76

A systemic approach to courseware engineering

Frédéric Blanc

API Laboratory—IUT Paul Sabatier
Toulouse
France

ABSTRACT

The development of an Intelligent Tutoring System is a complex process which requires multiple knowledge sources. The aim of Courseware Engineering is to reduce this complexity through flexible and domain independent environments designed to ensure client satisfaction using fast prototyping. An approach is presented for the production of complex educational software based on the notion of open systems. Courseware/learner interactions are considered the basis for adaptation and guidance. A development method is used based on simulation for predicting the possible behaviour of the system. Techniques come from Artificial Intelligence (qualitative reasoning and causal ordering) and allow a particular didactic strategy to be evaluated. Qualitative modelling is used to describe behavioural knowledge in terms of positive or negative influences of the learner's actions to simulate the partial functioning of the tutor during the development and after adapting behaviour during use.

Main conference themes: methodologies, software
Educational areas:
Study topics:
Secondary keywords: modelling, open systems, simulation

J. D. Tinsley et al. (eds.), *World Conference on Computers in Education VI*

INTRODUCTION

Development of educational software is a complex process which requires cooperation between users with various abilities: domain experts, pedagogical people, media specialists and software engineers.

Recent work has shown that, except in simple cases, special educational languages and author systems are not appropriate for developing complex software such as learning environments or Intelligent Tutoring Systems [1, 2]. Change of didactic strategies and adaptation of the learning process to the learner should be integrated. Courseware engineering should reduce the complexity by providing automated modelling of the learning process. We suggest a development approach based upon the notion of a complex open learner driven structure. Techniques are presented which are based on qualitative models for describing system behaviour in terms of causes and effects.

In the first part of this paper we define the notion Courseware System which is the basis of our approach. Then we present the systemic method based on simulation of qualitative models. Finally we present the software environment as it has been developed so far.

THE COURSEWARE SYSTEM

Objectives

An intelligent tutoring system allows changes of didactic strategy based on learner profile and activities. The dynamics require guidance to be adapted during the session and adequate flexibility [3].

Current approaches try build such software by identifying distinct subproblems [4]. These are *analytic methods* which require a precise typology of knowledge and which result in a closed system; these methods are:

- domain dependent specific for the particular type of tutor;
- centred on objects and data models, not on learner's activities.

As far as we can see these are not suitable for the production of flexible educational software. We suggest a *systemic method* for courseware engineering which allows collecting and organizing knowledge in a global, domain independent approach which ensures flexibility and guidance [5]. In this method:

- authors reason about tasks and activities, not only about system components;

- behavioural knowledge, more than structural knowledge is taken into account.

Theoretical aspects

We define a *courseware system* as a system of dynamically interacting didactic elements organized towards a goal, but modified and changed by interactions with the learner. Its dynamic behaviour is based on interactions and feedback loops [6].

The *interactions* between software and learner allow system evolution. The educational software evolves in interaction with the learner's activities and suggests in its turn tasks to make the learner progress (Fig. 1). It is able to perceive changes in the student's profile.

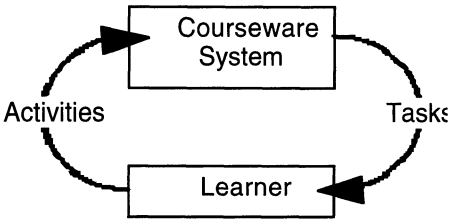

Fig. 1 Courseware system/learner interactions

To behave in a satisfactory way the system has to determine the didactic goals and maintain constraints by setting bounds. This implies **feedback** loops which stabilize the system around the didactic goals (Fig. 2).

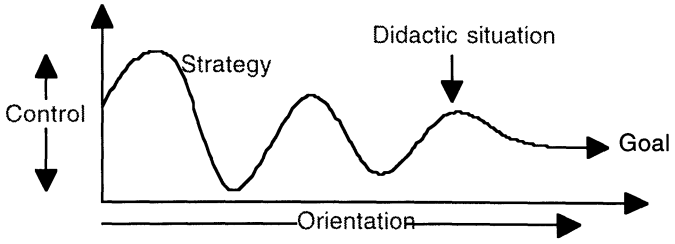

Fig. 2 System stability by feedback loops

Learner activities during the session, as well as the current state of the system influence learner guidance and system regulation. This influence is based on qualitative didactic expertise which judges the quality of the learning process (Fig. 3).

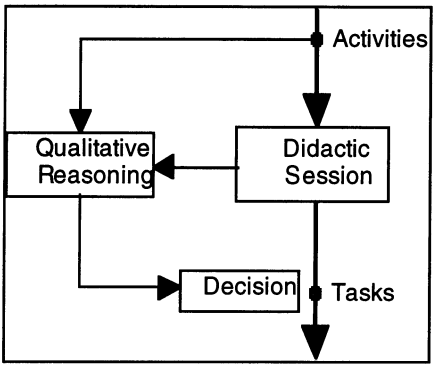

Fig. 3 Courseware system functioning

Courseware System development

In general in a courseware system there are three obvious constraints:

- decision rules which guide the learner and define system flexibility;
- qualitative expertise in support of decision making;
- analysis of learner tasks and activities which forms the basis of the qualitative reasoning.

To help finding decision rules, the adopted development method is aimed at building a first model of activities, tasks and qualitative expertise, and then executing this model to study its behaviour. The reality known by the pedagogue is compared with the results of this simulation and this serves as the basis for a modification of the starting model and for describing guidance rules (Fig. 4).

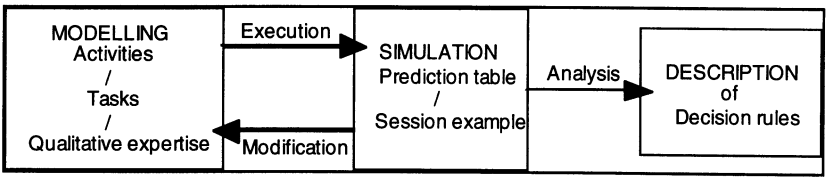

Fig. 4 Courseware system development process

SYSTEM MODELLING

In educational software production in general three types of complementary knowledge [1] play a role:

- mastery of the subject (pedagogical model);
- strategies to be used (didactic model);
- different tactics with respect to strategies (mediatic model).

The aim is to provide a framework into which authors can organize the didactic goals, the domain knowledge and in which the tasks, activities and factors can be identified which contribute to the success or failure of a particular teaching sequence. The analysis involves various experts: domain experts, pedagogues and software engineers.

System Analysis

Our method of analysis is based on a formal approach supported by logical notions. This mathematical basis allows precise definition of notions like completeness, consistency or correctness [7]. The method is supported by a formal specification language, Spec, in which first order logic expressions can be directly expressed [8]. Spec specifications are organized in units called modules which contain definitions for a set of concepts which are expressed in logical statements built from functions and relationships by using & (and), | (or), ~ (not), => (implies), <=> (if and only if) and the quantifiers all (for all) and some (there exists).

System analysis consists of four stages:

1. *Domain analysis*: specification of concepts defining domain knowledge. Aim of the analysis is to describe, to structure and to give typical values to objects as an environment for the simulation.

2. *Analysis of didactic goals*: definition of system goals to be reached by the learner integrated into learner orientation rules. The difference between the didactic goals and objects from the domain acquired during a session is used to assess learner progress.

3. *Task analysis*: tasks are defined to allow the learner to reach the didactic goals. A task is defined by a goal and constraints described in Spec. The pedagogue defines the general learning scenario which does not take the user activities into account.

The following expression shows an example of a task concerning questions about a concept:

concept setting_questions (c: CAI_concept) —task example
value (l: set {question})
where all (q: question **such that** q in l :: q.CAI_concept = c)

4. *Activity analysis*: the interaction between a subject (the learner) and his environment (the courseware) is analyzed by diagnosing learner actions: change in the factors which play a role in the learner's orientation, as shown through his activities (type of error, number of unfruitful tries...). These rules diagnose errors for which help is required. The following example is a very simple diagnosis of an error in questions associated with a concept (error_type is the concept associated with the exercise with the wrong answer).

concept question_error (c: CAI_concept) —activity/task diagnosis
value (t: error_type)
where some (q: question, a: answer **such that**
t = q.associated_concept ::
q = setting_exercises (c) **&** wrong_answer (q,a))

Qualitative modelling

Qualitative modelling attempts to formalize the reasoning of an engineer about a system. It is not a precise system description, but models the perception of the author (mental models) on the basis of qualitative data rather than precise numerical data [9]. It allows one to describe didactic situations in terms of positive or negative impacts of different factors on the quality of a given didactic strategy. From these descriptions the pedagogue can simulate activities to study and understand the behaviour of the tutor. This simulation allows one to take into account several successive situations to determine pro/con arguments for/against doing a particular action, then describing strategies or modifying the qualitative model.

There are three different formalisms: the first based on constraints definition [10], the second on the notion of component as described by qualitative differential equations [11], and the third based on the notion of process as an entity which regroups objects and their relations, and its activating conditions [12]. Thus the model consists of a set of causal variables and causal relationships representing the factors and links which express the qualitative expertise.

Causal variables
There are four types of causal variables an author must consider when developing his expertise:

- a *factor* used for capturing the learner's behaviour during the teaching session (endogenous variable): type of answer, number of errors, type of error...;
- *environmental variables* affecting the learner orientation (in principle the student profile);
- types of *interaction* defining the nature of interactions with the learner, in accordance with his profile and the way in which the session develops: exercise, help, course, comment...;
- *operation* (type of the associated problem: repetition, application, conceptualization ...), *importance* and *difficulty* defining characteristics of tasks.

Our objective is to use simple, but explicit causal variables. The aim is not to precisely define didactic situations, but to reason about descriptions which are qualitatively sufficient. Reasoning is done on a mental model of the situation, not on a strict, precise model.

Causal relationships
Causal relationships link variables to describe the behaviour of the system during the action of a user or a change of parameter. A causal relationship describes each influence in terms of its positive or negative impact during the learning process. Its recursive definition acts as a propagation rule of the learner actions through the whole system. It can be expressed by axioms where 'Inf+' describes a direct influence and 'Cause' represents an influence the impact of which is unknown (positive or negative).

```
concept Positive_influence (x: type, y: type)
value (b: boolean)
where b <=> some (z: type :: Inf+ (x,y)  |
                             (Inf+ (x,z) & Inf+ (z,y)) |
                             (Inf+ (x,z) & Positive_influence (z,y)) |
                             (Inf+ (x,z) & Cause (z,y)))
```

State of the system
Authors have to be able to measure the evolution of the system after the simulations are done. Three indicators are used:

- the **gap** with respect to the didactic goals;

- the **acquisition** rate of concepts;
- the **performance** in performing a task.

These indicators are barometers of the quality. These are dynamically updated during the session in accordance with positive or negative influences of the learner's actions. 'Increase' and 'Decrease' functions allow continuous modification of the state of the system by successive adjustments:

concept Positive_influence (d: diagnosis, dg: didactic_goal)
value (b: boolean)
where b <=> **some** (g: Gap :: Gap.current (dg, g) **&** Decrease (g,dg))

Temporal operators are associated with these indicators:

- **create** indicates the begin state;
- **current** indicates the current state;
- **goal** indicates the state to be reached.

For instance, 'Acquisition.create (CAI_concept, novice) fixes the acquisition rate for a concept at the beginning of the session and 'Acquisition.goal (CAI_concept, expert)' defines the goal for the acquisition of that concept.

COURSEWARE SYSTEM SIMULATION

The previous section has described the qualitative representation of a courseware system. The current paragraph is concerned with the simulation process, that is, the dynamic system behaviour resulting from the previous descriptions.

Simulation generally produces the set of possible system behaviour from an initial starting point. With the initial conditions fixed the pedagogue tries to predict the future system behaviour by modifying groups of variables (parameters). He is able to simulate a teaching session and can observe the system evolution. The process is of course iterative (Fig. 5).

Fixing the initial conditions
The starting values of variables (actual gap relative to each didactic goal) and indicators (acquisition rate required for each concept, level to be reached by the learner...) are fixed.

Predicting the system behaviour from the initial conditions
Prediction tables allow one to see the influences of a factor on the system starting from a given initial situation. An influence is propagated through the network of relationships and causal variables: its action can be followed through

the whole system. This mechanism is based on a test and case generation technique. This interrogates the model as a network of variables and relations placed in a given context (initial conditions) without simulating the actions in the model.

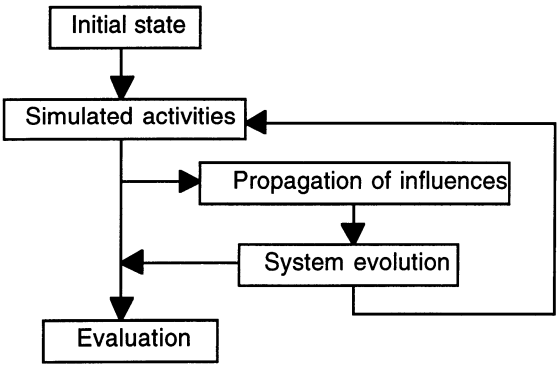

Fig. 5 The simulation process

Simulating the learner's activities
From fixed initial conditions the pedagogue is able to simulate a teaching session: setting an exercise, choosing an answer, setting the next exercise ... Then he successively plays the role of teacher and of student in a simulated situation.

Observing the system evolution
In addition to simulating a didactic session the environment supports model interrogation mechanisms. This allows one to know the current state of the system after the actions of the learner (their positive or negative influences on the system).

Simulation produces sequences of actions/observation. The pedagogue is thus able to predict the future system behaviour in a given situation and write adapted contextual guidance rules.

DESCRIPTION OF DECISION RULES

Courseware system dynamics consists of the combined action of feedback loops and activities. Feedback gives information about the state of the system integrated in the diagnosis of learner actions. The decision process therefore

involves activities, state indicators and environmental variables (student's profile).

Initially, the system is presumed to be at equilibrium. This equilibrium is disturbed by a change due to an action. The aim is to restore equilibrium by fixing tasks and stating control mechanisms (Fig. 2): after the learner activity the "disturbed" system has to react prescribing a task guaranteeing a good future behaviour.

The objective is to maintain coherence:

- fixing the learner's degree of liberty according to the current situation (*regulation* rules),
- guiding the learner to his didactic goals (*orientation* rules),
- controlling the learner towards the goals according to his degree of liberty (*control* rules),
- updating the student's profile (*revision* rules).

These decision rules are based on the implemented qualitative expertise which allow analysis of the current situation in the session:

Control rules act to keep the learner within the framework fixed by the pedagogue. These are based on the measurement of the current state (gap, acquisition), compared with the starting state which defines the learner's degree of liberty.

If Gap.current (didactic_goal, g1)		—current gap
	and Gap.create (didactic_goal, g2)	—starting gap
	and g1 > g2	—non authorized gap
Then	**If** Positive_influence (q, didactic_goal)	—type of q: question
	Then Set (q)	—setting right questions

Orientation rules are classical guidance rules: task prescription according to a diagnosis of activities.

If Acquisition.goal (CAI_concept, understanding)		—important goal
	and Difficulty.current (task1, hard)	—difficult task
	and non Error_type (CAI_concept)	—no error
Then	task2	—next task

Regulation rules allow dynamic modification of the learner's framework (changing his goals for example), using increases and decreases of the values of indicators.

If Level.goal (toto, deepening) —level to reach
and Difficulty.current (task, easy) —easy task
and Answer_type (question, CAI_concept, wrong)
—but errors
Then Acquisition.goal (CAI_concept, a)
Increase (a, CAI_concept)
—increase the acquisition rate required
Gap.goal (didactic_goal, g)
Decrease (g, didactic_goal) —the goal become less easy to reach

Revision rules are used to update the student's profile. In the following example the learner progresses because he did not make errors on questions.

If Level.current (toto, novice)
and non Error (CAI_concept1) —no error
and Answer_time (calculation, CAI_concept2) < 10
Then Level.create (toto, medium)

CONCLUSION

In this paper, we have presented an approach of complex educational software development based on the notion of open system.

This approach tries to really integrate the learner (activities and profile) at the beginning of the courseware design. It is a global, domain independent approach based on tasks and activity descriptions to ensure flexibility. Its primary contribution is a simple, clear system for qualitative modelling of the effects of actions on the system in a specific didactic situation (behavioural knowledge).

The artificial intelligence techniques and tools used, based on a formal specification language, the qualitative modelling and the simulation are being tested on an example problem concerning the functioning of aeroplane engines. We are currently experimenting with a version of the system which was designed to test our approach on this example. Implemented in PrologII+ it provides a Spec editor, tools to transform Spec code into Prolog (to ensure executability), an evaluation tool (completeness, consistency) and a simple simulation tool (Fig. 6). Possible future enhancements include mechanisms for representing time explicitly and integrating refined qualitative expertise.

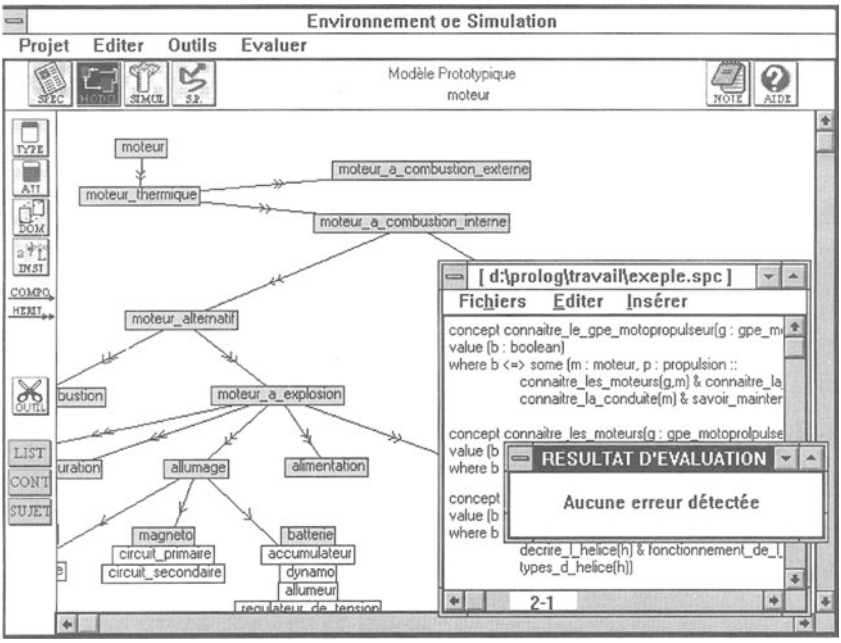

Fig. 6 The environment constructed

REFERENCES

1. Bessagnet, M.N., Nodenot, T. and Gouarderes, G. (1990) *A new paradigm: Software Engineering.* 5th IFIP World Conference on Computers in Education, WCCE'90, Sydney.

2. Major, N. and Reichgel, H. (1993) *COCA: a Shell for Intelligent Tutoring Systems.* Intelligent Tutoring Systems Conference, ITS'92, LNCS 608, Springer-Verlag.

3. Wenger, E. (1987) *Artificial Intelligence and Tutoring Systems.* Morgan Kaufmann Pub.

4. Schoenmaker, J., Nienhuis, J., Scholten, E. and Titulaer, J. (1990) *A methodology for Educational Software Engineering.* 5th IFIP World Conference on Computers in Education, WCCE'90, Sydney.

5. Woolf, B. (1991) *Representing, acquiring and reasoning about tutoring knowledge.* Intelligent Tutoring Systems—Evolutions in design. (ed Burns), Parlett, Redfield.

6. Andreewsky, E. (1991) *Systémique et cognition.* Dunod AFCET Systèmes.

7. Wing, J. (1990) *A specifier's introduction to formal methods.* IEEE Computer.

8. Berzins, V. Luqi (1990) *Software Engineering with Abstractions.* Addison-Wesley Publishing Company.

9. Clancey, W.J. (1989) *Viewing knowledge bases as Qualitative Models.* IEEE Expert.

10. Kuipers, B.J. (1993) *Reasoning with qualitative models.* Artificial Intelligence **59**, pp. 125-132.

11. De Kleer, J. and Brown, J.S (1986) *Theories of Causal Ordering.* Artificial Intelligence **29** , pp. 33-61.

12. Forbus, K.D. (1993) *Qualitative process theory: twelve years after.* Artificial Intelligence **59** , pp. 115-123.