International Journal of Artificial Intelligence in Education (1997), 8, 284-316

Authoring Simulation-Centered Tutors with RIDES

Allen Munro, Mark C. Johnson, Quentin A. Pizzini, David S. Surmon, Douglas M. Towne, James L. Wogulis

e-mail: munro@usc.edu, mjohnson@usc.edu, pizzini@usc.edu, surmon@usc.edu, dtowne@usc.edu, wogulis@usc.edu,

University of Southern California
Behavioral Technology Lab
250 No. Harbor Drive, Suite 309
Redondo Beach, CA 90277

Abstract. *RIDES* is an application for authoring and delivering interactive graphical simulations and tutorials in the context of those simulations. Some types of instruction can be generated automatically by exploiting structured simulation data. RIDES technology plays an important part in a number of related research and development projects. RIDES introduces innovations in the human-computer interface (HCI) of simulation-centered tutorial authoring systems, including support for constraint and event authoring, enforcing the primacy of reference, and permitting the authoring of procedure tutorials by interactive demonstration. RIDES simulations provide low level automatic support for detecting student operations and observations and are well-suited for teaching procedures. Authoring use has revealed a number of additional design considerations that should be taken into account in the development of future tutor authoring systems. Foremost among these is the need for an open architecture that supports collaboration among tutor components, including collaboration with types of tutor components that have not yet been developed.

INTRODUCTION

Intelligent tutoring systems (ITSs) often include interactive graphical simulations. For many types of tutoring, the use of an interactive graphical simulation helps to assure that what students learn is relevant to actual tasks that they must learn to perform, in a way that a primarily textual approach to learning interactions cannot. Interactive simulations can help to ensure that *performance* skills--as opposed to mere test-taking skills--are acquired as a result of tutoring. To date, most research projects on intelligent tutoring systems that have incorporated simulations have relied on low level tools (i.e., programming languages) to develop both the ITSs and the simulations. Reliance on such low-level development techniques naturally makes simulation-centered tutoring extremely expensive. It can also make it very difficult to determine what features of particular tutor are responsible for its efficacy (or lack thereof). The details of the *craft* of low-level development using programming languages can overwhelm the effects of the general principles that are followed in a particular tutor. An authoring system, by ensuring a uniform quality of low-level instructional interaction and by providing easily edited and modifiable tutorials (and simulations), can make it possible for developers to experiment with different high-level approaches to tutoring in a given domain.

RIDES is an X-Windows-based Unix application for interactively authoring graphical simulations and simulation-centered tutorials. RIDES (and a version of RIDES that lacks authoring features, called *sRides*--for *Student RIDES*) delivers simulation-centered tutoring to students. Because the simulation authoring system is designed to support tutorials, many types of instruction can be very rapidly authored, and many high quality instructional interactions are generated automatically.

Many tutors have now been built using RIDES, including two large tutors (and many smaller ones) that have been produced in our lab. Many more tutors have been developed at other sites. Developers at Armstrong Laboratory have created a large number of RIDES tutors, covering a wide range of subjects, including the operation of medical devices, the nomenclature and theory of orbital mechanics, the human circulatory system, and a wide range of tutors on the operation and maintenance of particular devices. RIDES has also played a role in a number of other research projects, as described below. Our experiences in collaboration with other tutoring projects lead us to believe that a more componential approach to tutorial delivery would significantly enhance the value of authored components such as RIDES simulations and RIDES procedure tutorials.

Background

The field of simulation in intelligent tutoring systems (ITS) research is a large and rapidly growing one, and this paper would be much larger if it were to review all the widely relevant work. On the other hand, the field of simulation-based tutor *authoring systems* is a very much smaller one. In this section, we briefly discuss several authoring systems for the development of simulations for learning.

In the past, many ITS research projects developed isolated intelligent tutors that were created using a variety of general-purpose tools, especially programming environments. There are several reasons why there are few exemplars of simulation-based tutor authoring systems. One of these is that the development of a usable authoring system for simulation-based tutors is a very large task. A good authoring system must have a very large number of quite different, easy-to-use authoring interfaces, or editors. It must be possible to draw (or import) and edit graphic objects, to specify the behavior of the objects, to enter tutorial information about the objects, to build complete lessons in the context of simulations, to build courses, to search for authored data, to debug simulations and lessons, and so on. The development effort is similar to that required to produce several different types of commercial software applications, such as a word processor, a spreadsheet, and a presentation system, and bind them all together in an integrated application suite. It is simply too arduous and expensive an undertaking to be undertaken casually as a part of a project to produce a tutor. It can only be justified if it can then be used to produce scores or thousands of different tutors at significantly lower cost than would be possible using less task-specific tools, such as programming languages.

STEAMER (Williams, Hollan, and Stevens, 1981; Hollan, Hutchins, and Weitzman, 1984) provided a direct manipulation simulation for students learning about a steam propulsion plant. The STEAMER project is an important spiritual ancestor of RIDES. It offered a discovery world for students and a demonstration platform for instructors, but it did not provide authoring tools for the development and delivery of instruction to the learner. Furthermore, simulations had to be written as conventional computer programs. An authoring tool was then used to link simulation values to STEAMER graphic objects. As the simulation values changed, so did the states of the STEAMER graphics.

Forbus (1984) developed an extension of STEAMER called the *feedback minilab*, which could be used to produce interactive graphical simulations without first writing separate simulation programs. This early authoring system provided a set of predefined components (such as valves and switches). Composing a device of these components determined the behavior of the simulated system as a whole.

IMTS (Towne and Munro, 1988, 1992) provided tools for authoring interactive graphical simulations of electrical and hydraulic systems, but the model authoring approach was limited. A library of graphic behaving objects could be composed, but the external effects of these objects had to be of either electrical, mechanical, or hydraulic type. IMTS supported troubleshooting assistance by a generic expert called Profile (Towne, 1984), but it could not be used to develop or deliver other kinds of instruction.

An early approach to direct manipulation instructional authoring was Dominie (Spensley and Elsom-Cook, 1989). That system, however, did not support the independent specification of object behaviors; the specification of simulation effects was confounded with the specification of instruction.

RAPIDS (Towne and Munro, 1991) and RAPIDS II (Coller, Pizzini, Wogulis, Munro, and Towne, 1991) were descendants of IMTS that supported direct manipulation authoring of instructional content in the context of graphical simulations. These systems provided a much more constrained simulation authoring system than is found in RIDES, and they did not offer authors low level control over instructional presentations. Like IMTS, these programs were available only on specialized AI workstations.

RIDES provides much more robust simulation authoring tools (described below) and instructional editing facilities than were to be found in RAPIDS and RAPIDS II. The RIDES system has some features in common with the SMISLE system (de Jong, van Joolingen, Scott, deHoog, Lapied, and Valent, 1994; Van Joolingen and De Jong, 1996), but is less restrictive about how simulations can be structured. SMISLE authors must separately specify an inner, 'real' level of behavior and one or more surface depictions of the behaving system. Similar effects can be achieved using RIDES, but they are not required. The SMISLE system also contains facilities for supporting student hypothesis formation, but lacks the unconstrained simulation authoring and instruction authoring capabilities of RIDES.

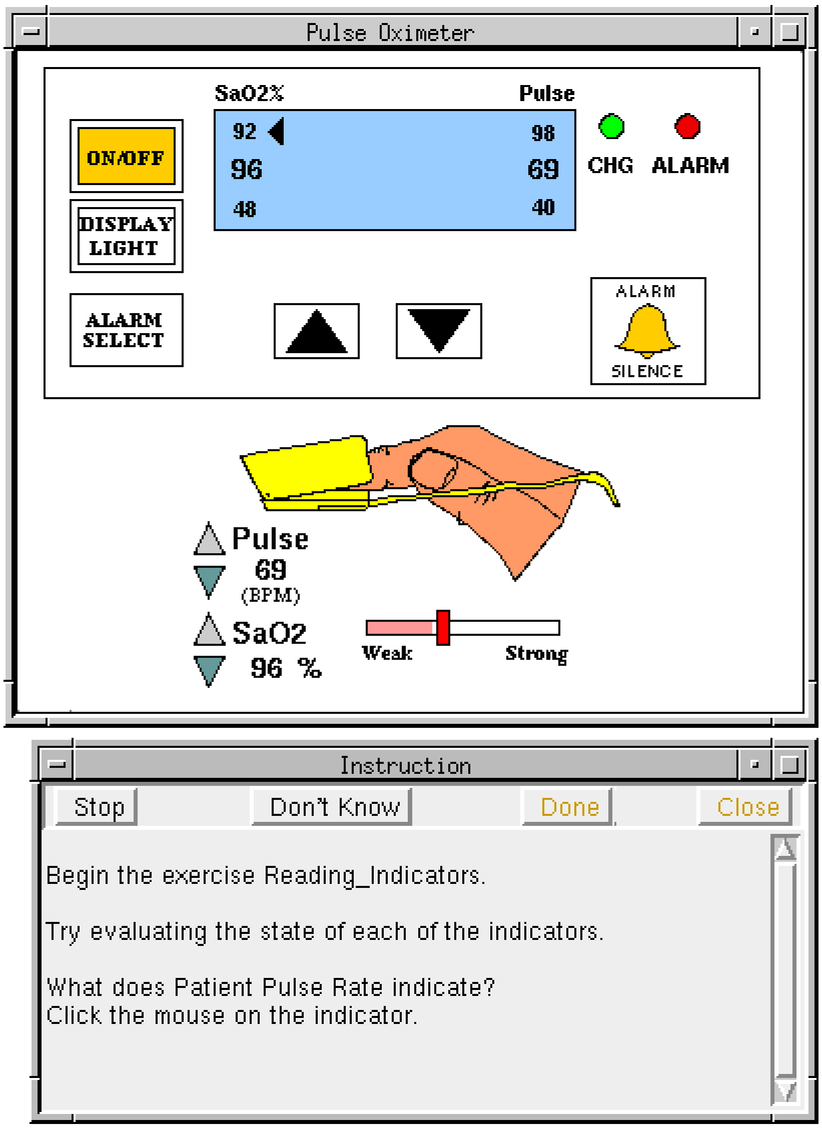
RIDES: A TUTOR DELIVERY AND DEVELOPMENT SYSTEM

RIDES is an integrated software environment for developing and delivering computer based tutorial instruction and practice that is based on graphical simulations. Using RIDES, authors can build interactive graphical models of devices or other complex systems and then swiftly build interactive lessons in the context of those graphical models. Students use RIDES to interact with the authored tutorials. If the author chooses, students can be allowed to explore graphical simulations in a free play mode, but, in addition, many types of structured tutorials can also be productively authored and delivered using RIDES.

Model Based Instruction

One of the key concepts in RIDES is that instruction takes place in the context of an authored graphical model of the particular domain. For example, if the domain is the operation of a certain item of medical equipment, then the appearance, behavior, and proper usage of that equipment is the domain to be learned. Figure 1 shows a student environment with two windows open. The lower window is the student instruction window, where instructional text and remedial directions are displayed. The upper window is one of several windows in this simulation that contain depictions of the domain of interest--here, a pulse oximeter.

Simulation scenes, such as the one with the title "Pulse Oximeter" in the figure, consist of a set of graphical objects in a window. Such scenes provide the context for all instruction in RIDES. Many elements of instruction can be automatically generated by RIDES because they are developed in the context of a structured domain model.



Begin the exercise Reading_Indicators.

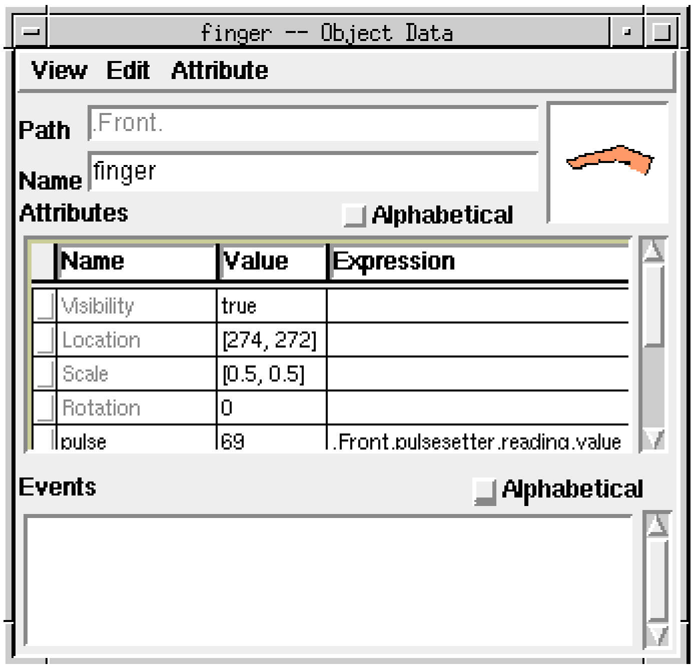
Try evaluating the state of each of the indicators.

What does Patient Pulse Rate indicate?
Click the mouse on the indicator.

Figure 1. A Student View of Instruction in the Context of a Graphical Model

Authors can build complex graphical models by copying library objects and pasting them into their scenes. They can also draw objects directly onto the scenes, using a palette of drawing tools, and can specify rules that control the values of object attributes. The finger in the figure above is one such object. Authors can open an *object data view* of the object, such as the one shown in the figure below. An object data view of an object shows its name, together with a list of its attributes with their values and their rules, if they have any. In this figure, there is a rule for the attribute named *pulse*.

The present value of this attribute is determined by the rule, which refers to the attribute of another object.



Path	.Front.
Name	finger

Attributes

Name	Value	Expression
Visibility	true	
Location	[274, 272]	
Scale	[0.5, 0.5]	
Rotation	0	
pulse	69	.Front.pulsesetter.reading.value

Figure 2. Object Data View for a Graphical Object

Some attributes directly control the appearance of graphical objects. When the values of such attributes are changed, objects may disappear, move, stretch, rotate, or undergo other visible changes.

RIDES has lesson-building tools that can exploit the information in a graphical model. For example, students can be taught the names of objects, because the simulation author gave the objects names that can be used by RIDES lessons. More interestingly, an instruction author can 'record' a procedure that students must learn, simply by carrying out the procedure. During tutoring, the student will be prompted to carry out the correct sequence of actions.

Because RIDES is an integrated development system for producing simulations and instruction presented in the context of those simulations, many aspects of the lesson-authoring process have been automated or streamlined.

Development with RIDES

RIDES offers a number of advantages to authors, to students, and to managers in comparison with other technologies for developing interactive graphical tutoring systems.

Simulation Based on Behaving Graphical Objects

At the simulation level, students don't merely interact with 'hot spots' on a screen, but rather with objects. This has numerous advantages.

- *For the student*
  More robust, more realistic simulations
  Support for free play immersive learning
- *For the developer*
  Potential for the **reuse** of objects
  Ease in developing more robust, flexible, and realistic simulations than can be achieved with the 'hot spot' approach

RIDES provides a range of graphic primitives, including lines, ellipses, rectangles, poly-lines, text, splines, and colored bitmaps. The inclusion of structured graphic types (as opposed to a bitmap-only approach found in some CBT authoring systems) makes it computationally practical to carry out operations such as scaling or stretching objects and rotation under simulation control. Such operations would be slower, jerkier, and visually less appealing if only bitmap objects were supported. In addition to encouraging the development of responsive and flexible simulations, non-bitmap graphical objects often have smaller memory requirements than bitmaps of the same size.

A number of aspects of the RIDES implementation of graphical objects enhance the authoring and maintenance of simulation behavior. In most cases, for example, an author can find out why an object behaves as it does by selecting the object, opening its data view, and looking at its rules. The binding of behavior to objects makes it easy for authors to debug and/or modify behavior in a simulation.

Instruction Grounded in Simulation

The instruction-authoring features of RIDES exploit the structure of simulations to make the authoring of certain types of lessons fast, accurate, and easily maintainable. Authors cannot direct students to put the simulation model into 'impossible' or internally inconsistent states, for example. Procedure training can be authored simply by carrying out a procedure.

Textual information can be entered in *knowledge units* that can be associated with model objects. Some types of lessons can be generated from textual discussions and from other types of information stored in these knowledge units.

Lessons can be played to students in three modes with differing levels of user responsibility. In *demo mode*, students only pace the progress of a lesson. In *practice mode*, students are required to carry out the actions of the lesson, but are given remediation when they fail to perform an action correctly. In *test mode*, students are not given remediation when they mis-perform a step during a learning sequence. It is not necessary to author these three types of lessons separately; all of them are created at once when the lesson is authored.

A number of highly structured kinds of instruction--called *patterned exercises* in RIDES--are authored quickly and directly from simulation data and a few specifications from instructional authors. The generated lessons can be customized and new, innovative lesson types can be built using more detailed instruction authoring tools that also interact directly with the authored simulation.

Courses Based on Objectives

A RIDES course is a set of learning objectives that must be realized by a student. Each objective is associated with a lesson for teaching that objective. A model of the knowledge of each student is based on the objectives of the course. Decisions about what lesson to present next are controlled by the relationships that hold among course objectives and by the state of the student model.

A Supportive, Highly Integrated Authoring Interface

The RIDES simulation-development environment is very natural for most authors with previous experience using window-based application software. To find out why an object is behaving in a certain way, the author can select the object and open a view that displays the rules that govern the object’s behavior. (This can be done while the simulation is running or when it is paused.) An author can modify simulations on-the-fly, even as the simulations are running. Authored changes in attribute values, relations, and events are reflected immediately in a running simulation. Similarly, direct manipulations of simulation objects that modify those objects’ values are immediately reflected in data views of those objects. The authoring system is designed to be highly transparent and consistent.

Authoring mistakes can be easily remedied by using the Undo feature, which supports multiple levels of *undo* and *redo*.

Behavior rules are automatically formatted in ways that help authors to immediately pick out object and attribute references, and that highlight any errors in such references.

Copying and pasting is well supported. In addition to conventional copy-and-paste capabilities, RIDES lets an author copy a named graphic object and paste it in a textual context, such as a behavior rule; the name of the copied object is placed at the insertion point in the text.

Two approaches for creating instruction are integrated in the RIDES authoring application: one for rapidly creating certain kinds of structured lessons, called *patterned exercises,* and one for building custom lessons. Both permit the author to carry out many aspects of lesson authoring by directly manipulating the interactive simulation in the same ways that students will be required to do.

Supplementary Utilities

An integrated debugger lets the author step through simulation execution, evaluating simulation expressions at the time of rule executions.

A list of any 'erroneous rules' that have not yet been repaired is maintained, and authors can directly access the data views for those rules from that list.

A 'find' utility lets authors search for named entities or text strings in RIDES, either unrestrictedly, or searching only in certain types of data.

A 'consistency checker' can be used to report on the presence of certain commonly observed possible problems in the RIDES document.

On-line Reporting to the Instructor Console

RIDES reports student progress to an instructor console program, called RADMIN. Instructors can observe which networked student stations are currently delivering RIDES instruction, what students are using those computers, which course and course objective each student is working on, and how long the student has been on that objective. The RADMIN program also provides instructor facilities for enrolling students at a tutoring site, and for setting up classes. The details of RIDES course administration are described in the *RIDES Administration Manual* (Munro and Surmon, 1996)

Overview of the Tutor Development Process

A model of development for RIDES authoring is shown in the Figure 3 below.

Authors must specify the objectives of a tutor, must relate those objectives to particular elements of instruction, and must develop a model of the domain of interest that can be used to author the lessons that will meet the instructional objectives. After the interactive simulation (the graphical model of the subject matter domain) is developed, the needed lessons must be built and linked to the objectives of the tutor. One or more prototype student models must then be built and the tutor can be tested for effectiveness with students. RIDES provides an integrated set of editors for carrying out all of the tasks shown in Figure 3.

Using the Course Editor, an author can design the objectives that are to be met by the tutor. Then the author can build an interactive simulation. This is often accomplished by a combination of pasting library objects and drawing and giving behavior to new objects. There are scene editors, object editors, attribute and event editors, simulation debuggers and other views and tools that can be used to develop and test simulations. The RIDES simulation authoring environment is so powerful and interactive that some users employ it not to produce tutors, but rather as an application prototyper or as an application development system.

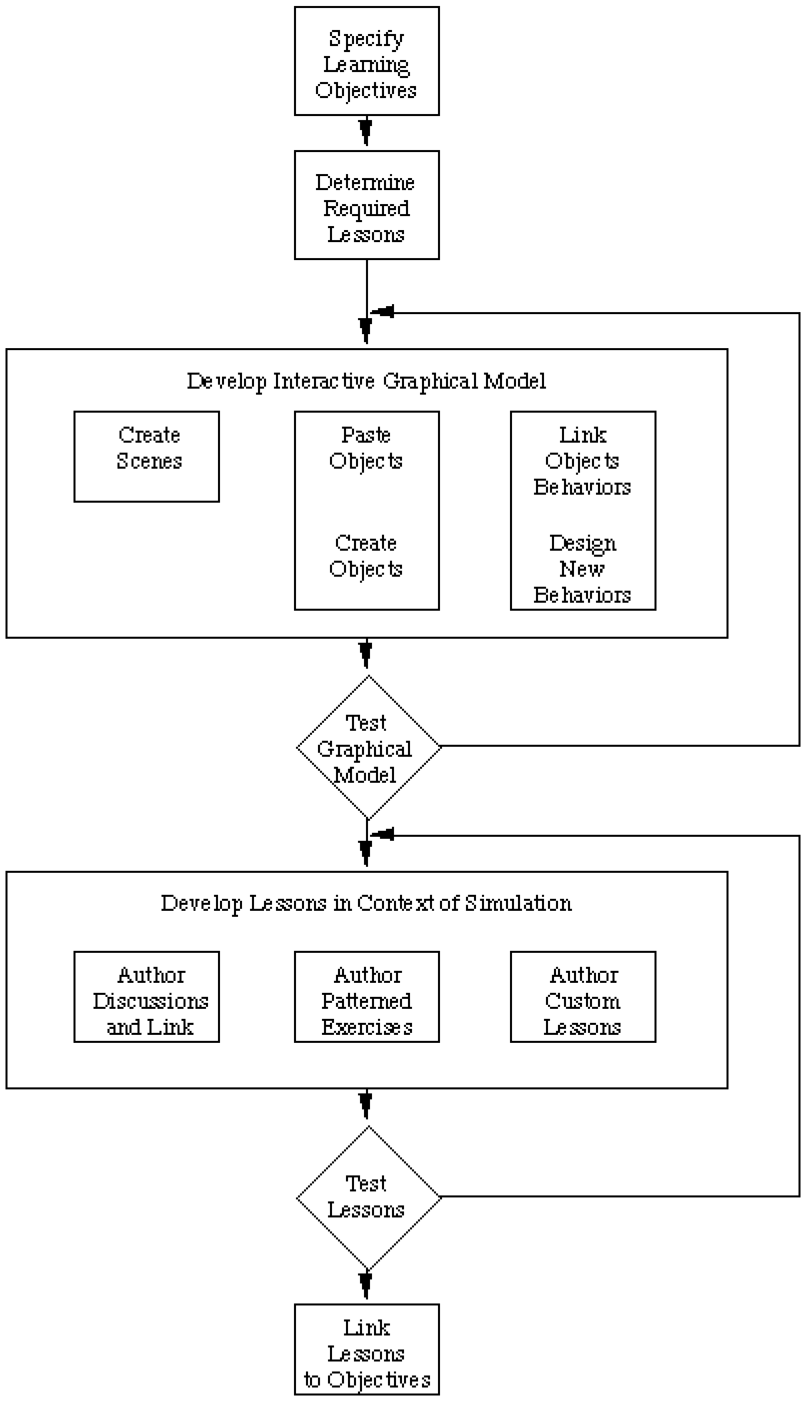

Specify Learning Objectives
Determine Required Lessons
Develop Interactive Graphical Model
Create Scenes
Paste Objects
Create Objects
Link Objects Behaviors
Design New Behaviors
Test Graphical Model
Develop Lessons in Context of Simulation
Author Discussions and Link
Author Patterned Exercises
Author Custom Lessons
Test Lessons
Link Lessons to Objectives

Figure 3. The Tutor Development Process in RIDES

Once a simulation has been built, the author can construct lessons, using either the patterned exercise authoring tools, which require only very simple choices, or using the custom authoring tools, which permit greater control over the instruction process. A combination of both patterned exercises and custom lessons is often used, and generated patterned exercises are sometimes edited using the custom instructional editing tools. Finally, the objectives that were designed in the first step of the tutor authoring process are linked to the authored lessons. RIDES can then present the complete authored course to students.

For a step-by-step account of how a simple simulation is authored and how simple lessons are developed in the context of such a simulation, see the RIDES QuickStart manual, which is available on-line at

http://btl.usc.edu/rides/documntn/qsMan/

HUMAN-COMPUTER INTERFACE ISSUES FOR INTERACTIVE TUTOR DEVELOPMENT

The RIDES project has addressed two fundamental research issues for human-computer interfaces (HCI). The first is, "Which HCI features can help in the development of behaving interactive graphical simulations?" The second is, "Which HCI features will best support an author's development of performance-oriented tutors in the context of interactive graphical simulations?"

HCI for Authoring Behaving Graphical Simulations

Figure 4 shows the RIDES graphic tool palette and examples of the nine primitive graphic types, together with a grouped object.

Any graphical object can be given a name and *attributes*. In RIDES, attributes are used to store values associated with objects. Values can be of six types: number, logical (boolean), text, point, color, and pattern.

Some object attributes are created automatically, while others are created by authors. For example, an author can add a new number attribute to an object, give the attribute a name, specify its current value, and, if desired, write a constraint that prescribes the attribute's value in terms of the values of other attributes. Attributes that are explicitly created by the commands of authors are called *authored attributes*. Attributes that are created automatically are called *intrinsic attributes*.

Most intrinsic attributes control aspects of the appearance of objects. Every graphical object has the attributes Visibility, Location, Scale, and Rotation. Each of these attributes controls some aspect of the appearance of an object. If Visibility is false, the object is not shown on its scene. Location is a point type value that prescribes the Cartesian coordinates of the object on the 'page' that is shown in a scene window. Changing the value of an object's Location will cause it to move to a different spot on its scene. The Scale attribute is also of type point; it specifies scaling factors that should be applied to the X and Y dimensions of the object. (Scaling is with respect to the size that the object had when it was originally drawn or otherwise created.) Rotation is a number attribute that controls the orientation of the object. Rotation values express the extent of an object's counter-clockwise rotation from its original orientation.

In addition to these four universal attributes, objects of particular graphic types have other intrinsic attributes. For example, lines have the additional attributes PenColor, PenStyle, and PenWidth. Filled graphic primitives, such as rectangles, ellipses, polygons, and splines, have these plus FillColor and FillPattern. Text primitives have a TextValue attribute that determines what text is actually displayed by the graphic, and so on.

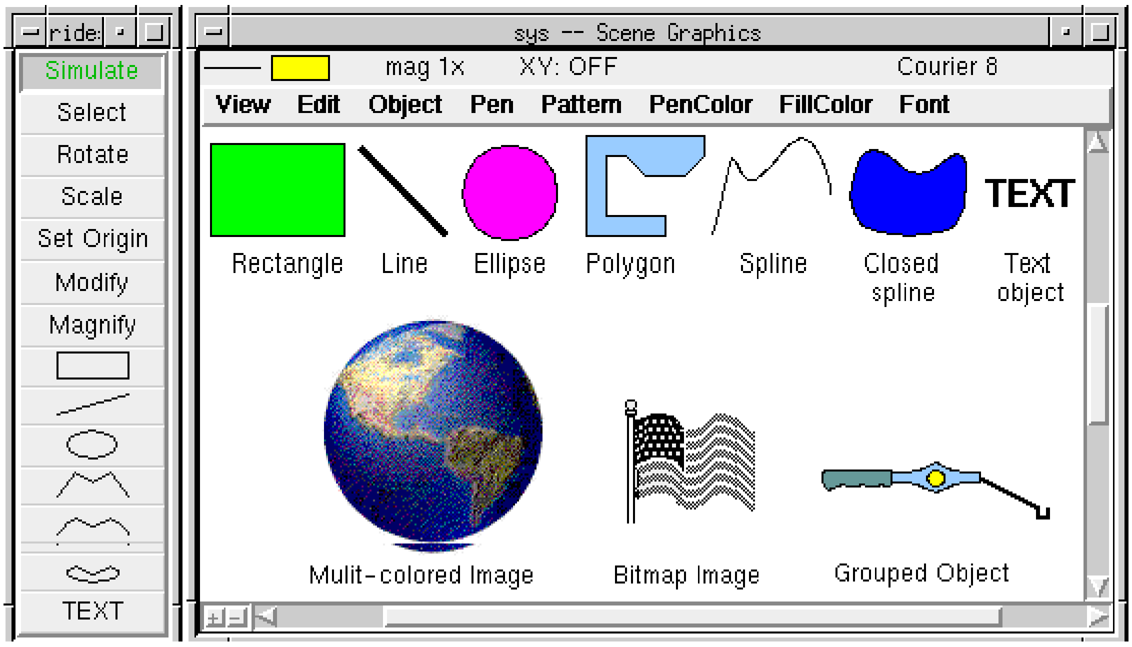


Figure 4. The RIDES Drawing Environment with Example Primitives (at top of the scene window)

Simulation authors can view an object's attributes by selecting the object and then opening its object data view, such as the one shown below in Figure 5.

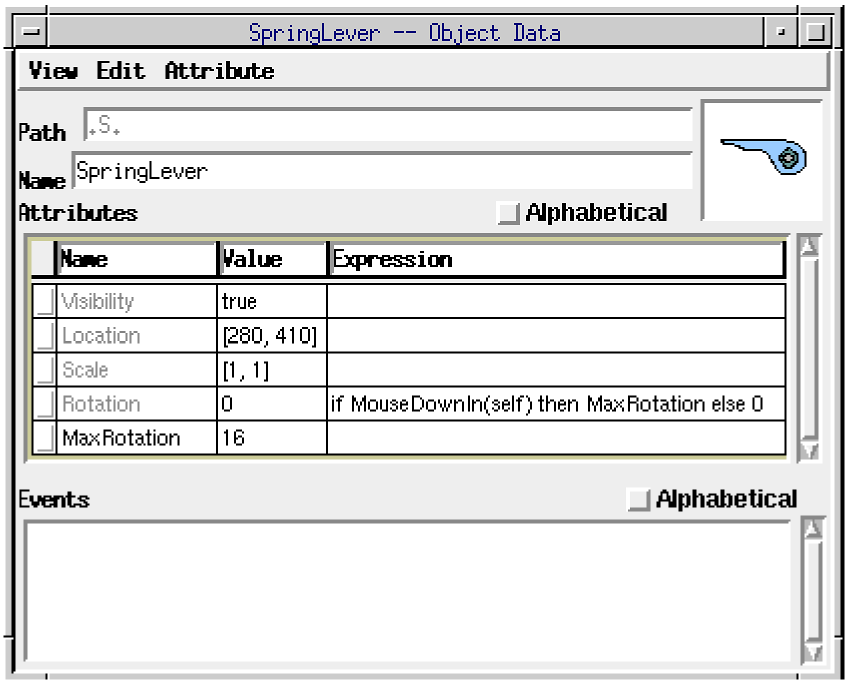
This is a grouped graphical object. (An author creates a grouped object by selecting a number of graphical objects and then issuing the Group command.) The PenColor and FillColor attributes of its component graphics can be observed by opening their object data views.

Attribute values can be changed in any of four ways: an author can manipulate the object with a tool in a way that changes an attribute value; a value can be typed into an attribute value cell in a data view; a relational constraint may change the value of the attribute; or an event statement that sets the value of the attribute may be executed.

1. *Tool Manipulation.* If an author manipulates an object with graphical tools or commands, corresponding attributes will be changed.

2. *Value Editing.* An object data view (shown in Figure 5, below) lists all the attributes of an object. Here the author can change the names of attributes and can enter new values. Changes to *intrinsic* attributes, such as Location or Rotation, have immediate graphic effects.

3. *Constraint Evaluation.* To the right of the value cell on each attribute line is a *constraint expression cell*. An author can enter a relation here that determines the value of the attribute. Figure 5, shows an object data view in which the Rotation attribute has a constraint. This constraint specifies that if the user holds the mouse button down while pointing to this object (named SpringLever) then its rotation will be 16 degrees; otherwise, its rotation will be 0.



Path .S.

Name SpringLever

Attributes

Name	Value	Expression
Visibility	true	
Location	[280, 410]	
Scale	[1, 1]	
Rotation	0	if MouseDownIn(self) then MaxRotation else 0
MaxRotation	16	

Figure 5. An Object Data View with a Constraint on the Value of the Rotation Attribute

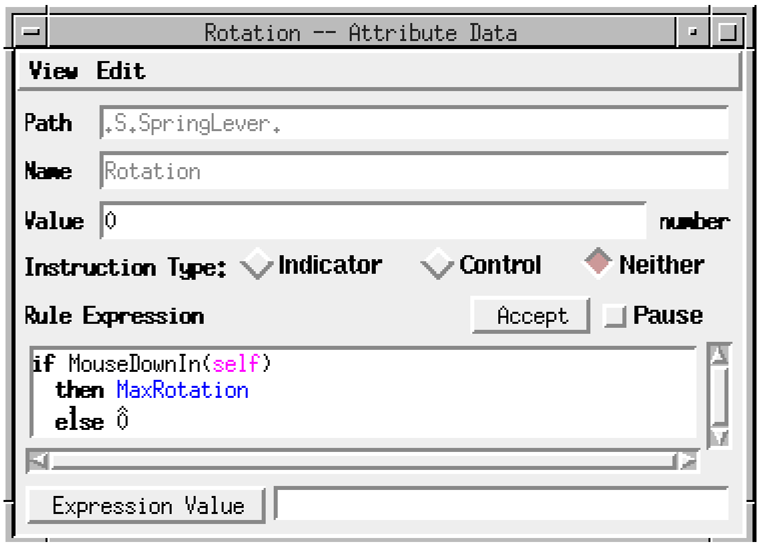
The value of attributes can be expressed by relations. For example, in

```
.S.MainSpring.Rotation * 2
```

the value of an attribute with this constraint will be twice the Rotation of the MainSpring object on scene S.

Constraints are written as expressions. Whenever any value that is referred to in a constraint expression changes, the expression is evaluated and a new value for the attribute is determined. It is not necessary to explicitly state that the expression above must be re-evaluated whenever the Rotation of the SpringLever on scene S changes. Authors need not concern themselves with when a value needs to be recomputed if that value is determined by a constraint.

There is an alternative way of viewing relational constraints in RIDES. An author can select an attribute row in an object data view, such as those shown in Figure 5, and can then open a more detailed *attribute data view*. Figure 6 shows the attribute data view for the Rotation attribute in Figure 5.



Path .S.SpringLever.
Name Rotation
Value 0 number
Instruction Type: Indicator Control [✓ Neither]
Rule Expression
Pause

```
if MouseDownIn(self)
  then MaxRotation
  else 0
```

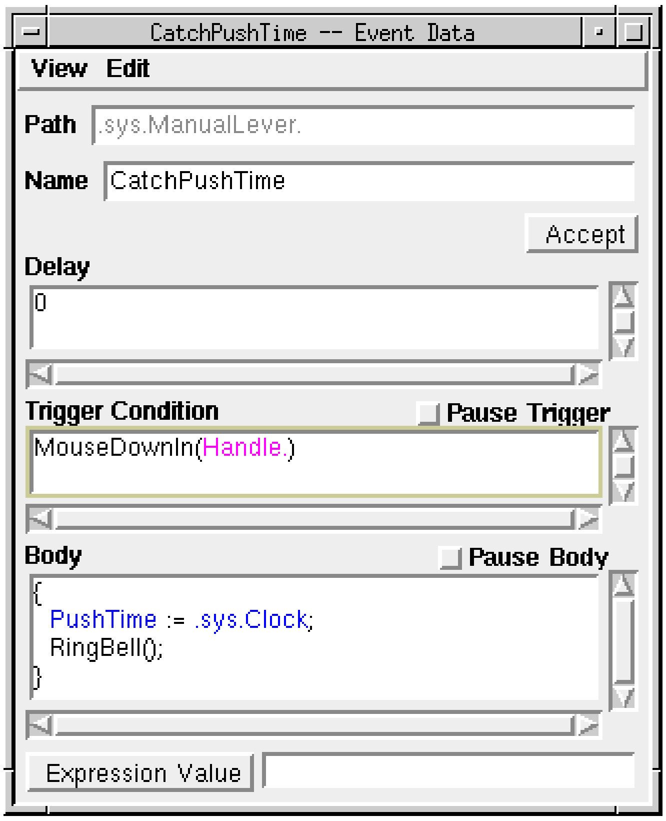
Figure 6. An Attribute Data View

This view includes a *code editor* sub-window--the frame labeled 'Rule Expression' in Figure 6. In code editor views, relational expressions are displayed with reserved words in bold and with automatic indenting. Object references (such as the *self* in this expression) are shown in a special object reference font and color. References to attributes are also shown in a distinctive color and font.

4. *Event Definition.* An alternative way of prescribing behavior is to write events that set attribute values and carry out other actions when triggered by some condition. In RIDES, an event consists of such a conditional expression, together with a delay expression, and an event body, which is a list of statements. When an event's conditional expression is evaluated and found to be true, the delay expression is evaluated to determine when the event statements should be carried out. In most cases the delay expression of an event is *0*, so the event body is carried out as soon as the event is triggered. Figure 7 shows a simple event data view.

It is possible to create events which belong to certain objects. Other events are global in nature and are simply stored in a list of unattached events.

Both types of behavior specification, constraints and events, have their uses. Constraint-based specifications promote a focus on *results* or *effects*. Event-based specifications promote a focus on *causes*. Constraint-based behavior specifications do not require expertise in controlling the flow of effects, and are therefore easier for many authors to make use of. Certain simulation effects, however, can only be achieved through event specifications.



Path .sys.ManualLever.

Name CatchPushTime

Delay

```
0
```

Trigger Condition

```
MouseDownIn(Handle.)
```

Body

```
{
  PushTime := .sys.Clock;
  RingBell();
}
```

Figure 7. An Event Data View

The Primacy of Reference. Another important element of the RIDES HCI for simulation behavior authoring is the primacy of reference.

- Correct references in code are displayed in a distinctive font and text color. Correct object references are shown in one color and correct attribute references are shown in a different color.
- Reference failures are also made self-evident in code views. Incorrect object references are shown in one distinctive font and color and incorrect attribute references are shown in the same font but with a different color.
- Objects can be selected and copied in any view; then, if a Paste action is carried out in any text editing view, an unambiguous name of the object is pasted in that context.

- If the name of an object or an attribute is changed, then any rule that referred to that object or attribute now refers to the new name.

These features were designed to help authors to better understand and edit behavior specifications in the constraints and events that they develop.

HCI for Authoring Procedure Tutorials

Authoring by Doing. RIDES provides a very simple interface for creating instructional vignettes that have one of a number of constrained structures, or patterns. Instructional units created using this interface are called patterned exercises. Figure 8 and Figure 9, below, show a patterned exercise being created. In Figure 8, the window at the left displays a simulation scene, while the window at the right shows the patterned exercise editor when the author has specified that a new exercise about operational setup (called "Procedure") should be created.

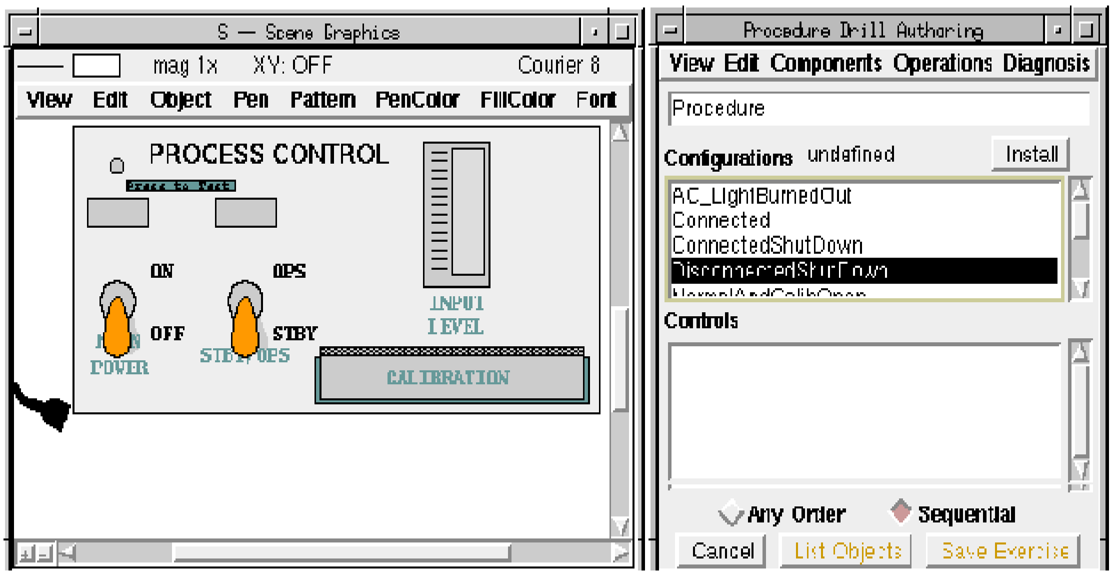


Figure 8. Beginning the Authoring of a Procedure Patterned Exercise

In Figure 9, the author has carried out several operations by clicking on the mouse-sensitive graphical objects, such as the power cord and switches. As each such control object is clicked, its name appears in the Controls list in the patterned exercise editor. At the same time that the actions are being recorded in the patterned exercise editor, they are generating graphical simulation effects in the simulation scene--switches change position, lights change colors or begin flashing, and so on. These graphical effects are brought about by the execution of previously authored simulation rules that determine the values of attributes of the graphical objects.

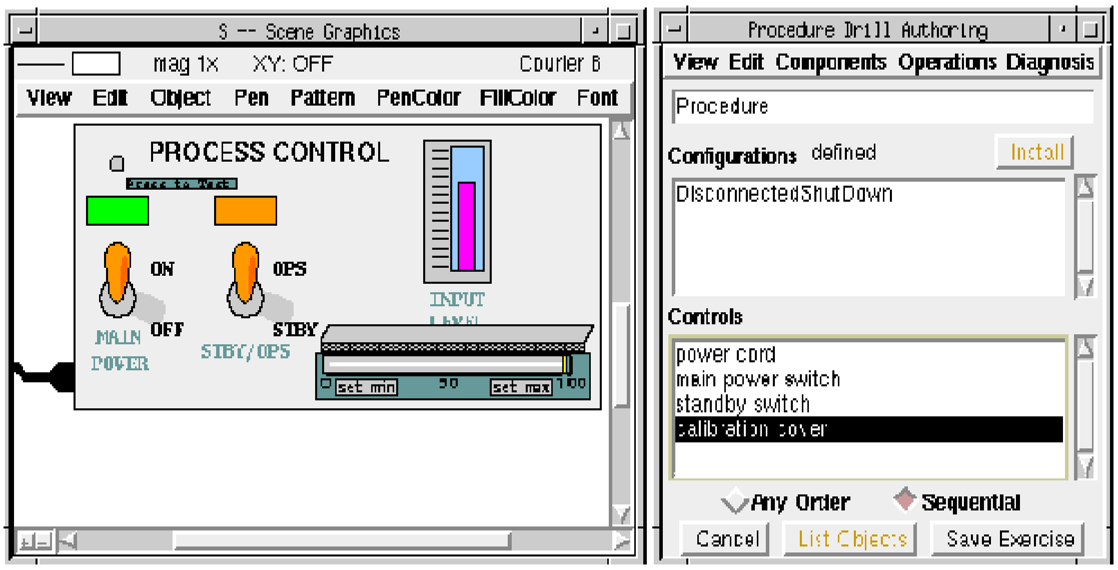


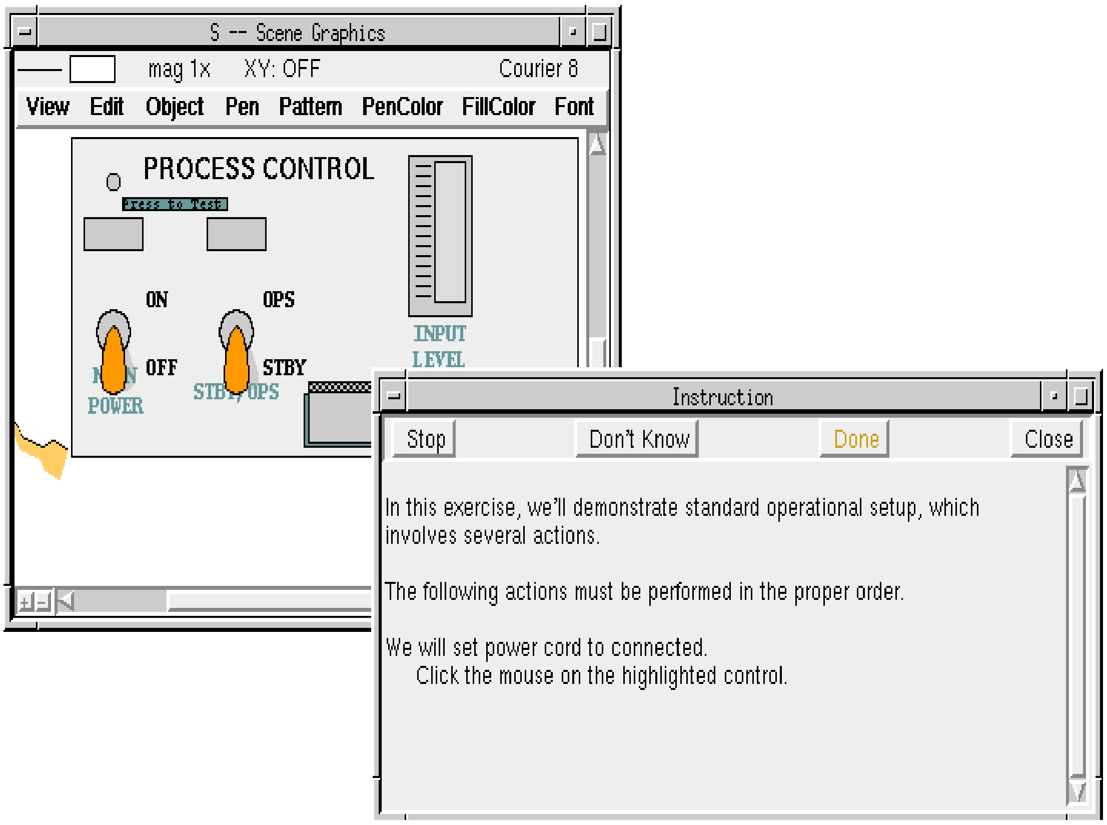
Figure 9. During the Authoring of a Procedure Patterned Exercise

By carrying out the sequence of actions that a student will be required to carry out, an author creates a *specification* of the exercise. When the author clicks the "Save Exercise" button, then a detailed instructional vignette is actually built and can be stored for later presentation to students.

Tutorial Modes. Instructional vignettes, such as the operational setup exercise (named "Procedure") specified in Figure 9, can be presented to students in any of three modes: a demonstration mode, a practice mode, and a test mode. This feature is an extension of a similar approach in our earlier RAPIDS II system (Towne and Munro, 1991). In each of these modes, essentially the same sequence of steps is undertaken, but with different levels of required student interaction and with different prompts presented to students. In the demonstration mode, each control action such as throwing a switch is taken automatically by RIDES. The student merely paces the presentation of text and actions by clicking a mouse button. See Figure 10.

In the practice mode, students are prompted to carry out certain actions. If a student fails to perform correctly, the simulation is reset to its prior state and the student is given another chance. After a specified number of attempts, the object that must be manipulated is highlighted and the student clicks the mouse to perform the action. The exercise then continues to the next required action. See Figure 11.

In the test mode of a procedure instructional vignette, students are told to carry out the next action in a sequence. If an action is not done correctly, then the correct action is carried out by RIDES and the student is told to carry out the next action. Student performance measures are recorded in the perform and test modes of instructional presentation.

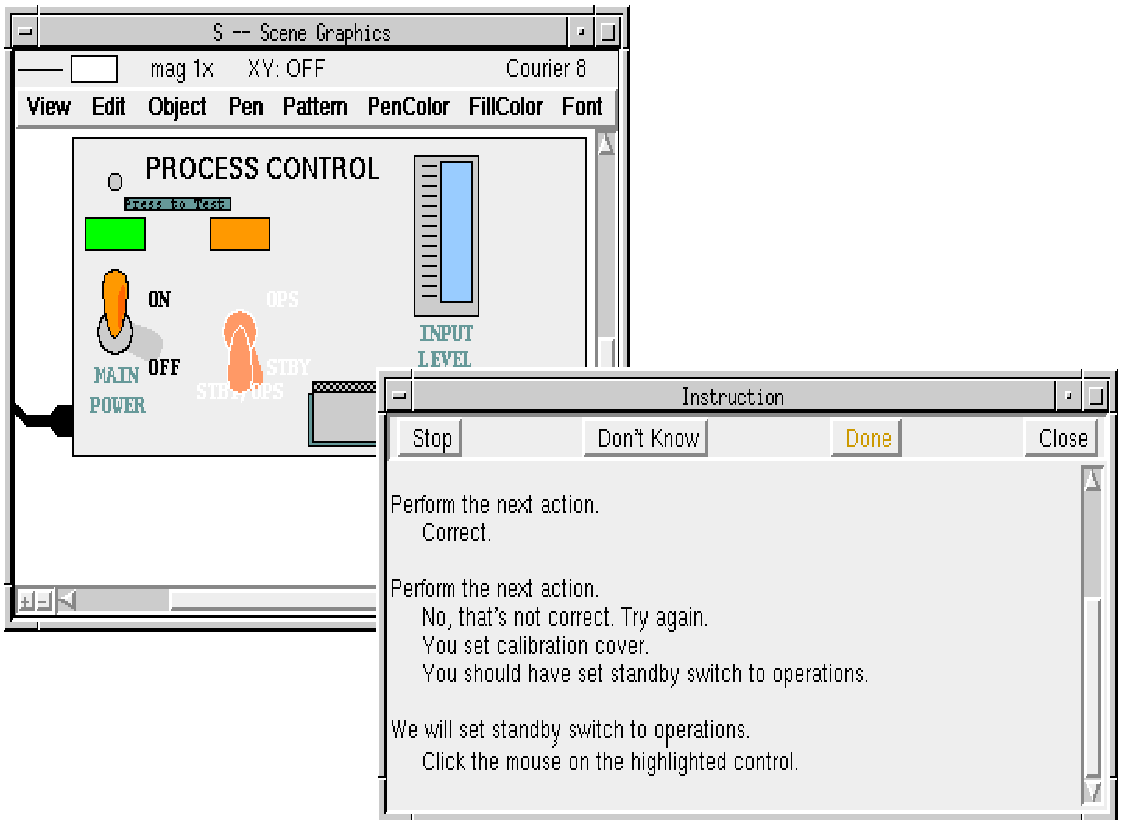


In this exercise, we'll demonstrate standard operational setup, which involves several actions.

The following actions must be performed in the proper order.

We will set power cord to connected.
    Click the mouse on the highlighted control.

Figure 10. Instructional Presentation in Demonstrate Mode



Perform the next action.
    Correct.

Perform the next action.
    No, that's not correct. Try again.
    You set calibration cover.
    You should have set standby switch to operations.

We will set standby switch to operations.
    Click the mouse on the highlighted control.

Figure 11. Automatic Student Remediation During Tutoring

Customizing Generated Tutorials. A good deal of instructional presentation can be easily generated using the procedure patterned exercise authoring interface. Exercises that can be presented in all three modes are generated in about the same amount of time that it takes to simply *do* the procedure. This is a highly productive approach to generating computer-based tutorials. In addition to being fast, the patterned exercise interface to procedure authoring has the advantage that it does not require extensive training for authors. Any subject matter expert who knows how a procedure should be performed can build an exercise. The author simply starts the procedure mode of the patterned exercise authoring tool by issuing the Procedure command from the Operations menu (see Figure 8) and then carries out the actions of the procedure.

The reason that so little instructional presentation authoring needs to be carried out is that the patterned exercise authoring tool exploits data that were created during prior simulation authoring. During the simulation authoring process, authors give names to objects. These names are used in composing text presentations to the student. In addition, if an object is one that the student can manipulate (by clicking or by dragging in it, for example), then the simulation author will mark one of its attributes as a Control attribute. The instructional presentations can then describe the state or 'value' of a control in terms of the values of the Control attribute. If the Control attribute of an on-off switch takes on the values "on" and "off" under the control of simulation rules, then the automatically generated instruction will tell the author to set the switch to "on" or to "off".

Another interface is also available for authoring instruction about procedures. This interface--which is called the *instructional unit editor*--can be used to build a complete instructional vignette from scratch, or it can be used to edit a lesson that was built using the patterned exercise editor.

Figure 12 shows the instructional unit editor being used to edit the procedure lesson that was generated in Figure 8 (using the patterned exercise authoring tool). In this view of the instructional unit, a tree structure displays the elements of the lesson. Two main types of nodes are found in an instructional tree: group nodes (the colored rectangles) and terminal nodes. The terminal nodes represent elementary instructional presentations of different types. For example, the first terminal node in the tree--the one labeled "text_explanation", at the top of the window--is a generated text presentation. This element contains the first text presentation that is shown in Figure 10, above. An instructional author can customize this instructional item. Double-clicking on this node opens a dialog in which the message can be customized, as in Figure 13, below. (In that figure, the author has already replaced canned text of the form "we will demonstrate Procedure" with "we'll demonstrate standard operational setup, which....")

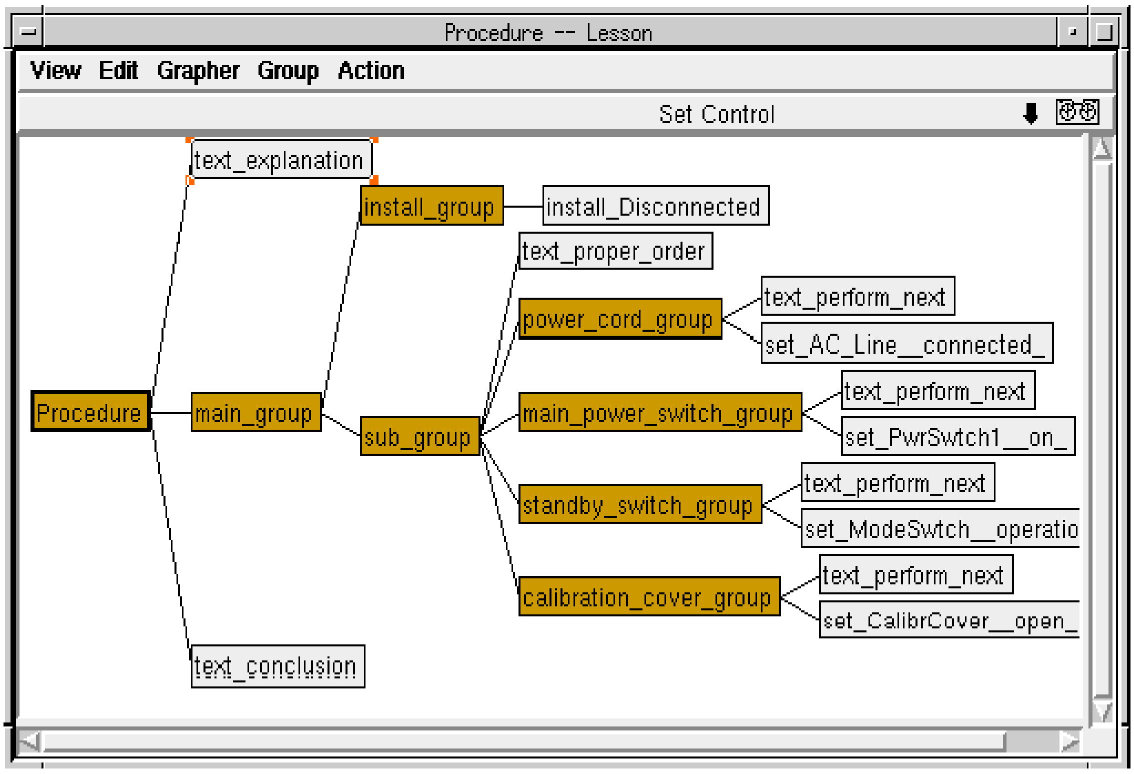
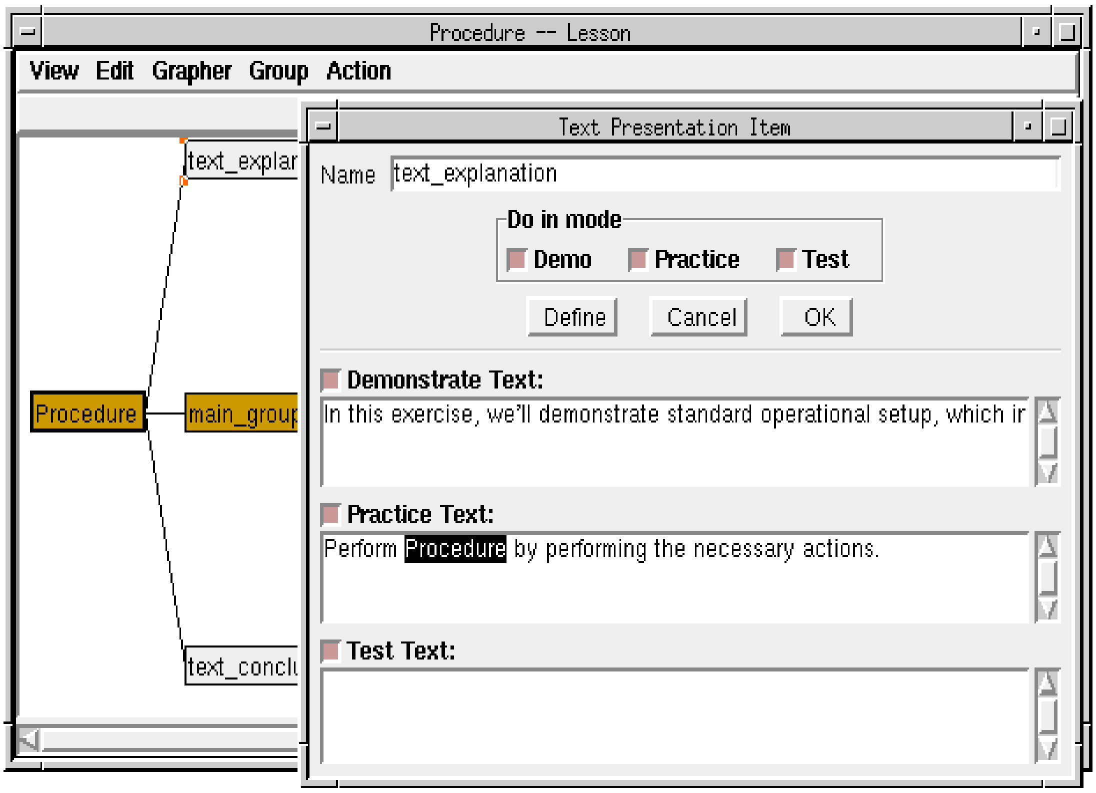


Figure 12. Viewing a Generated Lesson in the Instructional Unit Editor

Many instructional items specify more than textual presentations. The nodes that specify the control actions that a student is to take in carrying out a procedure are called Control instructional items. They, too, can be edited. Figure 14 shows a view of one of the control items in Figure 12, the one named "set_PwrSwtch1_on". In this dialog, an author can specify not only the textual prompts that should be given to students, but also what attribute value is the goal of the control item and even what object is the control.

A number of other instructional control aspects are also available to authors in these and other instructional item dialogs. These include whether and when an instructional item should time-out, how many attempts should be permitted before the answer is presented, how to weight the item's score, and whether to present the item in various lesson presentation modes. These are described fully by Munro and Pizzini (1996).

In addition to being able to edit the instructional items of a lesson, authors can delete, cut, copy, and paste these items. Their order can be altered by dragging them to new positions. Whole groups, as well as individual item nodes can be subjected to these operations.



Name text_explanation

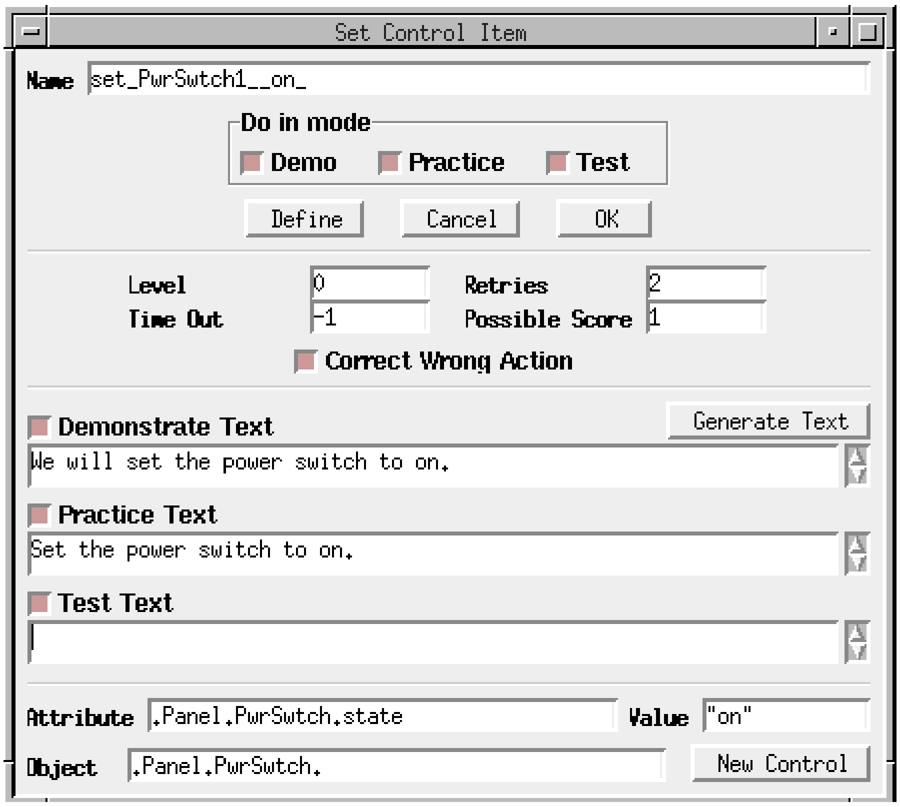
Do in mode
Demo Practice Test

Demonstrate Text:
In this exercise, we'll demonstrate standard operational setup, which ir

Practice Text:
Perform Procedure by performing the necessary actions.

Test Text:

Figure 13. Editing an Instructional Text Presentation



Name set_PwrSwtch1__on_

Do in mode
Demo Practice Test

Level	0	Retries	2
Time Out	-1	Possible Score	1

Correct Wrong Action

Demonstrate Text
We will set the power switch to on.

Practice Text
Set the power switch to on.

Test Text

Attribute .Panel.PwrSwtch.state Value "on"

Object .Panel.PwrSwtch.

Figure 14. Editing a Control Item Specification

New instructional items can be added to a lesson in the instructional item editor. Some types of items can be specified by issuing menu commands. Many of these commands automatically open dialogs such as those shown in Figure 13 and Figure 14 where authors can provide detailed item specifications. Other types of instructional items are created by choosing a 'tool' and carrying out an action (such as clicking on an object) in the simulation scenes. The instructional authoring tools are Perform Control, Read Indicator, and Find Object. If a graphical object is manipulated with the Set Control tool, then a Control instructional item is created. Clicking with the Read Indicator tool creates a type of instructional item that requires that a student tell what value an object is displaying. If the Find Object tool is used on an object, it generates an item that requires students to click on the object when its name is presented.

Just as in the case of the procedure patterned exercise editor, using the Set Control tool in the instructional unit editor can automatically create a series of steps in a procedure that students will be required to perform. As the instructional author carries out the actions, the simulation scenes update and new Control item nodes appear in the tree displayed in the instructional unit editor.

A glance at the custom authoring environment, shown in Figures 13-15 above, suggests that its use is significantly more complex than the use of the patterned exercise authoring environment, which is shown in Figure 8 and Figure 9. In fact, this is so. Instructional authors cannot make productive use of the custom instructional authoring interfaces unless they have a fairly robust understanding of the uses of the three instructional modes--demonstrate, practice, and test--and of the nature of the different types of elementary instructional presentations. (There are twenty-five such elementary instructional item types, including the text presentation type--shown in Figure 13--and the control item type--shown in Figure 14.) Full productivity with the custom authoring of certain types of items even requires an understanding of the same kinds of authored rule expressions that are used to build simulations.

In summary, one RIDES instructional authoring environment, the patterned exercise editor, requires no understanding of the structure of instructional specification. It can be used by authors with almost no technical understanding of the RIDES authoring system. A second instruction authoring environment, the instructional unit editor and its item dialogs, requires that authors understand more about the structure of simulations and simulation-based training specifications, but it offers customizable instruction authoring. In the RIDES instruction authoring environment, the tradeoff between ease of use and power has been resolved by providing two different kinds of instructional authoring tools. Because the data generated by the easy-to-use tool can be edited with the custom authoring tool, authors who choose to begin an instructional specification using the easier approach are not restricted from enhancing their specifications later in an environment that offers them more control.

INTEGRATION OF RIDES WITH OTHER RESEARCH EFFORTS

RIDES has played a major role in a number of other research efforts. Several such efforts are briefly described in this section.

REACT

In the REACT project, members of the research staff at the Jet Propulsion Laboratory collaborated with USC researchers in our lab to integrate RIDES with software developed at JPL. The JPL staff showed that RIDES could be used to carry out knowledge acquisition for complex domains (Hill, Fayyad, Santos, and Sturdevant, 1994; Hill, Chien, Smyth, Fayyad, and Santos, 1994). Their work studied the use of RIDES for knowledge acquisition about complex domains in the context of a simulation of Deep Space Network (DSN) subsystems. A special SunOS version of RIDES was developed that permits communication with the DSN monitor and control system. This made it possible to develop RIDES behavioral simulations for testing the advanced monitor and control system knowledge base.

Tactical Decision Making Simulation and Research

Using RIDES, Towne (1995) developed a system that provides a simulation of a shipboard radar that is tracking surrounding aircraft and ships. The simulation includes an authoring system for rapidly producing real-time tactical exercises. It has been utilized as a research tool capable of generating exercises and recording detailed performance data. Several features of this system specifically address learning research objectives, including the automatic generation of exercises, the automatic recording of the operator’s decisions (and their consequences) throughout a scenario, and an option for substituting a computer model for the human operator (in order to support research in machine learning).

VET

The VET project is a joint effort among the Lockheed Martin Artificial Intelligence Center, Behavioral Technology Laboratories, USC, and Information Sciences Institute. In the Virtual Environments for Training (VET) project, a version of the RIDES authoring and tutorial delivery system called VRIDES has been developed to work with collaborative applications, such as Vista, using TScript and the Vista communications bus. These extensions make it possible to use VRIDES to specify the behavior of 3D objects owned by Vista. They also make it possible for VRIDES to carry out some instruction and then to surrender control of instruction or guidance to Steve, an autonomous agent with procedural expertise in the modeled domain, as described below. Using VRIDES, we have developed a free play interactive simulation of the High Pressure Air Compressor (HPAC) on Arleigh Burke class ships. This simulation collaborates with a Vista model of the ship (and the HPAC in particular) and with Steve, an autonomous agent developed at ISI that has an understanding of HPAC procedures. A new simulation and tutorial on Gas Turbine Engine operation and inspection is now under development.

VIVIDS

The VIVIDS project seeks to extend RIDES to provide the capability to cost-effectively author and deliver interactive simulation-based instruction and training in virtual environments, collaborating with a variety of other software components. The VIVIDS authoring environment will let authors build simulations that can control the visual and interactive behaviors of modeled objects in a virtual environment. VIVIDS will also provide tools for productively authoring interactive tutorials and delivering them in virtual worlds. VIVIDS will support mixed-mode instruction, with some lessons presented using conventional 2-D graphics displays, while other lessons will require the use of 3D or even 3D-student-immersed environments. Authors will be able to create a single course that can control both kinds of lessons: 3D/Immersed and 2-D graphical.

DIAG

DIAG is a system for authoring and delivering instruction on device and system troubleshooting and maintenance (Towne, 1996, 1997). It is being developed entirely in RIDES. DIAG supports the development of a hierarchical model of a complex system. It will provide all the necessary functions for moving about in that model, for conducting tests, replacing suspected elements, and for consulting with a built-in diagnostic expert. During exercises, expert advisement is provided about the normality of indications, the significance of symptoms, and the implications of the test outcomes seen by the learner. After exercises, the learner can ask for 1) a debriefing in which DIAG points out the inferences that were possible from the learner's testing sequence, and 2) a demonstration of an expert troubleshooting strategy for the fault.

IETMs and RIDES

The Department of Defense Integrated Electronic Training Manual (IETM) specification prescribes an application of standard generalized markup language for the delivery of interactive technical manuals by computer (Fuller, Holloway, Jorgensen, and Brission, 1992; Jorgensen, 1994; Jorgensen and Bullard, 1994; Rainey and Fuller, 1994). This project seeks to integrate RIDES with markup-language-based presentation systems, such as IETMs. One outcome of this work has been the development of a facility to direct Netscape Navigator to present web pages to students, either under the control of RIDES simulation rules, or under the control of a tutorial. A second product of this work is a Java-based approach to presenting RIDES instruction in the Netscape Navigator world wide web browser. RIDES collaborates with an applet running in Netscape: the applet and Netscape together provide an instructional interface that is an improved version of the RIDES instructional text window.

FUTURE DIRECTIONS: COLLABORATING COMPONENTS

Two lessons that we have learned as a result of implementing RIDES and watching others use it to author and deliver tutors is that RIDES is too monolithic and that it is too restricted as to the platforms that can deliver authored tutors. These restrictions have, in some cases, made it difficult for colleagues in the intelligent tutoring systems research community to take advantage of aspects of RIDES that would have utility for their own research projects. For example, some researchers have expressed an interest in making use of the RIDES simulation development and delivery system, in coordination with a different tutor delivery system from the one provided by RIDES.

Our work on interacting tutor components has been influenced by continuing discussions with colleagues in the ITS community, especially at the ITS 96 workshops on *architectures and methods for designing cost-effective and reusable ITSs* and on s*imulation-based learning technology*, as well as at the workshop on *architectures for intelligent simulation-based learning* environments at AI-ED '97. Ritter and Koedinger (1996) have described a architecture for plug-in tutor agents that permits tutors to collaborate with applications that were not originally designed to provide services to tutors. Our focus is somewhat different, in that we hope to develop a set of standard services that simulations should be able to provide in order to support the widest range of useful tutoring interactions.

We envision an open architecture for simulation-centered tutors. Figure 15 shows a simplified schema for one such tutor based on five collaborating components. The components communicate and provide services to each other. (Some tutors would have more or fewer components. Only a subset of the required communication links is displayed in the figure.)

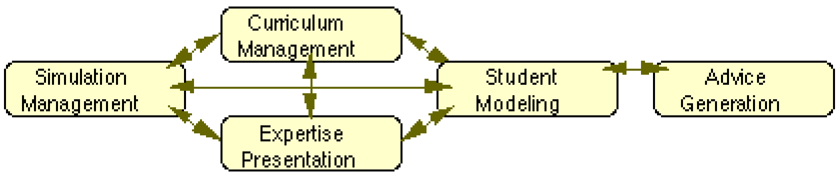


Figure 15. Collaborating Components in a Simulation Based Tutor

In this architecture, there can be many tutor components, including a simulation and interaction engine, one or more student modelers, expert advisors, procedure presenters, explainers, and so on. No one research or development group could be expected to produce leading edge versions of all these tutorial components without the use of higher-level tools. When every aspect of a tutor is developed using low-level tools by a single group, it is likely to have some strong and some weak components. This, together with the (for most practical purposes) monolithic nature of most experimental tutors, can make it very difficult to judge claims about the efficacy of the features of a particular type of tutor. The collaborating components approach would make it possible for a single simulation to be used with many other tutor components in order to compare the efficacy of a variety of tutorial approaches.

Student operations and observations. In a simulation environment, students carry out operations that change the state of the simulation. Certain changes that take place can be observed by students. In contrast with a generic simulation composition tool, one designed for use in tutorial systems will be able to report student operations and observations in a way that is useful to pedagogical components. The simulation authoring system will provide easy-to-use mechanisms for identifying simulation attributes and events that indicate that operations and observations have been conducted. Simulations built in this fashion will also be able to provide instructional interventions (at the direction of tutorial components) that require the students to make observations or to carry out operations, and they will be able to automatically generate useful textual directives or commentary.

Of course a wide range of other services--in addition to operation- and observation-based services--must also be provided by a simulation component for the other components of a tutor. These include putting the simulation into specified states, freezing and resuming simulation, making objects visually salient, hiding and displaying objects, and so on. Group learning support should be provided, so that several students can interact with the same simulation and yet receive appropriate individual tutoring.

Authoring tools and collaborating delivery components. In the collaborative components approach to tutor development, a series of authoring tools could be developed for producing the data required for each type of tutor component. See Figure 16.

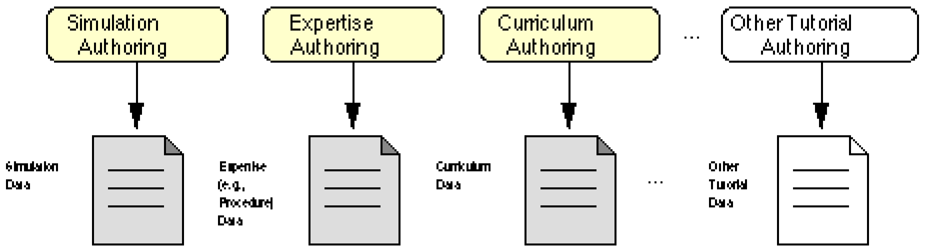


Figure 16. Authoring Tools for Tutor Components

The tools used to develop data required by simulation components may or may not be themselves multi-platform, but robust authoring tools will typically be larger executables than the multi-platform (typically Java-based) tutor components that deliver the authored simulations, expertise, curricula, etc. Figure 17 presents the notion that the authored data will be used by tutor components to deliver useful presentations or interactions to students.

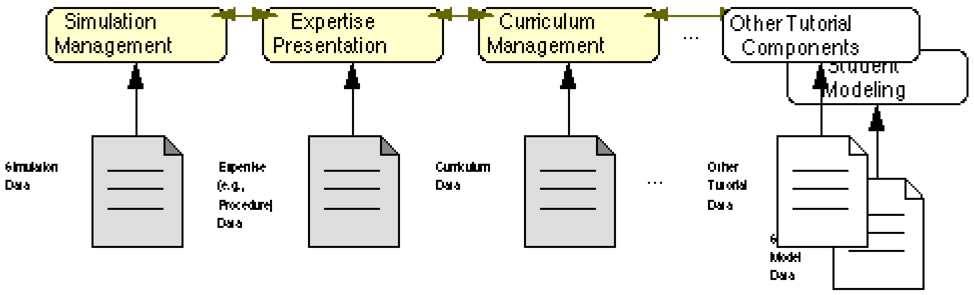


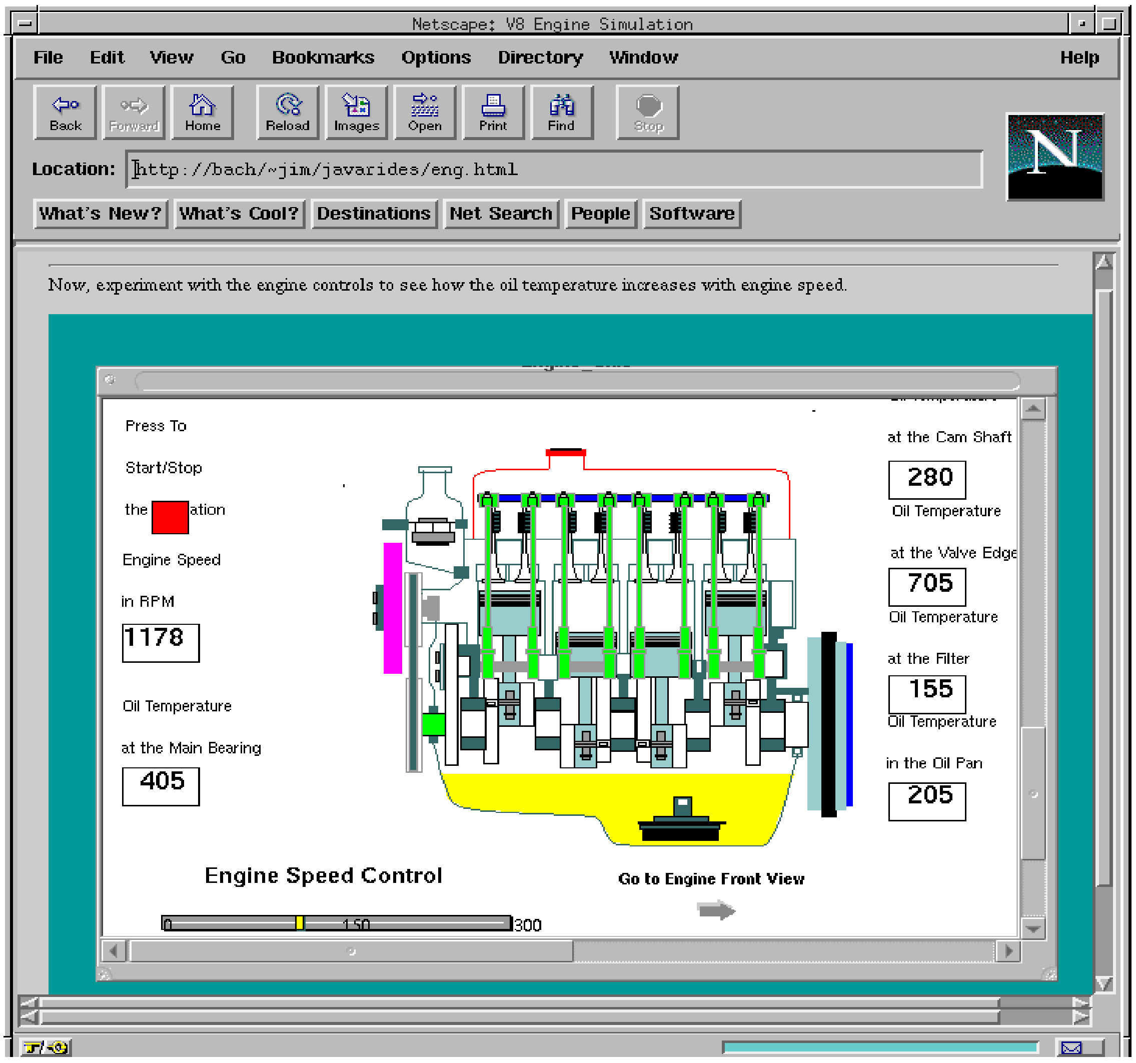
Figure 17. Tutor Components Are Responsible for Tutorial Delivery

Communications infrastructure. The current focus of our collaborating components research is what services should components be able to render to each other, rather than what communications infrastructure should be employed. Our preliminary implementations are being carried out in such a way as to support three approaches to module communication:

- CORBA
- Our own simulation tutor TCP/IP standard
- Java-to-Java function calls

Other groups of researchers, such as the IEEE P1484 working and study groups, are also working on communications infrastructure issues for tutoring systems. As new standards emerge, we expect to develop support for them. Information about the work of the P1484 group is available at http://www.manta.ieee.org/P1484/ . The work of the P1484.7 Tool/Agent Communication Working Group is described at

http://www.manta.ieee.org/P1484/sg-tp.htm .



Now, experiment with the engine controls to see how the oil temperature increases with engine speed.

Figure 18. Authored Simulation Running in Netscape Window

Multi-platform Authored Simulation Delivery. To explore the feasibility of a platform-independent collaborating components approach, we have developed an experimental Java-based simulation engine that can deliver interactive authored RIDES simulations on a wide range of platforms. Figure 18 shows a RIDES-authored simulation running in a Netscape Navigator window under the control of a Java-based RIDES player. Although the JavaRides player supports most of the RIDES simulation language, it lacks the communication features that would be required for a full tutorial component. It demonstrates the feasibility of the platform-independent delivery approach, however.

Summary

RIDES is a robust application for authoring highly interactive graphical simulations and tutorials that are delivered in the context of those simulations. RIDES supports the composition and delivery of instruction centered about a detailed model of a device or system of interest. The authoring system incorporates several innovative advances in human-computer interfaces (HCI) for tutor authoring systems. These include HCI features in support of simulation behavior specification and authoring-by-doing features for the development of instruction.

The RIDES system also provides course composition features, control over course delivery based on student model data, and support for centralized administration of courses and students.

RIDES has played an important role in several other research projects. The REACT project at JPL made use of RIDES in a project on knowledge acquisition for complex domains. The Tactical Decision Making Simulation project utilized RIDES to create a scenario authoring system in support of research on machine and human learning. The VET project (Virtual Environments for Training) has made use of RIDES to support the development of interactive behaving simulations in immersive 3D environments, and to support the delivery of RIDES-authored and autonomous-agent monitored instruction.

Ongoing and future research and development efforts are also building on the RIDES project. These include VIVIDS, a project to develop the capability to cost-effectively author and deliver interactive simulation-based tutoring in virtual environments on a variety of platforms. The DIAG project, which is developing a specialized authoring and delivery environment for troubleshooting training guided by a generic maintenance expert, makes use of RIDES as its development tool. Recent experimental developments using Java have shown the potential for creating a cross-platform tutor delivery system, with optional Internet or Intranet delivery and browser access.

In addition to many successes in HCI features and in providing low-level automatic support for simulation-to-tutor communications, the project has revealed the need for significant differences in the next-generation authoring system for simulation-based tutors. First, delivery systems should consist of communicating software components, rather than a monolithic application. This would make it possible for different types of tutorial agents, for example, to be used with a single interactive graphical simulation. Second, the simulation delivery system should be multi-platform, so that researchers and developers using a wide range of operating systems and environments can take advantage of the substantial investment made in creating an interactive simulation composition system such as RIDES.

For Further Information

Three major documents on RIDES authoring and administration are available at

http://btl.usc.edu/rides/documentn/

Many simulations and tutors have been developed using RIDES. Information about some of these tutors is available on the internet at our site:

http://btl.usc.edu/rides/examples/

and at this Armstrong Laboratory site:

http://www.brooks.af.mil/HSC/AL/HR/HRT/HRTI/icatt.htm

Acknowledgments

Major funding for this research was provided by the United States Air Force under contract F33615-90-C-0001. Additional support was provided by the Office of Naval Research under contracts N00014-93-1-1150 and F33615-90-C-0001. Our former colleague Lee Coller made design and implementation contributions to early versions of RIDES. Donna Darling has provided web page development and administrative support for the project. We thank the following people for helpful feedback based on their experiences with and/or analyses of RIDES: Susan Chipman, Albert Corbett, Michael Crumm, Zuzanna Dobes, Jim Fleming, Randall Hill, Carol Horwitz, Ken Koedinger, Len Mackie, Tim Miller, Wes Regian, Trish Santos, Chuck Swanberg, and Rusty Trainor. Many other authors and students have also made helpful comments and suggestions.

References

Coller, L. D., Pizzini, Q. A., Wogulis, J., Munro, A. & Towne, D. M. (1991) Direct manipulation authoring of instruction in a model-based graphical environment. In L. Birnbaum (Ed.), *The international conference on the learning sciences: Proceedings of the 1991 conference*, Evanston, Illinois: Association for the Advancement of Computing in Education.

de Jong, T., van Joolingen, W., Scott, D., deHoog, R., Lapied, L., Valent, R. (1994) SMILSLE: System for multimedia integrated simulation learning environments. In T. de Jong and L. Sarti (Eds.) *Design and production of multimedia and simulation based learning material*, Dordrecht: Kluwer Academic Publishers.

Forbus, K. (1984) *An interactive laboratory for teaching control system concepts*. (Tech. Report 5511). Cambridge, Massachusetts: Bolt Beranek and Newman Inc.

Fuller, J. J., Holloway, S., Jorgensen, E. L., and Brisson, J. B. (1992) *Military Specification MIL-D-87269 Data Base, Revisable: Interactive Electronic Technical Manuals, For the Support of*, MIL-D-87269 , Bethesda, Maryland: Tri-Service Working Group for Interactive Electronic Technical Manuals, 20 November 1992.

Hill, R., Chien, S., Smyth, C., Fayyad, K., and Santos, P. (1994) *Planning for deep space network operations,* Pasadena: Jet Propulsion Laboratory.

Hill, R., Fayyad, K., Santos, P., and Sturdevant, K. (1994) Knowledge acquisition and reactive planning for the deep space network. In *Working notes of the 1994 fall symposium on planning and learning: On to real applications,* New Orleans: AAAI Press.

Hollan, J. D., Hutchins, E. L., and Weitzman, L. (1984) STEAMER: An interactive inspectable simulation-based training system, *The AI Magazine,* 2.

Johnson, W. L., Rickle, J., Stiles, R. and Munro, A. (1996) Instructional agents in virtual environments. Submitted to *Presence*.

Jorgensen, E. L. (1994) *DoD Classes of Electronic Technical Manuals,* CDNSWC/TM-18-94/11, Bethesda, Maryland: Carderock Division, Naval Surface Warfare Center, April 1994.

Jorgensen, E. L., and Bullard, L. (1994) *Metafile for Interactive Documents (MID),* Bethesda, Maryland: Carderock Division, Naval Surface Warfare Center, November.

Munro, A. (1994) Authoring interactive graphical models. In T. de Jong, D. M. Towne, and H. Spada (Eds.), *The Use of Computer Models for Explication, Analysis and Experiential Learning*. Springer Verlag.

Munro, A. (1995) *RIDES QuickStart,* Los Angeles: Behavioral Technology Laboratories, University of Southern California.

Munro, A., Johnson, M. C., Pizzini, Q. A., Surmon, D. S., and Wogulis, J. L. (1996) A Tool for Building Simulation-Based Learning Environments, in *Simulation-Based Learning Technology Workshop Proceedings, ITS'96,* Montreal, Québec, Canada, June 1996.

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

Munro, A. and Pizzini, Q. A. (1996) *RIDES Reference Manual,* Los Angeles: Behavioral Technology Laboratories, University of Southern California.

Munro, A. and Surmon, D. S. (1996) *RIDES Administration Manual*, Los Angeles: Behavioral Technology Laboratories, University of Southern California.

Northrop Corporation. (1994) *Technical manual general system on-equipment maintenance: Landing gear B-2A aircraft*, TO 1B-2A-2-32GS-00-1, Los Angeles: Northrop Corporation.

Pizzini, Q. A., Munro, A., Wogulis, J. L., and Towne, D. M. (1996) The Cost-Effective Authoring of Procedural Training, in *Architectures and Methods for Designing Cost-Effective and Reusable ITSs* Workshop Proceedings, ITS'96, Montreal, Québec, Canada, June 1996.

Rainey, S. C., and Fuller, J. J. (1994) *Navy Interactive Electronic Technical Manual (IETM) Acquisition Guide, Initial Draft*, CDNSWC/TM-18-95/01, Bethesda, Maryland: Carderock Division, Naval Surface Warfare Center, October 1994.

Rigney, J. W., Towne, D. M., King, C. A., and Moran, P. J. (1978) *Field Evaluation of the Generalized Maintenance Trainer-Simulator: I. Fleet Communications System*. (Technical Report 89) Los Angeles: Behavioral Technology Laboratories, University of Southern California.

Ritter, S. and Koedinger, K.R. (1996) An architecture for plug-in tutor agents, *International Journal of Artificial Intelligence in Education*, **7**, 315-348.

Towne, D. M. (1984) A generalized model of fault-isolation performance. In *Proceedings, Artificial Intelligence in Maintenance: Joint Services Workshop*.

Towne, D. M. (1995) *ONR final report: A configurable task environment for learning research*. Los Angeles: Behavioral Technology Laboratories, University of Southern California, August 1995.

Towne, D. M. (1996) *DIAG: Diagnostic instruction and guidance--application guide*. Los Angeles: Behavioral Technology Laboratories, University of Southern California, October 1996.

Towne, D. M. (1997) Approximate Reasoning Techniques for Intelligent Diagnostic Instruction, *International Journal of Artificial Intelligence in Education*, **8**, 261-283.

Towne, D. M. & Munro, A. (1981) *Generalized maintenance trainer simulator: Development of hardware and software*. (Technical Report No. 81-9) San Diego: Navy Personnel Research and Development Center.

Towne, D. M. & Munro, A. (1984) *Preliminary design of the advanced ESAS System*. (Technical Report No. 105) Los Angeles: Behavioral Technology Laboratories, University of Southern California, December 1984.

Towne, D. M. & Munro, A. (1988) The intelligent maintenance training system. In J. Psotka, L. D. Massey, and S. A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned* (479-530). Hillsdale, NJ: Erlbaum.

Towne, D. M. & Munro, A. (1991) Simulation-based instruction of technical skills. *Human Factors*, **33**, 325-341.

Towne, D. M. & Munro, A. (1992) Two approaches to simulation composition for training. In M. Farr and J. Psotka (Eds.), *Intelligent instruction by computer: Theory and practice*. London: Taylor and Francis.

Towne, D. M., Munro, A., Johnson, M. C. (1982) *Generalized maintenance trainer simulator: Test and evaluation*. (Technical Report No. 98) Los Angeles: Behavioral Technology Laboratories, University of Southern California.

Towne, D. M., Munro, A., Pizzini, Q. A., Surmon, D. S., Coller, L. D., & Wogulis, J. L. (1990) Model-building tools for simulation-based training. *Interactive Learning Environments*, **1**, 33-50.

Spensley, F. and Elsom-Cook, M. (1989) Generating domain representations for ITS. In D. Bierman, J. Breuker, and J. Sandberg (Eds.), *The proceedings of the fourth international conference on artificial Intelligence in Education*. Amsterdam: IOS, 276-280.

Stiles, R., McCarthy, L., Munro, A., Pizzini, Q., Johnson, L., Rickel, J. (1996) *Virtual Environments for Shipboard Training*, Intelligent Ship Symposium, American Society of Naval Engineers, Pittsburgh PA Nov.

Van Joolingen, W.R. and De Jong, T. (1996) Design and implementation of simulation-based discovery environments: the SMISLE solution, *International Journal of Artificial Intelligence in Education*, **7**, 253-276.

Williams, M. D., Hollan, J. D., and Stevens, A. L. (1981) An overview of STEAMER: an advanced computer-assisted instruction system for propulsion engineering. *Behavior Research methods and Instrumentation*, **13**, 85-90.