

Authoring Content in the PAT Algebra Tutor **Steven Ritter*, John Anderson*, Michael Cytrynowicz+, Olga Medvedeva***

Abstract:

Most authoring tools for intelligent tutoring systems are targeted towards a broad range of applications. Such systems have expressive power but gain the complexity inherent in any general programming language. This paper considers a different kind of authoring tool, focused on creating content for a specific intelligent tutoring system. The resulting system, called pSAT, addresses the great demand for continuing development of content. A system like pSAT needs to be easily learned by end-users and needs to provide feedback adequate for the user to be able to determine that the system will correctly present the content under a wide range of user strategies, preferences and abilities. We focus on design principles driven by these considerations and conclude with a development strategy that begins with a closely-focused content authoring system and then broadens to a system that can more fundamentally affect the type of content presented by the intelligent tutoring system.

Keywords:

Intelligent tutoring systems, authoring tools, mathematics instruction

Demonstrations:

The Problem Situation Authoring Tool (pSAT) described in this article is available at <<http://domino.psy.cmu.edu:81/best/psat.html>>. An online version of the Practical Algebra Tutor, PAT OnLine, is available at <<http://domino.psy.cmu.edu/patonline.html>>.

Commentaries:

All JIME articles are published with links to a commentaries area, which includes part of the article's original review debate. Readers are invited to make use of this resource, and to add their own commentaries. The authors, reviewers, and anyone else who has 'subscribed' to this article via the website will receive email copies of your postings.

* *Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213, U.S.A.*
{sritter, ja, medol}@cmu.edu

+ *Corporate Design Group, Advanced Technology, Nortel, Canada, cytrynow@nortel.ca*

1. The PAT Tutor

Intelligent tutoring systems are generative. We expect them to be defined in a way that their instructional abilities can apply to a whole range of content, beyond that developed for the initial version of the system. Such systems should also allow a variety of different solutions for the particular exercises presented in the system. These factors present challenges for authors who wish to add content to existing systems. How can authors wishing to extend the system be assured that they are extending it in an appropriate manner? When adding new exercises, how can authors verify that the system will respond appropriately to the range of solutions that students might construct? In order to answer these questions, the author must come to an understanding of the tutoring system's encoding of the new content.

In this paper, we report on a content authoring system that we have developed for the Practical Algebra Tutor (PAT), an intelligent tutoring system for introductory algebra. In our design of this system, we have focused on ways to help an author understand how the system will respond when a student works with it. We emphasize aspects of the design that we believe to be general principles for design of all such systems.

PAT is an intelligent tutoring system developed to help teach introductory algebra¹. The system addresses mathematical modeling of problem situations which can be described by linear equations. A typical problem situation is shown in Figure 1. A web-based version of a portion of this tutor (Ritter, 1997) is now being tested.

1.1 Curriculum

The exercises presented by the tutor, the ordering of those exercises and the classwork performed outside of the tutor constitute the class curriculum. The class curriculum typically used with PAT was developed to reflect the recommendations of the National Council of Teachers of Mathematics (1989). These guidelines emphasize the importance of students being able to understand and appreciate the fundamental mathematics underlying real-world situations. They expect students to be able to understand the mathematical equivalence between different representations and to be able to map between them. That is, students are expected to understand the relationship between a problem situation, an equation, a graph and a table of values. These mappings are emphasized both within the tutor and in classwork. In a typical exercise in the tutor, students are asked to create a spreadsheet and graph representing the problem situation and to create algebraic expressions and solve equations relating to the problem situation.

The PAT curriculum is divided into lessons, which are further divided into sections. Each

section is defined by a set of skills that are required to successfully complete problems in that section. When students demonstrate mastery of a section (by achieving a level of competence on all underlying skills), the tutor promotes the student to a new section, which includes some new skills. For example, students might complete a section containing problem situations that can be modeled as linear equations with a positive slope. Once a student demonstrates mastery of that section, the student advances to a new section that introduces problems containing negative slopes. In this way, the tutor is able to automatically present problem situations to each student based on that student's demonstrated skill level.

Several studies (Koedinger, Anderson, Hadley and Mark, 1995; 1997; Koedinger and Sueker, 1996) have shown that PAT improves student learning relative to control classes, both in high school and college classrooms. In the 1996/97 school year, PAT was used in 22 high schools, 4 middle schools and 2 colleges. These schools include public and private schools in rural, urban and suburban locations. In a typical installation, students work with the PAT tutor in a computer laboratory two days a week. The other three days, their math class is in a standard classroom. The full PAT curriculum, ranging from simple problems with positive slope and no intercept to systems of linear equations in all four quadrants is sufficient to provide a year of instruction for most students.

We believe the success and wide adoption of the system is as much due to the printed curriculum as to the software. Bondaryk's description of the adoption process and the need for workshops (see Bondaryk, 1998, this issue) reflects our experience. Teachers need to understand and feel comfortable with the system and its place in the classroom before they will agree to use it. While we have sent out demonstration disks containing the software, we have found hands-on demos and workshops to be much more effective. Teachers and school administrators need to understand how a full-year class will operate, and it is very hard for them to get a sense of how the year will progress by only trying the software and reading the printed materials. A large part of our training for new teachers involves showing them how a classroom using the software and curriculum is managed and how it supports their curricular objectives.

1.2 The Student's View

Figure 1 shows an image from one problem in the PAT tutor. The student is presented with a problem in words and is asked to create a table (including algebraic expressions for the computed columns) and a graph that accurately describes the underlying mathematical relationships described in the text. Students may use an equation solving tool (itself an intelligent tutoring system) to find the solution to any equations that they create in the process of completing the table and graph. Other tools provide assistance for specific sub-tasks. For example, an "equation support" tool encourages students to use inductive reasoning to construct

an algebraic representation of the expressions in the table.

The behavior of the tools evolves along with each student's abilities. For example, early in the curriculum, the table tool operates as a simple table. Students at this level working on the problem presented in Figure 1 would enter "6" and "1100" in the row representing the values for question 1. Students are allowed to complete the problem in any order, and many will compute the specific instances required in the problem (the "question" rows) before the algebraic expression ("formula" row). If the problem came later in the curriculum, the table would act more like a spreadsheet and compute the "salary" column based on the formula that the student has entered for that column. Thus, early in the curriculum (where the "equation support" tool is of most use), students are encouraged to construct an algebraic representation based on induction from the way they have calculated numerical values for specific instances. Later, they are rewarded for using direct translation from text to algebra, since students who produce an algebraic representation before calculating specific instances will have the instances calculated for them. The graphing tool similarly evolves as students' skills increase.

The screenshot displays the PAT Tutor interface with several components:

- Problem Statement:** A text box containing a word problem about a car salesperson's salary and four questions. The problem states: "A car salesperson is paid a base salary of \$200 per month plus an additional amount of money for each car she sells. She sold four cars last month and received an additional \$500." The questions are:
 - How much would she make in one month if she sold 6 cars?
 - How much would she make in one month if she sold 0 cars?
 - If she made \$2000 in one month how many cars did she sell?
 - If she made \$500 in one month how many cars did she sell?
- Grapher:** A coordinate plane with 'CARS SOLD' on the x-axis (0-10) and 'SALARY (DOLLARS)' on the y-axis (0-10). Settings for 'CARS SOLD' and 'SALARY' are shown as 0 to 10 with a scale size of 1.
- Worksheet:** A table with columns for 'CARS SOLD' and 'SALARY' (DOLLARS). It includes a 'UNIT' row and a 'FORMULA' row. The data entered is:

	CARS SOLD	SALARY (DOLLARS)
UNIT	C	DOLLARS
FORMULA		$200 + 150C$
1	6	1,100
2	0	200
3	12	2,000
4		500
- Equation Solver:** Shows the steps: $500 = 200 + 150C$, Subtract 200, $300 = 150C$.
- Messages:** A text box with the prompt: "What can you do to both sides to get c by itself?"

Figure 1: PAT Tutor screen with partially-completed problem

1.3 Underlying Architecture

The PAT tutor was developed using the Tutor Development Kit (Anderson and Pelletier, 1991). Like all tutors developed with this system, it contains a cognitive model, which is a production system model representing competence in the target domain. The tutor provides feedback through model-tracing. As students work through their solution to the problem, PAT attempts to map student actions onto productions in its cognitive model. If a mapping can be found, the student's action is considered to be a correct step along the path to a solution. If PAT cannot find a mapping from the student's action to an action in the cognitive model, the tutor will indicate that the action is in error. The cognitive model contains many solution paths, so the tutor can follow individual students down the path they choose to pursue. In addition, there are incorrect "buggy" solution paths that the cognitive model recognizes. These buggy paths represent common student misconceptions. If the tutor detects that a student has followed one of these paths, it presents a message intended to clear up the student's misconception.

The cognitive model is, in some sense, an expert system, but it is an expert system that is constructed with the goal of modeling students' thinking about the problem situations which the tutor presents. Each rule in the cognitive model represents a component of the skill required to perform the task. In this way, rules can be mapped to cognitive skills as well as student actions. Previous work with tutoring systems of this type (e.g. Anderson, Conrad and Corbett, 1993) have shown that learning takes place at the level of these cognitive skills, and the tutor measures the progress of students with reference to these underlying skills.

From an authoring perspective, developing a cognitive model involves writing a set of rules that are sufficient to solve all problems within the target domain. Writing such rules is a specialized task that involves more-or-less equal parts of cognitive psychology and programming. However, writing the rules is only half the work involved in producing a working system. The system still needs a curriculum. Writing the curriculum involves creating problem situations that are appropriate for the system and deciding on grouping and ordering information appropriate for that set of problems. Cognitive modeling produces a computer program. Content authoring produces the data on which the program runs. The Tutor Development Kit (TDK) is a tool used to produce the cognitive model; the Problem Situation Authoring Tool (pSAT) is the system used to produce the content. Figure 2 shows the relationship between TDK, pSAT and the PAT system. TDK is used to produce the PAT system. The rules from TDK are programmed into pSAT, so that the content author will be able to see the runtime behavior of the system without starting PAT. A content author can then use pSAT to produce the curriculum database that the PAT system presents to students.

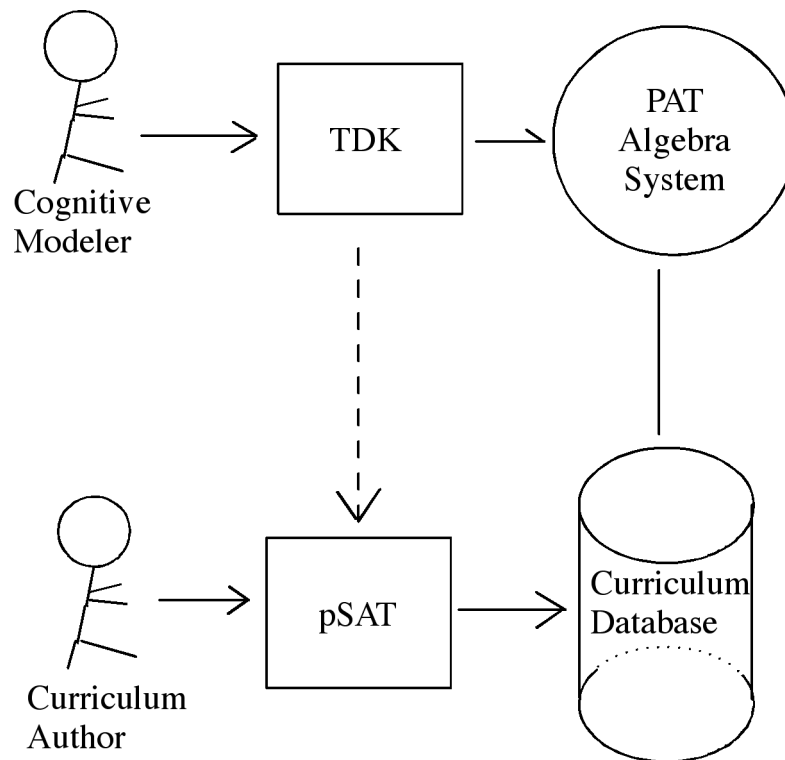


Figure 2: The complete authoring process for the PAT tutor. TDK is used to produce the rules in the PAT Algebra System. TDK rules are introduced to pSAT, which is then used to produce the curriculum database required by PAT.

2. The need for content authoring

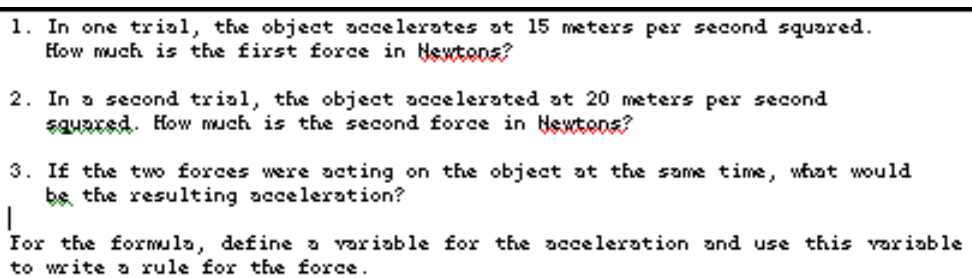
The kind of individualized instruction that PAT provides creates a need for a large number of problem situations. We do not expect any student to work through every problem situation, but we do strive to provide enough problem situations in each section so that every student can demonstrate competence on all skills in that section without repeating a problem. In the past year, we delivered over 300 problem situations with the system. This number of problems was sufficient to achieve this goal for most students ¹, although 400 problems would provide better coverage for some students who need extra help.

¹ *To prevent repetition of problems, the system automatically advances students to the next section if they have solved all problems in the section, regardless of whether they have demonstrated mastery of the skills in that section.*

The number of problems alone does not tell the whole story. There is a continuing need to add new problem situations to the system. This need is driven by several factors. First, we are continually changing the ordering and emphasis on different sections of the curriculum, based on the previous year's results. Some of these changes are driven by changes in the cognitive model, such as when our data suggests that an action that we were modeling as a single skill is better modeled as a combination of two or more skills. Other curricular changes are suggested by teachers or students or are driven by changes in the content or emphasis of the classroom curriculum from year to year.

A second factor in the continuing requirement for new problem situations is the use of the PAT system outside of the local Pittsburgh area. The problem situations encoded into the system were selected to be mathematically relevant but also to be interesting to our students. In many cases, the problems refer to Pittsburgh events, sports teams and local concerns. We feel that the addition of local content increases student interest, and we would like each location using the tutor to be able to customize the problem situations with local content. Students in Florida are less likely to be interested in problem involving shoveling snow than our Pittsburgh students.

The ability to add new content allows the possibility of developing curricula defined by a common theme. We could develop sections or lessons devoted to, for example, biology, history or sports. Some schools might wish to let students choose a theme which they are interested in. The existence of thematic lessons would allow schools to employ the tutors as an element of "math across the curriculum" programs. In these implementations, the PAT tutor would be used in a course outside of the math department, as a way of illustrating the mathematics inherent in the course's major subject. We have developed a small curriculum focused on physics problems that has been piloted in this manner (see Figure 3).

A screenshot of a physics mini-curriculum problem situation, enclosed in a black rectangular border. The text is displayed in a monospaced font. It contains three numbered questions and a final instruction. The first question asks for the first force in Newtons given an acceleration of 15 m/s². The second question asks for the second force in Newtons given an acceleration of 20 m/s². The third question asks for the resulting acceleration if both forces act simultaneously. The final instruction asks the user to define a variable for acceleration and use it to write a rule for the force.

1. In one trial, the object accelerates at 15 meters per second squared.
How much is the first force in Newtons?

2. In a second trial, the object accelerated at 20 meters per second squared.
How much is the second force in Newtons?

3. If the two forces were acting on the object at the same time, what would
be the resulting acceleration?

|
For the formula, define a variable for the acceleration and use this variable
to write a rule for the force.

Figure 3: Problem situation for physics mini-curriculum

If yearly changes in curriculum and customization for local content were the only factors driving problem authoring, we would be satisfied with a specialized authoring tool restricted in its use to well-trained individuals with sufficient technical and mathematical knowledge. However, we

have reason to believe that the process of authoring content for the system can provide teachers with new ways of thinking about their subject and their students.

As we have moved our tutoring systems out of the laboratory and into the classroom, we have become increasingly aware of the importance of providing adequate teacher training and support (Koedinger and Anderson, 1993). Only a small portion of this training and support is related to the technology itself. Much more of the teacher's adjustment to a classroom using PAT involves understanding the goals and methods of the curriculum. It is essential that the teacher is able to relate work done with the tutor to that covered in the classroom on the three days a week when the students are not using the tutor. Teachers can be more effective when they understand the principles behind the cognitive model embodied in the tutor. Many teachers report that the tutor helps them to think about the subject differently. The cognitive model underlying the tutor represents a way of thinking about algebra that is often new to teachers. Studies of students solving algebra problems have revealed that factors determining the difficulty of an exercise are often unappreciated by teachers (Nathan, Koedinger and Tabachneck, 1997). An understanding of the tutor's cognitive model, then, becomes a way of the teacher understanding how students think about algebra. This improved understanding can, in turn, lead to improved teaching, both with and without the tutor (Carpenter, Fennema, Peterson, Chiang and Loef, 1989).

One of the most effective ways to come to an understanding of the tutor's goals is to author curriculum for it. While we don't expect all teachers to author a substantial number of problems, we can expect each teacher to enter one or two, just to get an understanding of the process. We are in the process of incorporating problem authoring into our standard training for new teachers using the system. As a more ambitious goal, we would like to provide the ability for exceptional teachers at each site to add substantial content to the system. Our experience agrees with that described by Roschelle, Kaput, Stroup and Kahn (1998, this issue) in finding that, while many teachers profess a desire to author curriculum, only a small percentage follow through on this desire.

This discrepancy which might seem especially puzzling when you consider that teachers are very willing to author more traditional instruction (lectures and classroom demonstrations). Authoring for computer-based systems may be different because such authoring systems, including pSAT, require a significant amount of time, commitment and specialized knowledge to use, compared to writing a new lesson plan or setting up a demonstration. In both traditional and computer-based instruction there is little reward for teachers who take the time to create new material, other than the possibility (sometimes unrealized) that the material they create will make their own classes more interesting and informative for their students and themselves. Currently, the risk/reward ratio is sufficient to encourage only the most committed and

computer-literate teachers to author content for computer-based environments. Looking to the future, the Educational Object Economy ² (introduced in Spohrer, Sumner and Buckingham Shum, 1998, this issue) and the type of aggregation of content described by Marion and Hacking (1998, this issue) may provide rewards for teachers that extend beyond the impact of the new material on their own classes. The ability to distribute educational material widely and to reward (either monetarily or through increased status and recognition) the authors of this material may provide enough incentive to encourage a wider base of teachers to participate.

The rise of the World Wide Web provides the possibility that teachers from each site using the tutor could author content and be able to share that content with teachers and classes at other sites. The shared experience of problem authoring can create opportunities for teachers to discuss their approaches to teaching algebra and their understanding of students' thinking. We expect this sharing of content to lead to discussions among teachers about classroom experiences with the tutor and with algebra instruction in general. Eventually, we expect the authoring system to become one anchor in a virtual environment that supports communication between teachers about such issues as the best way to explain difficult concepts and the appropriateness of particular problem situations to their curricular goals.

Problem situations created with this authoring tool become objects in the object economy in several ways. Most obviously, they become content which other teachers can use within the PAT tutor. However, this is a fairly conventional use of the content, and is limited in application. More interesting is the possibility that these encoded problem situations will acquire value outside of the PAT tutor. The authoring tool's encoding becomes a description of the cognitive processes required to solve the problem. Teachers can read the problem text and understand more about the challenges that it poses to students. The problem can then be used to teach effectively outside of the PAT tutor, perhaps by extending pSAT to produce a paper-based lesson plan featuring the encoded problem situation.

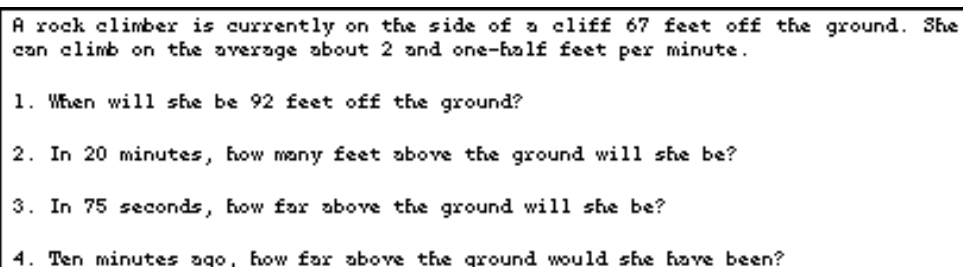
3. The Difficulty of Content Authoring

The development of a content authoring tool for the PAT system that would be usable by high school mathematics teachers is a formidable challenge. PAT's production rules are general enough to apply to any problem situation modeled by a linear equation, regardless of the subject matter or position in the curriculum. A large part of the student's problem solving process (and thus the tutor's cognitive model) is devoted to understanding the relationship between the text describing a problem situation and the underlying mathematics. In order for PAT to be able to assist a student in solving a problem, the tutor needs to understand this relationship. If a new problem is a simple variant of an existing problem (as would occur if the new problem differed only in the numeric constants or mathematically-unimportant details of the text), the

² *Educational Object Economy, The EOE Foundation* <<http://www.eoe.org>>

existing problem could be edited with a simple text editor to create the new problem. A more difficult process is required, however, when the new problem represents a fundamentally different way of presenting a mathematical situation. Since students are best prepared when they have had experience with the broadest range of problem situations, a good curriculum should include many different situation types. Our continuing improvements to the curriculum often include changes of this type.

In order to understand what we mean by a fundamentally different way of presenting a mathematical situation, compare the problem presented in Figure 1 to that in Figure 4. In the “rock climber” problem presented in Figure 4, the relationship between time and altitude is stated explicitly. Students may incorrectly encode the direction of this relationship or confuse the slope with the intercept, but almost all have no problem recognizing that the correct mathematical relationship has something to do with the phrases “67 feet off the ground” and “2 and one-half feet per minute.” In contrast, there is a different process involved in translating the wording of the “car sales” problem presented in Figure 1 to a symbolic form. In that problem, students need to understand that the most appropriate form of expression to compute salary involves changes after one month, not four months. In addition, they must realize that the proper way to calculate a one-month change when given a four-month change is to divide by four.



A rock climber is currently on the side of a cliff 67 feet off the ground. She can climb on the average about 2 and one-half feet per minute.

1. When will she be 92 feet off the ground?
2. In 20 minutes, how many feet above the ground will she be?
3. In 75 seconds, how far above the ground will she be?
4. Ten minutes ago, how far above the ground would she have been?

Figure 4: Problem situation in which rate of change and y -intercept are explicitly stated.

It should be clear that the tutor’s cognitive model treats these exercises differently. The tutor recognizes that they involve different component skills and require different solution strategies. Our concern here is how an author can encode the “car sales” problem in a way that makes these skills and strategies apparent. Ideally, authors would just type the problem text and be confident that it was understood. Unfortunately, the natural language understanding abilities required by that kind of solution are beyond our capabilities. Instead, the process of entering content requires entering the problem text as well as some hints about how to interpret the problem situation. Due to the imperfection of natural language understanding, a large part of the authoring process involves trying to understand and, when necessary, correct the system’s

(mis)understanding of the problem situation.

4. Designing the Content Authoring Tool

The preceding discussion has related some of the issues involved in developing a content authoring tool for the PAT tutor. Much of our discussion has focused on natural language understanding (and the absence of a reliable solution to that problem), and one might wish to attribute the difficulty of content authoring to the natural language problem. It is true that, if our tutoring system did not have to understand English-language problems, authoring would be easier. However, we believe that content authoring for intelligent tutoring systems faces a more fundamental problem. The author of content must understand what the system will do in presenting that content to a student. Since students will differ in their approaches and backgrounds when they confront that content, an author needs to know how those different approaches and backgrounds will be reflected in the tutor's behavior.

The first 300 or so problem situations currently available with PAT were entered by two teachers and one research programmer, using three successive versions of a content entry tool developed for use with PAT. Although we have seen gradual improvements in this authoring tool, it was apparent that we had not reached our goal of enabling any algebra teacher to be able to author content for the system in a short amount of time, following a minimal amount of training. Instead, we found that substantial training was required to master the authoring tool, and entry and testing of each problem situation took, on average 1 to 1.5 hours. Our latest revision of PAT's content authoring tool (dubbed pSAT, for "problem situation authoring tool") began as a collaboration with Carnegie Mellon's design department 3. One result of this collaboration has been a set of design goals that we believe need to be considered in creating an effective content authoring tool.

In particular, we advise authors of content authoring tools to:

1. *Reconsider the system's goals.* In particular, a content authoring tool must be aware of the context in which exercises are encountered. For our system, when developing an exercise, it was important to consider the place of an exercise in the larger curriculum.
2. *Make the abstract concrete.* Whenever possible, provide the author with examples, rather than abstract rules and grammars.
3. *Design for testing.* The system's behavior can vary depending on a student's background and approach to a problem. Content authors need to get a reasonable sense of how the system will behave in a variety of situations.

4. *Make the cognitive model visible.* This is, perhaps, the most fundamental principle. The author must get a sense of how the system thinks about an exercise.
5. *Build from specific to general.* Content authoring systems are, by definition, more narrowly focused than authoring systems that allow the complete creation of a cognitive model from the ground up. In many cases, however, the narrow focus can be broadened to address a much wider range of content than was originally envisioned.

None of these principles are unique to content authoring and many can be seen as specific instances of more general design principles (e.g. Norman, 1990). The remainder of this paper illustrates these principles with examples from the design and implementation of pSAT. It is hoped that these illustrations will provide more general guidance to developers of content authoring tools.

4.1 Design Principle 1: Reconsider the system's goals

Our first attempt at authoring content was simply a text editor. The content author was required to enter the representation of the problem situation by typing out the working memory elements representing the problem in the cognitive model. This was never meant as a problem authoring tool; it was simply a way to get started until a real authoring tool was developed. Still, subsequent authoring tools adopted the same goal as this system: the goal of the authoring tool was to produce a file which could be used by PAT. The task of creating that file was separated from that of associating the file with a particular section or lesson. Even after the second revision of the content authoring tool, the creation of sections and lessons involved constructing a text file describing the lesson, in a manner similar to the original content authoring tool.

Grouping problems into sections and lessons can be a deceptively tricky task. In order to do this properly, the author must understand the skills required to correctly solve the problem situation, the skills covered in the section and the position of the section within the curriculum. For example, if a section is intended to teach negative slopes, the inclusion of a problem situation using a positive slope will be wasted in that section. The tutor will never choose to show that problem to a student, since completing that problem could never increase the student's ability to solve problems with negative slopes.

In the worst case, it is possible to define a section such that no problem situations included in the section involve one or more of the skills that the system requires students to demonstrate mastery over in that section. In this case, no students will be able to graduate from the section, since they will never be given the opportunity to demonstrate their skills in that area.⁶ Although no author would intentionally structure the curriculum in this way, it is too easy for authors to

ignore the position of the section within the curriculum, leading to this kind of error.

As described above, the system changes the tools students are asked to use as they move through the curriculum. When tools for solving an exercise change, the skills required to solve the problem change as well. Thus, a problem situation may require a different set of skills for solution if it is included early in the curriculum than if it is included later in the curriculum. As a simple example, students working with a spreadsheet (as opposed to a table) may not use the skills involved in calculating the result of substituting a constant into an algebraic expression. (In our curriculum, they have typically demonstrated mastery of these skills in earlier exercises.) In this case, the way that the tutor treats an exercise and the skills that the tutor determines are required to complete the exercise depend on the position of the exercise in the curriculum, in addition to the content of the exercise itself.

The close relationship between particular problem situations and sections is reinforced by the conditions surrounding authorship. As mentioned earlier, one goal for the system is to provide enough content for each section so that no student repeats a problem. In a section involving a skill like graphing in the third quadrant, there are few good real-world situations that cover the appropriate skill. In such cases, the problem author approaches the task with the goal of thinking of a problem situation that addresses the appropriate skill, rather than starting with a problem situation in mind (as is often the case). Such authors would clearly benefit from a close relationship between problem authoring and curriculum structure.

In pSAT, there is no separation between authoring a problem situation and placing it into a curriculum (see Figure 5). We encourage authors to think about the output of the authoring task as a curriculum, not an individual problem file. pSAT includes several mechanisms to support this goal. When a problem situation is entered into the system, it can be placed into one or more “collections,” which are categories of problems grouped by any means the author desires. Some collections represent problems created by a single author, while others represent thematic or skill-based groupings.

Lessons and sections are represented as folders, in a manner similar to the Macintosh’s™ finder. Once a problem situation is entered into a collection, it can be dragged into one or more section folders. Section folders specify the skills to be mastered in that section. Run-time parameters (like the form of the spreadsheet/table and grapher) can be assigned to lessons, sections or individual problems within a section. As these parameters are assigned, pSAT verifies the consistency between the section skills and the problems assigned to the section.

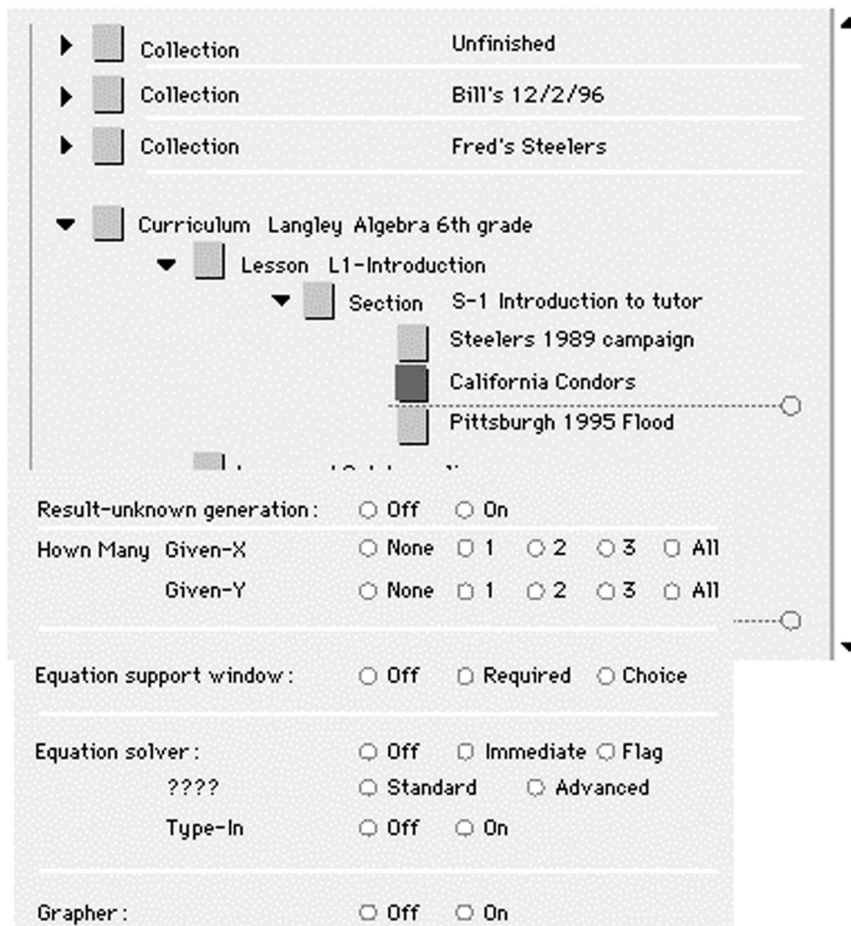


Figure 5: Organizing collections, lessons and sections and setting run-time parameters in pSAT.

4.2 Design Principle 2: Make the abstract concrete

The second design principle is illustrated by a difficult issue in authoring word problems for PAT. Consider the problem situation presented in Figure 1. Part of the solution to this problem would involve creating a spreadsheet with two columns, one representing the number of cars sold and the other representing the salary. In order for PAT to understand the student's solution to the problem, it needs to be able to tell whether a student's entry is appropriate for the column and which column the student's entry is supposed to represent.

Although PAT is designed to avoid natural language understanding problems, this is one area where we were unable to do so.⁵ In fact, we do not require as strong a solution as many natural language situations. It is acceptable for us to have a high rate of certain kinds of false alarms. Ungrammatical phrases may be acceptable, since we are really only concerned with the semantics of the phrase. In addition, it is acceptable for the column grammar to accept nonsensical phrases (such as “months earns salary”), since they are unlikely to be generated by students. On the other hand, the worst kind of error for the system to make is to interpret one column label as the other. In this case, the tutor would identify the student as entering a correct column label, but the student would have difficulty completing the rest of the spreadsheet in a way that would satisfy the tutor, since the tutor misunderstands a fundamental part of the student’s solution. This kind of error could occur, for example, if the system interpreted “receipts from cars sold” as a label for the “the number of cars sold.”

There are two basic approaches to solving this problem:

1. Provide natural language capabilities in PAT so that it can recognize appropriate labels. One version of this approach would be to extract appropriate column labels directly from the problem statement and identify semantically-equivalent phrases; another would be to have the author enter a single column label (or small set of column labels) and have the system generate synonymous phrases.
2. Have the author construct a set of allowable labels. This can be supplemented by some simple generalization routines at run-time. For example, we can recognize misspellings of acceptable labels and omissions or additions of function words.

The first approach might be the best one to implement, if we could do it properly. However the natural language problem is difficult and there is a real possibility of producing errors that result in the misattribution of a column label to the wrong column. A subtler problem with this solution is the fact that, if the natural language understanding is less than perfect, the author’s expectations of the system will not align with the system’s actual performance, and it would be nearly impossible for the author to know what was going wrong. It would be difficult for the author to understand why an unseen grammar (assumed to be the author’s own English grammar) would accept one phrase and not another. How could the system give the author feedback about its interpretation of a column label? Finally, the natural language solution presents a problem to the author who wishes to correct the system’s interpretation. If the system misinterprets the problem so that it assigns improper semantics to a column, how can the author correct this misinterpretation?

The second approach has the obvious drawback that it shifts the burden from the system to the

author. However, due to the difficulty of implementing the first class of solution, this is the approach we decided to take. This left us with the problem of identifying a reasonable system with which the authors could list appropriate column labels.

In previous versions of the content authoring tool, we simply had users list as many labels as they could. This system was used to author over 300 problems and proved to be inadequate. In many cases, students produced acceptable phrases that were not anticipated by the author. In some cases, the number of phrases coded becomes unwieldy, posing problems of organizing them in a reasonable manner. For example, the author of the problem shown in Figure 1 identified 18 different labels for the Y column. Other problems include close to 100 phrases for a label. Any attempt to list this many phrases leads to inconsistencies that make it difficult to predict which phrases will and which will not be accepted for a particular column. In some cases, a particular syntactic construction was defined for one key word but not for one of its synonyms. For example, the author of this problem included “amount of money she made” and “salary earned” but not “amount of salary she earned.” There is no principled way for a student to predict which phrases will be included and which will not.

The solution to this problem must involve a way of verifying that the set of appropriate phrases was complete (in the sense of defining each syntactic form over each appropriate synonymous term) or a way of generating phrases in a manner that encourages completeness. Our initial approach to this problem was to define a simple grammar that the author could use to generate phrases. However, it was soon clear that any such grammar would require a significant amount of time for authors to master and that it was difficult for authors to determine whether the resulting grammar was adequate.

Our solution (suggested by the third author) was to approach the problem from a design rather than a computer science perspective. Instead of focusing on the formal grammar describing appropriate phrases, we reconceptualized the problem as needing to provide a system that allowed the author to think of new words and phrases for the desired concepts. The resulting system allows authors to “brainstorm” about appropriate phrases, quickly see the phrases that are generated and eliminate those that are inappropriate.

Formally, the system implements a kind of grammar, but the author works with it in a visual manner. The resulting “phrase authoring tool” is remarkably simple and effective. First, the author enters a base phrase (see Figure 6). The base phrase becomes one appropriate phrase in the list, and each word in the phrase appears in a button along the left side of the phrase authoring tool. The author can then construct permutations of words in the base phrase by clicking on the buttons in any order. Once the author has exhausted the permutations suggested by the base phrase, that phrase can be edited. For example, the author could substitute the word

“makes” for “earns” in the base phrase “the amount of salary she earns in a month”. Next, the author can “regenerate” the phrases, which duplicates the existing phrases using “makes” in place of “earns”. This process ensures that syntactic constructions are repeated for all synonyms. A thesaurus provides access to potential words to use as synonyms. Additions and deletions from the base phrase can also be used to regenerate a set of phrases. For example, the introductory phrase “the amount of” could be added to the beginning of all phrases beginning with “salary.”

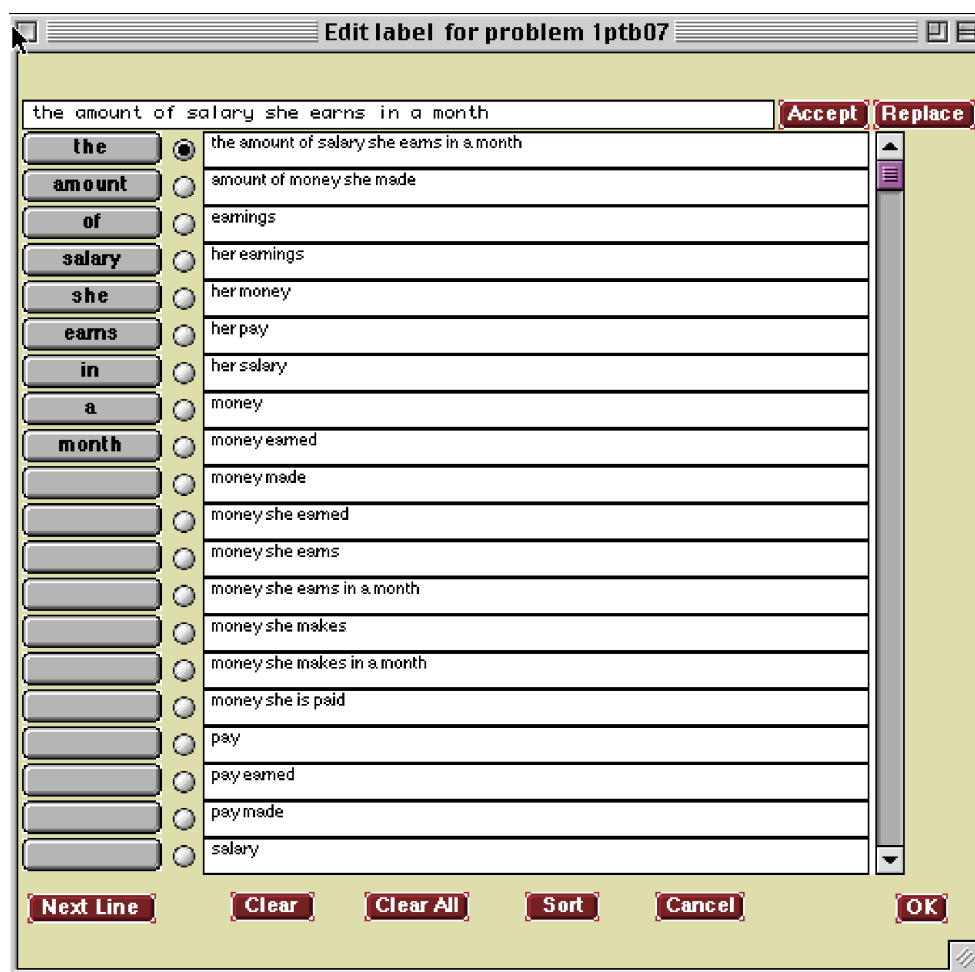


Figure 6: *The label authoring subtool*

This subsystem works well. We believe the major insight leading to the design of this system was

to think of the author not as trying to enter and organize a set of pre-existing phrases but as trying to brainstorm about the phrases themselves. Although the system can duplicate the behavior of a generative grammar, it does so in a very concrete manner. Instead of thinking about syntax abstractly, authors need only think about word order. Since the results of a “regeneration” are available immediately, it is easy to correct, adjust and add phrases that do not match the abstract pattern.

4.3 Design Principle 3: Design for testing

A significant task in designing an authoring system for a system that has an intelligent component is that, for any sufficiently sophisticated system, it will be hard to predict the system’s response in a given situation. Consider the situation of an author who has entered the problem situation given in Figure 1. After entering the problem, the author may want to test whether the expression “200+600x” (an error that ignores the fact that the bonus was paid for four cars sold, not one) will produce an error message when entered as a formula for the Y column. This seems, to the author, to be a mistake that students might make, so the author wants to find out what message will be given to a student who makes this error.

One obvious way to verify the behavior of the system is to run the system as a student would and produce the sequence of actions that you want to test. This was our approach in earlier versions of the content authoring tool.

In this example, the author could run the system as a student and try entering “200+600x” as the formula. Consider what the author would actually have to do to test this fairly simple behavior. First, the author exits the authoring environment and enters the student environment. Next, the author needs to decide which student actions need to be performed in order for the system to produce the error message (if, indeed, there is to be one). It seems reasonable to assume that the author must first enter a column label for the given column, but the author also needs to decide whether it makes a difference that the unit of measure for that column is also entered or that the label for the other column is entered. In the absence of a clear understanding of the cognitive model underlying the tutor, these are difficult decisions to make. Of course, the answers can be empirically determined by trying the entries in different ways, but the author quickly becomes enmeshed in a combinatorial explosion of possible conditions.

In general, testing a system is a task analogous to scientific discovery. The author has a hypothesis about how the system will work and needs to design experiments to test that hypothesis. This can be a difficult task because it requires the author to think of appropriate test conditions, to develop a system for running each condition identified (so as to ensure that none were omitted) and to keep careful records of the results of executing each condition (Klahr and

Dunbar, 1988). In the authoring case, there is an additional complication. When errors are found, the author must be able to correct the system's action, and the way to do this is not always obvious.

In practice, using earlier versions of the content authoring tool, authors would enter a new problem situation and test it two or three different ways before assuming that everything was correct. Even so, our observations of authors indicated that the majority of time spent with the tool was in testing and correcting the system. In many cases, errors persisted until the problems were encountered by large numbers of students, who sometimes approach the problem in ways not considered by the author.

In pSAT, testing is done differently. Instead of running through the problem as a student, the author can look at a canonical student solution. This canonical solution is annotated with information indicating the system's behavior in cases where students deviate from the solution. To verify the system's behavior in our example, the author enters the basic problem information. By going to the cell representing the given value for question 1, the author can see all help and bug messages for this cell and easily determine whether there is an appropriate message presented if a student enters 25. Of course, if desired, the author can also run the problem as a student.

The use of a canonical solution works well for our domain, in part because there are relatively few path dependencies. In the PAT tutor, students can come up with solutions that differ in many ways, but only the choice of a column for the dependent variable and a letter for the variable have implications for the completion of subsequent cells in the spreadsheet. In the programming and equation solving domains, however, choices at each point do lead to different alternatives at later points in the solution. Later, we describe an extension of this system that can assist in such cases.

4.4 Design Principle 4: Make the cognitive model visible

One of the advantages of the TDK is that it unites the cognitive model, help text and skills used for knowledge tracing into a single system (Anderson and Pelletier, 1991). This allows the author of a TDK-based tutor to write a cognitive model for performing the task in a manner similar to that used to write an ACT-R model. The tutor code and ACT-R code differ only in that the tutor rules need to be supplemented with templates for help text and identified with a user-readable skill name.⁷ In practice, authors of TDK-based tutors approach the problem in exactly this manner. They write and test cognitive models and only later add the features that allow the rules to be used in a tutor. This provides a flexible and familiar environment for these authors, but it poses a problem for the very different group of people who enter new content

(in the case of PAT, problem situations) into the system. The problem is that the cognitive model resulting from the task analysis is expressed only in the production system formalism used by the TDK. For content authors to be able to understand this formalism, they would need to learn to read the programming language in which it is expressed. This is clearly an unreasonable requirement.

Consider the following rules from the PAT tutor:

```

R1: IF a problem situation involves an initial value, THEN determine the
    magnitude of the initial value.

R2: IF a problem situation involves a constant rate of change, THEN determine
    the magnitude of the rate of change.

R3: IF your goal is to produce a formula representing a problem situation and
    the problem situation involves an initial value and a constant rate of
    change and the magnitudes of those values are known, THEN enter <rate of
    change>X<initial value>. [Help template 1: "Review the problem statement.
    What is the relationship between <heading> and <heading of x column of
    problem>?" Help template 2: "First consider the initial value of
    <heading>. Next consider how <short heading> will change as <heading of x
    column of problem> changes." Help template 3: "Write an expression that
    means the same thing as <heading> <sign operator of slope of expression>
    the change in <heading> for each <singular of unit of x column of problem>
    times <heading of x column of problem>.", Help template 4: "Write an
    expression that means the same thing as <intercept of expression> <sign
    operator of slope of expression> <absolute value of slope of expression>
    times <heading of x column of problem>."]
    
```

Rules R1 and R2 model understanding the problem text to the extent that the student can identify the magnitudes of the slope and intercept of the equation. Rule R3 takes that classification and uses it to determine an appropriate action to take. Rule R3 also shows the help text associated with this step in the problem solution.

These rules are clearly too complex and abstract (even in the simplified form shown here) to present to authors as a way of explaining the behavior of the system. However, a different presentation that recognizes the two kinds of rules in cognitive models is easily understood by authors. Cognitive models tend to have perception and action components (Simon, 1975). In the example rules, R1 and R2 are perceptual rules, because they involve extracting information from the problem statement and encoding the equivalent values in memory. R3 is an action rule, which takes the encoded values and suggests a course of action (as well as some help text).

While the TDK treats perception and action rules equally, pSAT treats them differently. In general, the conditions of perception rules are complicated and difficult to express easily. The fact that the problem shown in Figure 4 suggests a rate of change of 2.5 feet/minute depends

on a complex combination of semantic knowledge and rules of English grammar. A solution to the problem of understanding text does not yet exist and, even if it did, it would serve little purpose to try and explain how that solution applied to the problem that the author was trying to enter into pSAT. Instead, pSAT simply presents the output of the perception rules. In this case, the perceptual rules output a schema classification, which represents the process that the student needs to master in order to produce an algebraic representation relating the quantities and a set of numeric values that are used as the inputs to that process (further details about the schema encoding and operation of perceptual rules in pSAT are available in Ritter, 1998).

Action rules like R3 take the schema and numeric values output by the perception rules and suggest changes in the problem and, when appropriate, hint text. Thus, the author can understand an action rule simply by reading the problem-specific text that the action rule produces. For example, the author of the problem in Figure 4 could verify the appropriateness of rule R3 by reading the following problem-specific help text:

- 1 - Review the problem statement. What is the relationship between the height of the climber and the time?
- 2 - First consider the initial value of the height of the climber. Next consider how height will change as the time changes.
- 3 - Write an expression that means the same thing as the height of the climber plus the change in the height of the climber for each minute times the time.
- 4 - Write an expression that means the same thing as 67 plus 2.5 times the time.

By verifying that the generated help text (like that shown above) is appropriate for the problem, the author is verifying that the correct action rule has applied.

Consider what would happen if the system incorrectly generates a help message indicating that the formula should be $67-2.5x$. The author could check the help messages and see that the 3rd and 4th help messages used “minus” instead of “plus”. Inspecting the output of the perception rules would reveal that the system (due to a faulty perception rule) had inferred that the rate of change for this problem situation was -2.5 . The author could then correct the system’s encoding of the rate of change to be 2.5 , and this change will propagate to action rules. In this manner, every help and bug message referring to the rate of change automatically updates, in this case changing “minus” to “plus” in certain messages. Besides providing a convenient way to correct

the model, the method focuses the author on understanding the way that the system interprets the problem situation and the way that the cognitive model affects the messages that students see.

There is another example of the way that pSAT makes the relationship between problem encoding and particular messages apparent. As the author enters problem information, pSAT is continually generating and updating help and bug messages. As those updates propagate through the system, the spreadsheet cells (or grapher elements) to which these messages refer briefly flash. This gives the author immediate feedback on the set of elements that are affected by the content just entered.

4.5 Design Principle 5: Build from specific to general

As described to this point, pSAT is a tool that allows authors to enter problem situations, organize curricula and verify the behavior of the student system. The set of perception and action rules in pSAT was designed as a way of providing content authors with feedback about the behavior of the system on a particular problem situation. The system described is appropriate for our target authors, who are some subset of the teachers and/or curriculum administrators at each site. In this section, we consider extensions to this authoring environment that bring us from content authoring to authoring of cognitive models. We make this transition in two steps, describing authoring tools that serve different goals and authors with different amounts of technical expertise. The approach of moving from a specific content authoring tool to gradually more general authoring environments may hold promise for the design of such systems.

4.5.1 Customizing the Action Rules

At this level of authoring, we consider the perceptual rules to be fixed and inaccessible. That is, the portion of the cognitive model that determines which features are present in a particular problem situation is taken as given, although, as described earlier, the particular encoding assigned to a problem situation (i.e., the output of these rules) can be overridden.

There are two keys to allowing this level of authoring. First, we describe all properties of the encoding of the problem situation as properties of familiar objects in the system. For PAT, the main objects are the problem itself, the rows, columns and cells of the spreadsheet and the manipulable elements of the graphing tool. The second key to this kind of authoring is presenting the action rules in an accessible manner. To do this, we define a set of rule hierarchies, each applying to an individual element that the student would complete (see Figure 7). Each rule contains a single condition, with rules lower in the hierarchy inheriting the conditions of

rules above them. The top rule in the hierarchy simply specifies the action the student is taking (such as the cell in the canonical form of the spreadsheet). Properties of various objects can be themselves properties. For example, a cell has a column property, whose value is the column object containing that cell. Columns, in turn, have expressions, which are objects representing the algebraic expression used to calculate the column. Expression objects have a property called “form,” which is the general form of the expression (such as “mx” or “a(x+b)”). It is possible, then, for a rule attached to a cell to refer to the cell’s “form of expression of column” in either the condition or text of the rule. The help templates shown in rule R3 use this kind of notation. Although, in the current version of the system, properties and object references are written as text, the system could be extended to allow references to objects and properties to be identified visually by pointing to the appropriate objects.

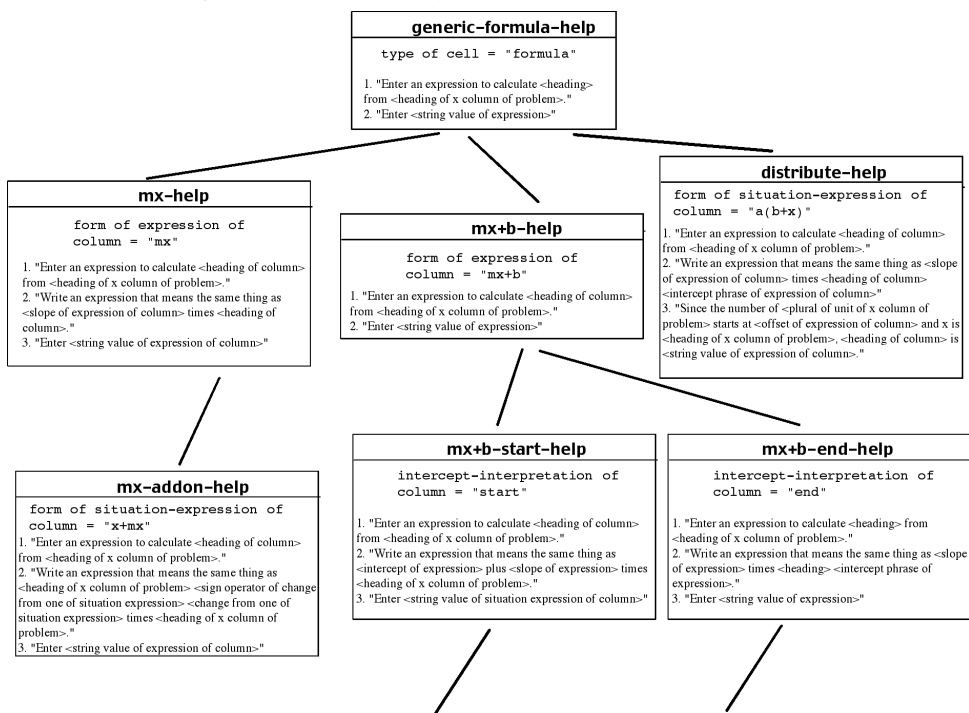


Figure 7: Part of the help rule hierarchy for the formula cell in the spreadsheet

4.5.2 Customizing the perception rules

This next level of generality allows true changes to be made to the cognitive model. From the content author’s point of view, the cognitive model running in pSAT provides a set of

suggestions for feedback to students in particular situations. We intend for the feedback automatically provided by the system to cover the vast majority of cases in every problem situation, but we have not, to this point, provided a way to extend the system to allow authors to modify the rules themselves. Such modification might be appropriate to implement extensions to the cognitive model or to the range of problem situation types that we wish to include in the system. Although PAT's cognitive model applies to all problem situations modeled by a linear equation, help and bug messages may be more appropriate to some kinds of problem situations than others. For example, in an early curriculum, we did not include any problems in which the relationship between the variables is represented as a percentage. Although these kinds of situations are modeled mathematically in the same way as other linear relationships, they can lead to different kinds of errors. For example, students are prone to model 25% as $25x$, rather than $0.25x$ in these types of problems. If we wish to provide bug messages that are specific to percentage problems, we need to modify the cognitive model.

The idea is not to replace the initial development of the system in a TDK-like tool. As described earlier, the TDK is a very flexible and powerful tool for developing the cognitive model. However, once the cognitive model is developed and debugged using a TDK-like tool, we can identify the perception rules and build them into the content authoring tool. This process makes the cognitive model more accessible to the content author and allows the addition of some extensions to the system's behavior. The resulting system allows a more general form of authoring than is possible with the first two levels of authoring.

The perception rules in the current system heuristically interpret the problem statement and other elements of the problem encoding to determine the encoding of the problem in terms of object properties. Currently, this is a programming task, but we are working on ways to make this task more accessible to non-programmers.

The general approach described here works well with the PAT tutor, in part, because most of the relevant objects in the PAT problem solving structure are fixed. For a given problem situation, every solution will contain a spreadsheet with a certain number of rows and columns and a graph with a certain, fixed set of labels, bounds and parameters to set. This is a peculiarity of the PAT tutor, not model-tracing tutors in general. In fact, the model-tracing tutors for programming (Corbett and Anderson, 1989; 1992), geometry proof (Koedinger and Anderson, 1993) and equation solving (Ritter and Anderson, 1995) do not share this characteristic. Those tutors exhibit strong path dependence, in that the assistance that they offer a student at any point in the solution is highly dependent on the student's decisions at earlier points in the problem. For example, a student solving the equation " $3x+4=12$ " with the equation solving tutor might choose to initially transform the equation into either " $x+4/3=4$ " or " $3x=8$ ". Clearly, the tutor's assistance on the second step of the solution would be dependent on the student's

first step. In fact, this path dependency is even more extreme than it may appear, since students are also allowed to follow paths that deviate from any direct solution. For example, a student could choose to transform the original equation into " $3x+16=24$ ". At this point, the tutor would suggest subtracting 16 from each side of the equation as the next transformation. Analogous situations occur in the geometry and programming domains.

Although this may appear to be a major difficulty, path dependency does not pose a major problem for the framework proposed here. The rule-based systems underlying these tutors all categorize problem situations into basic types, in a manner similar to the perception rules in PAT. For example, the equation solving tutor categorizes the equation " $3x+4=12$ " into an equation of the form " $y=mx+b$ ", with appropriate numeric parameters. PAT's advantage is that all possible decision points are laid out as the cells in the spreadsheet (or entries on the graph), where, in the more generative tutors, these decision points are constructed by the student. In these tutors, we cannot present a canonical solution which represents all decision points that students can reach. We can, however, present a set of such solutions that cover the most likely nodes. In a case like the equation solver, nodes can be presented to cover the set of nodes for which the tutor provides specialized instruction. The nodes need not be presented as a neat problem solution. Since testing the system as a student remains an option, it is not necessary that the set of canonical solutions cover every possibility, as long as the set is sufficient to give the author a reasonable sense of the system's interpretation of the problem.

5. Putting it together

Figure 8 illustrates the overall data flow in pSAT. Using the client-side application, the author enters the problem text, column headings, units of measure and given values. These are passed to the server. The server runs two sets of production rules. The first is primarily concerned with parsing the problem text to find referents for the column headings, formula and units of measure. For example, in the problem given in Figure 1, the system would identify the phrase "the number of cars she sells" as a referent for the column heading for the independent variable. The student system can highlight these portions of the problem text as a means of giving feedback, and text from these referents is incorporated into help messages. The problem author can review the referents and, if necessary, correct them.

This set of perception rules also interprets the text of the questions in the problem statement. In the problem given in Figure 1, the system would infer that the given quantity for question 1 is "6 cars". The number 6 is entered in two places: in the text of the question and as the given value for question 1. The perception rules contain information about units of measure and conversions between them, so that the system can determine whether a unit conversion is necessary in this question. If the unit of measure in the problem situation matches that given

for the column as a whole, the system verifies that the quantity given in the problem text is the same as the number specified as the given value for that question on the spreadsheet. If not, it checks whether the numbers are related in the way suggested by the conversion factor for their respective units of measure. If not, the author is alerted to the discrepancy, but, as in all aspects of the system, the author has the final decision.

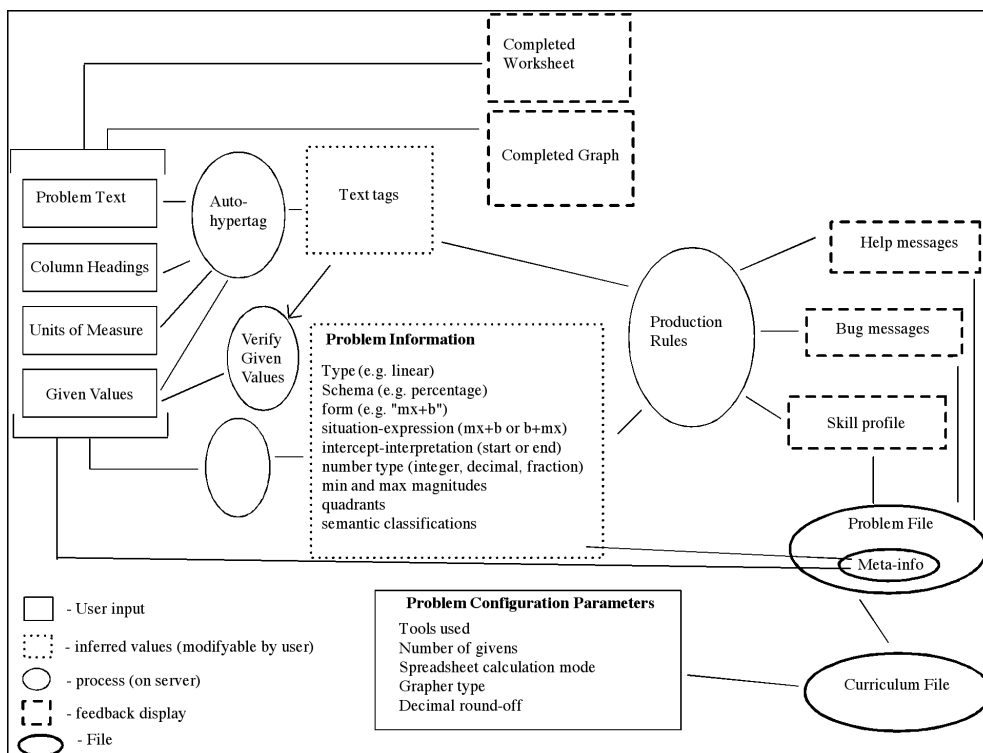


Figure 8: Overview of pSAT Architecture

The second set of perception rules classifies the problem on various dimensions, including the form of the mathematical expression underlying it, the general problem schema and the magnitudes of the numbers used in it. As with other inferences made by these rule systems, the author can override the system's inferences, thus determining the system's final behavior.

The outputs of these perception rules are fed to the action rules, which construct help and bug messages appropriate to the problem situation. These rules are able to run all of the time, and they will fire as soon as the information they depend on becomes available. Thus, as the author completes more of the problem encoding, more help and bug messages become available. At any

time, the author can ask to see the help and bug messages for any cell or entry in the graph.

When the problem is completed, the calculated portions of the graph and spreadsheet are displayed. This may seem like a minor point, but it helps to prevent many encoding errors. For example, authors may intend to construct a problem involving only integers, not realizing that one of the questions they have encoded will lead to a decimal answer. Similarly, authors may encode problems which, when graphed on a scale that displays all points, puts two or more points directly on top of one another. In several cases, this was an unintended consequence of entering problems involving large scales, and it was not discovered at authoring time, since graphical feedback was not immediately given.

When the author is satisfied with the problem encoding, the problem can be saved into one or more collections. A full curriculum can then be created (or an existing one modified) by dragging problems out of a collection and into a section “folder.” Finally, run-time parameters (specifying the form of the graph, capabilities of the spreadsheet and other factors) can be assigned to lessons, sections or individual problems within a section.

6. Breaking it apart

The architecture of pSAT lends itself well to component technologies. We have devised a high-level language for communication between the author’s input tools and the rule-based systems running on the server. Eventually, we would like to compose these rule systems into independent components, which could run either on the server or the client. If the rule banks were true components, other systems presenting word problems could use our perception rules, for example, as a portion of their understanding of the problems presented there. Similarly, our help-text generating action rules could be grafted onto an entirely different front-end which output the appropriate problem encodings.

We are in complete agreement with Roschelle et al.’s (1998, this issue) advocacy of component design combined with open standards, particularly as it applies to authoring tools. pSAT is, in part, an attempt to encourage a division of labor in authoring, where cognitive modeling can be performed by those skilled at the task and content authoring can be done by those in daily contact with the student (or by the students themselves). Within the content authoring tool, we have tried to separate the perceptual and action rule components, recognizing that creating perceptual rules remains a difficult programming task.

The big question in component design is always “what are the proper components?” Open architectures, distributed development and component design only make sense when the system is cut up in a way that makes sense to the participants. Components must be large enough to

constitute meaningful pieces to all developers yet small enough that developing a replacement for a particular component is not too large a task. We consider the pSAT architecture to be a first step in this direction, with further division into smaller components likely to follow.

7. Conclusion

The transition from a simple text editor to the pSAT authoring system has taken place over a period of four years. In that time, PAT has evolved and its success as an educational tool has allowed it to become more widely accepted as an integral part of the algebra curriculum at many schools. It is safe to say that, in the early development of PAT, the problem of content authoring was minimized. We treated content as equivalent to the authoring of the basic system itself: to be done once and only minimally changed over the years. The realization of the continuing importance of content authoring was only gradually made apparent.

In this paper, we have tried to share some of the lessons we learned in designing a content authoring tool to work with an existing tutor. Perhaps the most valuable lesson to learn is the importance of considering content authoring as an integral part of the design of intelligent tutoring systems. The pSAT system is still in an early state of its evolution and will continue to improve as we are able to apply these lessons more completely to its design. A primary direction for future work is to investigate the issues involved with moving from a tutor-specific authoring system to one that has more general applicability. We are also continuing to test the system with interested teachers and are exploring the barriers to wider use of the system by teachers using the PAT Algebra system in their classes.

References

- Anderson, J.R., Conrad, F.G. and Corbett, A.T. (1993). *The Lisp Tutor and Skill Acquisition*. In Anderson, J.R. (Ed.), *Rules of the Mind* (pp. 143-164). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Anderson, J.R., and Pelletier, R. (1991). *A Development System for Model-Tracing Tutors*. In *Proceedings of the International Conference of the Learning Sciences*, Evanston, IL., 1-8.
- Bondaryk, L.G. (1998). *Publishing New Media in Higher Education: Overcoming the Adoption Hurdle*. *Journal of Interactive Media in Education*, 98 (3).
<<http://www-jime.open.ac.uk/98/3>>
- Carpenter, T.P., Fennema, E., Peterson, P.L., Chiang, C. and Loef, M. *Using Knowledge of Children's Mathematics Thinking in Classroom Teaching: An Experimental Study*. *American Educational Research Journal*, 26(4), 499-531.

Corbett, A.T. and Anderson, J.R. (1989). *Feedback Timing and Student Control in the LISP Intelligent Tutoring System*. Proceedings of the Fourth International Conference on AI and Education, 64-72.

Corbett, A.T. and Anderson, J.R. (1992). *Student Modeling and Mastery Learning in a Computer-Based Programming Tutor*. In Frasson, C., Gauthier, G. and McCalla G. (Eds.), *Intelligent Tutoring Systems: Second International Conference Proceedings*, New York: Springer-Verlag, 413-420.

Klahr, D. and Dunbar, K. (1988). *Dual Space Search During Scientific Reasoning*. *Cognitive Science*, 12, 1-48.

Koedinger, K.R. and Anderson, J.R. (1993). *Effective Use of Intelligent Software in High School Math Classrooms*. Proceedings of the Sixth World Conference on Artificial Intelligence in Education, Charlottesville, VA: Association for the Advancement of Computing in Education, 241-248.

Koedinger, K.R., Anderson, J.R., Hadley, W.H., & Mark, M.A. (1995). *Intelligent Tutoring Goes to School in the Big City*. In Proceedings of the 7th World Conference on Artificial Intelligence in Education, Charlottesville, VA: Association for the Advancement of Computing in Education.

Koedinger, K.R., Anderson, J.R., Hadley, W.H., & Mark, M.A. (1997). *Intelligent Tutoring Goes to School in the Big City*. *International Journal of Artificial Intelligence in Education*, 8.

Koedinger, K.R. and Sueker, E.F. (1996). *PAT Goes to College: Evaluating a Cognitive Tutor for Developmental Mathematics*. In Proceedings of the International Conference on the Learning Sciences, Charlottesville, VA: Association for the Advancement of Computing in Education.

Marion, A. and Hacking, E.L. (1998). *Educational Publishing and the WWW*. *Journal of Interactive Media in Education*, 98 (2). <<http://www-jime.open.ac.uk/98/2>>

Nathan, M.J., Koedinger, K.R., & Tabachneck, H.J.M. (1997) *Teachers' and Researchers' Beliefs of Early Algebra Development*. In Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum.

National Council of Teachers of Mathematics (1989). *Curriculum and Evaluation Standards for School Mathematics*. Reston, VA: The Council.

Norman, D.A. (1990). *The Design of Everyday Things*. New York: Doubleday.

Ritter, S. (1997). *PAT Online: A Model-Tracing Tutor on the World-Wide Web*. In P. Brusilovsky, K. Nakabayashi & S. Ritter (Eds.), Proceedings of the Workshop on Intelligent Educational Systems on the World Wide Web, Kobe Japan.. The PAT Online system is available at <<http://domino.psy.cmu.edu/patonline.html>>.

Ritter, S. (1998). *The Authoring Assistant*. In B.P. Goettl, H.M. Half, C.L. Redfield and V.J. Shute (Eds.), *Intelligent Tutoring Systems* (pp. 126-135). Berlin: Springer-Verlag.

Ritter, S. and Anderson, J.R. (1995). *Calculation and strategy in the equation solving tutor*. In Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society.

Roschelle, J., Kaput, J. Stroup, W. and Kahn, T.M. (1998) *Scaleable Integration of Educational Software: Exploring The Promise of Component Architectures*. *Journal of Interactive Media in Education*, 98 (6). <<http://www-jime.open.ac.uk/98/6>>

Simon, H.A. (1975). *The Functional Equivalence of Problem Solving Skills*. In H.A. Simon, *Models of Thought* (pp. 230-244). New Haven, CT: Yale University Press.

Spohrer, J., Sumner, T. and Buckingham Shum, S. (1998). *Educational Authoring Tools and the Educational Object Economy: Introduction to this Special Issue from the East/West Group*. *Journal of Interactive Media in Education*, 98 (10). <<http://www-jime.open.ac.uk/98/10>>