Collaborative Authored Simulation-Centered Tutor Components

Allen Munro, David S. Surmon, Quentin A. Pizzini, and Mark C. Johnson

Behavioral Technology Laboratory
University of Southern California
250 No. Harbor Drive, Suite 309
Redondo Beach, CA 90277
munro@usc.edu

Abstract

As we have developed a number of authoring systems for simulation-centered tutorial development, each successive authoring system has achieved its original goals, and yet has been quickly shown to have limitations that motivated the next authoring system. The history of these systems reveals two emerging requirement themes. The first theme is that there is a need for more than one level of tutor development. For example, a two-level authoring system might provide an easy-to-use system for novice tutor developers and a deeper level of authoring for expert tutor developers. A second theme is that the tutor delivery system should be componential and open so that it can work with collaborating applications, some of which may not have even existed when the authoring system was originally designed. The second part of this paper describes one essential element of a componential approach to simulation-centered tutor development, the specification of simulation services that may be required by tutors.

I. Authoring Simulation-Centered Tutors: Lessons Learned

We have been involved in the development of a series of authoring tools for the development of tutors for many years (Towne, Munro, Pizzini, Surmon, Coller & Wogulis, 1990; Towne & Munro, 1991, 1992; Munro & Towne, 1994; Munro, Johnson, Surmon, & Wogulis, 1993; Munro & Pizzini, 1996; Munro, 1997). A primary motivation for building such authoring systems for developing tutors is to increase the cost-effectiveness of tutor development. Using a well-designed authoring system, an author should be able to much more rapidly develop, test, and modify a tutor than if a number of lower-level development tools, such as programming languages and expert system shells, are used. A cost effective approach will make feasible the development of a much larger number and wider variety of tutors than would be possible using more expensive techniques.

IMTS: The Intelligent Maintenance Training System (1984-1989)

This authoring system, described by Towne & Munro (1988), supported the development of interactive graphical simulations based on schematics (electrical, hydraulic, or mechanical). Authors could create behaving simulations by connecting behaving components from a parts library. Additional information about system level symptoms and causes could be entered in table form. A generic troubleshooting expert could assist a maintenance trainee in selecting tests to perform and in evaluating those tests. Figure 1 displays an example of a simple static state schematic simulation.

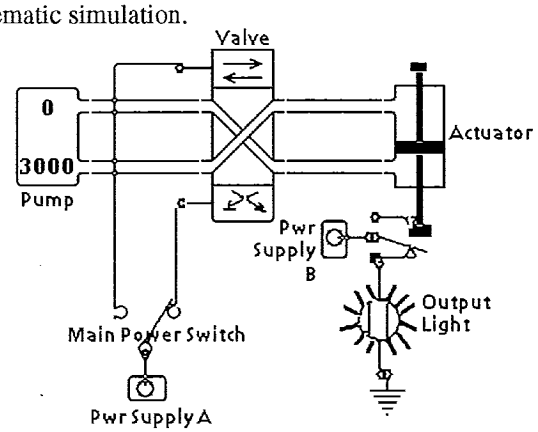

Figure 1. A simple static state simulation

In this figure, there is a switch under the control of the student. The student can observe changes in a valve, an actuator, and an output light. These are the *indicators* in the simulation. A good deal of useful instructional interaction with students can be phrased in terms of the states of *controls* (such as the switch) and such indicators. The simulation authoring system let developers draw objects in all their possible states and enter rules for transitioning among the states. A separate simulation scene editor was used to compose these authored objects into connected systems.

IMTS included an intelligent monitor of student performance called *Profile*. Profile watched students manipulating controls and noticed changes in indicators. It was designed for equipment troubleshooting (fault isolation) training. It made judgments about what malfunctions students should suspect, based on their actions and the indications they had observed. It could offer advice on the potential usefulness of tests and observations being considered by the student, and it could

advise about what should and should not be suspected, based on what had been done in the session.

Lessons Learned from IMTS. IMTS could be used to build equipment simulations of a certain class. With additional authoring about malfunction effects, it could also be used to deliver equipment troubleshooting instruction. Unfortunately, the naturalness and ease of use of the simulation authoring environment encouraged its application in domains for which it was not designed.

- *Simulation authoring systems should not be too special-purpose.*
  The ease of use of the simulation composition system generated demand from a wider authoring community than was originally envisioned. The special-purpose maintenance training focus was a negative for this larger group of authors. The range of systems that could be simulated was also found to be too restrictive. Simulations were based on object states and did not support continuous effects. Some authors needed deeper control over simulation objects and simulated systems. One way of describing the simulation authoring limitations of IMTS would be to say that it tried to be too intelligent. The authoring system itself had an understanding of hydraulic effects, electrical effects, and mechanical effects. As soon as authors tried to apply the system to simulating economic systems, chemical reactions, satellite orbits, or any other domain that did not fit into the 'equipment' orientation of IMTS, the authoring system kept authors from accomplishing what they wanted to.
- *It should be possible to author instruction, as well as to generate it at run-time.*
  IMTS also offered too little authoring control over instructional intelligence. Tutor development was fast and effective only if a particular type of maintenance tutor was wanted; no other type of tutor could be developed using the system.

RAPIDS: A Rapid Prototyping ITS Development System (1988-1990)

The lessons learned from IMTS were first applied to our RAPIDS project (Towne & Munro, 1991). The RAPIDS authoring system increased the openness and power of the simulation authoring system by letting authors build simulations with continuous value changes, as well as with state-oriented objects. It required authors to more fully specify how values were propagated in a system of connected objects, instead of trying to automatically apply physical laws to every object connection. This made it possible for authors to build systems that were outside the narrower domain of equipment simulations permitted by IMTS. Figure 2 below displays a simulation scene for a RAPIDS tutor about a helicopter blade-folding system.

RAPIDS also provided a more open instructional system. It supported a variety of simple instruction templates that authors could use to create simulation-centered tutorials. Authors would build complete graphical simulations using one set of authoring tools. Then, using a different set of instruction authoring editors, they would build lessons for delivery in the context of those simulations.

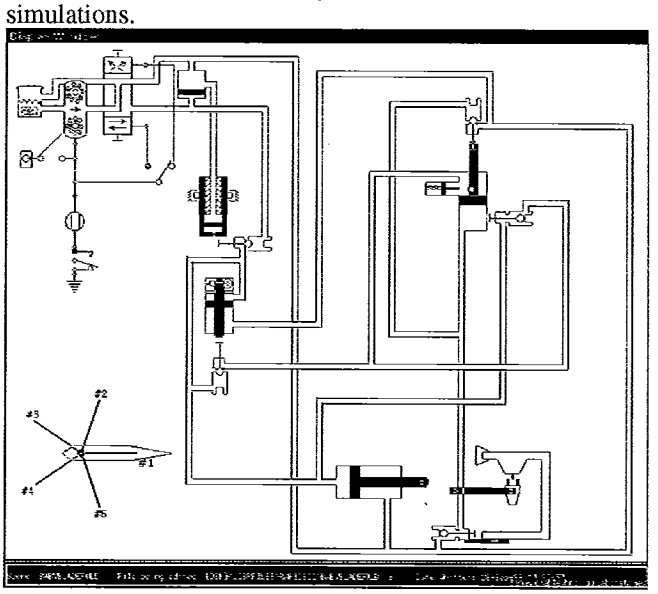

Figure 2. A schematic simulation with continuous effects

Lessons Learnedfrom RAPIDS. A lesson of the RAPIDS project was that still more control needed to be provided to authors. There was also demand for an easy-to-use simulation-centered authoring system that could be used on widely available workstations, rather than only on specialized Lisp workstations.

- *Simulation authors sometimes require very detailed control over object behavior.*
  Although the simulation approach was much more general-purpose in RAPIDS than in IMTS, some authors needed finer control over graphical features than it provided. Authors could control universal characteristics of objects like location, rotation, scale, and visibility, but they could not control the detailed, unique characteristics of most graphic primitives, such as fill pattern or color. RAPIDS still encouraged a state-oriented 'bitmap-style' representation of objects.
- *Ease of instruction authoring is good, but detailed control is necessary.*
  The instructional authoring system was easy to use, but it provided only a few types of structured lessons. Nonetheless, there was considerable interest in using RAPIDS. Authors were sometimes frustrated by their inability to make simple changes in the wording or structure of generated lessons.
- *Multiple development and delivery platforms should be supported.*
  Potential users were put off by the fact the RAPIDS, like IMTS, was available only on specialized Lisp machines.

RIDES: A Rapid ITS Development Environment (1990-1996)

The RIDES authoring system provides a much richer simulation authoring environment, one that encourages the development of continuous behaviors and that offered very fine control over the behavior of graphical objects. Simulations can control the specialized attributes of certain types of graphic primitives, including the colors and patterns of objects, the text of text primitives, and so on.

RIDES offers two levels of authoring for creating behaving objects in a simulation. The 'libraries' authoring level lets inexperienced authors select behaving objects (or groups of related objects) from libraries. See Figure 3, below. These objects can then be pasted into simulation scenes.

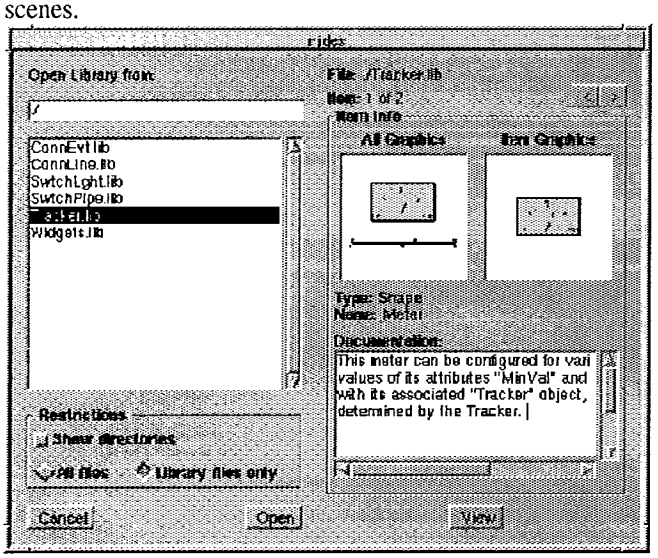

Figure 3. Selecting simulation objects from a library

New library objects can be created simply by selecting authored objects on a simulation scene and saving them in a library.

More expert simulation authors can develop novel new objects by drawing them (or by importing their graphics) and then writing rules, using the RIDES event and constraint editors, that control the behavior of those objects in RIDES simulations. See Figure 4.

RIDES also provides two levels of instruction authoring. A high-level authoring system permits subject matter experts who are not experienced instructional developers to quickly and interactively produce several different kinds of simulation-centered lessons. This approach to authoring is called *patterned exercise authoring* and is shown in Figure 5. A lower-level system lets authors who are more expert develop lessons with varied internal structures and with fine control over what is said to the students. This approach, called *custom instruction authoring*, is shown in Figure 6. In this figure, a lesson that was generated using a patterned exercise editor is being modified.

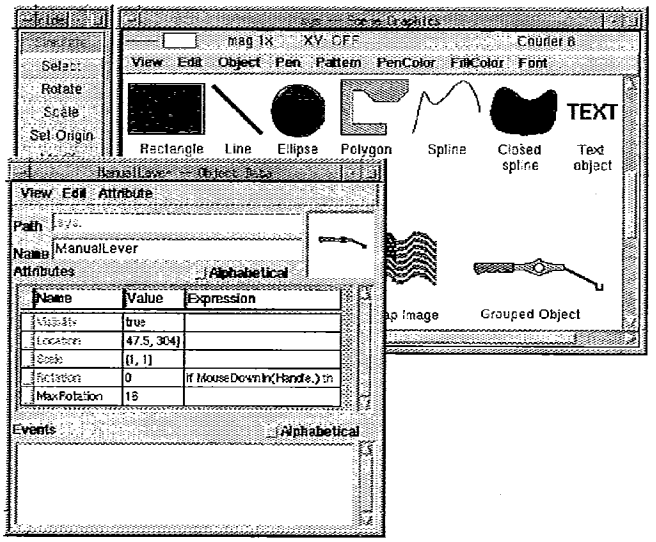

Figure 4. Custom simulation object authoring

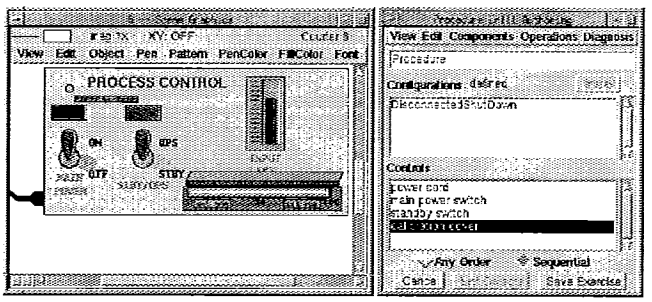

Figure 5. Authoring a patterned exercise

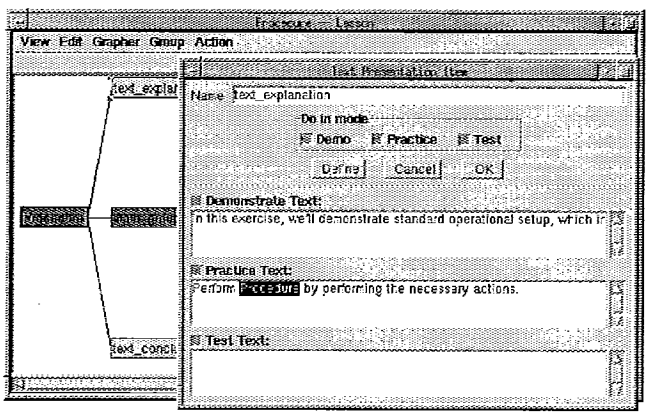

Figure 6. Modifying instruction in the custom lesson editor

As a result of applications of the lessons learned during the development of IMTS and of RAPIDS, RIDES has been widely experimented with in technical training applications. Its acceptance has undoubtedly been enhanced by the fact that it is an efficiently implemented application that runs on a variety of Unix workstations, including PCs running Linux.

Lessons Learned from RIDES. RIDES offers a much better range of development approaches that call for differing levels of expertise and offer different levels of

control than did our earlier authoring and delivery systems. In fact, the range of options for controlling simulations seems to be about right. However, still more instructional control is required by some authors. Furthermore, RIDES isn't multi-platform enough! Developers want to be able to deliver on Windows 95 and, as new platforms emerge, authors will want to deliver tutors on those platforms as well. Finally, RIDES could be more open and more componential, so that it could be more widely used by our research colleagues.

- *Instruction authors need as much power and flexibility as do simulation authors.*
  Even the *custom* instruction authoring in RIDES is still somewhat too constrained. It needs to be more flexible, in order to give advanced developers more control over the structure and flow of tutorials. There should be a deep level of representation of instruction that provides the power and flexibility of a programming language. Many casual authors would never use this deep level of instruction, but it should be available for those who need it.
- *More platforms should be supported.*
  In addition to Unix workstations, many developers have requested support for Windows NT and Windows 95—and, ideally, future platforms as well. At least it should be possible to deliver simulations and simulation-centered tutoring on virtually any platform, even if the authoring system is not universally available.
- *An open architecture of collaborative components.*
  Not every feature of the RIDES simulation-centered tutorial development and delivery system is required for every project to which it has been applied. In particular, some of our research colleagues at other institutions have wanted to use portions of the RIDES system, such as its simulation engine or its course administration system, while substituting for other portions of the system their own applications. An architecture of collaborative components—some providing simulation, others instructional or course management—defined with open communication standards, would facilitate the integration of our work with that of our colleagues.

Summary of Lessons Learned

Support two or more levels of authoring. We have learned that widespread acceptance of an authoring system requires that each of the major aspects of the system should support at least two levels of authoring: a very easy-to-use interface that can be employed with little training to productively generate useful tutorials, and a very flexible and powerful system for tutorial elaboration and editing by well-motivated expert developers. Our latest authoring environment for simulation-centered tutor development, VIVIDS, will support two levels of simulation authoring, similar to those of RIDES. Similarly, VIVIDS will support two levels of tutorial authoring: a high level specification that is very easy to generate for certain classes of predefined tutorial structures, and a detailed tutorial scripting level, which will be more powerful than that provided in RIDES.

Make the authoring system extensible. In VIVIDS, authors can add newly defined behaving simulation objects to the libraries, where they can be utilized by others. We plan to provide a mechanism for allowing advanced tutor developers to create their own templates for rapid instruction authoring using high-level specifications.

Make the delivery system open and componential. Many colleagues have expressed an interest in using only a portion of our tutor delivery system, e.g., only our simulations, in conjunction with their own tutorial components. Our current VIVIDS delivery system is a monolithic application, but we are architecting a new version that will support much greater openness. Our next generation VIVIDS delivery system, which we call JavaVivids, will consist of a confederation of collaborating applications, including a simulation engine, tutorial modules, a course control system, and a student modeling system. Our research partners and other advanced developers will be able to replace one or more of these components with their own collaboration-compliant applications.

Support platform independent delivery of tutors. JavaVivids will be implemented in Java (a high-speed optimized simulation engine component may be available for certain platforms), so that the same tutor delivery system can be used on any platform that supports Java.

II. Collaborating Components

The most complete and capable of the authoring systems developed at our laboratory are RIDES (the Rapid ITS Development System) and its immediate descendent, VIVIDS. The key concepts in these systems are

- Instruction in an interactive graphical simulation environment
- Direct manipulation authoring of simulations
- Facilities for authoring content-based help
- A highly productive approach to authoring structured lessons in the context of interactive simulations
- Courses organized around learning objectives

Many interactive graphical simulation-centered tutors have been built using RIDES and VIVIDS, both in our laboratory and at other sites. Most of these are centered around 2D simulations, such as those shown in Figure 7, below.

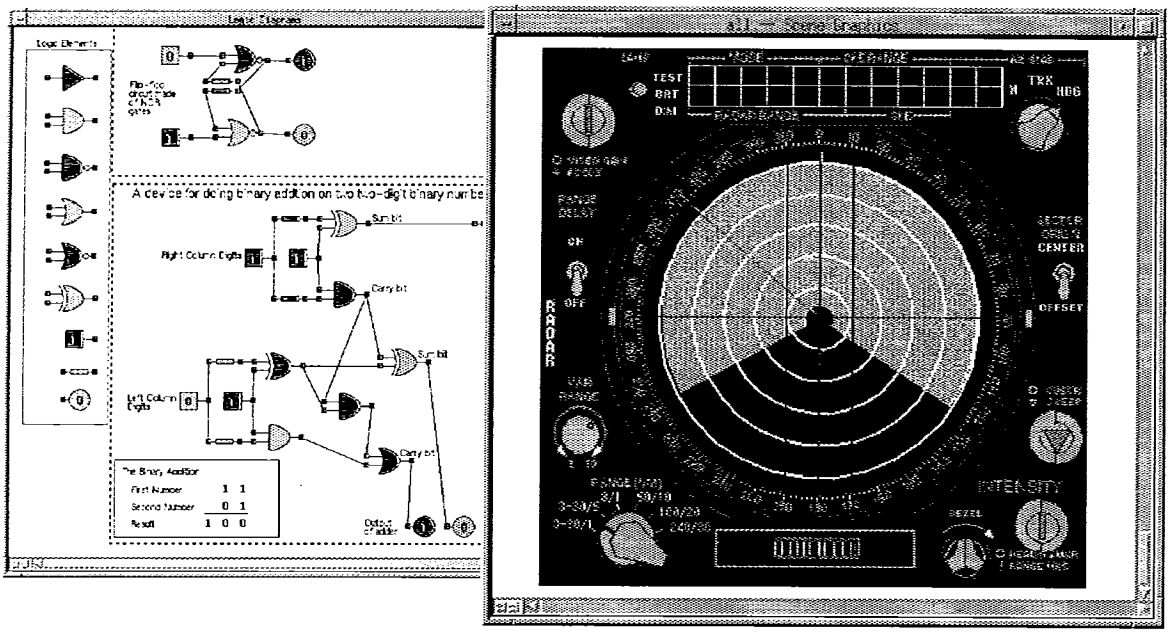

Figure 7. Two 2D simulations built in RIDES

Several substantial 3D simulations have been built using VIVIDS, including the one shown in Figure 8.

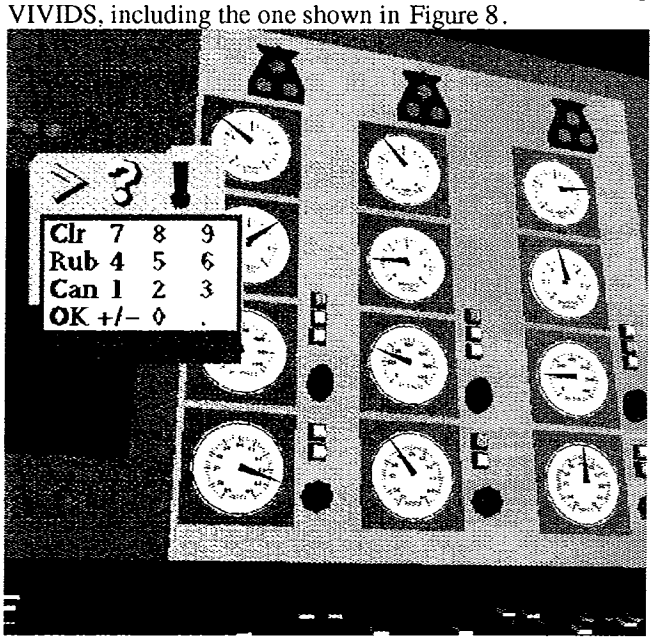

Figure 8. A 3D simulation Built with VIVIDS

RIDES and VIVIDS 1 are monolithic development-and-delivery environments that provide a wide range of user interface features in support of simulation development, tutorial development, and course development. These features include the following:

Simulation Authoring Features

- Behavior specified by Constraints
  - Advantages: Modularity, locality of effect, cause tracing, un-authored flow of control
- Behavior specified by Events
- Library Objects
  - Both graphics and behavior
- Links to content based help, hypertext-like browsing
- Feedback about reference in behavior authoring editors
- Special reference typography, error highlighting, pasting objects as references where appropriate, name changes automatically update in referenced contexts
- Multiple levels of Undo and Redo
- Transparency of Effects
  - Appearance changes can be accomplished by editing object data in interactive editors, or by using graphical tools. The effects of the graphical tools are reflected in the object data editors.
- Smart Copy and Paste
  - Parallel behaviors are maintained when a multiple selection set is copied and pasted.
- Find tool
  - Supports searching for named objects and attributes, searching in constraints or events, searching among attribute values.
- Simulation debugger
  - Supports step-by-step execution, tracing, breaks

Instruction Authoring Features

- Build certain types of tutorials by demonstrating procedures in the simulation
- Modify and customize such generated tutorials in an interactive lesson editor
- Robust linkage between simulation and instruction
- Simulation name changes automatically reflected in instruction, etc.
- Objectives-based course design
- Trace lesson execution
- Log student actions
- Student performance reporting over network

VIVIDS-1 Collaboration in the VET Project

We have been conducting a project in collaboration with the Lockheed Artificial Intelligence Center and Information Sciences Institute to explore the potential of virtual environments for technical training—VET (Stiles, McCarthy, Munro, Pizzini, Johnson, & Rickel, 1996; Johnson, Rickel, Stiles & Munro, in press).

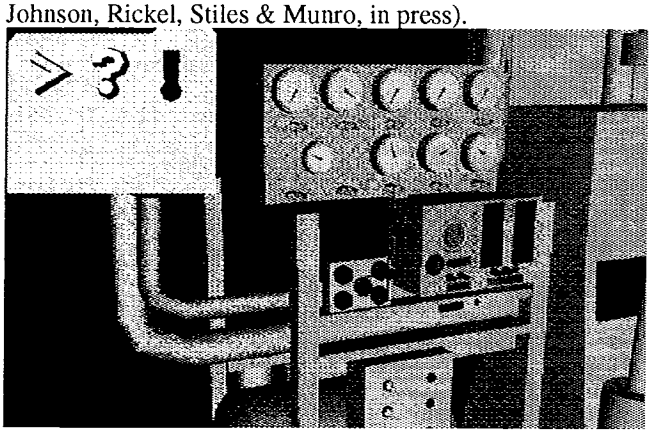

Figure 9. Virtual environment with VIVIDS instructional interface

The VET project makes use of three major collaborating software applications: VIVIDS—from our lab, Vista—a 3D interactive environment from Lockheed Martin research group headed by Randy Stiles, and Steve—an autonomous agent developed by Lewis Johnson and Jeff Rickel at Information Sciences Institute.

The success of this project has increased our interest in developing a more open approach to tutor components. In this brief paper, we focus on a single question: What are the core low-level services that an instruction module should be able to request of a simulation module?

Open Architecture Tutors

A successful architecture for simulation-centered learning environments must be extremely robust and very open. It is very important that the architecture itself not be closely wedded to one particular theory or approach to tutoring in the context of simulations, because we must expect that our theories will evolve and improve as we conduct research and as we observe the tools that we develop in use. In fact, and ideal architecture will encourage the development of a variety of different instructional approaches in the context of interactive simulations.

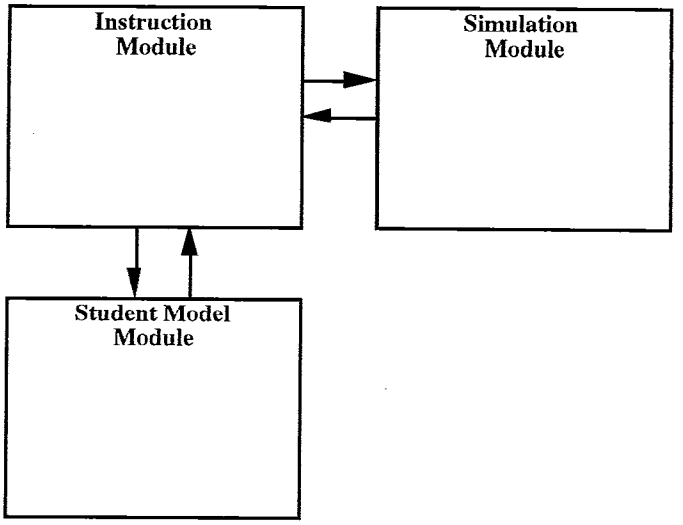

Figure 10. Oversimplified approach to tutor modules

Figure 10 presents a conventional simple high-level architecture for a tutoring system. In this much-oversimplified figure, we assume that pedagogically relevant domain knowledge can be found in the component labeled 'Instruction'.

As we have wrestled with the design and implementation of a real-world approach to these modules, we have found it useful to focus on widely required primitive instructional services for simulation-based tutors.

To make the instructional services problem tractable, we envision an instruction system in which there are low-level instructional services (such as presenting a piece of text to a student or requiring that a student touch or click on a simulation object), and higher level instructional modules, such as one that presents previously scripted lessons or one that monitors complex student problem-solving activities and offers advice and correction. The high-level instructional modules express themselves by calling on low-level instructional services. Some of these low-level instructional services may have nothing to do with the simulation module. For example, instructional text may be presented in a window that is not under the control of the simulation module, or such text may be presented using a text-to-speech or pre-recorded voice system. Figure 11 sketches a portion of this type of system.

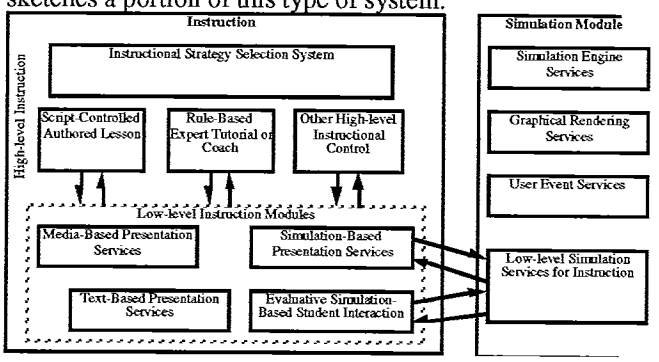

Figure 11. More detailed approach to tutor modules

Below we lay out preliminary lists of

- Simulation-based *presentation* services to be provided by a low-level instruction module, and
- Evaluative simulation-based student interaction services that are to be provided by low-level instruction in collaboration with the simulation.

We are attempting to identify the primitive vocabulary of simulation-centered tutorial events. We ask for feedback from our colleagues in the simulation-centered tutoring community. Are the proposed sets of low-level services adequate for the requirements of the high-level tutorial modules that they plan to develop in the future?

In what follows, we have organized a list of proposed low-level student interaction services for tutors into four groups.

1 Non-simulation presentation services

2 Non-simulation student judging services

3 Simulation presentation and monitoring services

4 Simulation judging services

The first two groups are not of special concern for this paper, and are not meant as an exhaustive list. Our focus is on the third and fourth sets of services, which relate to authored simulations.

1 Presentation services (not simulator-based)
These services are non-evaluative. The instructional component may provide these services directly, or may request them from one or more *presentation* or *media* service providers. These components are designed to collaborate in a complete tutor delivery system.

- openTextPresenter
- closeTextPresenter
- repositionTextPresenter
- clearTextPresenter
- presentText
- presentURL
- presentMPEG
- presentAVI
- ...

2 Student judging services (not simulator-based)

These services require an action of the student, and must evaluate the quality of the student's response, but they are not dependent on the presence of a simulation.

- Get menu answer
- Get keypad answer
- Get typed answer
- Get spoken answer

Of special interest to us, given our focus on simulation-centered tutorials, are the low-level services that a tutor may reasonably expect a collaborating simulation to provide.

3 Simulation-based presentation and monitoring services

These services are non-evaluative. The instructional component can simply request the service and expect a confirmation that the simulation has carried it out.

- *openSimulation*—A simulation produced by an authoring system will typically be saved as data in a file that can be opened by a collaborating *simulation player*. Hard-coded executable simulations would lack the ability to open different simulations.
- *closeSimulation*—When a simulation is closed, any visible characteristics of the simulation (e.g., open windows) should disappear.
- *openScene*—Not all simulations will support the delivery of multiple views (in, for example, different windows). Those that do, however, should offer the instruction module the service of opening a requested scene.
- *closeScene*—If a simulation can open scenes, it should also be able to close them at the request of the tutor.
- *saveConfiguration*—Some simulations, such as those we have developed, have a mechanism for capturing the complete *state* of the simulation and restoring that state upon request. This is important for many complex simulations, because the tutor wants the system to be in a particular state before it begins to discuss features of the simulated world that may depend upon the state. Some tutors need to capture a state of the simulation before letting the student work with the simulation. That way, if the student fails to carry out an assigned task, the tutor can restore the state of the simulation before remediating the student.
- *installConfiguration*—Restore a previously saved state of the simulation.
- *startSimulator*—It can be very useful to freeze and unfreeze a simulation during the course of a tutorial.
- *stopSimulator*
- *lockOutStudent*—Sometimes a tutor wants simulation effects to continue (time to pass, animated effects to take place, etc.) while denying the student the ability to carry out any manipulations in the student environment.
- *setSimulationClock*—Some simulations may exhibit behaviors that are dependent on the simulated time.
- *setClockMultiplier*—It is often useful to be able to compress or expand the time scale, so that students can be shown processes that are too fast (or too slow) to be observed ordinarily.
- *executeEvent*—Some simulations have defined events (such as the appearance of an enemy on a radar screen or the start of a coolant leak in a nuclear power plant) that the tutor may want to invoke under conditions of its own choosing.
- *setAttribute*—The concept of setting a simulation attribute to a particular value is similar to that of invoking an event, but it gives a tutor more intimate control over the characteristics of a simulation, because events must be pre-authored to be available in the simulation.

- *getAttributeValue*—A tutor can ask the simulation for an attribute value so that it can accurately refer to or discuss that value in the current simulation state. It can also make teaching decisions based on features of the simulation state that arises as a result of student manipulations.
- *getExpressionValue*—This service allows a tutor to send the simulation any arbitrary expression in the format supported by the simulation and to receive back the current value of the expression. This allows the tutor to refer to values that may not be explicitly computed in the simulation (e.g., "The current output of the five generating units is 1.21 gigawatts.").
- *addExpressionListener*—This service allows a tutor to announce to a simulation that it would like to be informed whenever the value of a particular expression changes. A common use of this service is to monitor some aspects of the state of a simulation after directing the student to achieve a particular goal in the simulated world. When the expression of interest becomes true, the student has succeeded.
- *removeExpressionListener*—Lets the tutor unregister a previously expressed interest that is no longer needed in the tutorial.
- *highlightObject*—A simulation should offer the tutor some mechanism for making graphical components of a simulation visually salient. In RIDES and VIVIDS, highlighting is accomplished by rapidly alternating the colors of the object or objects that are to be highlighted.
- *unhighlightObject*—The tutor also needs some way to make objects stop being so visually salient. A useful variant form of this service is to *unhighlightObject All*, which restores all the graphics that are currently being highlighted to their normal visual appearance, as prescribed by the current state of the simulation.
- *addObjectListener*—A tutor can ask a simulation to tell it whenever a student touches a simulated object. This service can be used in combination with the lockOutStudent service to let the student point out objects without manipulating them. It can also be used to monitor object manipulations.
- *removeObjectListener*—Lets the tutor unregister a previously expressed interest in the student's object touches.
- *carryOutUserAction*—This option lets the tutor "reach over the student's shoulder" to carry out a manipulation in the simulated environment. This service is very useful for demonstrating to students how to carry out procedures. It is also necessary for remediating erroneous actions in some simulation-centered tutors.

4 Simulation-based student judging services

These services may not actually be services of the simulation itself, because they can be composed from the low-level services described under **3**, above. However, they are very commonly utilized in tutorials developed using the RIDES and VIVIDS authoring systems, and are likely to often prove very useful as low-level simulation-centered instruction services. Here are examples of these services in action.

- requireSetControl

This low-level instruction service employs a number of the low-level simulation services described above to require that a student carry out a step in a procedure in the simulation environment. It judges each student manipulation, and resets the simulation to its prior state when incorrect manipulations are carried out. After an author-specified number of unsuccessful attempts, the correct object to manipulate is visually highlighted, and the action is carried out for the student..

- requireReadIndicator

Students are required to touch the named indicator and then to tell what value is displayed by the indicator. (In RIDES and VIVIDS simulations, numeric values are entered using a graphical keypad, while other values are entered using a popup menu.) If the student fails, automatic remediation points out the indicator and says what value it displays.

- requireFindObject

The FindObject instructional service only requires that a student touch a named object. It offers remediation through highlighting automatically after an author-determined number of attempts..

- requireStateAttainment

In RIDES and VIVIDS, students can be required to achieve a particular state of the simulation. For example, the tutor can specify that the student must ensure that certain operational indicators are displaying their normal values. At the author's discretion, the tutorial can notice as soon as the student succeeds and continue with the lesson, or the tutorial can require that the student notify that the state has been achieved (by clicking the *Continue* button). The tutorial can point out to the student what aspects of the required goal have not been achieved. If the student fails to succeed in achieving the designated state, a previously recorded procedure for doing so can be demonstrated.

A Call for Comments

The list of simulation services in **3** above is a candidate list of universal low-level services that tutors might expect simulations to provide. What is missing? What additional simulation services do your tutors require? What additional low-level services do your simulations provide for tutors? Please address comments and suggestions for service list modifications to `munro@rcf.usc.edu`

Acknowledgments

The RIDES and VIVIDS projects are sponsored by the United States Air Force under Contract F33615-90-C-0001. Jim Fleming of Armstrong Laboratory serves as technical program manager. The developers of RIDES and VIVIDS include Mark Johnson, Allen Munro, Quentin Pizzini, David Surmon, Douglas Towne, James Wogulis, and Lee Coller. Design suggestions, exhaustive testing, encouragement, and support were provided by Michael Crumm, Donna Darling, Zuzanna Dobes, David Feldon, Randall Hill, Carol Horwitz, Len Mackie, Wes Regian, Trish Santos, Chuck Swanberg, Rusty Trainor, Josh Walker, and others.

The VET project is sponsored by the Office of Naval Research under Contract No. N00014-95-C-0179 awarded to Lockheed Martin Co. This work is being performed in collaboration with colleagues at the Lockheed Martin AI Center—Randy Stiles, Laura McCarthy, and Sandeep Tewari—and with colleagues at USC Information Sciences Institute—including Lewis Johnson, Jeff Rickel, and Rich Angros.

References

Horwitz, C., Fleming, J.L., and Munro, A, *Demonstration of RIDES: An Authoring Shell for Simulation-Based Instruction* in the Systems Demonstrations Handbook, ITS'96, Montreal, Québec, Canada, June 1996.

Johnson, W. L., Rickle, J., Stiles, R. and Munro, A. Instructional agents in virtual environments. To appear in *Presence*, in press.

Munro, A. 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, 1994.

Munro, A. *RIDES QuickStart*, Los Angeles: Behavioral Technology Laboratories, University of Southern California, 1997.

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

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

Munro, A. & Towne, D. M. Productivity tools for simulation centered training development, *Educational Technology Research and Development*, 1994.

Pizzini, Q.A., Munro, A., Wogulis, J.L., and Towne, D.M., 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.

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

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

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

Towne, D. M. & Munro, A. Supporting diverse instructional strategies in a simulation-oriented training environment. In J. W. Regian and V. J. Shute (Eds.), *Cognitive approaches to automated instruction*. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., 1992.

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