Articulate Software for Science and Engineering Education

Kenneth D. Forbus
Computer Science Department/School of Education and Social Policy
Northwestern University
1890 Maple Avenue
Evanston, IL, 60201, USA
Email: forbus@nwu.edu

To appear in: Forbus, K., Feltovich, P, and Canas, A. *Smart Machines in Education: The coming revolution in educational technology.* AAAI Press.

1 Introduction

Improving science and engineering education is a critical problem for technological societies, who, in addition to needing scientists, engineers, and technicians, need a scientifically literate population in order to make wise decisions. We believe a new kind of educational software, *articulate software*, can help solve this problem. Articulate software understands the domain being learned in human-like ways, and can provide explanations and coaching to help learners master it. Articulate software is made possible by advances in artificial intelligence, particularly in *qualitative physics*, combined with the ongoing revolution in computer technology. This chapter explores the ideas underlying articulate software and describes two architectures for articulate software that we have developed:

- *Articulate virtual laboratories* (AVLs) help students learn by engaging them in conceptual design tasks. We illustrate this architecture with two examples: CyclePad, an AVL for engineering thermodynamics which is now routinely used at a number of universities worldwide, and FAVL, which is designed to help students understand feedback systems.
- *Active Illustrations* provide an interactive simulation medium that enables students to experiment with physical phenomena, providing conceptual explanations as well as traditional simulator outputs. We illustrate this architecture with several examples, ranging from stand-alone simulation laboratories (e.g., Evaporation Laboratory, Mars Colony Ecosystem) to game-style simulations embedded in explanatory hypertexts (i.e., the Principles of Operations manual for a virtual space probe).

We describe the scientific and pedagogical principles underlying these architectures, and summarize some of the lessons we have learned by building and deploying them. We end with some suggestions for what is needed to bring these architectures into widespread use.

2 Qualitative Physics and Articulate Software

Creating new kinds of educational software has been one motivation for qualitative physics since its inception (cf. Forbus & Stevens, 1981; Brown, Burton & de Kleer, 1982; Hollan, Hutchins, & Weitzman, 1984). There are two reasons why qualitative physics is particularly appropriate for application to science and engineering education. The first is that *qualitative physics represents the right kinds of knowledge.* Much of what is taught in 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. Consider the water cycle, a key topic in middle-school science curricula. Understanding this cycle requires understanding the kinds of forms that water can be in (e.g., liquid water, water vapor, snow and ice), the sorts of places it can be (in bodies of liquid water, underground, in the air), and the processes that transform and move it from place to place (e.g., flows, evaporation, condensation, freezing, etc.). Traditional mathematical and computer modeling languages do not attempt to formalize such notions because they are designed for expert humans who already know such things. For example, the conceptual understanding that a simulation designer used to create a simulator typically resides at best in the program's documentation, and at worst only in the designer's mind. For many

purposes this opacity is fine; however, the lack of tight coupling between concepts and their software embodiment makes it difficult for most educational software to explain its results.

On the other hand, uncovering how we think about physical entities and processes is one of the central scientific goals of qualitative physics. Progress in qualitative physics has led to new modeling languages that describe entities and processes in conceptual terms, embody natural notions of causality, and express knowledge about the modeling process itself (cf. Forbus, 1984, Weld & de Kleer 1990; Falkenhainer & Forbus, 1991; Forbus 1996). These languages provide new capabilities for domain content providers of science education software. By embedding human-like models of entities and processes in software, the software's understanding can be used to provide explanations that are directly coupled to how specific results were derived. These explanations can delve into topics that traditional software cannot handle, e.g., why a process was considered to occur or why a specific approximation makes sense. Figure 1 illustrates how qualitative physics encodes some sample everyday concepts.

What a student sees	**What the software knows**
"Evaporation of water from the cup"	`(evaporation PI2)`
"Temperature of the water depends on its heat."	`(qprop (temp water6)` `(heat water6))`

"Heat flow occurs when two things are touching and their temperatures are unequal. Heat goes from the hotter one to the cooler one."	`(defprocess heat-flow`
	`:participants`
	`src a thermal-object`
	`dst a thermal-object`
	`:conditions`
	`(> (T src) (T dst))`
	`:consequences`
	`(I+ (heat dst) (hf self)`
	`(I- (heat src) (hf self))`

Figure 1: Qualitative reasoning provides formal languages for conceptual knowledge

The second reason that qualitative physics is particularly apt for science and engineering education is that *qualitative physics represents the right level of knowledge.* We believe that the tendency for engineering education to be highly mathematical at the expense of qualitative understanding is counterproductive. Rare is the instructor who does not lament that students memorize formulae without understanding basic principles. Indeed, cognitive scientists have extensively documented the existence of persistent misconceptions that survive college training in domains such as physics (cf. Gentner & Stevens, 1983). We believe that a strong quantitative sense of the world is crucial for engineering. However, we believe that principles governing a domain (i.e., the laws, mechanisms, and causal relationships) need to be mastered at the qualitative level to provide the kind of deep, robust understanding that engineering education seeks to impart. Typically, expositions of such knowledge are centered around mathematical

equations (particularly those which can be solved analytically), in the form of derivations that mimic the structure of proofs. While these equations are important, only a small subset of students gain the desired understanding from this method of presentation. An alternative is to focus more on teaching qualitative principles directly. Good textbooks attempt to do this by introducing ideas in qualitative terms before diving into quantitative details, but they rarely linger at the qualitative level. One reason for shortchanging qualitative understanding is the lack of a systematic formal vocabulary for qualitative knowledge, which makes this knowledge harder to articulate than quantitative knowledge.

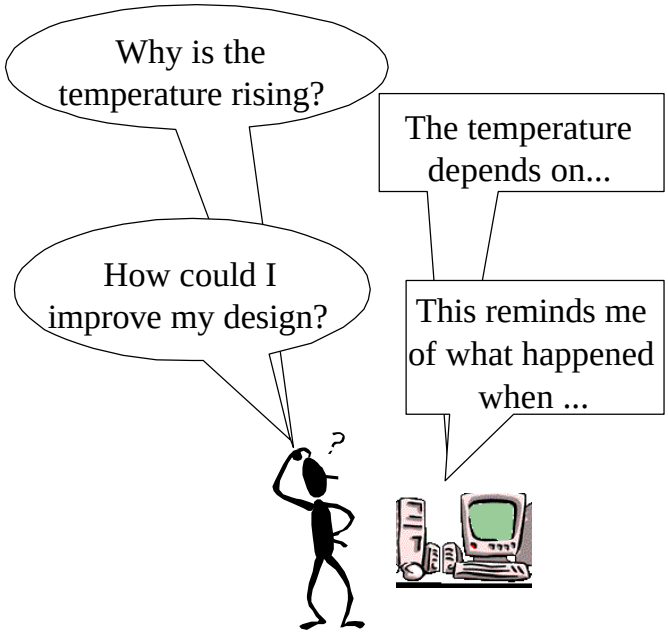

Figure 2: Articulate software provides explanations

Qualitative physics provides such vocabularies, and we hope that as these ideas become more widespread, engineering educators will be able to use them to express aspects of their expertise that are currently described as "intuition" or "art".

Whether or not one believes that engineering education must be heavily mathematical, it is impossible to make such a claim about pre-college science education. Students learn calculus, at best, at the end of high school, and sometimes only encounter algebra at the start of high school. Making students memorize differential equations in the guise of teaching them science simply isn't an option. Even formal algebraic models are not feasible for elementary and middle school students. On the other hand, students are taught about the entities, relationships, and processes needed for a qualitative understanding of phenomena. They learn what kinds of objects there are (e.g., bodies of water, clouds) and what parameters exist (e.g., temperatures, pressures, dew points). They learn partial information about relationships between parameters (e.g., "the rate of evaporation depends on the water temperature"). They learn when various relationships are relevant and what physical phenomena they are tied to (e.g., that boiling in the open air occurs at a constant temperature). In other words, the qualitative mathematics developed in qualitative physics provides exactly the right level of language for expressing relationships between continuous properties for pre-college science students.

Properties of articulate software. What these two claims suggest is that qualitative physics can provide the ability to create much smarter educational software: software whose models of the world have much in common with people's mental models. Such software can use this understanding to explain, coach, and scaffold students in a variety

of ways (Figure 2). We call such software *articulate software.* Articulate software should have the following properties:

- It should be *fluent*. 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 should 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.

There are potentially many types of articulate software. The rest of this chapter describes two architectures for articulate software that we have developed. These architectures provide students with experiences that would often be too expensive, time-consuming, or dangerous to deal with in the physical world. *Articulate virtual laboratories* provide students with design experiences, highly motivating settings for learning principles. *Active Illustrations* provide students with simulations that can be used to explore phenomena. The next two sections examine each architecture in turn.

3 Articulate Virtual Laboratories

Design activities provide powerful motivation and meaningful contexts for learning fundamental physical principles. In designing a household refrigerator, for instance, students quickly discover that water makes a poor working fluid because its saturation curve requires very low operating pressures to achieve vaporization at typical operating conditions. Design requires that students use knowledge in an integrated fashion rather than memorizing isolated facts. Getting students to think in design terms leads naturally to building a strong interest in understanding complex, real-world relationships: The question "Why did they design it that way?" can be asked of any artifact in the world around us. Design environments that provide appropriate scaffolding for students, so that they can focus on particular areas of interest, could prove invaluable for instruction in basic science as well as engineering, and could better motivate interest in science learning.

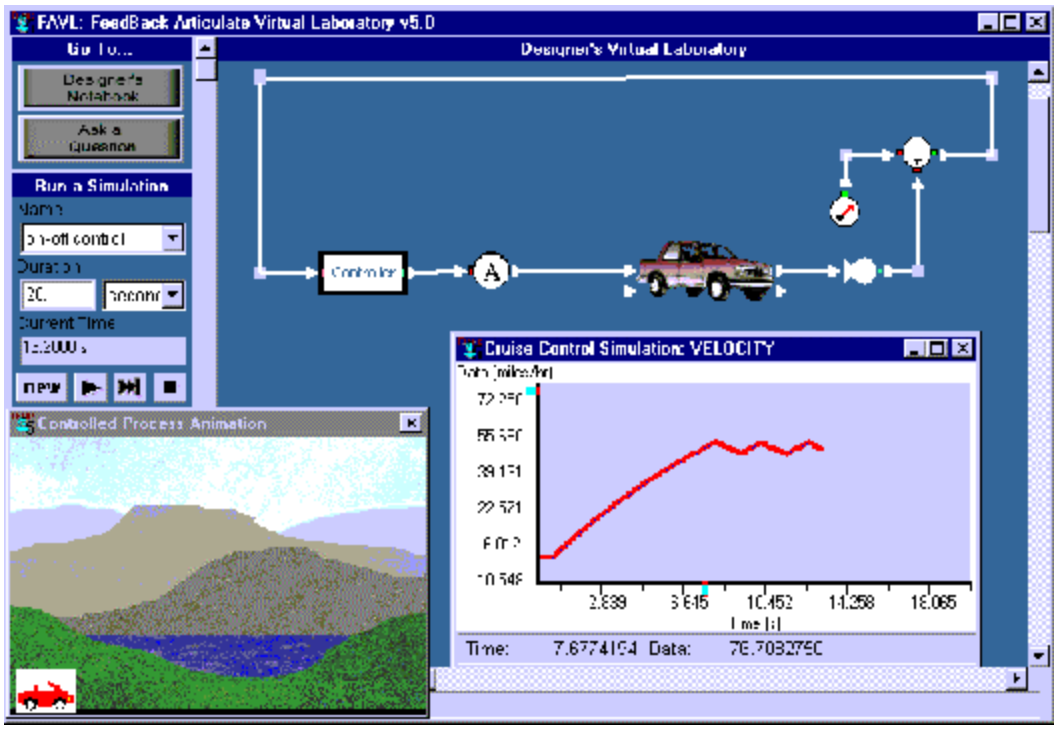

Figure 3: FAVL helps students learn principles of feedback

The articulate virtual laboratory architecture we have developed addresses this need. Like existing virtual laboratories (e.g., Electronics Workbench, Interactive Physics) it includes a software environment for creating and analyzing designs without the expense (and sometimes danger) of creating physical artifacts. Unlike existing virtual laboratories, it provides explanation facilities and coaching, to help guide the student.

We have constructed two articulate virtual laboratories to date: CyclePad (Forbus & Whalley, 1994; Forbus et al 1999) helps engineering students learn engineering thermodynamics by supporting students in designing and analyzing thermodynamic cycles. The Feedback Articulate Virtual Laboratory (FAVL) (Forbus, 1984; Ma, 1998, 1999) helps high school students learn the principles of feedback by designing controllers (see Figure 3). Since CyclePad is already in routine use, we will focus our discussion on it.

3.1 CyclePad: An AVL for engineering thermodynamics

CyclePad (Forbus & Whalley, 1994; Forbus et al. 1999) is an articulate virtual laboratory for engineering thermodynamics. The analysis and design of thermodynamic cycles is a major task that drives engineering thermodynamics (cf. Whalley, 1992). A thermodynamic cycle is a system within which a working fluid (or fluids) undergoes a series of transformations in order to process energy. Every power plant, engine, refrigerator, and heat pump is a thermodynamic cycle. Thermodynamic cycles play much the same role for engineering thermodynamics as electronic circuits do for electrical engineering: A small library of parts (in this case, compressors, turbines, pumps, heat

exchangers, and so forth) are combined into networks, thus allowing a potentially unlimited set of designs for any given problem. (Practically, cycles range from four components, in the simplest cases, to networks consisting of dozens of components.) One source of the complexity of cycle analysis stems from the complex nature of liquids and gases: Subtle interactions between their properties must be harnessed in order to improve designs. Cycle analysis addresses questions such as the overall efficiency of a system, how much heat or work is consumed or produced, and what operating parameters (e.g., temperatures and pressures) are required of its components. As in many engineering design problems, an important activity in designing cycles is performing sensitivity analyses, to understand how choices for properties of the components and operating points of a cycle affect its global properties.

To illustrate, consider the power generation cycle shown in Figure 4. Air from the atmosphere (S2) is compressed, which raises its pressure and causes its temperature to rise. More heat is added in the combustion chamber by injecting fuel and igniting it. Energy is extracted by expanding the gas through Turbine 1, and the gas is reheated and then passed through Turbine 2 to extract yet more energy. One consequence of the Second Law of thermodynamics is that a cycle must reject some heat as waste. A clever feature of this cycle is that it recycles some of this heat, using it to drive another cycle. This occurs via the heat exchanger, which heats the working fluid in the second cycle. We will return to the subcycle shortly. In the main cycle, the waste gases are exhausted back to the atmosphere after going through the heat exchanger. This return to the atmosphere is represented by a cooler, which enables us to take into account the heat lost in this transaction. Returning to the heat exchanger, the heat transferred from the gas cycle is sufficient to vaporize the working fluid (in this case water) in the lower cycle into superheated steam, which is passed through Turbine 3 to extract yet more work. Finally, the steam is condensed back into water, exhausting more heat to the atmosphere, and is pumped back into the heat exchanger to complete the cycle. A thermodynamics expert would recognize this as a combined cycle, where a Brayton gas cycle with reheat drives a Rankine vapor cycle.

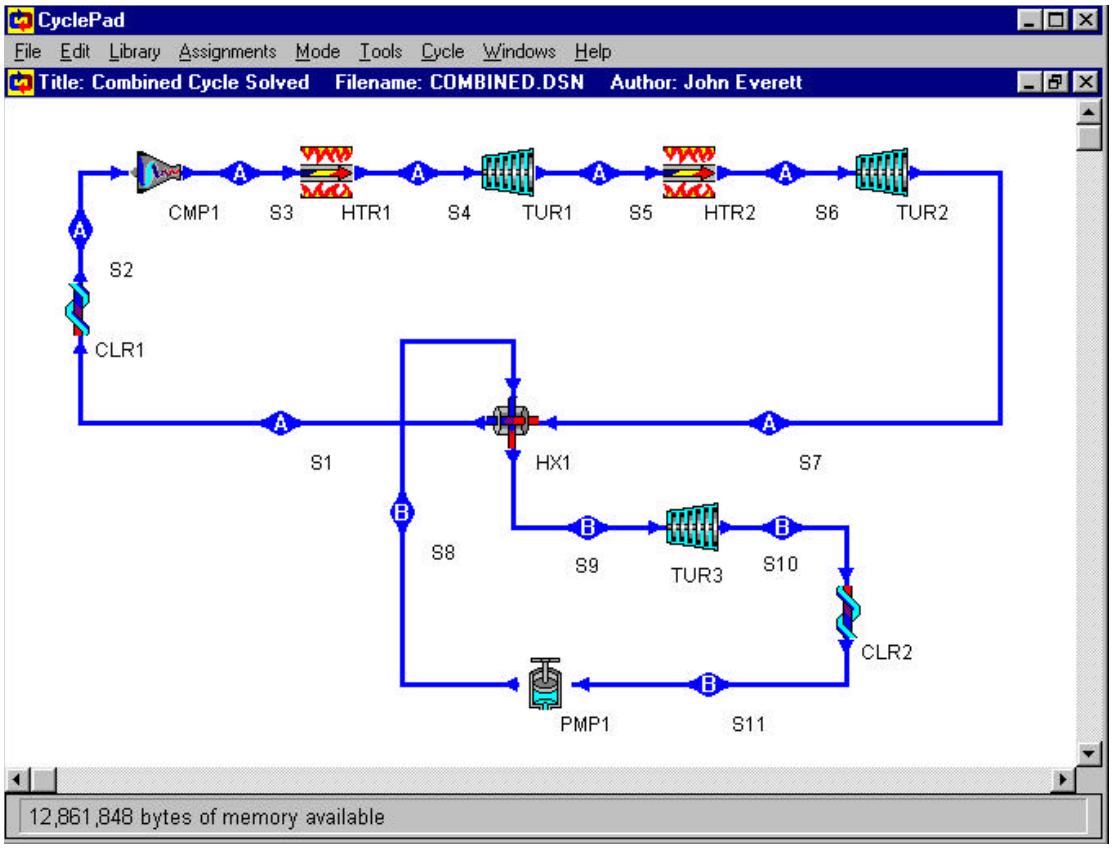

Figure 4: Example of a CyclePad design.

In thermodynamics education for engineers, cycle analysis and design generally appear towards the end of their first semester, or in a second course, since understanding cycles requires a broad and deep understanding of the fundamentals of thermodynamics. However, even the most introductory engineering thermodynamics textbooks tend to devote several chapters to cycle analysis, and in more advanced books the fraction devoted to cycles rises sharply. Indeed, some textbooks focus exclusively on cycle analysis (e.g. Haywood, 1985). Aside from their intrinsic interest, the design of thermodynamic cycles provides a highly motivating context for students to learn fundamental principles deeply.

There are several reasons why thermodynamics is especially suitable for virtual laboratories. First, physical laboratories involving even simple thermodynamic cycles can be expensive and dangerous, due to the need for high temperatures and pressures, along with dangerous chemicals (e.g., fuels, ammonia). Second, many of the most interesting systems, such as jet engines and power plants, are far too expensive and time-consuming to build as student projects. Third, creating working physical artifacts requires what is known as detailed design, where many concrete choices are made, in addition to conceptual design, where the basic properties of a system are figured out. It

is conceptual design that provides the most pedagogical value in teaching fundamental principles. For example, designing a working jet engine requires selecting an appropriate curvature for the turbine blades and worrying about shape and weight tradeoffs for every component. Designing a working power plant that uses water as its working fluid requires designing draining, cleaning, and lubrication systems. These topics are important for advanced engineering courses specializing in those areas, but are irrelevant and distracting when trying to help students master the fundamentals of the domain.

3.2 Using CyclePad: An example

When students start up CyclePad, they find a palette of component types (e.g., turbine, compressor, pump, heater, cooler, heat exchanger, throttle, splitter, mixer) that can be used in their design. Components are connected together by *stuffs*, which represent the properties of the working fluid at that point in the system.

An important aside: There are actually two perspectives one can take on thermodynamic cycles: Steady-flow versus closed cycles. In steady flow analyses one treats the fluid flowing through the system as essentially varying with location rather than time. Typical applications of steady-flow analyses include power plants and jet engines. In closed cycle analyses one follows a volume of fluid through a set of changes imposed by physical processes, regardless of their physical location. Typical applications of closed cycle analyses include automobile and diesel engines. Closed-cycle problems are also heavily used early in thermodynamics courses to focus students on the particular properties of substances and processes. While the laws of thermodynamics are the same for both perspectives, how the laws are applied varies as a consequence of the ontological choices each perspective makes. CyclePad supports both perspectives, but we only discuss steady-flow systems here for simplicity.

Once students put together the structure of the cycle, they continue to use CyclePad to analyze the system. The student enters assumptions such as the choice of working fluid and the values of specific numerical parameters. In addition to numerical assumptions, *modeling assumptions* can be made about components. For example, a turbine can be assumed to be adiabatic (i.e., no heat is lost from it to its surroundings), isothermal (i.e., no temperature change across it), or isentropic (i.e., constant entropy). Such modeling assumptions can introduce new constraints that may help carry an analysis further. They can also introduce new parameters (e.g., the efficiency of the turbine if it isn't isentropic) whose values the student must appropriately constrain.

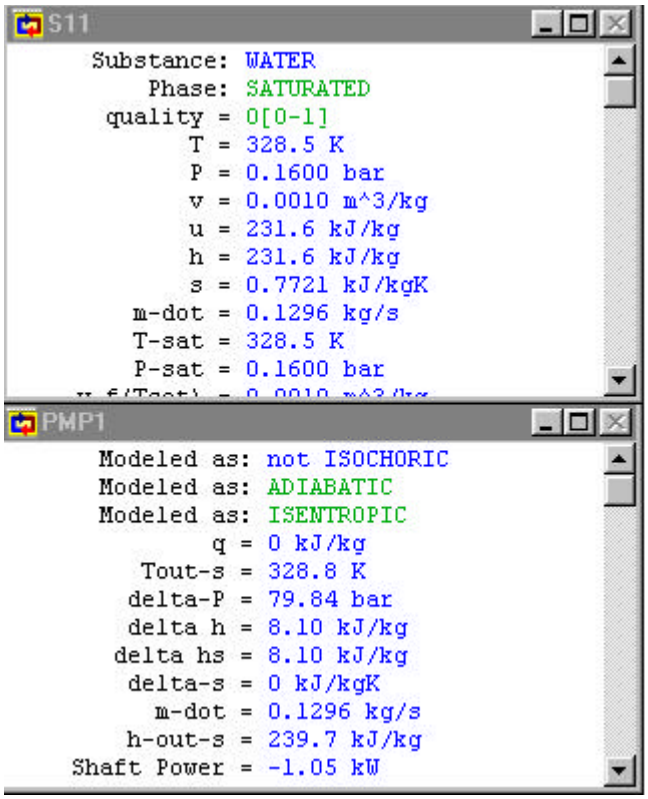

Figure 5: Meters provide details about parameter values. Green indicates assumptions and blue indicates derived values.

CyclePad accepts information incrementally, deriving from each student assumption as many consequences as it can. *Meters* associated with each component and stuff are available to describe its properties (see Figure 5). Like physical meters, these meters display numerical parameters of the entity they represent in a compact tabular form. Unlike physical meters, they also display the modeling assumptions the student has made so far, and what other assumptions might yet be made. At any point questions can be asked, by clicking on an element of a meter display to obtain the set of questions (or commands) that make sense for it. The questions and answers are displayed in English. They include links back into the explanation system, thus providing an incrementally generated hypertext (see Figure 6). This hypertext is important for several reasons. First, it helps students understand the indirect consequences of their assumptions. Second, the nature of the domain is such that students often make inconsistent modeling assumptions[1]. CyclePad detects and flags such contradictions, and the hypertext system enables the student to explore the subset of assumptions responsible and decide which to retract.

[1] The historical interest in perpetual motion machines and its manifestations today (the "free energy" inventors and those dabbling in "over unity technologies") suggests that this problem is not limited to students.

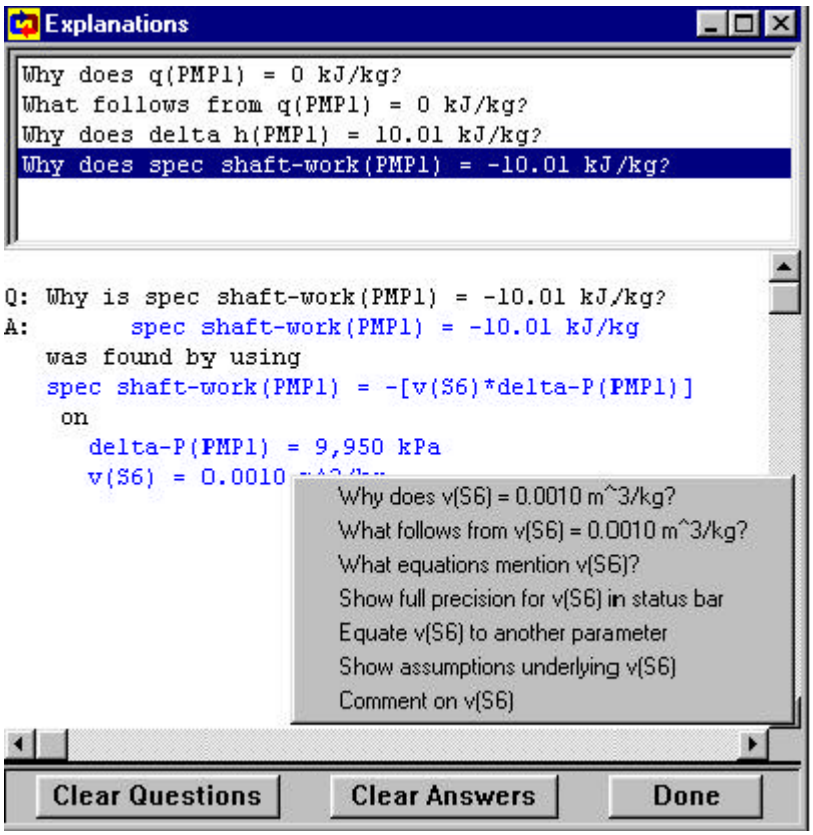

Figure 6: CyclePad's hypertext explanation system

In addition to the hypertext system, CyclePad incorporates several other tools and capabilities that help students understand their design. CyclePad will automatically create a T-S (Temperature-Entropy) diagram on request, a graphic commonly used by engineers to understand the global properties of a cycle. CyclePad includes an optional economic analysis model, so that designs can be evaluated on the basis of their cost of construction and operation. CyclePad can carry out sensitivity analyses, enabling students to explore how changes in one parameter affect another. Reflecting on graphs of such sensitivity analyses is useful in understanding the properties of their design and how modifications might affect it. For instance, students might discover that to obtain the desired thermal efficiency with their current design the operating temperature would have to be raised so high that the materials to build it would be too costly.

CyclePad includes two kinds of coaching. The *on-board coach* provides rapid feedback for some commonly encountered problems. For example, it recognizes the most common type of contradiction (i.e., requirements that push a state point outside the bounds of the property tables) and provides visualizations of the progress of the analysis (i.e., how much has been pinned down about particular aspects of the cycle so far). It can make suggestions about reasonable ranges for parameter values, based on its understanding of

the teleology of the cycle. The *email coach* provides additional assistance with analysis and design. The most novel facility in the email coach is the ability to make suggestions about how to improve a student's design, based on analogies with a library of expert-authored cases. The coach provides step-by-step instructions illustrating how this suggestion can be applied to the student's design. It does not, however, evaluate whether or not this suggestion is a real improvement – that evaluation provides a valuable learning opportunity for the students.

3.3 How CyclePad helps students

The design of CyclePad was driven by addressing the needs of instructors in teaching engineering thermodynamics. A variety of common problems arise when teaching students how to design and analyze thermodynamic cycles:

1. Students tend to get bogged down in the mechanics of solving equations and carrying out routine calculations. They avoid exploring multiple design alternatives and avoid carrying out trade-off studies (e.g., seeing how overall cycle efficiency varies as a function of turbine efficiency versus how it varies as a function of boiler outlet temperature). Yet without making such comparative studies, many opportunities for learning are lost.
2. Students often have trouble thinking about what modeling assumptions they need to make, such as assuming that a heater operates isobarically (i.e., no pressure drop across it), leading them to get stuck when analyzing a design.
3. Students typically don't challenge their choices of parameters to see if their design is physically possible (e.g., that their design does not violate the laws of thermodynamics by requiring a pump to produce rather than consume work).
4. Students typically have no basis for relating the values they calculate to the physical world and their everyday experience. The units of thermodynamic quantities, such as kilowatts, are not as accessible as pounds or feet. This lack of intuition about, for instance, whether 10,000 kilowatts is enough to light a room or a city causes students to treat thermodynamics problems as abstractions divorced from practical application.

These considerations drove the design of CyclePad. Here are how the features shown is the previous section address these problems:

1. CyclePad handles routine calculations, including equation solving and property-table interpolation. By facilitating sensitivity analyses, CyclePad encourages students to develop their intuitions through trade-off studies.
2. CyclePad's interface makes modeling assumptions explicit and highly salient. It helps them keep track of the consequences of their modeling assumptions.
3. CyclePad detects physically impossible designs, using a combination of qualitative constraints and numerical reasoning. It alerts students about such problems, and supports their investigations and resolution of them through its generated hypertext system.

4. CyclePad includes benchmarks that help ground parameter values in real-world examples, and a web-based design library whose entries are accessed based on analogies with the student's design.

3.4 How CyclePad works

CyclePad uses a combination of artificial intelligence techniques to provide the abilities outlined in the previous section. These are described in detail elsewhere (Forbus et al 1999), here we summarize them to highlight how they contribute to scaffolding student learning.

```
(defEntity (Abstract-hx ?self ?in
                        ?out)
  (thermodynamic-stuff ?in)
  (thermodynamic-stuff ?out)
  (total-fluid-flow ?in ?out)
  (== (mass-flow ?in)
      (mass-flow ?out))
  (parameter (mass-flow ?self))
  (parameter (Q ?self))
  (parameter (spec-Q ?self))
  (heat-source (heat-source ?self))
  ((parts :cycle) has-member ?self)
 (?self part-names (in out))
 (?self IN ?in)(?in IN-OF ?self)
  ?self out ?out)(?out out-of ?self))

(defAssumptionClass
  ((abstract-Hx ?hx ?in ?out))
    (isobaric ?hx)
    (:not (isobaric ?hx)))
```

```
(defEntity (Heater ?self ?in ?out)
  (abstract-Hx ?self ?in ?out)
  (?self instance-of heater)
  (heat-flow (heat-source ?self)
             (heat-source ?self)
              ?in ?out)
  ((heat-flows-in :cycle)
     has-member (Q ?self))
  (> (Q ?self) 0.0))

(defEquation Hx-law
  ((Abstract-Hx ?hx ?in ?out))
  (:= (spec-h ?out)
      (+ (spec-h ?in) (spec-Q ?hx))))

(defEquation spec-Q-definition
  ((Abstract-Hx ?hx ?in ?out))
  (:= (spec-Q ?hx)
      (/ (Q ?hx) (mass-flow ?hx))))
```

Figure 7: Samples from CyclePad's knowledge base

3.4.1 CyclePad's Knowledge Base

The domain knowledge in CyclePad is represented using techniques from qualitative physics (Forbus, 1984) and compositional modeling (Falkenhainer & Forbus, 1991). The knowledge required to support design and analysis goes far beyond just a set of equations, as the examples in Figure 7 illustrate. CyclePad's domain theory includes:

- *Physical and conceptual entities:* These include components such as compressors, turbines, pumps, and heat exchangers; physical processes such as compression, combustion, and expansion, and the representations of the properties of the working fluid between them. CyclePad's knowledge base currently contains over 29 entity definitions.
- *Structural knowledge:* What kinds of relationships can hold between components, process occurrences, and the descriptions of working fluids that connect them. CyclePad's knowledge base currently contains 34 structural facts.
- *Qualitative knowledge:* This includes the kinds of physical processes that can occur inside components, or in the sequence of operations in an open cycle. Physical processes constrain the parameters of the situation. For instance, the temperature of the working fluid coming into a heater cannot be higher than the temperature of the

working fluid when it leaves. CyclePad's knowledge base currently contains definitions of five fundamental physical processes.

- *Quantitative knowledge:* This includes equations that define relationships between the parameters of the constituents of a cycle, numerical constants (i.e., molecular weights), and tables of property values for substances (e.g., saturation and superheat tables). CyclePad also automatically derives equations for global properties. For example, equations for net work and heat flows into and out of the cycle are derived every time the structure of the cycle changes. CyclePad's knowledge base currently contains 167 equations, and saturation tables and superheat tables for 10 substances.
- *Modeling assumptions:* Modeling assumptions describe what simplifications can be made about a component or process during an analysis. For instance, the pressure drop across a boiler is typically ignored in conceptual design because it is negligible for the purpose of the analysis. Rather than stipulating a particular pressure drop, it is simpler to assume that the heater used to model a boiler is isobaric, i.e., has no pressure drop. CyclePad's knowledge base currently contains 10 types of modeling assumptions
- *Assumption classes:* Assumption classes help structure reasoning by organizing modeling assumptions into sets. When an assumption class holds, one assumption from it must be included in the model of the cycle for the model to be complete. CyclePad's knowledge base currently contains 14 assumption classes.
- *Economic model:* Economic tradeoffs are key issues in design. CyclePad incorporates standard engineering cost estimating functions that extrapolate capital costs for a cycle based on the size of the components, generally estimated by mass-flow. CyclePad contains information about several different materials, including stainless steel, nickel alloy, titanium, and molybdenum. Each material has limits on the temperatures (high and low, the latter for cryogenic applications) that it can endure. A special material, Unobtainium, with extraordinary properties (including price) is useful for suspending the economic constraints on a particular device or subset of devices. CyclePad also estimates the resulting weight of the cycle as a function of the materials employed, which may be a critical constraint, for example, in the design of an aircraft engine.

CyclePad's knowledge base is powerful enough to handle a wide variety of analyses found in introductory and advanced thermodynamics textbooks.

3.4.2 CyclePad's Analysis Methods

A student's activities with CyclePad shift between creating and/or editing the structure of the cycle and analyzing the properties of the cycle by supplying assumptions about its constituents. CyclePad interactively and incrementally derives the consequences of each student assumption. This work is performed via antecedent constraint propagation, with the derivations being recorded in a logic-based truth maintenance system (LTMS) (Forbus & de Kleer, 1993). At any point the student can ask for explanations of derived values, the indirect consequences of particular assumptions, what equations might be

relevant to deriving a particular value, and other similar queries. These explanation facilities exploit the dependency network created in the LTMS.

Explanations in CyclePad are represented by *structured explanations*, an abstraction layer between the reasoning system and the interface. The reason for this layer is that the reasoning system needs to be optimized for performance, while the interface needs to be optimized for clarity, and these goals are often incompatible. The structured explanation layer provides summarization, hiding aspects of how the reasoning system works that are irrelevant to the student. It also provides reification, making explicit dependencies that would otherwise be implicit, such as the various methods that could be used to derive a desired parameter.

Automating the tedious calculations involved in using thermodynamic equations and providing clear explanations of how the student's assumptions were used provides substantial scaffolding. Students can focus on thinking about the thermodynamic consequences of their assumptions, rather than using their calculators to solve routine equations. The LTMS also provides a useful mechanism for detecting and recovering from contradictory assumptions. For instance, if the parameters supplied by the student imply that physical laws are violated (i.e., that a turbine consumes work rather than generates it), this fact along with the subset of assumptions responsible is brought to the student's attention for correction.

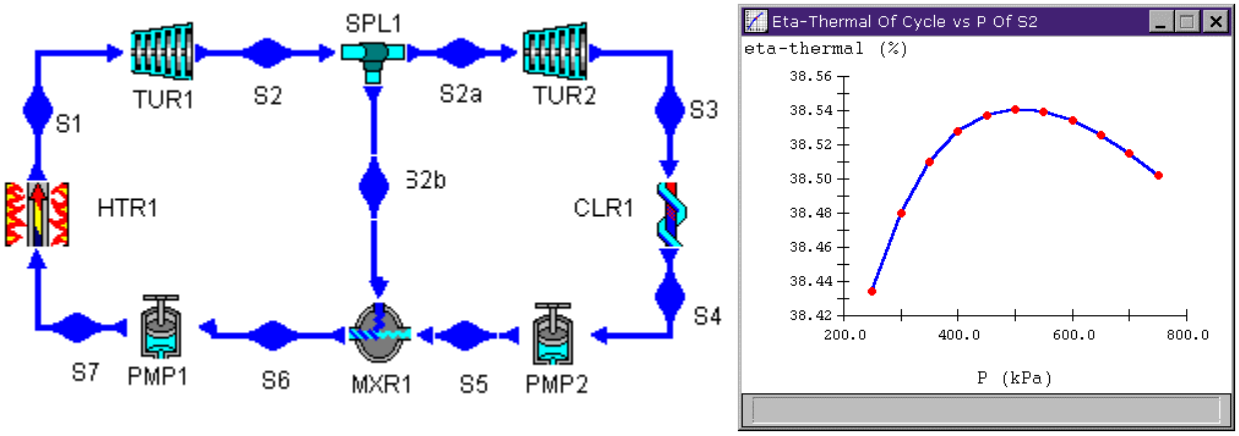

Figure 8: Sensitivity analyses show impact of design decisions, revealing underlying principles. Here the effect of feedwater pressure on thermal efficiency in a regeneration cycle is being explored.

CyclePad provides other analysis tools in addition to constraint propagation. It automates the process of performing sensitivity analyses, which involve seeing how a change in one parameter affects another parameter (e.g., how the boiler pressure affects the thermal efficiency of the cycle), using the dependency network in the LTMS to identify relevant parameters and automatically derive the necessary equations (see Figure 8). Such analyses are viewed as important by instructors for gaining a deeper appreciation of the

domain. CyclePad provides visualization tools that make apparent how parts of the cycle contribute to its overall performance. Graphical information about the bounds of available property tables, and in some cases automatically generated T-S (temperature versus entropy) diagrams, are also available. An on-line help system that describes the program's operation and knowledge is included.

Building student intuitions about the meaning of the properties of thermodynamics and helping them achieve a quantitative "feel" for the subject is an important pedagogical problem. Students initially know so little about thermodynamics and cycles that they can have problems spotting problems in their designs. For example, experienced designers will note that low quality (i.e., too much liquid in the mixture) in the working fluid exiting a heat engine's turbine is likely to cause damage to the turbine blades. Consequently, they will attempt to adjust the system's parameters to increase the exit quality, or failing that, make a structural alteration to the cycle. To spot problems like this and understand how to fix them requires knowledge of how function relates to structure. For example, low exit quality is only a problem if the cycle is intended as a heat engine. In a cryogenic cycle, turbines can be used to cool the working fluid sufficiently to cause precipitation, because a resisted expansion results in a greater drop in the working fluid temperature than a throttled expansion. Thus in the case of a cryogenic cycle we might be aiming for low quality. Giving advice about cycle parameters, therefore, requires understanding the intended purpose of the system and the functional roles each component plays in achieving that purpose.

CyclePad incorporates Everett's Carnot teleological recognition system (Everett, 1999) to understand the intended function of the cycle, in order to provide advice about values of cycle parameters. Different components can play different functional roles. For example, a mixer may act as a simple way to join flows, as a heat-exchanger, or as a jet-ejector, in which a high-velocity jet of fluid entrains and compresses another inlet stream. Understanding the intended function of a system requires assigning functional roles to each component and recognizing any larger-scale plans that the configuration of roles represent, such as regeneration. Carnot uses evidential rules and Bayesian inference to suggest plausible functional roles for each component in a student's cycle. The evidential rules provide evidence either for or against a particular role. This evidence is used to update the prior probability of each role for each component. The evidential reasoning is included in CyclePad's explanation system, so that students can find out why (and with what certainty) a particular role is believed and can also get an explanation of why other potential roles were rejected (see Figure 9).

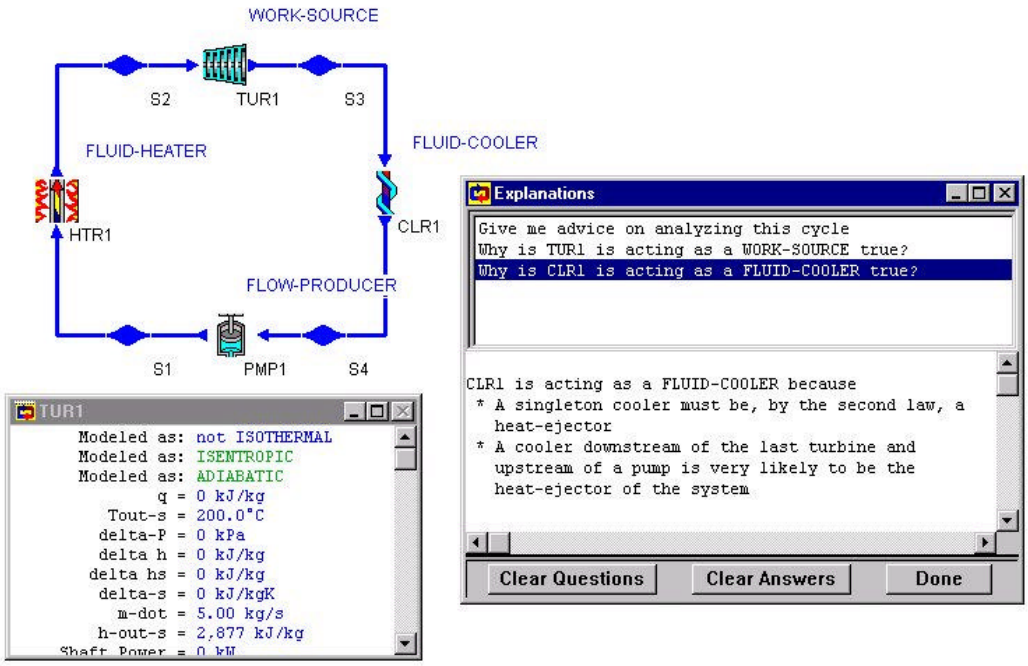

Figure 9: CyclePad uses evidential reasoning to infer student intent

CyclePad combines Carnot's teleological inferences with *norms* to generate advice for adjusting parameters. A norm is a range for a component's parameter that is appropriate based on the component's functional role. For example, the temperature of the steam leaving a Rankine cycle boiler typically falls in the range of 300-600ºC. Lower temperatures result in inadequate efficiency whereas higher temperatures require uneconomically expensive materials in the downstream components. In contrast, the range of temperatures for the refrigerant leaving the coils of a refrigerator (which are modeled as a heater) is quite different, typically in the range of 5-15ºC. Inferring the role a component is playing is therefore essential to providing relevant advice to the student. Our knowledge base currently contains eighteen norms, between two and six per component depending on the number of potential roles for that component.

When the Analysis Coach is invoked, Carnot infers the teleology of the cycle. The functional roles assigned to each component are used to retrieve applicable norms, which are checked against known parameter values. Any violations or suggestions are noted using CyclePad's explanation system, providing explanatory text associated with each norm. In addition to being used to provide on-board advice, Carnot's teleological representations also play an important role in our case-based design coach, described below. The insight is that similarity in intended function and qualitative properties are better predictors of a case's relevance than the specific numerical values involved in it.

3.4.3 CyclePad's Coaching

CyclePad's on-board coaching facilities are supplemented by an email-based coaching system. We turned to email because CyclePad is used by students in a variety of institutions spread across the planet. Since CyclePad is distributed via the web, there is some likelihood that students have network access. Students can use an email system built into the software to send their current design and a query about it to our coach, which runs on a server at Northwestern. The coach is implemented as part of a *RoboTA Agent Colony* (Forbus & Kuenhe 1998), a software architecture designed for providing distributed learner support. Email to a RoboTA is handled by a Post Office Agent, which ascertains which member of the colony is best able to handle it. The *CyclePad Guru* is the agent designated for CyclePad-related messages. The kinds of messages supported by the CyclePad Guru are

1. *Turning in an assignment.* We are experimenting with a system that enables instructors to create assignments, including evaluation rubrics, that enable students to submit their solutions via email. The idea is to make it easier for instructors to collect student work and have the mechanical aspects of their evaluations applied automatically.
2. *Asking for help with a contradiction.* The coach provides some general feedback in response to this case.
3. *Asking for help in completing an analysis.* The coach provides advice based on an expert model of how to analyze cycles, pointing out the kinds of assumptions that might be appropriate to make given the student's progress.
4. *Asking for help in improving a design.* The coach provides suggestions for improving the design, based on a case library of design transformations

Design coaching is the most novel feature of the CyclePad Guru. We have two goals in giving design advice. First, we want to nudge students in useful directions, rather than solving problems for them. Consequently, the Guru provides plausible specific suggestions, but does not attempt to validate those suggestions in the students' context. Understanding why a suggestion will or will not work in a particular circumstance is an important learning experience that we want students to have. Second, we want to motivate students to dig more deeply into the nature of thermodynamics-ideally, to immerse themselves in the culture of engineering thermodynamics by studying real-world systems and how they are connected to the assignments they are grappling with. Consequently, the Guru uses case-based coaching to generate design advice, motivating students to explore the case deeply by showing exactly how that case might be relevant to the improvement they are trying to make.

```
From: robota@godzilla.cs.nwu.edu
Date: Mon, 20 Sep 99 00:49:47 -0600
To: forbus@nwu.edu
Subject: The CyclePad Guru's response to your message: Need help improving my
Rankine cycle

You asked for help with your design.
I have 2 suggestions.
============================================
Suggestion #1
Your problem reminds me of a method: increasing boiler temperature in a Rankine
cycle.  Increasing the boiler temperature increases the efficiency of the cycle.
You can find out more about this at <URL>.
Here is how you might apply this to your design:
1. Increase T(S2).
============================================
Suggestion #2
Your problem reminds me of a method: reheat in a Rankine cycle.  Reheat adds
another heater and another turbine.  The second heater, a reheater, heats up the
working fluid at the turbine
outlet, and the second turbine extracts yet more work from that.  This increases
efficiency because more heat is being added when the steam is still at a
reasonably high temperature.
You can find out more about this at <URL>.
Here is how you might do this with your design:
1. Disconnect the outlet of TUR1 from the inlet of CLR1.
2. Create a new heater, which we'll call HTR2.
3. Connect the outlet of TUR1 to the inlet of HTR2.  Let's refer to the
properties of the working fluid there as S5.
4. Create a new turbine, which we'll call TUR2.
5. Connect the outlet of HTR2 to the inlet of TUR2.  Let's refer to the
properties of the working fluid there as S6.
6. Connect the outlet of TUR2 to the inlet of CLR1.  Let's refer to the
properties of the working fluid there as S7.
You might find the following assumptions relevant or useful:
1. Assume that the working fluid at S5 is saturated.
2. Assume quality(S5) = 1.0000[0-1]
3. Assume that HTR2 is a reheater.
4. Assume that HTR2 works isobarically.
5. Assume that HTR2 is made of molybdenum.
6. Assume that HTR2 burns natural-gas.
7. Assume that TUR2 works isentropically.
8. Assume that TUR2 is made of molybdenum.
9. Assume that the working fluid at S7 is saturated.
10. Assume quality(S7) = 1.0000[0-1]
============================================
```

Figure 10: Design advice from the CyclePad Guru

A sample of design advice from the Guru is illustrated in Figure 10. The Guru has access to a library of cases, each describing a particular change to a design and what it is intended to accomplish. These changes can be either tuning the parameters of the cycle (i.e., increasing the operating temperature of a boiler to increase efficiency) or a structural change in the cycle (i.e., adding reheat to a cycle to enable more work to be extracted). Cases are authored by domain experts, using CyclePad and an HTML editor. Notice that a URL is supplied as part of the advice. These web pages describe the general principle involved in the library case, illustrated through a concrete example. The concrete

example used in the case is generated by the domain expert, using CyclePad. The domain expert describes a transformation that implements the principle by making the appropriate changes to this design. A *case compiler* uses this information to compute a description of the transformation that can be used in analogical reasoning. Consequently, domain experts only need to be able to use CyclePad plus an HTML editor in order to add cases to the design library.

Given a student's design, the Guru uses a cognitive simulation of similarity-based retrieval, MAC/FAC (Forbus, Gentner, & Law, 1995) to retrieve relevant cases. Concrete advice as to how to apply the idea of the case to the student's design is generated by a cognitive simulation of analogical matching, SME (Falkenhainer, Forbus, & Gentner, 1989; Forbus, Ferguson & Gentner, 1994). The use of cognitively motivated analogical processing software has two advantages over the typical state of the art in case-based reasoning (CBR) systems. First, most CBR systems require hand-indexing of new cases by experts familiar with both the domain and the retrieval system. By using MAC/FAC, we exploit human-like similarity computations to automatically retrieve cases without indexing. Second, most CBR systems use simple lists of features as their representation medium. By contrast, CyclePad designs are (internally) full predicate calculus descriptions, encoding relational structure such as the steps required to achieve a design modification. These richer relational structures lead to analogical inferences by SME, that are turned into step-by-step instructions on how to apply the case to the student's design.

Using a distributed coaching system has its disadvantages. It requires students to have access to email. It involves a delay in responding to a student's request, which may not be as effective as providing an immediate response. This is an inevitable limitation of email as a transport mechanism. Prior requests do not affect the answer returned, i.e., one cannot enter into a correspondence with this coach. Creating a software coach capable of natural language conversations with students and maintaining an ongoing model of them and their progress would be an excellent research project, but is extremely difficult. On the other hand, by putting complex coaching facilities on a server at our site, we can make improvements in coaching strategy without asking users to reinstall our software. The potential value of a distributed coach becomes especially apparent when considering the issue of extending and maintaining a case library. A large, rich case library with lots of associated media (e.g., pictures of the real physical systems corresponding to the CyclePad design) is probably best treated as a network resource, rather than installed on each student machine. We are forming an editorial board for the web-based design library, to ensure quality control, and encouraging submissions from CyclePad experts worldwide, much in the manner of the Eureka community-maintained database of tips (Bell et al 1996).

3.5 Discussion

CyclePad has been distributed for free via the Web since September 1997 and has been used in classrooms scattered all over the world. As of September 1999, we had over

2500 distinct downloads from 63 countries. While some downloaders never use the software or do not find it to their liking, we know from surveys and email feedback that a number of instructors have adopted it successfully and use it in their courses in a variety of ways. Although the project is now over, we will continue to distribute CyclePad and run the CyclePad Guru server, and will make CyclePad's source code publicly available through an open-source license.

CyclePad provides strong evidence for the utility of articulate virtual laboratories. It has been adopted by instructors in a variety of educational institutions for both introductory and advanced courses. Some institutions use it with traditional textbooks, while others are developing new curricula around it. In universities where we have direct collaborators, we have seen various benefits of CyclePad. For example, advanced thermodynamics students at the US Naval Academy were able to tackle more complex term projects than they were able to previously, resulting in some cases in publishable technical papers (cf. Wu & Burke, 1998; Wu & Dieguez, 1998).

In Engineering Technology curricula, i.e., curricula aimed at producing technicians rather than engineers, students often learn calculus later than thermodynamics. This makes the analysis-heavy approach of standard thermodynamics courses even less useful for this population. CyclePad provides a “simulated hands-on” experience for such students, helping them build solid, accurate intuitions about thermodynamics. (Baher, 1998). For example, at University of Arkansas, Little Rock, students use CyclePad in laboratory exercises to experiment with systems that would be too expensive or dangerous to physically build.

The design approach of articulate virtual laboratories fits quite naturally into many advanced thermodynamics courses. Regrettably, in the United States this has not been true of introductory courses. Traditional thermodynamics courses, like many current engineering courses, are analysis-centered, lavishing classroom time on mathematical derivations of thermodynamic principles at the expense of helping students understand the principles themselves and their implications. Many courses still spend time teaching students how to do complex analyses, including table interpolations, with just a simple calculator, even though as practicing engineers they will have more sophisticated computer support. This necessarily reduces the time available for understanding the principles of thermodynamics and time available for learning design skills. This has been a significant barrier in introducing CyclePad in introductory courses. Indeed, using CyclePad in such courses can lead to drops in student performance, since students are being tested on mechanical calculation skills that in practice are automated. This problem is analogous to the introduction of calculators into mathematics education. The introduction of intelligent systems that handle more of the analytic load of engineering tasks suggests rethinking what we should be teaching and how it can be taught. For example, in pilot studies we have experimented with exercises where students use CyclePad to do simple design and optimization tasks, weighing their written reports as to the "how" and "why" of their work as much as the specific answers they provided. A positive trend is the recent interest by ABET, the US engineering education standards organization, on infusing design tasks throughout engineering curricula.

The articulate virtual laboratory architecture CyclePad embodies can, we believe, be fruitfully applied to many other engineering domains. The nature of the analysis tools will vary from domain to domain. AVLs for electronics or chemical engineering might end up looking very much like CyclePad, whereas AVLs for mechanism design or computer programming might be able to utilize similar structured explanation systems and distributed coaching, but with very different analysis and design methods. AVLs could make spreading design work through the engineering curriculum much more practical, for instance by providing support for portfolio assessment.

We also believe that with appropriately simplified domains, articulate virtual laboratories could also be used in science teaching. Design activities are commonly used in constructivist learning systems and curricula because they are so motivating (cf. Papert 1980; Lehrer 1998). The National Science Education Standards have identified design activities as a means of motivating learning of scientific content and process as well as a vehicle for understanding the technological world for K-12 education (National Research Council, 1996). Experience with physical systems is often an important aspect of learning through design, but AVLs could provide valuable complementary activities, and make rich design activities possible in domains for which it is now impossible. For example, CREANIMATE (Edelson, 1992) used the idea of modifying animals as a motivation for students to watch videos that showed how animals behave. While this video-driven case-based approach has its attractions, an AVL for such a domain would provide much richer explanations and more freedom for students to explore animal behavior and biomechanics.

4 Active Illustrations

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. Students 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 capabilities. An active illustration can be thought of as a hands-on museum exhibit, consisting of a virtual artifact or system, and (ideally) a guide who is knowledgeable about the exhibit and enthusiastically helps 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.

4.1 Example: The Evaporation Laboratory

Suppose a student is interested in how evaporation works. Since evaporation happens in everyday circumstances that are neither dangerous nor expensive to set up, it can easily be experimented with. 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 an unused computer, the student starts up an Active Illustration on evaporation, to try to gain some insights in minutes instead of days.

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. 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 requesting a plot of 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. (This is obviously not an experiment that is easily carried out in the physical world.) Qualitatively the behavior is the same, but the higher thermal conductivity of diamond means that the temperature of the diamond cup will quickly become close to the ambient temperature, and indeed leads to increased evaporation (Figure 11).

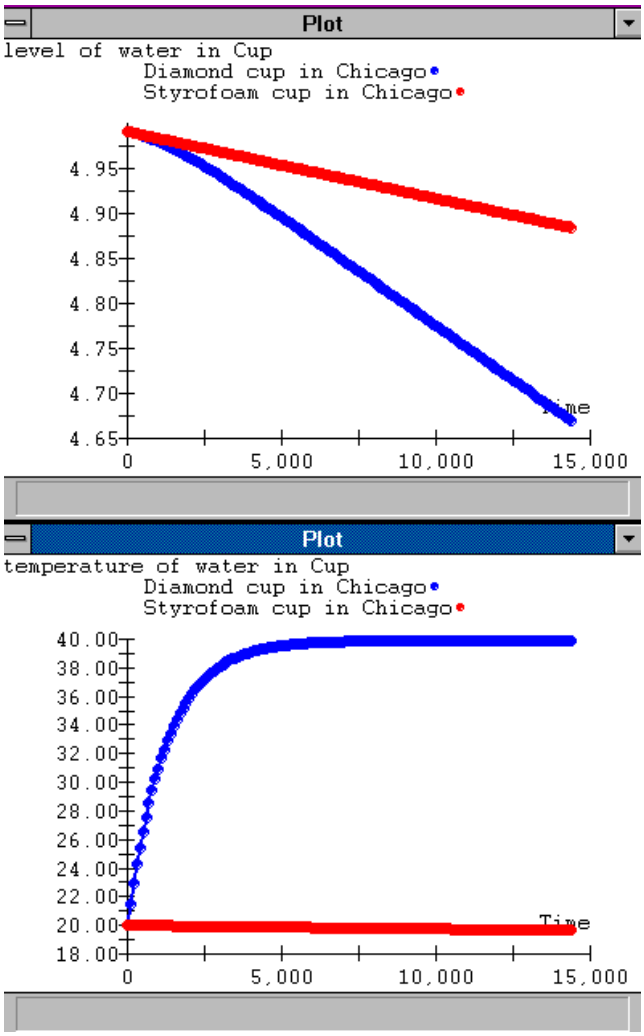

Figure 11: Students can compare behaviors quickly across multiple simulations

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.

4.2 How Active Illustrations work

The principle component of active illustrations for dynamical systems are *self-explanatory simulators* (Forbus & Falkenhainer, 1990; Iwasaki & Low, 1992; Amador, Finkelstein, & Weld, 1993). A self-explanatory simulator combines qualitative and numerical representations to provide both accurate quantitative descriptions of behavior

and conceptual explanations of it. The conceptual explanations are in terms of what physical processes are occurring in the system being simulated, and the causal relationships that govern its behavior. As the Evaporation Laboratory example showed, a self-explanatory simulator can describe at every point in the simulation exactly what is happening in the system being simulated and why. These explanations can in theory range from qualitative, causal explanations suitable for novices to sets of ordinary differential equations suitable for an expert audience. (We have focused on the former so far since many of our simulators have been designed for middle-school students.)

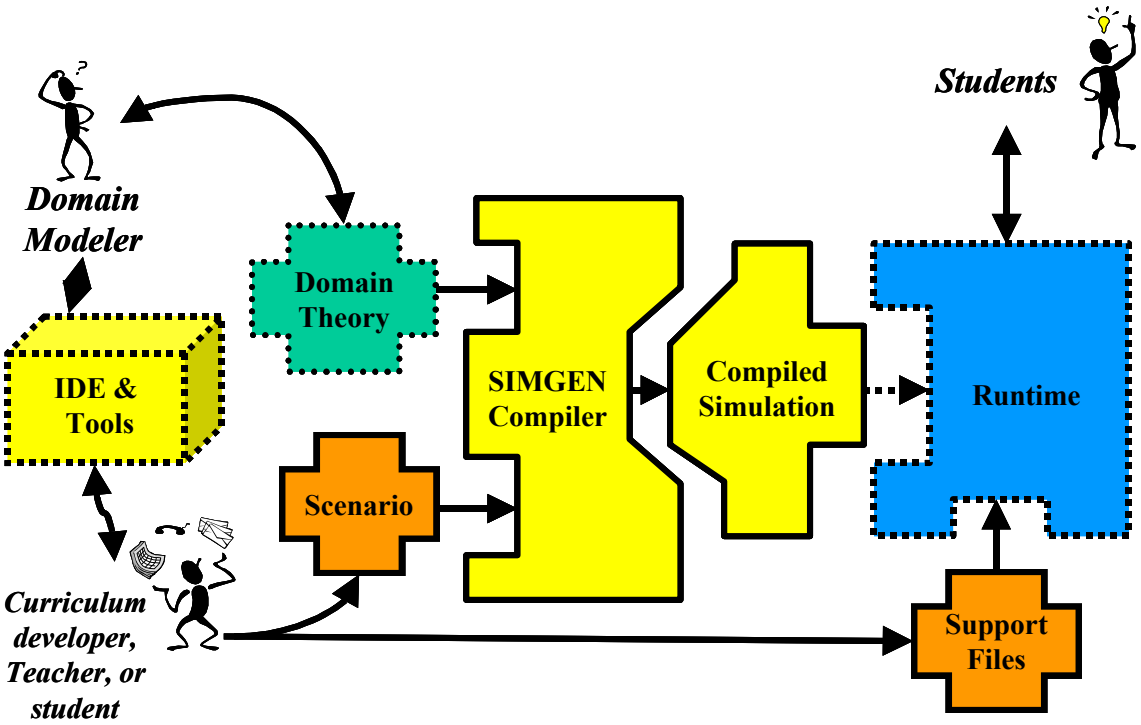

Figure 12: The process of creating self-explanatory simulators

Traditional simulators can be difficult to build and tune, so it might at first seem that self-explanatory simulators must necessarily be more complex. This is not the case. In fact, self-explanatory simulators can be constructed automatically, using AI techniques whose general form and operation are inspired by watching human simulation authors work. A person writing a simulator must first decide exactly what phenomena need to be simulated -- what should be included and what should be left out. For example, in simulating global warming, including the thermal effects of the oceans is important, whereas the gasses produced by cigarette smoking is not. Once the phenomena to include have been decided, appropriate mathematical models must be found or derived. From these mathematical models simulation code is written, either from scratch or by assembling predefined modules. In the ideal case, the conceptual understanding process that the simulation author went through is well-documented somewhere, perhaps even in material accessible to the simulation users. In reality, such documentation is rare, and often produced by reconstruction rather than during construction. This can lead to

problems, as when the simulated behavior clearly is not consistent with the explanations about how it is generated.

A compiler for self-explanatory simulators operates in much the same way (Forbus & Falkenhainer, 1995). The overall development process is illustrated in Figure 12, and the details of the compilation process are illustrated in Figure 13. It relies on a *domain theory* that describes relevant physical phenomena in general terms. Given a specific system to write a simulator for, the compiler starts by figuring out which general descriptions from the domain theory need to be used to understand the system (e.g., in the Evaporation Laboratory, heat flow to and from the atmosphere through the cup needs to be considered as well as evaporation of water from the cup). The compiler starts by creating a conceptual, qualitative description of the system, identifying what physical processes and parameters are relevant. This conceptual understanding is then used to retrieve mathematical models from its domain theory, in the form of equations or code fragments, that are assembled into a quantitative model of the system. The compiler then translates this quantitative model into efficient simulation code. Writing simulation code can be complicated, since changes in the phenomena occurring can lead to significant changes in the mathematical model. For instance, the set of equations that hold when simulating water heating on the stove is very different from the appropriate mathematical model needed to simulate that water boiling. The qualitative model provides the necessary framework for detecting such potential situations, and for writing code to handle them properly.

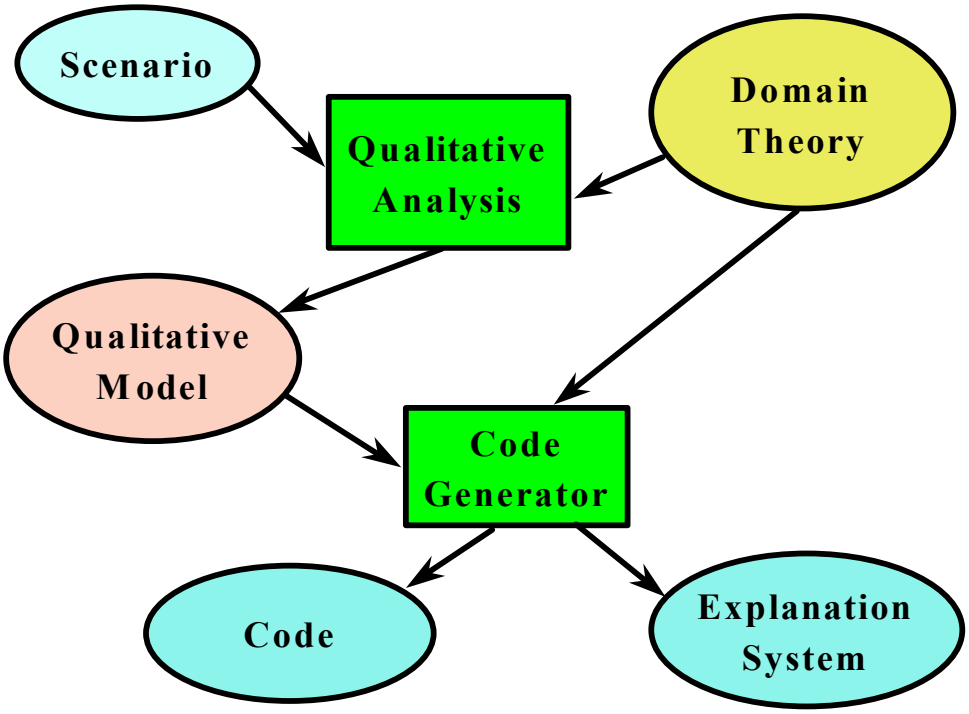

Figure 13: Automatic compilation process for Self-explanatory simulators

The rich explanatory power of self-explanatory simulators comes from exploiting the fact that the simulation compiler itself has a conceptual understanding of what is being simulated. In addition to producing traditional simulation code, the compiler also produces a compact *structured explanation system* (Forbus & Falkenhainer, 1995) that embeds its conceptual understanding of the system into the simulator it builds. Thus the explanations used to explain a simulation are based on the explanations used to generate the simulator itself. The link between the numerical simulated behavior and the conceptual descriptions is maintained by tracking the corresponding qualitative distinctions as they change over time. For instance, when physical processes start or stop or when objects come into existence, disappear, or change in a very significant way (e.g., phase changes), such physical events are noted in a *history* (Hayes, 1985) that provides a qualitative summary of the behavior. This history provides the bridge between the numerical behavior and the causal understanding of the system.

From an algorithmic perspective, we note that self-explanatory simulators can be compiled in polynomial time, as a function of the size of the system to be simulated (Forbus & Falkenhainer, 1995). This is important for scaling up: Simulators involving thousands of parameters can be created quickly. It is equally important to note that the simulators produced are compact and efficient. All qualitative reasoning is done at compilation time, not run time. Thus the simulators produced run asymptotically close in speed to an equivalent numerical simulator for the same system. The only extra overhead is the maintenance of the history, and this requires only a few extra tests per time step and

only requires space proportional to the qualitative complexity of the behavior (i.e., the number of significant physical events), rather than as a function of the time step chosen. This makes them practical in a wide variety of circumstances. For instance, we have run simple simulators on MS-DOS palmtops (8mhz, 640KB of RAM), and as Java applets on web pages.

Turning self-explanatory simulators into active illustrations involves two issues: Selecting the right levels of explanation, and providing the illusion of interacting with a physical system, rather than a complex piece of software. We discuss each in turn.

Providing appropriate levels of explanation: The structured explanation system internally contains the full range of representations used to create the simulation. Not all explanations are appropriate for all audiences: As noted above, middle-school students cannot be expected to understand differential equations. Our solution has been to put filters on the explanation system, to hide information that would be inappropriate for the intended audience. For middle-school students, for example, we focus on the kind of causal information that students are supposed to be learning. As the interaction earlier demonstrated, questions that students can ask include what can affect a parameter and what can it affect. The answers they receive are in terms of causal qualitative relationships (*influences*, in the terms of qualitative process theory (Forbus, 1984)), e.g. "`X can be affected by…`" in the dialog above. 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[2].

Even within the level of causal explanations, it is sometimes useful to filter out information. For example, in the Evaporation Laboratory the concept of thermal conductivity is something that we, in the role of curriculum designers, want the student to discover, rather than telling them about it explicitly. (The inclusion of a diamond cup is intended to lead students in this direction. Few students can resist trying the diamond cup, and since diamond has a thermal conductivity that is orders of magnitude larger than most substances, they are faced with some dramatic behavior differences to explain.) We tackle this problem by a "can't say, don't tell" policy in the software. Each element in the structured explanation system has a natural language phrase associated with it. These phrases are generated semi-automatically by the compilation process; they can be edited separately after the simulation is compiled to support localization. If an explanation element does not have an associated natural language phrase, the explanation system will not use it in any explanation it constructs. Editing tools are provided that enable curriculum designers to adjust the explanation system in this way.

Providing the illusion of interacting with a physical system: Initializing the parameters of even a simple simulation can be complicated, since the choice of parameters must be made with an eye towards physical consistency. Yet the expertise needed to evaluate physical plausibility is part of what we want students to learn from doing simulation experiments. This is a conceptual problem, not a standard HCI problem. Providing a large menu of numerical and logical parameters, even in the cleanest, well-organized GUI, can easily lead to bewilderment. Our solution is to simplify this process by using a

[2] Several teachers have recommended adding a "nerd switch" so that interested students could see the equations. We have not alas had the resources to do this yet.

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, 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). Figures 14 and 15 show the catalog contents currently used in the

Figure 14: Catalog of cups for the Evaporation Laboratory. Students can change the amount and temperature of the water for whatever cup they choose.

Evaporation Laboratory. 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

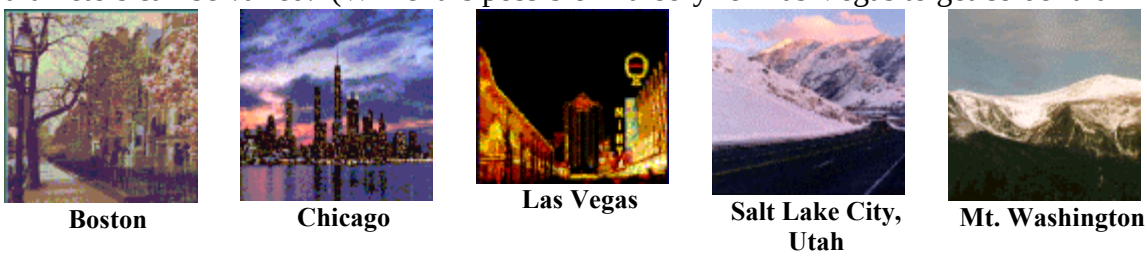

Figure 15: Catalogs of environments used in the Evaporation Laboratory

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 also provide pedagogical benefits, by helping the student see relationships between physical objects and circumstances and their properties. Props also provide a simple path to customization: Adding props representing familiar objects and situations (e.g., a student's favorite cup or home town) also provides a simple form of customization that can make software more engaging.

4.3 How Active Illustrations can be used

There are several settings in which Active Illustrations can be used. We discuss each in turn.

Simulation Laboratories. Student can use Active Illustrations as a laboratory for running experiments, such as the Evaporation Laboratory above. The Evaporation Laboratory and several other simulation laboratories have been publicly available from our web site for several years now, and the reasons given for downloading range from intended classroom use to science fair projects. We are also developing two new simulations, an ecosystem for a hypothetical Mars base and a solar house simulation, to be used in curricula we are developing in collaboration with teachers from the Chicago Public Schools[3] and as motivating phenomena in new research we are doing on helping middle-school students learn how to create models.

Hypermedia component. Active illustrations can be a powerful new type of media in hypermedia systems. A student might start using an Active Illustration included to provide a concrete example of some phenomenon, and branch back out to the rest of the hypertext network based on the concepts in the Active Illustration's explanation system. For example, the on-line Principles of Operation Manual for a simulation based on NASA's Deep Space One autonomous spacecraft used several Active Illustrations to enable players to experiment with basic principles of rocketry and orbits[4].

Virtual artifacts in shared virtual environments. Virtual environments are being explored by many groups as environments for students to interact with each other and instructors in an arena designed to support learning. Because interaction is computer-mediated, such spaces provide additional opportunities for software-based coaching and assessment of student progress. In collaboration with Ken Koedinger and Dan Suthers, we have explored the use of Active Illustrations in *Science Learning Spaces* that support reflection and coaching (Koedinger et al. 1999). In addition to providing preconstructed virtual artifacts, efforts are underway to develop a *construction kit* approach to enable students to significantly modify existing objects, and even create new designs (Erignac, 2000).

With the exception of on-line construction kits, the research groundwork for these applications is already in place. Three things are needed for broad-scale deployment. First, significant investment in software engineering is needed. The software prototypes described above are exactly that: research prototypes. They are robust enough that they can be used in schools, but only by developers with strong expertise. Making it easy for curriculum developers, teachers, and students to create simulators will require making the runtime shells far more robust. Better tools for design, debugging, and tuning of domain theories and simulators are needed, combined in a supportive simulator development environment. Second, libraries of domain theories, created by experts, are needed. With off-the-shelf domain theory libraries and the automated modeling capabilities of self-

[3] This is thanks to the National Science Foundations' Center for Learning Technologies in Urban Schools, a joint project of Northwestern University, University of Michigan, and the Chicago and Detroit public school systems.

[4] http://www.qrg.nwu.edu/projects/vss/docs/index.html

explanatory simulation compilers, the burden of modeling will be greatly reduced for curriculum designers, enabling them to focus more on pedagogical issues. Third, we need to learn how to best exploit this new technology in curricula and activities that achieve educational goals.

5 Discussion

Advances in artificial intelligence, particularly in qualitative reasoning, provide the scientific foundation for new kinds of educational software. *Articulate software*, this chapter has argued, has revolutionary potential for science and engineering education. I believe that software that embodies a conceptual understanding of its domain can help students learn better. As the CyclePad experience shows, articulate virtual laboratories can be valuable in engineering education. As the Active Illustrations we have built suggest, simulators that provide causal, qualitative explanations can help students explore complex physical phenomena.

The examples presented here are, I think, only the beginning. The architectures described here can be applied to a broad set of phenomena and systems to support science and engineering education. And other architectures for articulate software could also be valuable. Consider these:

- *Articulate training simulators*. Combine self-explanatory simulators of a complex system that people operate (i.e., ships, aircraft, spacecraft, power plants) with a model of the goals and context of a system and the procedures for operating that system, to teach someone how to operate that system. By context, I mean what the system is used for and what social and economic, as well as physical, constraints govern its operation. Tankers should not produce oil slicks, for example. The context provides the background needed for the simulator to understand why the procedures are the way they are, and the potential consequences of mistakes. This understanding can be used to set up challenging problems for trainees, and provide better post-mortems than would otherwise be possible (cf. Wilkins & Bulitko, 1999).
- *Articulate game engines*. Computer games can provide a highly motivating setting for students, who happily learn complex ideas for the sake of successfully interacting with and in a simulated world. Often the simulated world underlying these games (e.g., SimCity, SimEarth, Civilization) operate by combining dynamical models with a spatial, map-like model of some sort (e.g., a cellular automata). Domain theories that describe the physics and economics of the simulated world could be used in compiling game engines that embody that conceptual understanding, as a new form of self-explanatory simulator. This conceptual understanding can then be used by in-game tutors, coaches, and opponents (Dobson & Forbus, 1999).

In the long run, I believe software that understands in a human-like way what is to be learned, and uses that understanding to help people learn, will be ubiquitous in education. Someday we will not give students educational software that does not contain such understanding, any more than we would today give them software without a graphical

user interface. There are still many challenges -- scientific, software engineering, and pedagogical design -- to be met before that day will come, but even our first steps are providing enough benefits to convince us that the journey is worth it.

6 Acknowledgements

The work described here rests on basic research funded by the Office of Naval Research: The qualitative reasoning ideas have been developed through the support of the Computer Science Division, and the analogical processing ideas have been developed through the support of the Cognitive Science Division. The research on articulate virtual laboratories was supported by the Applications of Advanced Technology program of the Education and Human Resources Directorate of the National Science Foundation. The research on active illustrations was supported by grants from NASA JSC and NASA Ames, and by DARPA through the Computer Aided Education and Training Initiative (CAETI). Paul Feltovich, Joyce Ma, and Karen Carney all provided valuable suggestions that improved this paper.

7 References

Amador, F., Finkelstein, A. and Weld, D. Real-time self-explanatory simulation. *Proceedings of AAAI-93*.

Baher, J. (1998). How Articulate Virtual Labs Can Help in Thermodynamics Education: A Multiple Case Study. Paper presented at the Frontiers in Education 1998 Conference, Tempe, AZ.

Bell, D.G., Bobrow, D.G., Raiman, O., and Shirley, M.H., 1996, "Dynamic Documents and Situated Processes: Building on local knowledge in field service," IPIC'96, The International Working Conference on Integration of Enterprise Information and Processes, "Rethinking Documents", Cambridge, MA.

Brown, J.S., Burton, R. & de Kleer, J. Pedagogical, natural language, and knowledge engineering techniques in SOPHIE I, II, and III. In Sleeman, D. and Brown, J.S. (Eds.), *Intelligent Tutoring Systems*, Academic Press, 1982.

Dobson, D. and Forbus, K. 1999. Towards articulate game engines. *Proceedings of the 1999 AAAI Spring Symposium on AI and Computer Games.* AAAI Press, March, 1999.

Edelson, D. C. 1992. When should a cheetah remind you of a bat? Reminding in case-based teaching. *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, July 1992.

Erignac, C. 2000. Interactive semi-qualitative simulation. Proceedings of the 14th international workshop on qualitative reasoning (QR2000), Morelia, Mexico. June, 2000.

Everett, J. O. 1999. Topological Inference of Teleology: Deriving Function from Structure via Evidential Reasoning. *Artificial Intelligence* 113(1-2): 149-202

Everett, J. and Forbus, K. 1996. A garbage-collecting truth maintenance system. Proceedings of AAAI-96.

Falkenhainer, B. and Forbus, K. Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence*, **51**, (1-3), October, 1991.

Falkenhainer, B., Forbus, K., Gentner, D. The Structure-Mapping Engine: Algorithm and examples. A*rtificial Intelligence*, **41**, 1989, pp 1-63.

Forbus, K. Qualitative Process theory. *Artificial Intelligence*, **24**, 1984.

Forbus, K. “An interactive laboratory for teaching control system concepts” BBN Technical Report No. 5511, January 1984.

Forbus, K. “Qualitative Reasoning”. *CRC Handbook of Computer Science and Engineering*. CRC Press, 1996.

Forbus, K. & de Kleer, J. *Building Problem Solvers*, MIT Press, 1993.

Forbus, K. and Falkenhainer, B. Self-explanatory simulations: An integration of qualitative and quantitative knowledge, *Proceedings of AAAI-90*.

Forbus, K. and Falkenhainer, B. 1995. Scaling up Self-Explanatory Simulators: Polynomial-time Compilation. Proceedings of IJCAI-95, Montreal, Canada.

Forbus, K., Ferguson, R. and Gentner, D. 1994. Incremental structure-mapping. *Proceedings of the Cognitive Science Society,* August.

Forbus, K., Gentner, D. and Law, K. 1995. MAC/FAC: A model of Similarity-based Retrieval. *Cognitive Science*, 19(2), April-June, pp. 141-205.

Forbus, K.D., & Kuehne, S.E. (1998), RoboTA: An agent colony architecture for supporting education, In Proc. 2nd International Conference on Autonomous Agents (Agents '98), ACM Press, pp. 455-456

Forbus, K. and Stevens, A. ``Using Qualitative Simulation to Generate Explanations" Proceedings of the Third Annual Conference of the Cognitive Science Society, August 1981

Forbus, K. and Whalley, P. (1994) Using qualitative physics to build articulate software for thermodynamics education. *Proceedings of AAAI-94*, Seattle.

Forbus, K.D., Whalley, P., Everett, J., Ureel, L., Brokowski, M., Baher, J. and Kuehne, S. (1999) CyclePad: An articulate virtual laboratory for engineering thermodynamics. *Artificial Intelligence*. **114**, 297-347.

Gentner, D. and Stevens, A. (Eds.) 1983. *Mental Models*. LEA Associates.

Hayes, P. (1985). Naive Physics 1: Ontology for liquids. In Hobbs, R., & Moore, R. (Eds.), Formal Theories of the Commonsense World. Norwood, NJ: Ablex Publishing Corporation.

Haywood, R. W. *Analysis of Engineering Cycles: Power, Refrigerating and Gas liquefaction Plant*, Pergamon Press, 1985.

Hollan, J., Hutchins, E., & Weitzman, L. 1984. STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*, **5**(2), 15-27.

Iwasaki, Y. & Low, C. Model generation and simulation of device behavior with continuous and discrete changes. *Intelligent Systems Engineering*, **1**(2), 1993.

Koedinger, K. R., Suthers, D. D., & Forbus, K. D. (1999). Component-based construction of a science learning space *International Journal of Artificial Intelligence in Education,* 10, 292-313.

Lehrer, R., Erickson, J. (1998) The Evolution of Critical Standards as Students Design Hypermedia Journal of the Learning Sciences; v7 n3-4 p351-86

Ma, J. 1998. A Computer-Based Learning Environment for Teaching High-School Students Feedback Control through Design. Paper presented at the Frontiers in Education Conference, Tempe, AZ.

Ma, J. (1999). A Case Study of Student Reasoning About Feedback Control In a Computer-Based Learning Environment. Paper presented at the Frontiers in Education Conference, San Juan, PR.

National Research Council (1996). *National science education standards.* Washington, DC: National Research Council.

Papert, Seymour, 1980, *Mindstorms: Children, Computers and Powerful Ideas*. NY, Basic Books.

Ritter, S. and Koedinger, K. (1996) Towards lightweight tutor agents. ITS'96 Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs, Montreal, June 10th.

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

Reiter, E. & Mellish, C. Optimizing the costs and benefits of natural language generation, *Proceedings of IJCAI-93*, 1993.

Rickel, J. and Porter, B. 1994. Automated modeling for answering prediction questions: Selecting the time scale and system boundary. *Proceedings of AAAI-94.*

Rutherford, F. & Ahlgren, A. 1990. *Science for All Americans*, Oxford University Press.

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

Stallman, R.M. and Sussman, G.J. 1977. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence* **9**, pp. 135—196.

Weld, D. and de Kleer, J. (Eds.) 1990 *Readings in qualitative reasoning about the physical World*, Morgan-Kaufman.

Whalley, P. *Basic Engineering Thermodynamics,* Oxford University Press, 1992.

Wilkins, D. C. and Bulitko, V. V. (1999), "Automated Instructor Assistant for Ship Damage Control," *Proceedings of the Eleventh Conference on Innovative Applications of Artificial Intelligence, Orlando, FL, July.*

Wu, C., & Burke, T. J. (1998). Intelligent computer aided optimization on specific power of an OTEC Rankine power plant. Applied Thermal Engineering, 18(5), 295-300.

Wu, C., & Dieguez, M. (1998). Intelligent computer aided design on optimization of specific power of finite-time Rankine cycle using CyclePad. Journal of Computer Application in Engineering Education, 16(1), 9-13.