A Task-Specific Architecture for the Generation of Intelligent Tutoring Systems

Eman El-Sheikh
Department of Computer Science
University of West Florida
11000 University Parkway
Pensacola, FL 32514 USA
eelsheikh@uwf.edu

Jon Sticklen
Intelligent Systems Laboratory
Computer Science and Engineering Department
Michigan State University
East Lansing, MI 48824 USA
sticklen@cse.msu.edu

Abstract
There is a need for easier, more cost-effective means of developing intelligent tutoring systems (ITSs). A novel and advantageous solution to this problem is the development of a task-specific ITS shell that can generate tutoring systems for different domains within a given class of tasks. Task-specific authoring shells offer flexibility in generating ITSs for different domains, yet are powerful enough to build knowledgeable tutors. In this report, we describe the development of an architecture for the generation of intelligent tutoring systems for various domains by interfacing with existing generic task-based expert systems, and reusing other tutoring components.

Introduction
The need for effective tutoring and training is mounting, given the increasing knowledge demand in academia and industry. Rapid progress in science and technology has created a need for people who can learn knowledge-intensive domains and solve complex problems. The incorporation of artificial intelligence techniques and expert systems technology to computer-assisted instruction (CAI) systems gave rise to intelligent tutoring systems (ITSs), i.e., systems that model the learner’s understanding of a topic and adapt the instruction accordingly. Although ITS research has been carried out for over two decades, few tutoring systems have made the transition to the commercial market. One of the main reasons for this failure to deliver is that the development of ITSs is difficult, time-consuming, and costly. There is a need for easier, more cost-effective means of developing tutoring systems.

In this research, we describe a novel ITS development methodology for the generation of tutoring systems for a wide range of domains. We focus on the development of an ITS architecture that interacts with any generic task-based (GT) expert system to produce a tutoring system for the domain knowledge represented in that system.

For many years, researchers have argued that individualized learning offers the most effective and efficient learning for most students (Bloom 1984; Cohen et al. 1982; Juel 1996). Intelligent tutoring systems epitomize this principle of individualized instruction. Recent studies have found that ITSs can be highly effective learning aides (Shute and Psotka 1996). Shute evaluated several ITSs to judge how they live up to the two main promises of ITSs: (1) to provide more effective and efficient learning in relation to traditional instructional techniques, and (2) to reduce the range of learning outcome measures where a majority of individuals are elevated to high performance levels. Results of such studies show that tutoring systems do accelerate learning with no degradation in final outcome.

Although ITSs are becoming more common and proving to be increasingly effective, a serious problem exists in the current methodology of developing intelligent tutoring systems. Each application is usually developed independently from scratch, and is very time-consuming and difficult to build. In one study of educators using basic tools to build an ITS, results indicated that each hour of instruction required approximately 100 person-hours of development time (Murray 1998). Another problem is that there is very little reuse of tutoring components between applications or across domains. The dilemma that Clancey and Joerger noted at the First International Conference on Intelligent Tutoring Systems that “…the endeavor is one that only experienced programmers (or experts trained to be programmers) can accomplish…” still faces us today (Clancey and Joerger 1988).

Authoring tools for ITSs are not yet commercially available, but authoring systems are available for traditional CAI and multimedia-based training. However, these systems lack the sophistication required to build “intelligent” tutors. Commercial authoring systems give instructional designers and domain experts tools to produce visually appealing and interactive screens, but do not provide a means of developing a rich and deep representation of the domain knowledge and pedagogy. Indeed, most commercial systems allow only a shallow representation of content.

Moreover, a gap exists in current authoring systems between the tutoring content and the underlying knowledge...
organization. There is a need for ITS authoring tools that can bridge this gap by making the organization of the knowledge used for tutoring more explicit.

The motivation for our work comes from the need for reusable intelligent tutoring systems and from the leverage that the generic task (GT) development methodology offers in solving this problem. The assumption of the GT approach is that there are basic “tasks” - problem solving strategies and corresponding knowledge representation templates - from which complex problem solving may be decomposed (Chandrasekaran 1986). GT systems are strongly committed to both a semantically meaningful domain knowledge representation, and an explicit inferencing strategy. The architecture developed can interact with a GT-based system, and produce an effective tutorial covering the domain knowledge represented in the problem solver.

This approach facilitates the reuse of tutoring components for various domains. The ITS shell can be used in conjunction with any GT-based expert system, effectively allowing the same tutoring components to be plugged in with different domain knowledge bases. In other words, the tutoring overlay can be used to generate an ITS for a domain by linking to a GT expert system for that domain. The same tutoring overlay can be used as-is to generate an ITS for other domains by linking to different GT expert systems. The ITS shell is domain-free, and this allows it to be reused for different domains. Our approach makes the underlying knowledge organization of the expert systems explicit, and reuses both the problem solving and domain knowledge for tutoring (El-Sheikh 1999; El-Sheikh and Sticklen 1998).

Leverage of a Task-Specific Framework for ITS Generation

Task-specific authoring environments aim to provide an environment for developing ITSs for a class of tasks. They incorporate pre-defined notions of teaching strategies, system-learner interactions, and interface components that are intended to support a specific class of tasks rather than a single domain.

Task-specific authoring systems offer considerable flexibility, while maintaining rich semantics to build “knowledgeable” or “intelligent” tutors. They are generally easy to use, because they target a particular class of tasks, and thus can support a development environment that authors can use with minimal training. A task-specific ITS shell also supports rapid prototyping of tutoring systems since different knowledge bases can readily be plugged into the shell’s domain-free expert model. In addition, they afford a high degree of reusability because they can be used to develop tutoring systems for a wide range of domains, within a class of tasks. Moreover, task-specific authoring environments are likely to be pedagogically sound because they can utilize the most effective

instructional and communication strategies for the class of tasks they address.

Driven by the need for achieving cost-effective and reusable ITSs, several research efforts were initiated that aimed to develop task-specific authoring tools or environments for tutoring systems. One such environment is IDLE-Tool, the Investigate and Decide Learning Environments Tool (Bell 1999). IDLE-Tool supports the design and implementation of educational software for investigate and decide tasks, which are a type of goal-based scenarios. Another example of task-specific authoring environments is TRAINER (Reinhardt 1995), a shell for developing training systems for tasks such as medical diagnosis.

The Generic Task Expert Systems Development Methodology

The generic task approach is a semantically motivated approach to developing reusable software - in particular reusable shells for knowledge-based systems analysis and implementation. Each GT is defined by a unique combination of: (1) a well-defined description of GT input and output form, (2) a description of the knowledge structure that must be followed for the GT, and (3) a description of the inference strategy utilized by the GT. To develop a system following this approach, a knowledge engineer first performs a task decomposition of the problem, which proceeds until a sub-task matches an individual generic task, or another method is identified to perform the sub-task. The knowledge engineer then implements the identified instances of atomic GT building blocks using off-the-shelf GT shells by obtaining the appropriate domain knowledge to fill in the identified GT knowledge structure. Having a pre-enumerated set of generic tasks and corresponding shells guides the knowledge engineer during the analysis phase of system development. Several atomic generic tasks, such as structured matching, hierarchical classification, and routine design, have been identified and implemented.

This framework focuses on hierarchical classification (HC), a knowledge representation and inferencing technique for selecting among a number of hierarchically organized options. Knowledge is organized in the form of a hierarchy of pre-specified categories. The higher level categories represent the more general hypotheses, while the lower level categories represent more specific hypotheses. Inferencing uses an algorithm called “establish-refine” in which each category attempts to establish itself by matching patterns of observed data against pre-defined matching patterns, and then refined by having its sub-categories attempt to establish themselves. Pruning the hierarchy at high levels of generality eliminates some of the computational complexity inherent in the classification problem.
An Architecture for ITS Generation from Generic Task Expert Systems

The GT framework is extended by developing an ITS architecture that can interact with any GT-type problem solver to produce a tutoring system for the domain addressed by the problem solver. The learner interacts with both the tutoring system shell (to receive instruction, feedback, and guidance), and the expert system (to solve problems and look at examples), in an integrated environment as shown in figure 1.

The architecture for a tutoring extension to the generic task framework is shown in figure 2. The architecture consists of three main components: (1) a GT expert system, (2) a component for extended domain knowledge for tutoring, and (3) an ITS shell. The GT expert system is used by the tutoring shell and the learner to derive problem solving knowledge and solve examples. The extended domain knowledge component stores knowledge about the domain that is necessary for tutoring, but not available from the expert system, such as pedagogical knowledge. The ITS shell has four main components: the expert model, student model, instructional manager, and user interface.

The expert model component of the ITS shell models the structure of a GT expert system. Rather than re-implement the expert model for each domain, the ITS shell interfaces with a GT system, through the knowledge link depicted in figure 2, to extract the necessary knowledge for each domain. This facilitates the reuse of the instructional manager, student model, and user interface components for different domains.

Linking the ITS’s expert model to the problem solver deserves special consideration. Rather than encode domain knowledge explicitly, the expert model extracts and utilizes the domain knowledge available in the expert system. Thus, the quality of the tutoring knowledge is affected by the knowledge representation used by the expert system. The GT methodology’s strong commitment to both a semantically meaningful knowledge representation method, and a structured inferencing strategy allows the extraction of well-defined tutoring knowledge. The expert model extracts three types of knowledge:

- Decision-making knowledge
- Knowledge of the elements in the domain database
- Knowledge of the problem solving strategy and control behavior

To make the knowledge available to the ITS, the expert system must use a knowledge representation that supports tutoring. Generic task expert systems have well defined knowledge structures and reasoning processes that can be reused for tutoring support. A typical GT system is composed of agents, each of which has a specific goal, purpose, and plan of action. The expert system solves problems using a case-by-case approach. Individual cases can be extracted from the expert system, to present as either examples or problems for the learner to solve. The expert model of the ITS shell can extract the following types of case-based knowledge for tutoring:

- Case name and description
- Input variables and values
- Explanation of the output generated

The expert model uses this knowledge, along with an encoding of the expert system’s structure, to formulate domain knowledge as required by the ITS.

The reusable ITS shell has four main components: the expert model, student model, instructional manager, and user interface. The expert model component and how it interacts with a GT expert system was described above. Next, the other components are described briefly.

The architecture adopts a simple, yet beneficial approach for student modeling that utilizes a task model of the expert system. For the purpose of student modeling, the expert system actually represents a model of how an expert would solve problems in the domain. The student model compares the performance of the student during problem solving to the expert system’s solution to make inferences about how the learner is learning. An overlay model is used to assign performance scores to the problems that the learner solves. Each question that the learner answers is assigned a score based on how many hints and/or attempts the learner needed, and on whether or not the learner was
able to determine the correct answer. Each topic has an overall score that is computed as the average score of all the questions covered on that topic. The instructional manager uses this information provided by the student model to direct the instruction. For example, if the learner’s overall topic score is low, the tutor presents more examples. If the score is high, the tutor can ask the learner to solve more questions, or move on to the next topic.

The instructional manager uses an instructional plan that incorporates a cognitive apprenticeship approach; learners move from looking at examples to solving problems as their competence level of the domain increases. The curriculum includes:

- Presentation of an overview on the domain topic. This gives the learner an introduction to problem solving in the domain, including a description of the input variables and output alternatives of the problem solving process.
- Presentation of problem solving examples on each topic of the domain.
- Asking the learner to solve problems covering the domain topics.

The curriculum gives the author flexibility in determining the content of the examples and questions presented to the user. The author selects these from the set of cases defined in the expert system. The author can also determine the number of questions to ask the user for each topic. The pedagogical strategy also includes techniques for giving the learner feedback and hints. When the learner solves a problem during a tutoring session, the tutor gives the learner appropriate feedback according to whether the learner solved it correctly or not. If the learner’s answer is incorrect, the tutor gives the learner a hint specific to the mistake committed, and then re-asks the question. The pedagogical strategy supports giving the learner up to three levels of hints and attempts to solve a problem, after which the tutor presents the correct answer if the learner still did not answer the question correctly.

The architecture employs a simple user interface that has been designed for tutoring using a problem solving approach. Since the tutoring content is mainly generated from the expert system, the user interface is tailored to allow the learner to interact with both the ITS shell and expert system in a transparent manner. The domain knowledge presented through the user interface is generated from the underlying expert system.

For the ITS architecture to produce effective tutoring, it needs the appropriate domain and pedagogical knowledge about the domain. Additional pedagogical knowledge is obtained from the extended knowledge component and used by the ITS shell in presenting an overview of the domain to the user. This knowledge includes:

- A high-level description of the domain.
- The list of examples used in the curriculum.
- The list of questions used in the curriculum.

The ITS author specifies what topics to be included in the tutoring curriculum by defining the list of examples and questions in the extended knowledge component.

**Implementation**

The conceptual framework described above was implemented as an architecture named Tahuti, which consists of three main components: the GT-based expert system, the ITS shell, and the extended knowledge module.
The architecture runs as a CD-based platform-independent environment, and was developed in Smalltalk. Any hierarchical classification-based GT expert system can be plugged into the architecture to generate tutors for different domains. The extended knowledge module is implemented as a structured text file. It includes a high level description of the problem and the topics to be covered in the curriculum as examples and questions.

To generate an ITS for a certain domain using the Tahuti architecture, an ITS author perform three main steps:
1. Identify an existing GT-based expert system for that domain, or develop one using the Integrated Generic Task Toolset developed and used at the Intelligent Systems Laboratory, Michigan State University (Sticklen al. 2000).
2. Construct the extended knowledge module, by filling in the template provided as a structured text file with a description of the domain, and a list of the examples and questions to be used in the curriculum.
3. Generate the intelligent tutoring system. This step is largely automated. The ITS author links the expert system and extended knowledge module to the ITS shell, by identifying the file names, after which, the ITS is generated automatically.

Conclusions

This research work addresses the need for easier, more cost-effective means of developing intelligent tutoring systems. We suggest that a novel and advantageous solution to this problem is the development of a generic task-specific ITS shell that can generate tutoring systems for different domains within a class of tasks. Task-specific authoring shells offer flexibility in generating ITSs for different domains, while still being powerful enough to build knowledgeable tutors.

We have formulated a technique for leveraging the knowledge representation and structure of the Generic Task expert systems framework, and reusing that knowledge for tutoring. More specifically, we have presented an architecture for generating intelligent tutoring systems for various domains by interfacing with existing HC-based GT expert systems, and for reusing the other tutoring components. Among other features, the architecture employs a runnable deep model of domain expertise, facilitates fine grained student diagnosis, offers an easy method for generating ITSs from expert systems, and allows the core ITS shell components to be reused with different knowledge bases.

Future work includes testing the architecture with several knowledge-based systems to generate different tutors, and evaluating the tutors generated within instructional settings, for example, as classroom learning aides. Moreover, by incorporating different instructional strategies or content formats, the architecture developed could be used as a tool for the design and evaluation of different learning environments, and as an important component of a virtual laboratory for experimentation with new learning approaches and technologies.

References