Creating an Adaptive Training System: Integration of the SMART Student Model in a RIDES Tutor

Carol D. Horwitz
Command Technologies, Inc.
cdh@count.brooks.af.mil

Valerie J. Shute
USAF Armstrong Laboratory
vshute@cobra.brooks.af.mil

J. L. Fleming
USAF Armstrong Laboratory
fleming@alhrt.brooks.af.mil

7909 Lindbergh Drive
Brooks AFB, Texas 78235

Abstract

This paper describes the technical investigation, design, and implementation of integrating the Student Modeling Approach for Responsive Tutoring (SMART) into a tutor built with the Rapid ITS Authoring Development Shell (RIDES©). The effort resulted in RIDES software components and a definitive methodology that can be used to instantiate the SMART approach to instruction in a RIDES tutor.

Background and Goal

The Rapid Intelligent Tutoring System Development Shell (RIDES©) was built by Behavioral Technology Laboratories (BTL) of the University of Southern California under contract to Air Force Armstrong Laboratory. This tool was designed to cost effectively develop, deliver and maintain intelligent computer-based tutors for field and laboratory applications. Application of such training systems have the potential of doubling the instructional effectiveness of more traditional instructional approaches. Also, the initial investment required to convert to these systems is paid back and cost savings begin to be realized within 2 years (Regian et. al. 1996). The full functionality of the RIDES authoring shell is documented elsewhere (see Munro 1997). This paper will only present those features and elements of the authoring shell relevant to this technical investigation and the determined course of action.

In general, the RIDES Authoring Shell allows developers to rapidly 1) implement a graphical simulation of any device or process, 2) develop instructional exercises using the simulation as the basis for interaction with the student, 3) easily modify or extend both the simulation and the instruction, 4) tie instructional exercises to specific learning objectives, and furthermore, to a course plan, and 5) adapt instruction based on student performance.

Typically, a course of instruction in RIDES consists of a set of learning objectives. Each objective is associated with a RIDES lesson and can be satisfied by "successful" completion of that lesson, where "success" is defined by a score considered by the instructor to be "passing." A course structure is created by defining some objectives as prerequisites for others. The RIDES shell manages lesson presentation, taking into account the student's performance on each attempt of a lesson and which objectives are prerequisites for others. Although any performance or simulation parameter may be tracked as part of the student model, an integral part of the model in a traditional RIDES course is the student's score and number of attempts for each learning objective. The student model is updated continuously as the student progresses through the course.

Although many courses can be adequately addressed using this approach, instructors may choose to utilize more advanced sequencing mechanisms in an effort to realize more effective tutoring. The RIDES simulation engine and relation rule capabilities facilitate implementation of more sophisticated student modeling approaches and the use of such student models in lesson selection.

One such advanced sequencing mechanism is the Student Modeling Approach for Responsive Tutoring (SMART). The SMART student modeling paradigm is designed to enhance the efficacy of automated instructional systems across a broad range of domains (Shute 1995). The target domain knowledge is derived via a principled cognitive task analysis procedure, resulting in three distinct outcome types: symbolic knowledge(SK), procedural skill(PS), and conceptual knowledge(CK). Knowledge types are in turn described by a set of curricular elements(CEs) arranged in an inheritance hierarchy (Figure 1). This hierarchy represents the prescriptive order of knowledge acquisition.

The student model is comprised of performance values for each CE reflecting the inferred probabilistic value of mastery. Problems in the tutor are used to evaluate sets of CEs. The basic structure of the tutor is to provide instruction, then problem sets, starting with the lowest order set of CEs in the hierarchy. As questions in a problem are completed, the student model is updated for each CE currently being evaluated. In a SMART tutor, a critical component of evaluating the student's value of probable mastery is the amount of feedback required to accomplish a task. When mastery level is eventually achieved for all CEs in the set, the tutor progresses on to the next set of CEs, as described by the inheritance hierarchy.

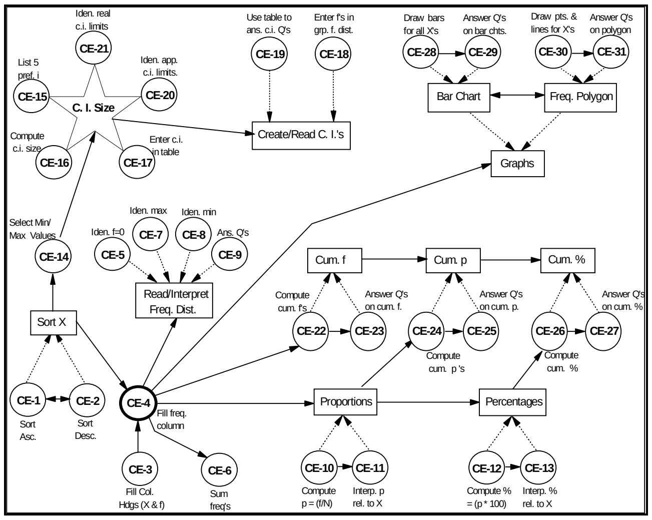


Figure 1. Hierarchy of a subset of curricular elements in Stat Lady (Shute 1995)

SMART has been used effectively in Stat Lady (Shute 1995) and the PC-IMAT tutor (Schuler and Wulfeck, in press). These tutors use an interactive, graphical interface to present domain knowledge to students. Both tutors were implemented using Visual Basic, a generalized programming language. In contrast, the RIDES shell is directed toward the production of simulation-oriented instruction. Could a RIDES tutor be adapted to use the SMART paradigm of instruction? If so, then productivity in tutor development and maintenance could be afforded by the RIDES shell, and efficiencies in student learning could be provided by use of the SMART technology.

Functional Specifications

The SMART student modeling paradigm was reviewed, with particular attention to the functionality of SMART in Stat Lady (Shute 1995) and the design of the generic SMART module for the PC-IMAT tutor (Lefort and Dubbs 1995). The core components of the SMART technology were noted. In general, SMART works as follows: (a) calculates probabilistic mastery levels via a set of regression equations, (b) evaluates what learners know in relation to these low-level bits of knowledge/skill, (c) tailors the curriculum per learner through micro-and macro-adaptive modeling techniques (Shute 1993), and, (d) adapts to both domain-specific knowledge/skills as well as general aptitudes. A more detailed summary of SMART concepts is delineated below:

(1) Curriculum elements (units of instruction) represent the complete set of knowledge and skill elements that comprise the curriculum. These are arranged in an inheritance hierarchy. Figure 1 illustrates a subset of the curriculum from a group of CEs covered in Stat Lady. (For more on this, see Shute 1995.)

(2) Each new chunk of instruction introduces the next set of CE(s) which in turn are evaluated during problem solution in the tutor. Furthermore, each question within a problem set posed by the tutor is associated with one specific CE, so blame assignment (and consequent remediation) is precise and timely.

(3) A value which represents the learner's probable mastery of the curricular element, p(CE), is maintained for each CE. The program allows for continuous representation of the learner's probable mastery values, employing regression equations to compute new p(CE) values (Figure 2). Each of the four equations is linked to the learner's required level of assistance in the solution of a problem involving one or more of the CEs. In other words, the equation invoked is tied to the actual number of hints (i.e., level of feedback) provided by the system to the learner.

h = # of hints required for correct response
X = p(CE) value

$$\text{if } h = 0,\ X_{n+1} = 0.3026 + 1.4377X_n - 0.7207X_n^2$$
$$\text{if } h = 1,\ X_{n+1} = 0.3316 + 0.2946X_n + 1.1543X_n^2 - 0.9507X_n^3$$
$$\text{if } h = 2,\ X_{n+1} = -0.0117 + 0.5066X_n + 0.3518X_n^2$$
$$\text{if } h = 3,\ X_{n+1} = 0.0071 + 0.6001X_n + 2.5574X_n^2 - 1.4676X_n^3$$

Figure 2. Regression equations used to update the student model

(4) SMART is initialized based on pretest performance data, where each pretest item is scored, in real-time, from 0 to 1, with partial credit given where appropriate. The pretest contains items assessing all CEs. This provides the potential for pre-assessed abilities (per CE) which influences tutor delivery. A learner is placed within the curriculum at the lowest level (given the CE hierarchy) instructional piece that presents a CE where their p(CE) value is below some pre-established mastery criterion (e.g., < .70).

(5) The hints given are progressively more explicit (ranging from level 1, vague, to 3, specific).

Moreover, the feedback is specific to the particular problem being worked on, and sensitive to the number of retries. It is provided in response to erroneous inputs, not explicitly requested by the student.

Our technical investigation then turned to reconciling these essential features of SMART with the functionality available in a typical RIDES tutor, or through adaptations to be implemented using the RIDES rule language and simulation engine (Munro et al. 1993). Since one of the major design principles in the development of RIDES was to facilitate a variety of pedagogical approaches, we expected to utilize the traditional simulation, lesson, course and student model components of the shell to achieve a SMART-RIDES tutor. We looked for the parallels between the five components of the SMART paradigm, previously listed, and a simple student modeling approach derived from traditional RIDES course development (Munro 1996). Our analysis determined the following:

(1) There exists a strong parallel between the inheritance hierarchy of curricular elements used in SMART, and the RIDES approach to authoring a course as a series of objectives and any prerequisite relationships that may hold among the objectives. Thus, RIDES authors can build an objective hierarchy by defining one objective as an *enabling objective* for another.

(2) A RIDES lesson corresponds to an instructional piece in a SMART tutor that introduces new CEs, then follows with problem sets to evaluate those CEs. Each *problem* can be described in the RIDES lesson by a *group node*, a component of the RIDES lesson editor. Other components of the lesson editor include a wide variety of *scored* instruction items to measure student knowledge, e.g., multiple-choice question, keyboard question, find task, set-control task. Individual scored items, i.e., questions in the problem, can be associated with specific CEs through a user-defined attribute.

(3) In a traditional RIDES course, the student model consists of the *retries* and *score* attribute values of each learning objective. As the student works through the course material and these attribute values change (i.e., she attempts the learning objective), the corresponding values in the student model are automatically updated by RIDES. Actually, a student model in RIDES can contain *any* RIDES attribute, including new attributes designed by the RIDES author. So, building a student model comprised of the probable mastery values, p(CE), for all curricular elements in the hierarchy is facilitated by the RIDES shell. These p(CE) values will be automatically updated in the student model as the student progresses through the course. Authored instruction can re-calculate the appropriate CE value, as questioning progresses, and a continuous representation of probable mastery values can be maintained in the student model, according to the equations in Figure 2.

(4) The RIDES student model is an ASCII text file, so it is easily readable and editable. This facilitates use of pretest performance measures to prime the p(CE) values in the student model. RIDES tutors allow students to exit the tutor at any point, saving the student model to disk and resuming instruction at the lowest level unattained objective (assuming an objective hierarchy) upon re-entry.

(5) The instruction lesson editor in RIDES facilitates specification of *number of retries* allowed for each scoring item. Also, for each scoring item, RIDES simulation and instruction can access the current number of student attempts to successfully answer the question. This functionality is key to providing more progressively explicit hints in response to erroneous student inputs.

Design Issues

Given the parallels listed above, we decided not to compromise the instructional authoring productivity inherent in the RIDES shell. Consequently, we could make full use of the traditional RIDES instruction engine to maintain individual student models, enable student logins and course selection, and facilitate instructional delivery and student monitoring through the RIDES student instruction window.

RIDES relational expressions, used to effect the simulation, can also be used to dynamically affect instructional presentation within a lesson. However, this approach would incur a lot of overhead, custom authoring attributes and rule expressions for many individual RIDES instruction items. Our goal was to produce a SMART-RIDES tutor while minimizing the need for customization. We hoped to achieve this goal by using RIDES simulation objects to monitor instructional presentation and student performance, and continuously update student p(CE) values according to the regression equations established by the SMART paradigm. These simulation objects could be imported for use in any RIDES tutor, effectively automating the SMART functionality, and thereby minimizing custom instruction authoring. Such objects would materialize through use of the RIDES Library facility, the RIDES rule language and its interpretive nature, and the ability for a RIDES simulation to "author" rules dynamically. However, realization of this plan required us to deal with a few limiting factors identified in RIDES.

Although RIDES permits authoring relationships between instructional objectives, i.e., an objective hierarchy, RIDES course management restricts that relationship to be expressed in terms of a "passing" score for the enabling objective. The score for a learning objective (lesson) in RIDES is usually the accumulation of points received for each scoring item in the lesson. In a SMART-RIDES tutor, we want mastery of specific CEs to serve as the enabling requirement for teaching subsequent CEs. Therefore, our design requires authoring a rule expression in the score attribute of each objective to assign a "passing" score if and only if mastery is obtained for all CEs taught in that objective.

Another service imposed by our design plan is the name of the instruction item, if any, which is currently in play. Our prototype efforts were able to obtain this information in a rather contrived manner from the debriefing capability of RIDES. When *debriefing* is turned on in RIDES, a play-by-play of student performance on each instruction item is dumped to an ASCII file. Based on this requirement in implementing SMART, we proposed a RIDES modification to provide a *currentInstruction* system attribute that would always contain the name of the instruction item currently being played.

The technical investigation concluded that dynamic problem generation (within a lesson) was not key to the SMART technology. That is, once the tutor selected a set of CEs according to the current state of the student model, the specific content of the problem used to evaluate mastery of CEs in the problem set was not pivotal. It should be stressed that while a RIDES lesson could contain a large pool of problems, limitations to the sequencing of problems within the lesson and the ability to dynamically generate the expected student tasks or answers for any given problem were limited. Sequencing *(random or sequential)* can only be modified during instruction authoring. If the pool of problems is to be presented randomly, *randomness* is controlled by RIDES and is not user-definable. If problems are to be presented in a specific order, they are required to appear in the lesson in that order and cannot be changed dynamically during tutoring. Instruction items requiring student completion (i.e., tasks or questions) are authored very specifically in RIDES, for the most part. Although this technical investigation did not produce an example of dynamically generating problem content, it is realized that a RIDES author might achieve this through careful simulation and instruction planning. The correctness of any task established by dynamic problem content would have to be measurable by loosely defined *goal items* or *read indicator items.*

One final design hurdle was a more elegant way for the simulation to discontinue the current lesson when mastery of the CEs in that lesson has occurred. In a traditional RIDES course, each lesson would be played to its entirety. Authors can dynamically alter this behavior by placing a rule expression in individual *doDemo* and *doPractice* attributes of instruction items or instruction group nodes. Again, in an effort to minimize the required custom authoring, we proposed a RIDES modification to the rule language. This modification allows a RIDES simulation event to generate a *stopLesson* command to the RIDES instruction engine.

The Derived Scheme

Since RIDES provides a powerful, object-oriented programming environment, the scheme which evolved from this study defines only one possible solution, and generally serves to illustrate the potential of implementing advanced sequencing models in RIDES tutors.

The derived scheme assumes a pre-defined CE hierarchy for the target domain. Consider a hypothetical SMART-RIDES B-E (Basic Electricity) Tutor based on some of the

- **Relationships Among V, I, R**
  - Voltage (V) is equal to current (I) multiplied resistance (R), or V=I*R
  - When the current (I) goes up or down and resistance (R) stays the same, this implies the voltage should also go up or
  - Current (I) is equal to voltage (V) divided resistance (R), or I=V/R
  - If the voltage goes up or down and resistance stays the sam, this implies that current will go up or down with the
- **Current, in a simple series circuit, something**
  - The current at one point in an piece of wire is equal to the current at point in the uninterrupted piece of
  - The current is the same before and after voltage source
  - The current is the same before and after resistor
- **Voltage drop, in a simple series**
  - The voltage drop across all components of a series net sums up to voltage of the ENTIRE net
  - Voltage drop is lower across any component of a series net than across whole net

Figure 3. Some electricity principles (Shute 1993)

electricity principles defined within the Ohm Tutor (Shute, 1993), shown in Figure 3. If we represent these principles as curricular elements 1 through 9, we can define a simple CE hierarchy for the basis of the B-E Tutor (Figure 4).

The RIDES simulation should provide an interactive environment to illustrate key domain concepts to the student. We assume that each CE in the hierarchy can be evaluated with a RIDES *scoring instruction item* (e.g., menu question, goal, set-control). The student demonstrates knowledge of the CE by either correctly answering the question posed or performing the required interaction(s) with the RIDES simulation. The student's current level of knowledge about the domain is represented by the p(CE) values maintained in a smart course objective, and subsequently saved in the student model.

Our approach also requires the RIDES simulation to implement a hinting mechanism. The hint would appear in any RIDES scene, and its content driven by the task at hand and the current number of attempts to answer correctly. A *hintnum* counter is established with this scheme for this purpose. Three levels of assistance (i.e., hints) are assumed in the scheme to maintain consistency with the regression equations defined in SMART.

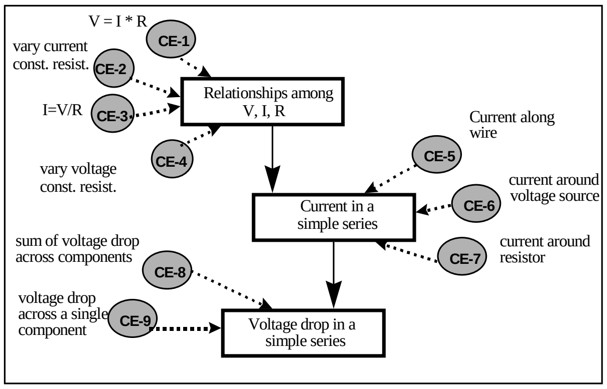


Figure 4. CE Hierarchy for a SMART-RIDES B-E Tutor

Once the simulation is authored, the RIDES lesson editor is used to create individual lessons, each of which addresses some small fixed set of CEs. Although there is no rule for how many CEs are optimally grouped in a lesson, a range of 1 - 4 is suggested. A single *problem* in the lesson addresses several CEs, concurrently, through its set of specific tasks or questions. A RIDES lesson to teach *current in a simple series circuit* as proposed for the B-E Tutor is shown in Figure 5.

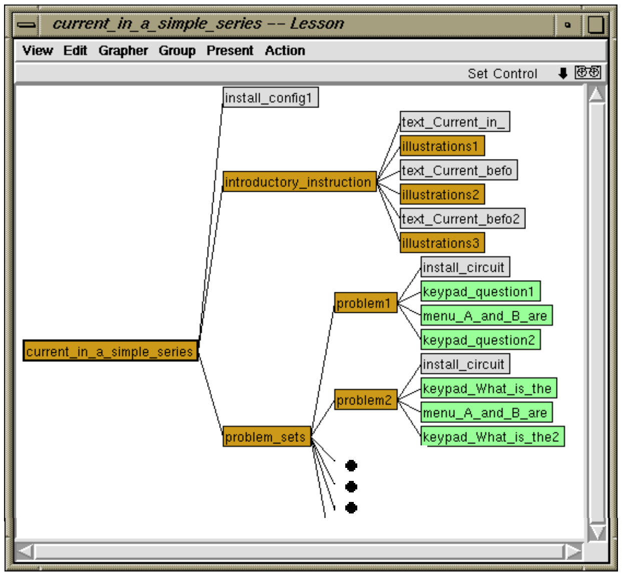
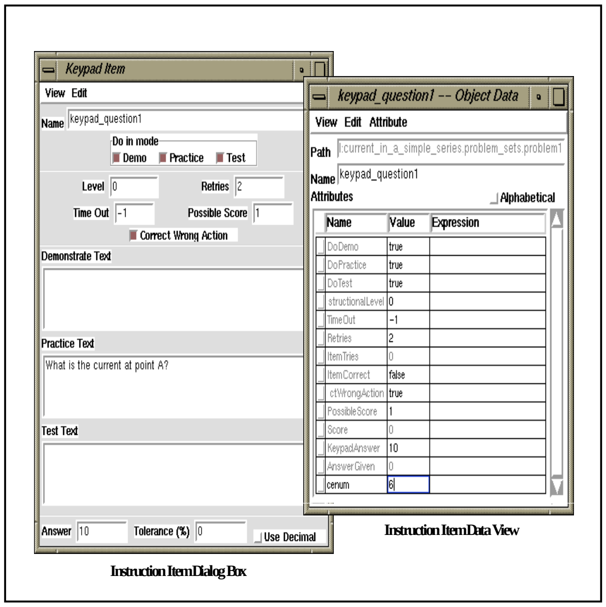


Figure 5. Teaching Current in a Simple Series (B-E Tutor)

Since RIDES lessons cannot "loop," each lesson should contain the maximum number of *problems* required to demonstrate mastery. Problems can be presented in a predetermined or random order within each lesson, and the lesson continues until either the student reaches mastery level for all relevant CEs, or all problems are exhausted. The examples given here do not address the latter condition, but there are many options available to the RIDES author.

Mastery evaluation of each CE in the problem is measured through the student's attempt(s) to accomplish a given task or answer a specific question in the problem. In general, RIDES authors customize instruction in a tutor through instruction item dialog boxes in the RIDES lesson editor. However, even more control of instruction is available to a RIDES author through the instruction item data view. Using the lesson editor and the instruction item data view, the RIDES author adds an attribute to each scored instruction item to associate it with a specific CE. In the example in Figure 6, a keypad question is used in a problem to evaluate the student's understanding of CE6, given a simple series circuit.



Keypad Item
View Edit
Instruction Item Dialog Box
keypad_question1 -- Object Data
View Edit Attribute
Instruction Item Data View

Name: keypad_question1
Do in mode: [✓ Demo] [✓ Practice] [✓ Test]
Level: 0 Retries: 2
Time Out: -1 Possible Score: 1
[✓ Correct Wrong Action]
Demonstrate Text:
Practice Text: What is the current at point A?
Test Text:
Answer: 10 Tolerance (%): 0 Use Decimal

Path: l:current_in_a_simple_series.problem_sets.problem1
Name: keypad_question1
Attributes Alphabetical

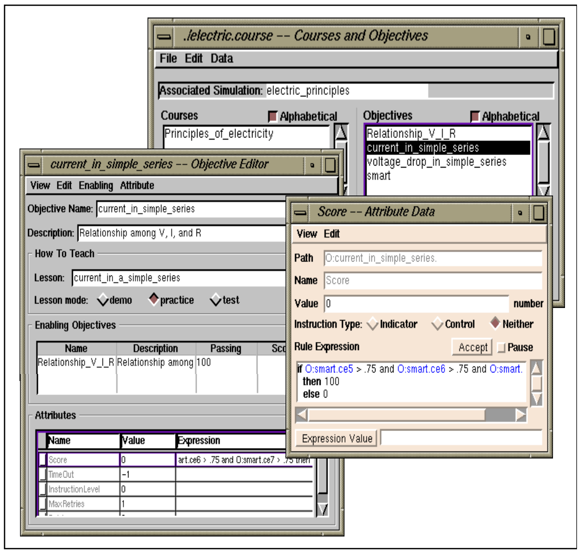
Name	Value	Expression
DoDemo	true	
DoPractice	true	
DoTest	true	
structionalLevel	0	
TimeOut	-1	
Retries	2	
ItemTries	0	
ItemCorrect	false	
ctWrongAction	true	
PossibleScore	1	
Score	0	
KeypadAnswer	10	
AnswerGiven	0	
cenum	6	

Figure 6. Associating a question or task with a specific CE

The pre-defined CE hierarchy is further imposed on the flow of instruction in our SMART-RIDES tutor with a three-step procedure, using the RIDES *Course and Objectives Editor*: 1) Define a learning objective for each sibling set of CEs, and tie each objective to the lesson that presents and evaluates that instructional piece; 2) Establish the enabling objective(s) for each defined objective, if any; and 3) For the *score* attribute of each objective, author a rule that checks for mastery level of all CEs covered by that objective. Figure 7 illustrates some of the steps of this procedure for the SMART-RIDES B-E Tutor.

To this point, we have authored all of the instructional pieces and their relationships, creating a course which ultimately refers to the current student p(CE) values.

However, we are lacking the RIDES "code" required to monitor instructional presentation and student performance, and continuously update student p(CE) values according to the applicable regression equation for each task attempted by the student. Here, our scheme relies on three SMART-RIDES library objects we designed to monitor performance and update student values. Working with the course of instruction authored as described above, the library objects we developed serve to: a) recognize when a CE-specific question/task is presented; b) consider the number of hints required to correctly complete the question/task; c) re-calculate the p(CE) value; and d) stop the lesson when all CEs in that lesson have reached mastery level. Our hope to minimize custom instruction authoring is realized, for the most part, through use of these SMART-RIDES library objects. Basically, the objects are imported with the *Open Library* command into any RIDES tutor being designed to use the SMART technology. Authors wishing to implement a SMART-RIDES tutor need only concern themselves with a few special attributes in the objects.


./electric.course -- Courses and Objectives
File Edit Data
current_in_simple_series -- Objective Editor
View Edit Enabling Attribute
Score -- Attribute Data
View Edit
Accept
Expression Value

Associated Simulation: electric_principles

Courses ☑ Alphabetical: Principles_of_electricity

Objectives ☑ Alphabetical: Relationship_V_I_R; current_in_simple_series; voltage_drop_in_simple_series; smart

Objective Name: current_in_simple_series
Description: Relationship among V, I, and R
How To Teach
Lesson: current_in_a_simple_series
Lesson mode: demo [✓ practice] test

Enabling Objectives

Name	Description	Passing	Sco
Relationship_V_I_R	Relationship among	100	

Attributes

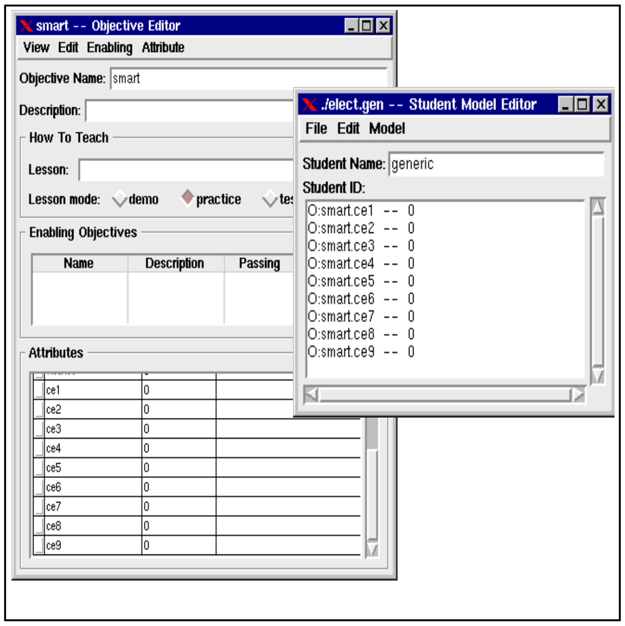
Name	Value	Expression
Score	0	art.ce6 > .75 and O:smart.ce7 > .75 then
TimeOut	-1	
InstructionLevel	0	
MaxRetries	1	

Path: O:current_in_simple_series.
Name: Score
Value: 0 number
Instruction Type: Indicator Control [✓ Neither]
Rule Expression ☐ Pause

```
if O:smart.ce5 > .75 and O:smart.ce6 > .75 and O:smart.
 then 100
 else 0
```

Figure 7. Defining a SMART-RIDES course for the B-E Tutor

As mentioned, the SMART-RIDES objects apply the regression equations, deriving each new p(CE) value from the previous p(CE) value. Therefore, curricular element values have to be accessible throughout the course of instruction However, there are features in RIDES, not specifically addressed in this paper, that allow the simulation to revert to some prior state during instruction. While the specific features are not pertinent, at this point, the effects are. Because implementation of SMART requires p(CE) values to persist until actively re-computed, we must maintain these p(CE) values outside of the simulation attributes. Our scheme requires the values to reside in a *smart* objective, which is defined as part of the SMART-RIDES course. The smart objective is considered a "dummy" objective because it will never be presented to the student during the course of instruction, i.e., it is not tied to a RIDES lesson. Instead, the author uses it to define a number attribute that maintains the student's probable mastery value, p(CE), for each curricular element in the course. These attributes are then also used to create the student model (Figure 8).


smart -- Objective Editor
View Edit Enabling Attribute
./elect.gen -- Student Model Editor
File Edit Model

Objective Name: smart
Description:
How To Teach
Lesson:
Lesson mode: demo [✓ practice] tes

Enabling Objectives

Name	Description	Passing

Attributes

ce1	0
ce2	0
ce3	0
ce4	0
ce5	0
ce6	0
ce7	0
ce8	0
ce9	0

Student Name: generic
Student ID:

```
O:smart.ce1  --  0
O:smart.ce2  --  0
O:smart.ce3  --  0
O:smart.ce4  --  0
O:smart.ce5  --  0
O:smart.ce6  --  0
O:smart.ce7  --  0
O:smart.ce8  --  0
O:smart.ce9  --  0
```

Figure 8. Defining the CE attributes and generic student model

Conclusions

Intelligent Tutoring Systems (ITS) (Regian and Shute, 1992), are often characterized through their use of: 1) elaborate models of domain knowledge; 2) a software simulation that provides the student with interactive practice opportunities; and, 3) pedagogical software to make run-time curriculum decisions and manage learning activities. Although the instructional effectiveness of these advanced automated instructional systems has been demonstrated many times (Shute and Regian 1993), the costs of development and maintenance can be prohibitive.

Tools such as RIDES promote productivity in the development of simulation-based instruction. The pedagogy implicit in a RIDES tutor is a simple instruction sequencing mechanism based on an objective hierarchy. SMART, an advanced sequencing mechanism, defines a highly effective pedagogical algorithm that is domain-independent. A SMART-RIDES tutor would allow us to increase the effectiveness of computer-based instruction, without sacrificing productivity in development and maintenance.

This study resolved that the RIDES authoring shell offers enough flexibility to accommodate the SMART student modeling paradigm, without an excessive increase in the workload generally required to produce a RIDES tutor. Since this investigation did not produce a domain-specific tutor and associated empirical data, this remains an enticing exercise for future research.

Acknowledgment

We would like to thank SSgt. Brian Bagnetto for technical assistance in developing the prototypes and contributions to the technical investigation.

References

Munro, A. 1996. The RIDES Quick Start Manual. Unpublished technical report.

Munro, A. 1997. The RIDES Simulation-based Instructional Authoring System, USAF Armstrong Laboratory Technical Report. Forthcoming

Munro, A., Johnson, M.C., Surmon, D.S., and Wogulis, J.L. 1993. Attribute-centered simulation authoring for instruction. In Proceedings of AI-ED 93 World Conference on Artificial Intelligence in Education.

Regian, J. W., Seidel, R., Schuler, J., and Radtke, P. 1996. Functional Area Analysis of Intelligent Computer-Assisted Instruction. Training and Personnel Systems Science and Technology Evaluation and Management Committee (TAPSTEM) report to the Deputy Undersecretary of Defense for Research and Engineering.

Regian, J. W. and Shute, V. J. 1992. Automated instruction as an approach to individualization. In Regian, J. W. and Shute, V. J. eds. *Cognitive approaches to automated instruction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Schuler, J.W. and Wulfeck, W.H. 1997. Design and development of the Personalized Curriculum for Interactive Multisensor Analysis Training (PC-IMAT) system, Technical Report, San Diego, CA: Navy Personnel Research and Development Center. Forthcoming

Shute, V. J. 1993. A comparison of learning environments: All that glitters... In S. P. Lajoie and S. J. Derry eds. *Computers as cognitive tools* (pp. 47-74). Hillsdale, NJ: Erlbaum.

Shute, V. J. 1995. SMART: Student Modeling Approach for Responsive Tutoring. *User Modeling and User-Adapted Interaction: An International Journal* 5: 1-44.

Shute, V. J., and Regian J. W. 1993. Principles for Evaluating Intelligent Tutoring Systems. *Journal of Artificial Intelligence & Education* 4 (2/3), 245-272.