances regulating the use of either one, are of a deeply qualitative ground. If the *value* of a factor is indeed calculated from two or three numerical parameters, the *context* in which they are defined depends on whether we have to timewise move an unique amount or a series of amounts, identical or not, or conversely to compute an equivalent annual amount, etc. In fact, this context corresponds to the type of conditions that govern the investment, or *investment conditions type*, without respect to the amounts and durations involved, and is thus essentially qualitative.

This characteristic allows us to qualitatively describe relationships between factors which are basically quantitative: the *transitivity* ($\Phi_{AF,i,n} = \Phi_{AP,i,n} \times \Phi_{PF,i,n}$) and *inversion* ($\Phi_{FP,i,n} \times \Phi_{PF,i,n} = 1$) relations. Moreover, this qualitative description (including the notation) leads to a significant improvement of the pedagogical approach used to teach this subject to university students.

1.3 Abstraction and complexity levels

As it turns out, every factor introduces an *intermediate abstraction level* between the concepts implied the equation defining it. For example, in the case of formula (1), or equivalently formulæ (2) and (3), we have (see figure 1):

- at the bottom of the hierarchy, the interest rate and the number of periods, basic concepts "making technicalities explicit";
- above them, concepts more fundamentally related to the cost engineering problem being solved, namely the present and future values of the investment at hand;
- between these two levels, the intermediate one brought by the introduction of the factor Φ_{FP} or Φ_{PF}.

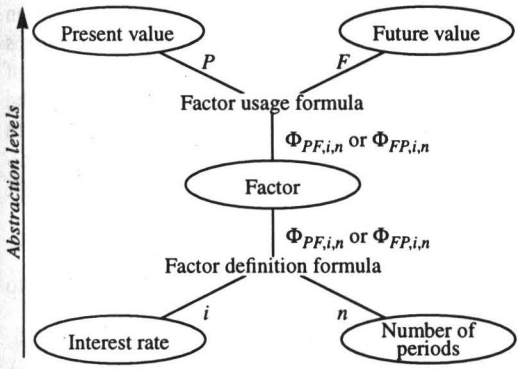

Figure 1 — Representation of a factor as a concept.

That intermediate status of the factor, originally just an intermediate variable in calculations [see formulæ (2) and (3)], makes it appear as a pedagogically oriented concept, which clearly separates

- the computational, quantitative aspect of the factor definition, i.e. the interest rate and the number of periods
- from the practical, qualitative aspect of the factor usage in a domain problem, i.e. the present and future values.

This follows the theory [Lenat & al., 1979; Malec, 1989] according to which the use of multiple *abstraction levels* eases the modelling process and simplifies inferences which may be made on the domain concepts.

Most interestingly, our scaffolding approach can be made more general: we may present and use higher-level factors built upon these first ones. Indeed, "above" Φ_{FP} and Φ_{PF}, we can define factors to express the present and future values of a series of identical amounts (and vice versa), which are a first way to generalize this concept hierarchy. For example, the Φ_{AP} factor is indeed a sum of Φ_{FP} factors:

$$\Phi_{AP,\,i,\,n} = \sum_{k=1}^{n} \Phi_{FP,\,i,\,k} = \sum_{k=1}^{n} (1+i)^{-k} = \frac{(1+i)^n - 1}{i(1+i)^n}$$

where the last expression results from computing the geometrical series shown. This example constitutes a proof of (4), but also and mainly shows that the Φ_{AP} factor is at a higher level than Φ_{FP}. Note that this refers to a *complexity level* rather than an abstraction level, since it is due to the way the Φ_{AP} factor is defined and computed.

In the sequel of the cost engineering course, other abstraction or complexity levels appear when introducing the factors related to investments made of periodic amounts in arithmetic or geometric progression, and the factors related to various depreciation types.

2 Problem-solving knowledge

In order to describe the problem-solving knowledge, we now present the general characteristics regarding problem-solving knowledge modelling in a KH-ITS (§ 2.1). As we did in section 1, we then present our model in the cost-engineering domain (§ 2.2).

2.1 General

The second type of knowledge is specific to KH-domains, henceforth to KH-ITSs. We call it *problem-solving knowledge* (PSK). It contains all inferential processes used to solve a problem resulting from the instantiation of a practical situation based on the domain knowledge [Kowalski, 1979; Patel & Kinshuk, 1996]. In other words, in order to be able to solve a problem, the problem-solving knowledge needs a theoretical background, which is found in the domain knowledge. The processes stored in PSK may be represented in various ways, using either or all of logic [Kowalski, 1979], procedural networks [Brown & Burton, 1978], semantic networks with procedural attachments, (augmented) transition networks, production rules [Goldstein, 1979; Anderson & Reiser, 1985], etc.

The main system activities centred on this knowledge type are:

- providing the inferential tools for problem solving;
- providing the inferential tools for coaching a student in a problem-solving session.

The main advantage of separating the problem-solving knowledge from the domain knowledge is that it emphasizes the distinction between the domain itself and the skills used to solve a practical problem in that domain, thus simplifying the learning process. That knowledge separation into DK and PSK is common to all KH-domains; this is why we believe that KH-ITSs, which are aimed at helping the student to learn how to solve problems, should display the same knowledge separation.

Besides, we can use — we believe in a novel way — that separation between DK and PSK to define four *generic operating modes* in a KH-ITS, based on the type of knowledge involved (DK or PSK), and on who "generates" it (the system or the student).

- In *domain-presentation mode*, the student asks the system some information about a domain theoretical element, and the system reacts by transferring to the student the required information or knowledge. The knowledge involved in this category is always DK, system-generated.
- In *demonstration mode*, the student asks the system to solve a practical problem or to coach him while he solves a problem. In the first case, the problem typically comes from the student himself, whereas in the latter one the problem typically comes from the tutoring sys-

tem. In either case, the main level of knowledge involved is PSK, system-generated.

- In *domain-assessment mode*, the system prompts the student to develop a domain element, and the student thus expresses his understanding of that element. If judged necessary, the system may then intervene to correct that understanding. The knowledge involved in this mode is essentially DK, student-generated.
- Finally, in *exercising mode*, the system prompts the student to solve a practical problem. The student then solves it step by step, showing what he understands of the involved problem-solving knowledge and of the associated domain knowledge. If necessary, the system may decide to intervene in order to help him reach his goal or to correct it. The knowledge involved in this mode is naturally PSK, student-generated.

2.2 Application to cost engineering

In the particular case of cost engineering, various problem-solving activities can be identified:

1. identify the given problem data;
2. identify the expected result;
3. draw a temporal diagram to represent the relevant events;
4. apply a formula;
5. compare amounts located at the same date;
6. compare amounts located at different dates;
7. add amounts situated at the same date;
8. add amounts situated at different dates;
9. choose a reference date;
10. move an amount from one date to another;
11. collapse a series of periodic amounts into one single amount;
12. explode an amount into a series of periodic amounts;
13. etc.

As it turns out, many problem-solving tasks may be divided into smaller ones, letting us introduce the notion of *complexity levels* in these tasks. For example, the comparison of two amounts situated at different dates implies:

- first, choosing a reference date at which to make the comparison;
- then, moving either (or both) amount(s) from its present date to the reference date;
- finally, comparing the amounts now both located at the same reference date.

These subactivities (of types 9, 10, and 5 respectively in the sample list above) thus appear to be of a lower complexity level than the initial one (of type 6). However, it is interesting to note that, although activity 6 turns out to be more complex than activity 5 (the latter is part of the former), both are stated using the same *abstraction level.*

It may also happen that some lower-level activities can *only* appear as components of a higher-level one. For example, the activity "drawing a temporal diagram" (type 3 above) implies the following tasks, which can only be accomplished as part of that activity (hence their identification from 3a to 3d):

3a. draw a timeline to encompass all periods implied by the problem data;

3b. draw arrows representing the amounts involved in the problem data;

3c. if necessary, split an amount (or each amount in a series) to simplify the computations;

3d. qualitatively draw as a special arrow the expected result of the computations to be made.

3 Tutoring knowledge

We now briefly present the *tutoring knowledge* (TK) in order to help the reader to better apprehend the relationships of that knowledge with DK and PSK. This third type of knowledge contains all tutoring processes enclosed in the ITS. It is not directly related to the teaching domain or to problem solving, but helps the student to understand, assimilate, and master the knowledge included in DK and PSK [Gagné & Trudel, 1996].

The main system activities using TK are:

- ordering and formatting the topics to be presented to the student;
- monitoring a tutoring session, i.e., triggering the various tutoring processes according to the system tutoring goal and the student's actions; such monitoring may imply giving explanations, asking questions, changing to another type of interaction, etc.;
- in a KH-domain, while the student is solving an exercise, monitoring the student's problem-solving activities: understanding and assessing these activities, giving advice to correct or optimize them, giving hints or partly solving the exercise at hand (as required by the student or by the tutoring module), etc.;
- continuously analysing the student's progress in order to optimize the tutoring process.

The advantage of separating the tutoring knowledge from the knowledge of the domain to be taught has been emphasized long ago [Goldstein, 1977; Sleeman & Brown, 1982; Clancey, 1986; Wenger, 1987], and lies in the reusability of TK in various domains. In the case of KH-domains, the domain to be taught clearly encompasses both DK and PSK; indeed, the term "domain knowledge" applies to DK if referring to the knowledge type, and to DK + PSK if referring to the knowledge to be acquired. Therefore, as shown earlier, in a KH-ITS, knowledge ends up being separated into three categories rather than two.

We believe that the tutoring processes are triggered by tutoring goals which depend on the current educational setting and learning context. In the current state of our research, our assumption is therefore the following: the underlying hierarchy or hierarchies governing the way they interact with one another is not related to these processes *per se*, but rather to the goal to be attained when they are invoked. If that assumption turns out to hold, then the *dynamic structure of educational goals and subgoals* — which itself depends on the student's desires or abilities, the main underlying objective of

the tutoring system, the student's state (e.g. of tiredness, etc.) and performance, etc. — will determine the succession of tutoring processes activated and tutoring interactions taking place. To our knowledge, the use of abstraction levels to induce a dynamic hierarchy of tutoring goals is new, as is the assumption that such a hierarchy will play a major role in activating the various tutoring processes and student–system interactions.

4 Educational interests of the abstraction and complexity levels

In the above sections, we have described a complexity- and abstraction-level approach to help model the three types of knowledge involved in a KH-ITS. In this section, we present the educational interests of our model. Sections 4.1 to 4.3 focus on the type of knowledge respectively presented in sections 1 to 3. Section 4.4 summarizes that discussion with some overall pedagogical interests of our approach.

4.1 Domain modelling

The factor hierarchy described in section 1 lets us derive an *order for the presentation of factors* to the student, from the lowest (simplest) level up to the highest, i.e. with increasing understanding complexity. That does not imply that such an order is unique, or even the best (e.g. a student's personal interests might make another order more motivating for him), but it is justified by our model. This presentation order may itself induce a possible *order for prerequisites*; e.g., if a student experiments difficulties to deal with Φ_{AP}, has he well mastered Φ_{FP}, a conceptually simpler factor?

Moreover, the intermediate abstraction levels will permit our ITS to exhibit a *sharper error modelling*. For example, the source of an understanding error concerning either relation in equation (2) or (3) is much easier to identify using the corresponding factor, either as a definition error or as a usage error, than an error concerning the global equation (1), where the definition and application relationships are not made explicit, and are therefore impossible to distinguish. Similarly, an error using a Φ_{AP} factor may be diagnosed as resulting from an insufficient mastery of the simpler factor Φ_{FP} as concept (which in turn will be diagnosed as related either to the definition, or to the usage of that concept).

Finally, in a first development stage, this modularity will ease our *defining the exercise types* to be implemented into the ITS, and will ease the tutor module task of *choosing the exercise type* to challenge the student with. Later, once the basic system is operational, that modularity will help us develop an automatic *exercise generator* dealing with the domain elements to be mastered by the student. That approach will then help the student to acquire a better critical mind about the relative importance of *know-how* knowledge vs. *domain* knowledge.

4.2 Problem-solving modelling

The problem-solving activities briefly presented in section 2 naturally display abstraction and complexity levels. Indeed, a standard cost engineering problem can be divided, possibly in more than one way, into major steps, which can then be split into simpler substeps. Such "relatively complex" cost engineering activities shown in section 2.2 were activity 3 (draw a temporal diagram) and activity 6 (compare amounts situated at different times). Using activities 5 and 6 as examples, we also made a distinction between the complexity level, based on the execution process, and the abstraction level, based on expressiveness or scope.

Such abstraction and complexity levels will let us introduce multiple levels of explanations, which will ease their tailoring, vary the answers to the student's questions, and adapt to the theoretical and practical reminders needed by the student.

Moreover, such an approach will lead the student to focus specifically on the activities for which he needs more tutoring, with the abstraction and complexity levels appropriate to his individual case.

That tutoring may take the form of explanations, guidance, hinting, even partially solving the exercise on which the student is currently working.

Table 2 — Characteristics of the four typical functioning modes of an ITS based on a problem-solving domain.

Functioning mode		Domain-presentation mode	Demonstration mode	Domain-assessment mode	Exercising mode
Main type of knowledge involved		Domain knowledge	Problem-solving knowledge	Domain knowledge	Problem-solving knowledge
Student's main goal		To learn		To assess his learning	
Direction of the knowledge transfer		System → Student		Student → System	
Typical interaction	Trigger (start)	The student asks the system...		The system prompts the student...	
		some information about a domain theoretical element	to solve a practical problem or to coach him while he solves a problem	to develop a domain element	to solve a practical problem
	Knowledge exchange	The system presents...		The student presents his view of...	
		the requested element	a possible solution to the requested problem	the requested element	a possible solution to the given problem
	Result (closure)	The student expresses his understanding...		The system assesses the student's answers, and possibly corrects them	
		of the element	of the problem solution		

4.3 Tutoring modelling

As presented in section 2, the distinction between DK and PSK leads to four natural functioning modes: the domain-presentation mode, the demonstration mode, the domain-assessment mode, and the exercising mode. Their main characteristics are recalled in Table 2.

The abstraction levels of the tutoring goals aimed at by the system when invoking the tutoring processes is likely to result in a chain of recursive calls of these processes. This recursivity will or will not be direct, depending on the tutoring interaction types being chained: the system might freely decide to temporarily change between interaction types. However, the potential complexity of this chain is only apparent: because of the tutoring goal abstraction hierarchy, each newly invoked process will be called with a *narrower scope*, which naturally eliminates the risk of "forgetting" the initial tutoring goal or running into an infinite loop.

4.4 Overall interests of these abstraction and complexity levels

Abstraction levels are certainly not new. What we think is new is to use them in a systematic way to shed a uniformizing light on the system design and operation, and to make it more user-friendly once implemented.

First, they may help to better tailor the system tutorial interventions to fulfil the student's needs and the system tutoring goals, thus improving its conviviality and efficiency.

Then, all the capabilities presented above should result in smoother, more "natural", human-like interactions with the student. This improved ability to reproduce a human teacher's behaviour contributes again to make the system more user-friendly.

Finally, although that aspect is not in the scope of this paper, our refining of the three types of knowledge as described in sections 1 to 3 paves the way for the implementation of a structured error model, and eventually a student model.

Conclusion

This presentation of a possible knowledge structure for KH-domains, which emphasizes the separation between domain knowledge and problem-solving knowledge, shows how a general functioning theory of such an ITS — namely the four functioning modes described in section 4 — can naturally be derived.

Moreover, the abstraction and complexity levels highlighted throughout this paper can be used as a common paradigm to help finding an appropriate representation for each knowledge type, and thus help creating more efficient ITSs. More generally, this paradigm can shed a uniformizing light on the system design, although it has never been used in a systematic way in the design or implementation of an ITS.

We thus hope to bring some contribution to the general and important problem of finding a generic architecture for intelligent tutoring systems.

Bibliographic references

Anderson John R. (1986) *Cognitive Modelling and Intelligent Tutoring*. National Science Foundation (Washington, DC).

Anderson, John R. & B. J. Reiser (1985) "The LISP Tutor". *Byte* **10**, no. 4, p. 159–175.

Brodie, Michael L., John Mylopoulos & Joachim W. Schmidt, eds. (1984) *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*. Springer Verlag (New York).

Brown, John S. & Richard R. Burton (1978) "Diagnostic models for procedural bugs in mathematical skills". *Cognitive Science* **2**, p. 155–192.

Clancey, William J. (1986) "Qualitative student models". *Annual Review of Computer Science* **1**, p. 381–450.

Clocksin, William F. & Christopher S. Mellish (1981) *Programming in Prolog*. Springer-Verlag (Berlin).

Díaz de Ilarraza Sánchez, Arantza & Isabel Fernández de Castro, eds. (1996) *Computer-Aided Learning and Instruction in Science and Engineering*, Proceedings of the *Third International Conference, CALISCE'96*, San Sebastian (España), 29–31 July 1996. LNCS 1108, Springer (Berlin).

Findler, Nicholas V., ed. (1979) *Associative Networks*. Academic Press (Orlando, FL).

Frasson, Claude, Gilles Gauthier & Alan Lesgold, eds. (1996) *Intelligent Tutoring Systems*, Proceedings of the *Third International Conference, ITS'96*, Montréal (Canada), 12-14 June 1996. LNCS 1086, Springer (Berlin).

Gagné, Denis & André Trudel (1996) "A highly flexible student-driven architecture for computer-based instruction". [Frasson & al., 1996], p. 66–74.

Goldstein, Ira P. (1977) *The Computer as Coach: an Athletic Paradigm for Intellectual Education*. AI Memo 389, AI Laboratory, Massachusetts Institute of Technology (Boston, MA).

Goldstein, Ira P. (1979) "The genetic epistemology of rule systems". *International Journal of Man–Machine Studies* **11**, no. 1, p. 51–77.

Kim, Won & Frederick H. Lochovsky, eds. (1989) *Object-Oriented Concepts, Databases, and Applications*. ACM Press, Addison-Wesley Publ. (Reading, MA).

Kowalski, R. (1979) *Logic for Problem Solving*. North-Holland (Berlin).

Lelouche, Ruddy & Jean-François Morin (1996) "The formula: a relation? Yes, but a concept too!". [Díaz de Ilarraza Sánchez & Fernández de Castro, 1996], p. 176–185.

Lenat, D., F. Hayes-Roth & P. Klahr (1979): *Cognitive Economy*. Working Paper HPP-79-15, Stanford Heuristic Programming Project. Stanford University (CA), June, 46 pages.

Malec, J. (1989) "Knowledge elicitation during dynamic scene description". *ACM–SIGART Newsletter: special issue on knowledge acquisition*, no. 108, April, p. 162–163.

Patel, Ashok & Kinshuk (1996) "Applied artificial intelligence for teaching numeric topics in engineering disciplines". [Díaz de Ilarraza Sánchez & Fernández de Castro, 1996], p. 132–140.

Sleeman, D. H. & John Seely Brown, eds. (1982) *Intelligent Tutoring Systems*. Academic Press (London).

Sowa, John F. (1984) *Conceptual Structures*. Addison-Wesley (Reading, MA).

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