Self-Explanatory Simulators for Middle-School Science Education: A Progress Report

Kenneth D. Forbus
Qualitative Reasoning Group
The Institute for the Learning Sciences
Northwestern University
1890 Maple Avenue
Evanston, IL, 60201, USA

Abstract

This essay describes a project in progress that is applying qualitative physics to create new kinds of educational software. We first outline why qualitative physics is an important technology for science education. One architecture we are developing, *active illustrations*, is described next. The rest of the paper describes our progress towards making this software available in schools.

Introduction

Creating new kinds of educational software has been one motivation for qualitative physics. Our research has brought us to the stage where we are now creating such software, and focusing some of our efforts on investigating how its educational benefits can be optimized. This essay describes one architecture of the three that we are developing: The incorporation of self-explanatory simulators into *active illustrations*, systems that provide an environment for guided experimentation. We start by examining why qualitative physics is useful for science education, and then describe the active illustrations architecture. We then discuss some of the issues that have arisen in moving our software from laboratory to classroom, and our plans for deployment.

Why Qualitative Physics is Useful for Science Education

There are many motivations for research in qualitative physics, ranging from improving our understanding of human cognition to making new kinds of software systems for various applications. While there are many important applications for qualitative physics, we believe that educational software, particularly for science education, constitutes an extremely important application area, for two reasons:

Qualitative physics represents the right kinds of knowledge. Much of what is learned about science in elementary, middle, and high school consists of causal theories of physical phenomena: What happens, when does it happen, what affects it, and what does it affect. The representational issues that are the central concern of qualitative physics are exactly those which must be addressed by domain content providers for science education software.

Qualitative physics represents the right level of knowledge. The prevailing attitude about engineering education is that it must be heavily mathematical. Whether or not one agrees that that is how it should be, it is impossible to make such claims about pre-college science education. Students learn calculus, at best, at the end of high school, and algebra at the beginning of high school. Making students memorize differential equations in the guise of teaching them science simply isn't an option, and even formal algebraic models are not feasible for elementary and middle school students. The qualitative mathematics developed in qualitative physics provides the right level of language for expressing relationships between continuous properties for pre-college science students.

We see qualitative physics as essential for creating a new generation of educational software and activities for science education. We believe such software should have the following properties:

- It should be *articulate*. The software should have some understanding of the subject being taught, and be able to communicate both its results and reasoning processes to students in comprehensible forms.
- It should be *supportive*. It should include a mentoring component consisting of coaches and tutors that scaffold students appropriately, taking care of routine and unenlightening subtasks and helping students learn how to approach and solve problems.
- It should be *generative*. Students and instructors should be able to pose new questions and problems, rather than just selecting from a small pre-stored set of choices.
- It should be *customizable*. Instructors must be able to modify, update, and extend the libraries of phenomena, designs, and domain theories used by the software, without needing sophisticated programming skills. This simplifies maintenance and provides scalability.

The Active Illustrations Architecture

The power of illustrative examples is well-known in education. Traditional media offer high authenticity but low interactivity. Textbook illustrations and posters can provide thought-provoking pictures, tables, charts, and other depictions of complex information. Movies and video can provide gripping dynamical displays. But none of these media provide interaction. A student intrigued by a picture of a steam engine in a textbook (or a movie of a steam engine) cannot vary the load or change the working fluid to see what will happen. They cannot ask for more details about explanations that they don't understand. They cannot satisfy their curiosity about how efficiency varies with operating temperatures by testing the engine over ranges of values. The *Active Illustrations* architecture uses AI techniques to provide such interactive services. An active illustration can be thought of as a hands-on museum exhibit, consisting of a virtual artifact or system, and a guide who is knowledgeable about the exhibit and enthusiastically helps you satisfy your curiosity about it. Active illustrations support student explorations, by allowing students to change parameters and relationships to see what happens. They are articulate, in that students can ask why some outcome occurred or some value holds, and receive understandable explanations that ultimately ground out in fundamental physical principles and laws.

For dynamical systems, an active illustration can be viewed as a self-explanatory simulator (Forbus & Falkenhainer, 1990, 1995; Iwasaki & Low, 1992; Amador, Finkelstein, & Weld, 1993) combined with *visualization tools*, an *explanation presentation system*, and a *coach*. The visualization tools provide graphing and report generation services. The explanation presentation system filters and translates the conceptual and mathematical explanations provided by the self-explanatory simulator into terms appropriate for the student. The coach helps the student set up and interpret the simulation results, using the explanation system of the simulator as its domain knowledge and using an explicit task model of student activities (e.g., predicting a value, constructing a causal theory).

The domain we are using to explore this architecture is middle-school Earth Science, focusing on the processes that underlie the weather. We are building a sequence of active illustrations, starting with laboratories for exploring fundamental processes such as evaporation and phase changes, and expanding to system-level models of the hydrological cycle and the atmosphere. These systems will be field-tested in the DODEA school system, a collection of schools run by the US military for the children of personnel stationed abroad, under the auspices of DARPA's Computer-Aided Education and Training Initiative (CAETI).

Example: The Evaporation Laboratory

Suppose a student is interested in how evaporation works. Since evaporation happens in everyday circumstances, the student begins to set up different jars of water, varying in width and amount of water, and measures their initial level. The student places these jars on the window ledge in the classroom, and looks for something else to do while waiting for the outcome of the experiment. Seeing a free machine, the student starts up a simulation lab on evaporation, to try to gain some insights in minutes instead of days.

(It should be noted that we are not arguing that simulated laboratories can or should substitute completely for physical labs. We believe that each has its role in education. Without physical labs, simulated labs seem pointless: Physical labs provide an environment where an instructor can link abstract concepts to their manifestation in the physical world. Experiments in the physical world ground classroom learning. But on the other hand, physical experiments can be expensive, time-consuming, and dangerous. Simulation labs enable students to experiment with ranges of phenomena that they simply couldn't in physical labs. Moreover, simulation labs offer opportunities for students to receive additional instruction from software-based coaches, which can help reinforce what happens in the classroom and support distance learning.)

The student's interaction with the simulation laboratory starts with setting up a scenario. The student selects, from an on-screen catalog, a cup to use in an experiment. The cups are all the same shape and size, but they are made from a variety of materials, ranging from Styrofoam to tin to titanium and even diamond. The student chooses a styrofoam cup, since such cups are common in their environment. From another catalog, the student selects an environment to place the cup in. Since it is hot outside, the student selects Chicago in the summer, and sets the simulator to run for four hours of virtual time. A few moments later, the simulation is finished. The student notices, by plotting how the level of water in the cup changes over time, that there is a slow but measurable decline. Using the explanation system, the student finds the following summary of the behavior:

```
Between 0.0 and 14400.0 seconds:
  evaporation from Cup occurs
  flow of heat from Atmosphere to water in Cup
       occurs
  there is water in liquid form in Cup
  water in Cup touches the atmosphere
```

The student follows up by using the hypertext facilities of the explanation system:

```
In Styrofoam cup in Chicago,
mass of water in Cup can be affected by:
  water loss via evaporation from Cup
In Styrofoam cup in Chicago,
```

```
water loss via evaporation from Cup can be
affected by:
  vapor pressure of Atmosphere
  saturation pressure of Atmosphere
  surface area of water in Cup
  temperature of water in Cup
```

At this point the student conjectures that higher temperature should lead to more evaporation. To confirm this conjecture, the student runs a second simulation, using a diamond cup this time to increase the flow of heat from the atmosphere. Qualitatively the behavior is the same, but the higher thermal conductivity of diamond means that the temperature of the diamond cup is closer to ambient, and indeed leads to increased evaporation (Figure 1).

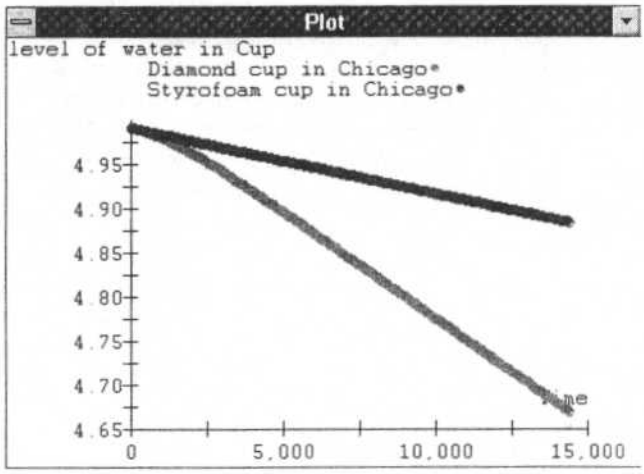

Figure 1: Comparison of evaporation from Styrofoam and diamond cups, summertime in Chicago

The student might continue their explorations by deciding to see what happens with the same cup on the top of a mountain, where it would be very cold, or in the tropics, where the temperature could be adjusted to be the same as on the desert, but with a much higher relative humidity. These explorations can be accomplished in minutes, with reports produced for further comparison and reflection.

From Laboratory to Classroom

Anyone who has tried to transform research software into a fielded application knows that many of the issues that arise bear little relation to the issues that motivated the research. This project is no exception.

One issue we are exploring is the different settings in which this architecture can be instantiated. The most obvious is stand-alone software, where our software is the only software the student is interacting with and it is running directly on the student's machine. However, there are two other settings for this architecture that are worth exploring, because of their potential impact on education and training. The first setting is embedding active illustrations in other kinds of software. Active illustrations are a natural type of media for hypermedia systems, for example. Similarly, when combined with knowledge of operating procedures and appropriate peripherals, active illustrations could provide more intelligent training simulators. The second setting is integrating active illustrations into shared virtual environments, to create *learning spaces* that students can use to interact with both the software and other students, who may be distributed in different locales. Because interaction is computer-mediated, such spaces provide additional opportunities for software-based coaching and assessment of student progress. Such learning spaces could also facilitate distance learning, since they enable the formation of communities of interest that are less limited by geographical boundaries. The creation of such learning spaces is an important goal of the CAETI community.

The need to explore these different settings raises some interesting software engineering issues. Our solution is to produce self-explanatory simulator runtime libraries which can be incorporated in different kinds of software. The first library is written in C++, in the form of a Windows dynamic linked library (DLL), for producing small-footprint systems. The second library is written in Common Lisp, to maximize flexibility and to provide cross-platform portability. Our SIMGEN Mk3 compiler (Forbus & Falkenhainer, 1995) produces Common Lisp simulators that can be used directly with the Lisp runtime library, and also produces C++ source code for a simulator, again in the form of a Windows DLL for easy embedding.

We have created two "shells" which use these libraries. The first is a GUI-based shell, written in C++, which provides a small memory footprint stand-alone system. The second is a client/server shell, with transactions mediated via MCP[1], so that the student/simulator interactions can be broadcast through a MUD, and thus available for coaching, tutoring, and evaluation software.

In both shells so far, the visualization capabilities we provide are simple plotting routines. This appears adequate for our target tasks. The explanation presentation system has a set of fixed policies, based on the needs of middle-school students. We heavily filter access to the information available in the self-explanatory simulator's explanation system by limiting the types of queries allowed. For instance, the only questions one can ask about a parameter are what can affect it and what can it affect. This filtering is important to avoid distracting or confusing students – while the explanation system can produce the set of differential equations that held at any point during a simulation, showing this information to students at that age would generally be a mistake. Instead,

[1] MUD Communications Protocol, a message protocol for software which needs to interoperate with MUDs. See http://jhm.ccs.neu.edu:7043/help/subject!mcp for details.

we focus on the kind of causal information that they are supposed to be learning. The answers to questions about parameters, for instance, concern the existence of influences, e.g. "`X can be affected by...`" in the example dialog. While the explanation system knows the type and sign of the influence, this information is suppressed because it is something that the student should be learning, along with the relative magnitudes of various effects.

Coaching raises a variety of issues. An important problem is creating explicit models of the learning tasks that students might engage in with this architecture, so that a software coach can recognize what they are doing and provide assistance as appropriate. For instance, three kinds of experiments that a student might perform using a simulation are

- *What happens experiment:* Look at the process structure in the simulation and see how it evolves over time. (Example: "What happens to water in a diamond cup in Las Vegas?")
- *Factor relevance experiment:* Run the simulator to create a baseline history. Then vary one factor, without varying others. Run the simulator again with the new set of initial conditions, to create a new history. Compare the baseline and the new history to assess the impact of the change. (Example: "How does air temperature affect evaporation?")
- *Achieve result experiment:* Figure out the difference between a baseline history and the desired history. Use the causal account so far of the phenomena to figure out which parameters can be perturbed in order to bring this change about. Explore the space of such changes to find a set of perturbations from the baseline initial state so that the simulator generates a history with the desired properties. (Example: "How can I make water in Chicago evaporate faster than in Las Vegas?")

Explicit representations of these and other types of experiments will provide the information needed for a coach to communicate with a student about their goals, and to gauge the student's activities against their stated goals. We are collaborating with other members of the CAETI community to create coaches, exploiting the "opening up" of the simulator/interface link provided by the client/server model to simplify experimentation.

Another issue which arises in fielding simulators to be used by students is the problem of initialization. Providing a large menu of numerical and logical parameters, even in the cleanest, well-organized GUI, can easily lead to bewilderment. We simplify this process by using a metaphor from drama – the idea of a *prop*. A prop on a stage represents something in the imagined world being created on-stage. In our simulators, props represent a coherent subset of the simulator's parameters that naturally make sense to consider together. Each simulator has a set of catalogs, each catalog containing props that impose different constraints on a particular subset of the simulator's parameters. In the Evaporation Laboratory, for instance, there are two catalogs of props, cups and environments. The choice of cup constrains the shape and dimensions of the cup, as well as its thermal conductivity (e.g., the thermal conductivity of diamond is orders of magnitude higher than just about anything else). The choice of environment constrains the temperature and pressure and vapor pressure of the atmosphere, as well as the limits over which these parameters can be varied. (While it is possible in theory for Las Vegas to get colder than the top of Mt. Everest, it would be very surprising, and providing constraints that prevent two props from being identical in the simulator helps maintain the suspension of disbelief.) In addition to solving the technical problem of setting up a simulation, props should also provide pedagogical benefits, by helping the student see the mapping between physical objects and circumstances and their properties. It also provides a simple path to customization: Adding props representing familiar objects and situations (e.g., a student's favorite cup or their home town) also provides a simple form of customization that can make software more engaging.

Finally, there are still interesting qualitative physics issues that must be addressed. First, the need to make robust self-explanatory simulators that operate over a wide variety of conditions has required richer domain theories, and we expect this enrichment to continue. Second, in the process of creating domain theories, the constraints of flexibility and explanatory simplicity are often in opposition. Ideally the domain modeler should ignore explanatory simplicity and focus on flexibility, and the explanation presentation system would take care of suppressing unnecessary detail. In practice this can be difficult. Finally, the model formulation algorithm we are using currently is very simple, and cannot handle automatic selection of temporal grain size or spatial resolution. Creating such algorithms will require improvements in compositional modeling, along the promising directions explored by Rickel (Rickel & Porter, 1994) and by the KSL group (Levy, Iwasaki, & Fikes, 1995).

Deployment and Other Plans

The Evaporation Laboratory was demonstrated at a CAETI meeting in March 1996. The client/server version of the software was used, with communications routed to a MUD so that evaluation agents and other software could listen in. The stand-alone version of the Evaporation Laboratory is being delivered to the CAETI testing facilities in April, as part of the process of migrating the software to DODEA test schools. If all goes well, the stand-alone system will be in use as of September 1996.

In addition to extending our domain theory and creating new simulators, there are two other activities we are conducting in the development of this architecture:

TUI software bots. Today's MUDs are almost entirely text-based. Limitations in communications bandwidth over most of the planet, and limited education budgets, means that many MUD-based learning spaces will continue to be text-based for years to come. Consequently, providing a *Text-based User Interface (TUI)* for active illustrations, in the form of a MUD-based software robot, can open up this technology to an even broader audience. We also suspect that text-based bots could provide some advantages that complement those available with GUIs. First, text provides higher-resolution imagery than graphics, assuming that the student's imagination is engaged. Second, the metaphors for interaction with bots are different: the bot can take on the role of an assistant, carrying out experiments in a virtual environment, or as a purveyor of phenomena, helping to spark a student's interest. We are creating such a softbot, using Sibun's Salix natural language generation techniques to provide explanations. A key issue in creating a useful softbot is handling discourse well enough in a MUD-based environment to keep students engaged.

Authoring Environment for Active Illustrations. To make this technology widely available requires getting ourselves out of the loop in creating new active illustrations. Our goal is to make an authoring environment that is friendly enough that curriculum developers and experienced teachers can create new active illustrations, using off-the-shelf domain theories. If we succeed, it could fundamentally change the economics of using simulation in science education, and hopefully help improve science education substantially.

Acknowledgments

This work is sponsored by Advanced Research Projects Agency of the US Department of Defense, under the Computer Education and Training Initiative. The basic research on self-explanatory simulators and compositional modeling is supported by the Computer Science Division of the Office of Naval Research. Aaron Thomason is responsible for C++ programming, Penelope Sibun is responsible for explanation generation and softbot design.

References

Amador, F., Finkelstein, A., & Weld, D. (1993). Real Time Self-Explanatory Simulation. Proceedings of AAAI-93. 562-567.

Forbus, K., & Falkenhainer, B. (1990). Self Explanatory Simulations: An integration of qualitative and quantitative knowledge. Proceedings of AAAI-90. 380-387.

Forbus, K., & Falkenhainer, B. (1995). Scaling up Self-Explanatory Simulators: Polynomial-time compilation. Proceedings of IJCAI-95. 1798-1805.

Iwasaki, Y. & Low, C.M. (1992). Device Modeling Environment: An Integrated Model Formulation and Simulation Environment for Continuous and Discreet Phenomena. Proceedings of CISE-92.

Levy, A., Iwasaki, Y., Fikes, R. (1995) Automated Model Selection based on Relevance Reasoning. KSL Technical Report KSL-95-76. Stanford University, November, 1995.

Rickel J., & Porter, B. (1994). Automated Modeling for Answering Prediction Questions: Selecting the Time Scale and System Boundary. Proceedings of AAAI-94. 1191-1198.

Sibun, P. (1992) Generating Text without Trees. *Computational Intelligence* 8(1) 102-122.