Reverse-Engineering of an Intelligent Tutoring System

Dunia RAMAZANI & Claude FRASSON
Université de Montréal
Département d'informatique et recherche opérationnelle
C.P. 6128, Succursale A
Montréal (Québec) Canada H3C 3J7

Abstract:
The design complexity of an intelligent tutoring system (ITS) leads to think about the possibility to re-use existing ITS. However, when we want to analyze or restructure an ITS according to a specific view, available specifications are often incomplete and not adapted to new goals. To solve this problem we have experimented a reverse-engineering technique in order to produce a specification of the architecture from the existing code.
We have applied this technique to Wordtutor, an existing ITS prototype. We have extracted from the source code a description in form of communicating state machines. This description was then translated into an executable specification which was validated. Results obtained led to efficient recommendations for a new distributed Wordtutor. The technique proved useful to accelerate the process of ITS specification

Mots-clés: Reverse Engineering, Tutoring Systems, Distributed AI
1. Introduction

The design of an intelligent tutoring system (ITS) is a complex process which remains often specific to the domain to implement [15]. A multidisciplinary approach is necessary, involving abilities in AI, psychology, pedagogy, didactic, ergonomy of human-machine interface. This characteristic and the lack of tools and methodology to support specification, design and implementation of ITS do not facilitate producing specifications of a new ITS in a reasonable time.

For many research teams [6, 11], it would be useful to re-use specifications of existing prototypes for elaborating new specifications. We have developed [6] Wordtutor, an ITS prototype in word processing which uses several hundred of rules implemented in ART [1] and an object database that contains the subject matter (word processing). Transporting this prototype in another environment would require a lot of software and human resources.

We expected to re-use Wordtutor in order to produce a distributed ITS based on a client-server type architecture. The problem, however, is that as in many similar situations in which a complex prototype has been experimentally implemented (because in such type of research, specifications are not known in advance), we do not have a complete specification of architecture and internal process that could be simulated.

This problem is not new. In 1988, McCalla noted the difficulty to transfer methods and issues obtained in ITS [11]. "programs do not specify a semantics for the theory that they are encoding... Programs do not allow the theory they are encoding to be viewed at varying grain sizes. They are uniform, fairly low-level, large, and often very cumbersome implementations of the theory. They do not illuminate any of the meta-issues about where a theory applies and does not apply, and what the theory's strengths and weaknesses are...This makes it difficult to understand what a program does." Moreover, its is very difficult to interpret the objectives of a large scale program. High level descriptions given by the developers are often inconsistent with the programs. The main reason is that it is often difficult to apply software engineering principles in research prototypes and particularly produce a relevant documentation including the different versions of the prototype.

In order to contribute towards a solution to this problem, we have elaborated a reverse-engineering technique which takes the existing code of Wordtutor as input and produces a specification of cooperation and synchronization of the different components. The resulting specification was transformed into an executable specification using a formal description [4]. The specification was finally used as a support for in depth analysis of Wordtutor architecture. It served for defining the architecture and protocol of a distributed Wordtutor.

The purpose of this paper is to illustrate the principle of reverse-engineering applied to an ITS. We briefly present the main concepts of Wordtutor then we describe the reverse-engineering technique used and the issues of the experiment. We show the main characteristics of the distributed Wordtutor and finally, we discuss the advantages and limits of the approach.
2. Wordtutor

Wordtutor is an ITS prototype in word processing developed in the Heron lab at Université de Montréal [10]. It provides capabilities to plan learning activities and observe a student interacting with a microworld through an interface, teaches knowledge specific to the domain of word processing and adapts itself to the reactions and knowledge of the student. Each knowledge element of the curriculum corresponds to either a "know", "can-do" or "can-use", category of knowledge. Wordtutor is based on a didactical cycle that is carried out in three steps: planning, monitoring and updating. The first step (planning) uses the current state of student's knowledge to find a sequence of tasks which will constitute the next activity to transmit to the monitor. During the monitoring phase, the student does a series of actions which are analyzed. A global evaluation is given to the curriculum which will update the state of acquired knowledge and transmit it to the planner for a new cycle.

The architecture includes four main components: the Curriculum, the Planner, the Monitor, the Interface, and two sub-components: the expert of the domain and the diagnostic module, used by the Monitor. Each component can be considered as an agent with its own knowledge base and expertise. Expertise of an agent is defined by a set of methods (procedures and functions). For instance, the Curriculum contains methods for updating the knowledge base and selecting the next activity to be taught to the student. In the same way, the Planner and the Interface are agents with their knowledge base and expertise. System operation is based on a set of "view-point" [10] defined for each agent that communicate via a black board [8]. Each agent is able to place requests on the black board and react to requests addressed by other agents. The components function as follows:

**Curriculum:** its goal is to give contextual information to the Planner (based on the organization of the subject matter), the objectives to perform (which are in the syllabus) and the results of the student. It suggests a set of teaching units to the Planner and responds to its requests using methods.

**Planner:** it proceeds in five steps, (1) elaboration of a pedagogical plan that will serve to ask the Curriculum about subjects to present to the student, (2) selection of learning activities that will be communicated to the Monitor, (3) selection of the most appropriate activity according to the needs of the student, (4) association of activity to a set of tasks (videos, examples and exercises) and (5) revision of the pedagogical plan according to student's actions received by the Monitor. Several hundred rules, coded in ART [1], coordinate these different planning steps.

**Monitor:** it proceeds in four steps, (1) generation of exercises on particular subjects, (2) recording of student's actions, (3) appropriate interventions and (4) producing an evaluation of student's actions. All those steps require the use of the Interface and for (1), (3) and (4) the use of expert and diagnostic modules.

Recently [14], we decided to implement a distributed version of Wordtutor intended to operate through an EtherTalk network (Ethernet-AppleTalk) linking Macintosh and Sun
workstations. Was the initial code adapted to this new goal? How was the development state of the communication protocol? In order to re-use some Wordtutor modules and also to test the validity of the first prototype we experimented a reverse-engineering method.

3. Reverse-engineering
Reverse-engineering is a process which consists in analyzing the target system in order to identify system components and their relations, then to obtain representations of the target system in another formalism or at a more abstract level [9]. The goal is not to modify or reproduce a system but rather extract the design decisions and proceed to a synthesis of abstract representations of the new target system.

The main objectives (and advantages) of this process are: handling complexity, productions of new representations, recovery of information lost during the coding phase, detection of side effects, synthesis of high level specifications and facility of reusability. Reverse-engineering (compared to classical forward engineering activity in figure below) depends on the ability to recognize, understand and manipulate the design decisions contained in the source code.

Figure 1: Forward engineering and reverse-engineering
The degree of reverse-engineering depends on the type of result we expect to obtain (for instance highlighting data flow diagrams) and the step of the development cycle we would like to reach. The internal activity of most systems can be defined using communicating finite state machine [2]. In that case, transition diagrams indicate the internal operation of the system according to events and corresponding actions. The domain of communication protocols uses widely the formalism of communicating finite state machines, due to the mathematical foundation of finite state automata and its usefulness. According to Woolf [16], a model of the internal operation of an ITS can be built using finite automata, due to a periodic change of states: pedagogical states, presentation states and evaluations states. Wordtutor was entirely defined in this way, evolving from a class of states to another class through a didactical cycle.

Figure 2 illustrates the different steps of reverse-engineering process used on Wordtutor. The first step consisted in developing an algorithm [13] to transform the source code into a specification in terms of communicating finite state machines. The second step used Estelle, a language intended to describe distributed systems [5], to obtain an executable specification. This formal description technique was selected by ISO for describing open systems standards. Estelle is supported by an environment (called Véda [5]) which allows simulation and verification of specifications (step 3). Finally, the results of the simulation are carefully examined by an expert in ITS design to give specific recommendations in terms of implementation, architecture, etc...

Figure 2 : Reverse-engineering of WordTutor
The results obtained with Wortutor brought to light:
- the flow of existing communication among the different modules,
- the transitions between the different states,
- the detection of possible malfunctions and deadlocks (for instance it would have been possible to detect if the didactical cycle could be blocked)
- a high volume of communication between the planner and the curriculum.

The following sample is a small subset of the results of the analysis realized in Véda. It indicates that 95 transitions were executed among 81 states and no problems were detected.

```
states limit 100000
states exception stop
on success trace 10
on success ignore
on error trace 10
on exception stop
on deadlock trace 10
on deadlock ignore

compressed state size = 12 bytes
Number of states : 81
Number of transitions : 95
duration : 0 mn 2 s

Number of error transitions : 0
Number of success transitions : 0
Number of deadlock states : 0
```

It is important to notice that specification particular to the distributed ITS can be added and experimented in step 3. Successive evaluations lead to a refinement of the target architecture.

**4. Distributed WordTutor**

The set of different results from the experiment and the expected functionalities of the distributed Wordtutor led to clear recommendations:

- in order to reduce some high volume of data transfer it would preferable to separate the Planner from the Curriculum.
- as the student model is specific to each user, it is preferable to implement it in totality in a client process (a simulation of this possibility led to good results). It's the same for the Monitor who is in charge of the presentations to the student through the interface;
- the Curriculum, in a client process, consists in a copy of the subject matter part that the Monitor has to present to the student. This part is variable according the user. So, in the present approach the Curriculum makes requests to the database of the subject matter contained in a central server;
- the planner is an intermediate module which transfers planning requests and results obtained by the Monitor to the planning server. This one can perform its task independently before transferring the new plan to the client process.
Figure 3 precises a possible architecture in which two central servers (represented by Sun stations) contain one the ART system, and the other the subject matter to teach (including video sequences) in an object database. Each Macintosh supports a client process composed by: a student model (M-E, here not integrated in the Curriculum) distinct of the curriculum, a planner (p), a monitor (M), a curriculum (c) and an interface. Notice that new Macintosh have a graphical interface that incorporates facilities for presenting multimedia applications.

This distribution of functionalities offers the advantage to limit messages exchange through the network to a subset related to a specific domain (for instance planning). More details on this architecture are given in [14], where the distributing process is illustrated.

5. Discussion
The source code of an ITS is composed of functions, procedures and rules. For functions and procedures, the formalism of communicating state machines is adequate, because they can be represented by algorithms and finite automata. For rules, the formalism comes up against the inference mode. If it is possible to simulate a forward chaining mode with automata, it is not the case for backward chaining which lead to numerous problems in size and complexity. The formalism cannot be applied for detailed descriptions.

The reverse-engineering technique was however useful for several reasons. We obtained several clarifications on the communication protocol used in Wordtutor. Solutions for distributing functionalities of the new prototype appeared easily and presented the
advantage to be simulated. We obtained a more clear view of the final design decisions of the first prototype and could justify the new design of the distributed system.

Reverse-engineering of an ITS can be enhanced by the simulation of the behavior of the ITS along with its specification. The design process is accelerated and the decisions evaluated. New systems can be tested and integrated with previous prototypes. Reusability is then possible with this technique.

The development of formal specification of a complete distributed ITS architecture could be achieved by including additional tools in the reverse-engineering process in order to allow simulation of more detailed functions and generation of code from executable specification.

Acknowledgements
We would like to thank Gregor von Bochmann for his support in theoretical foundations.

Références
Mémoire de maîtrise, Département d'informatique et de Recherche Opérationnelle, Université de Montréal, Montréal, Québec, Canada

