FlexiTrainer: A Visual Authoring Framework for Case-Based Intelligent Tutoring Systems

Sowmya Ramachandran, Emilio Remolina, and Daniel Fu

Stottler Henke Associates, Inc., 951 Mariner's Island Blvd. #360, San Mateo, CA, 94404
{sowmya, remolina, fu}@stottlerhenke.com

Abstract

The need for rapid and cost-effective development Intelligent Tutoring Systems with flexible pedagogical approaches has led to a demand for authoring tools. The authoring systems developed to date provide a range of options and flexibility, such as authoring simulations, or authoring tutoring strategies. This paper describes FlexiTrainer, an authoring framework that enables the rapid creation of pedagogically rich and performance-oriented learning environments with custom content and tutoring strategies. FlexiTrainer provides tools for specifying the domain knowledge and derives its power from a visual behavior editor for specifying the dynamic behavior of tutoring agents that interact to deliver instruction. The FlexiTrainer runtime engine is an agent based system where different instructional agents carry out teaching related actions to achieve instructional goals. FlexiTrainer has been used to develop an ITS for training helicopter pilots in flying skills.

Introduction

As Intelligent Tutoring Systems gain currency in the world outside academic research, there is an increasing need for re-usable authoring tools that will accelerate creation of such systems. At the same time there exists a desire for flexibility in terms of the communications choices made by the tutor. Several authoring frameworks have been developed that provide varying degrees of control, such as content, student modeling and instructional planning [3]. Some allow the authoring of simulations [2], while some provide a way to write custom tutoring strategies [1, 4]. However, among the latter type, none can create tutors with sophisticated instruction including rich interactions like simulations [3]. Our goal was to develop an authoring tool and engine for domains that embraced simulation-based training. In addition, our users needed facilities for creating and modifying content, performance evaluation, assessment procedures, student model attributes, and tutoring strategies. In response, we developed the FlexiTrainer framework which enables rapid creation of pedagogically rich and performance-oriented learning environments with custom content and tutoring strategies.

FlexiTrainer Overview

FlexiTrainer consists of two components: the authoring tool, and the runtime engine. The core components of the FlexiTrainer authoring tool are the Task-skill-principle Editor, the Exercise Editor, the Student Model Editor, and the Tutor Behavior Editor. The *Task-skill-principle Editor* enables the definition of the knowledge of what to teach and includes the following default types of knowledge objects: tasks, skills, and principles. These define the core set of domain knowledge. The *Exercise Editor* facilitates the creation of a library of such exercises for the tutor to draw upon as it trains the students. The *Tutor Behavior Editor* has the author specify two kinds of knowledge: how to assess the student and how to teach the student. Both types of knowledge are captured in the form of behavior scripts that specify tutor behavior under different conditions. These behaviors are visualized in a "drag and drop" style canvas.

Except for the Behavior Editor, all the other editors employ a uniform method for creating knowledge structures. An atomic structure consists of a *type* which is a set of properties common to a number of instances that distinguish them as an identifiable class. For example, the author may want to define "definition" as a separate knowledge type by creating a "definition" type with properties "name", "description", and "review content". An instance would be a definition of "groundspeed" with values filled in, such as "speed relative to the ground" and "ground speed review.html".

Types and instances provide a way for gathering knowledge. Ultimately, there are two ways in which the knowledge will become operational: evaluating and teaching the student. The ways in which the training system fulfills these functions are driven by behavior scripts that dictate how the training system should interact with the student.

FlexiTrainer's behavior model is a hierarchical finite state machine where the flow of control resides in stacks of hierarchical states. Condition logic is evaluated according to a prescribed ordering, showing very obvious flow of control. FlexiTrainer employs four constructs: *actions*, which define all the different actions FlexiTrainer can perform; *behaviors* that chain actions and conditional logic; *predicates*, which set the conditions under which each action and behavior will happen; and *connectors*, which control the order in which conditions are evaluated, and actions and behaviors take place. These four allow one to create behavior that ranges from simple sequences to complex conditional logic. Figure 1 shows an example "teach for mastery" behavior invoked whenever the student wants to improve his flying skills. It starts in the upper left rectangle. The particular skill to practice is determined by the selectSkill behavior. Once the skill to practice is chosen, the teachSkill behavior is invoked: it will pick an exercise that reinforces the skill (and is appropriate for the student mastery level) and then will call the teachExercise behavior to actually carry out the exercise. If the student has not taken the assessment test yet, he will take the test before any skills are selected.

Instructional agents carry out teaching-related actions to achieve instructional goals. The behaviors specified with the Behavior Editor define how agents satisfy different goals. The engine also incorporates a student modeling strategy using Bayesian inference.

So far the FlexiTrainer framework has been used to develop an ITS to train novice helicopter pilots in flying skills [5]. We plan to add other functionality such as: ability to support development of web-based tutoring systems; support for creating ITSs for team training; a pre-defined library of standard tutoring behaviors reflecting diverse instructional approaches for different types of skills and knowledge.

The work reported here was funded by the Office of the Secretary of Defense under contract number DASW01-01-C-5317.

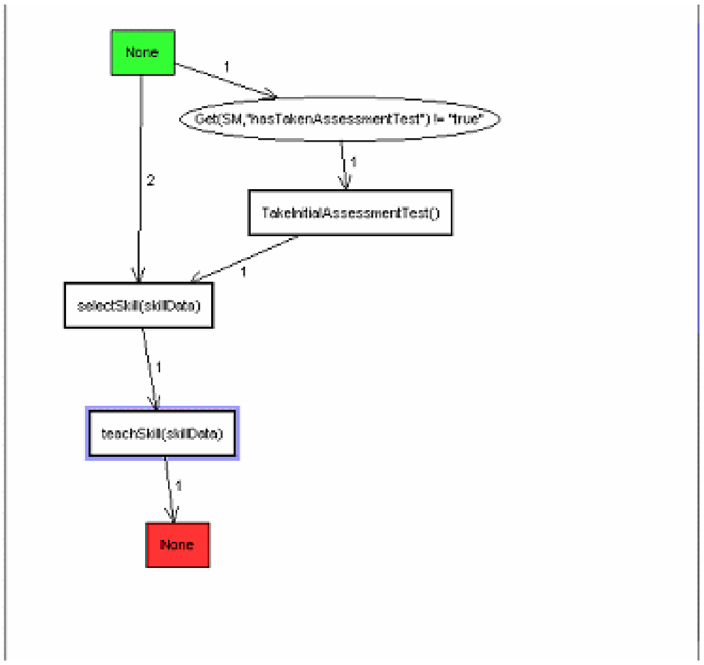
None
1
Get(SM,"hasTakenAssessmentTest") != "true"
2
1
TakeInitialAssessmentTest()
1
selectSkill(skillData)
1
teachSkill(skillData)
1
None

Fig. 1. Example of a dynamic behavior specification

References

1. Major, N., Ainsworth, S. and Wood, D. (1997) REDEEM: Exploiting symbiosis between psychology and authoring environments. *International Journal of Artificial Intelligence in Education*, 8 (3-4) 317-340.
2. Munro, A., Johnson, M.C., Pizzini, Q.A., Surmon, D.S., Towne, D.M. and Wogulis, J.L. (1997). Authoring simulation-centered tutors with RIDES. *International Journal of Artificial Intelligence in Education*. 8(3-4), 284-316.
3. Murray, T (1999). Authoring Intelligent Tutoring Systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10, 98-129.
4. Murray T. (1998). Authoring knowledge-based tutors: Tools for content, instructional strategy, student model, and interface design. *Journal of the Learning Sciences*, 7(1).
5. Ramachandran, S. (2004). An Intelligent Tutoring System Approach to Adaptive Instructional Systems, Phase II SBIR Final Report, Army Research Institute, Fort Rucker, AL.