Four Easy Pieces: Development Systems for Knowledge-Based Generative Instruction

Patricia Y. Hsieh, Henry M. Halff, Carol L. Redfield

International Journal of Artificial Intelligence in Education (1999), 10, 1-45

Four Easy Pieces: Development Systems for Knowledge-Based Generative Instruction

Patricia Y. Hsieh, Henry M. Halff, Carol L. Redfield, *Training Technology Division, Mei Technology Corporation, 8930 Fourwinds Dr., Suite 450, San Antonio, TX 78239*

Abstract. The Experimental Advanced Design Advisor (XAIDA) is a system for the development of computer-based maintenance training. XAIDA acquires knowledge of a device from a subject matter expert and applies common maintenance-training procedures to generate interactive training from the description. XAIDA provides maintenance training in four areas: the physical characteristics of a device, its theory of operation, operating and maintenance procedures, and troubleshooting. XAIDA relies on an instructional device known as a transaction shell, an instructional procedure applicable to particular instructional objectives of a specific type. XAIDA employs a different transaction shell for each of the four above-mentioned areas, and each shell employs a knowledge structure appropriate to the shell. Semantic networks represent physical characteristics and procedures; causal reasoning schemes represent theory of operation; and fault trees represent troubleshooting. Each shell provides a browser that is used to present knowledge to the student and a practice environment that promotes skill acquisition under the guidance of an intelligent tutoring system. Subject-matter experts create device descriptions using a WYSIWYG knowledge acquisition system that makes for extremely efficient development. This paper describes each shell in detail, summarizes the research that has been done, and describes where XAIDA stands in relation to other knowledge-based authoring tools.

RAPID DEVELOPMENT SYSTEMS FOR KNOWLEDGE-BASED GENERATIVE INSTRUCTION

This paper describes a research and development effort aimed at making it easy for subject-matter experts to develop Interactive Courseware (ICW). By "easy" we mean that courseware development should be as effortless as, say, spreadsheet development. We aim for development efficiency on the order of ten hours of development for each hour of instruction. Our ideal subject-matter expert is a maintenance technician with basic computer skills and no experience in ICW development. The ICW that we seek to produce draws on the best techniques available in the field, including conventional computer-based instruction, multimedia, exploratory environments, and intelligent tutoring systems. It adds to existing instructional media the same value that a knowledgeable instructor brings to instruction.

Our approach is grounded in knowledge-based, generative instructional methods, and, in particular, Merrill's (1993) notions of instructional transactions and transaction shells. A transaction is an instructional procedure for meeting a particular instructional objective such as being able to tie one's shoe or answer questions about the characteristics of an automobile engine. A transaction shell is a generic form of a transaction. It might, for example, describe a procedure for instruction in step-by-step procedures in general or for associative learning in general. Transactions are produced by adding particular subject matter knowledge (e.g., the steps in tying one's shoe or the characteristics of an automobile engine) to a transaction shell.

Transaction shell theory is therefore founded on the hypothesis that one can factor subject-matter knowledge and instructional procedures. We do not claim that the hypothesis is original with Merrill. However, his ideas were the immediate inspiration for XAIDA. Some sense of the

history of the project can be obtained from Hickey, Spector and Muraida (1991), Spector (1990), and Spector, Polson and Muraida (1993).

Although the factorability hypothesis would be difficult to defend in general, it should hold within limited content domains (Halff, 1993). The domain of interest to us is equipment maintenance. Building on a conceptual design for maintenance training in Halff (1990a), we are developing four computer-based transaction shells, each of which addresses a particular aspect of maintenance training. These aspects include a system's physical characteristics, its theory of operation, its operating and maintenance procedures, and troubleshooting procedures. The four shells and their associated software comprise a system called (for historical reasons only) the Experimental Advanced Instructional Design Advisor (XAIDA). The view of XAIDA as applicable to maintenance training only should not be taken too seriously. It has found application in the closely related field of medicine, and, like any good computer tool, has found application in many areas for which it was never intended.

Our purpose in presenting this work is twofold. First, we hope to provide evidence concerning transaction theory's central hypothesis (the factoring of knowledge and instructional methods) and thereby advance instructional theory in general. Second, we hope to raise some of the more important issues that arise in practical implementations of knowledge-based generative instruction and to contribute to their resolutions.

The Structure of XAIDA (and of This Paper)

XAIDA consists of two computer programs and a corpus of associated data. A program called Develop is a knowledge acquisition system used by a subject-matter expert to specify the knowledge and materials making up the subject matter of instruction. The output of Develop is an XAIDA Knowledge Base. This Knowledge Base specifies the structure of knowledge in the domain and the materials used in interactions with students. Deliver is an instructional delivery program that uses transaction shells to mediate the interaction between a student and the Knowledge Base. Describing XAIDA's approach to instruction is therefore a matter of addressing four topics: (a) the representation of subject-matter knowledge, (b) the instructional materials associated with knowledge structures, (c) the instructional procedures embodied in Deliver, and (d) the knowledge acquisition procedures embodied in Develop.

These topics are handled differently by each of XAIDA's four transaction shells. The representation of a system's physical characteristics, for example, is quite different than that of its theory of operation. Hence, what students need to learn about these two aspects of a system is different. The instructional techniques and materials differ as well. Table 1 provides a sense of the main characteristics of each shell, in a way that illustrates their differences. In spite of these differences, each of the shells brings two basic modules or capabilities to the task of instruction: a browser that is used to present knowledge to the student and a practice environment that promotes skill acquisition under the direction of an intelligent tutoring system.

We need to mention that XAIDA is a work in progress, and each shell is at a different stage of development. The Physical Characteristics shell is the most developed. It is both usable and effective in properly supported applications. The Theory of Operation shell is likewise fully functional. It has not had the benefits of extensive testing and refinement that have accrued to the Physical Characteristics shell. The Procedures and Troubleshooting shells exist only as functional specifications and working Deliver prototypes.

The body of this paper describes each shell in terms of topics a–d listed above. Our main focus is on knowledge representation and instructional methods. Within the latter, we focus on intelligent tutoring techniques. The focus on intelligent tutoring is partly in order to address the purpose of this special issue of the IJAIED but also because intelligent tutoring techniques inevitably stress the transaction-shell approach in deeper and more interesting ways that other less interactive instructional methods.

The paper concludes with a brief summary of XAIDA research, a description of where XAIDA stands in relation to other ITS authoring tools, and directions for future research and development.

Table 1. XAIDA's Transaction Shells

Transaction Shell	**Nature of Knowledge**	**Instructional Objective**	**Knowledge Representation**	**Instructional Method**
Physical Characteristics	The structure of the system, the location and other characteristics of the system, its modules, and its components	Recall of system characteristics in a variety of situations	Semantic network based on the structure of the system	Structure-directed presentation of system characteristics and practice in recall using diverse exercise formats.
Theory of Operation	Variables characterizing system behavior, their possible values and their functional relations	Infer the values of some variables from knowledge of the values of others	Causal reasoning scheme	Presentation of salient cases (both student- and instructor-generated) about system behavior along with exercises requiring inferences about system behavior.
Procedures	The steps of a procedure, characteristics of the procedure and of each step including potential mistakes and mishaps.	Recall the steps of a procedure; the fundamental characteristics of the procedure and of each steps, including potential mistakes and mishaps. Recall methods for error avoidance and recovery.	Semantic network based on the steps of the procedure.	Stepwise presentation of the procedure; stepwise practice of recall; practice in error recognition and recovery; global recall practice.
Troubleshooting	Successive partitions of the system used to progressively isolate a fault, observations used in the isolation process	Isolate a fault by narrowing its locus to successive subregions of the system.	Discrimination net (called a fault tree) with regions at its nodes and observation at its branches.	Troubleshooting practice in which faults are progressively added to the practice environment in the order dictated by the fault tree.

PHYSICAL CHARACTERISTICS

Maintenance training typically provides for extensive treatment of the physical characteristics of the systems addressed in the training. Students must learn the structure of the system, that is its modules and components. They must learn where these parts are located. They must master their salient characteristics such as their functions, capacities, and limitations. In conventional

instruction students must exhibit mastery of these facts by passing tests consisting of questions about the system.

XAIDA's Physical Characteristics shell is designed to instill this knowledge in students. The type of knowledge is fundamentally associative, and we therefore use semantic networks for knowledge representation. Instructional capabilities include both a browser used to present instructional materials and an intelligent tutoring system (ITS) that can be used to practice recall of the characteristics.

Knowledge Representation: Semantic Networks

XAIDA uses semantic networks to represent the physical characteristics of a device. In this respect XAIDA follows a tradition that dates back to the dawn of intelligent tutoring systems (Carbonell, 1970). A semantic network (Collins & Quillian, 1969; Collins & Loftus, 1975) is a set of triples, called facts in XAIDA, of the form (subject, attribute, object), for example, (filter basket, part of, coffee pot). Figure 1 depicts a fragment of a Physical Characteristics semantic network. Several points about this network are worth mention:

1. Central to XAIDA's knowledge representation for Physical Characteristics is the part structure of the device being represented. This structure is represented in the semantic network through the attribute "is part of". We use the term "part" to refer to any element of the part structure, including the top-level element. We reserve the terms "system" or "device" to refer to this top part.
2. In a Physical Characteristics semantic network the subject of a fact is always a part.
3. The network contains location information. Associated with each part is a background graphic in which all of its subparts are visible. Associated with each subpart is a region of the background called a locator that denotes the location of the subpart. Such a locator is shown in Figure 1 for the Crew Drain Valve.
4. The developer can create any number of attributes deemed important for device understanding and create facts for those attributes. The objects in these facts can either be other parts (e.g., (Crew Drain Valve, connected, Buildup and Vent Valve) as shown in Figure 1) or text (e.g., (Troop Oxygen System, function, provide oxygen to troops) as shown in Figure 1).

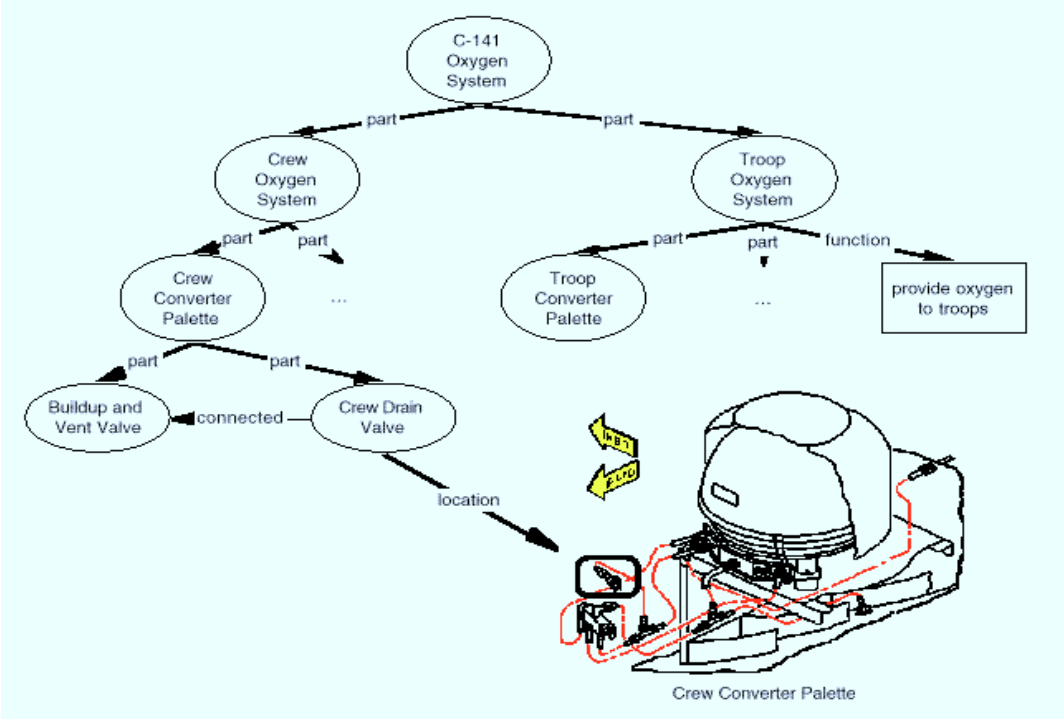

Figure 1. Physical Characteristics Semantic Network Fragment.

Instructional Materials

As mentioned above, the elements of the knowledge representation described here each have associated instructional resources that are used to convey information about the element to the student. These materials can take on several forms.

Materials for parts. Each part has a name, a text string that both XAIDA and the student use to refer to the element named. Each part has a description, a short body of text describing the part. Each part has a bitmapped graphic called a background, a picture of the device that shows each of its subparts. The term "background" was chosen because it is usually presented with an overlay of locators that show the locations of subparts.

A variety of instructional resources can be associated with each part. These resources can be as simple as a bit of text or as complex as an HTML file linking the lesson to the World Wide Web. Any computer file with an associated executable (player) can be designated as a resource, and any number of resources can be associated with a part.

Materials for facts. XAIDA also generates instructional materials from facts. Each component of a fact (part, attribute, and value) has a name. These names fill slots in templates to create sentences of various types pertaining to the fact. One template states the fact in a simple declarative form. Other templates put the fact in question form for practice purposes and in forms suitable for feedback during practice. Any of these templates can be customized on an attribute-by-attribute basis.

Instructional Delivery

In providing instruction on a device's physical characteristics, XAIDA constructs a lesson outline based on the part structure of the device (see Figure 1) and covers each part in outline order. A browser is used to present the materials related to each part in outline order. In addition, a practice session is inserted after all parts of any part with subparts have been covered. Thus, the first few topics covered in connection with Figure 1 are

C-141 Oxygen System,
Crew Oxygen System,
Crew Converter Palette,
Buildup and Vent Valve,
Crew Drain Valve,
Practice,
remaining parts of the Crew Oxygen System,
… .

The following sections describe the operation of the browser and practice capability.

The Physical Characteristics Browser

The Physical Characteristics Browser has three functions. It presents an overview of a part; it provides systematic step-by-step coverage of the part's characteristics; and it supports selective review of material previously presented.

Overview. Each part is introduced to the student with a one- or two-part overview. First, the part is shown in the context of its parent part by highlighting the part's locator in its parent's portrayal. This step is omitted in the case of the system itself, which has no parent. Second, the part is shown close up, along with its description (Figure 2). If the part has sub-parts, the locations of the sub-parts will also be indicated using labeled locators; furthermore, if the student happens to pass the cursor over one of these locators, a text description of the corresponding sub-part is presented.

Presentation. The second step of the overview (Figure 2) introduces the presentation of the part's characteristics, namely, its resources, facts, and subparts. During the presentation of a

part, XAIDA presents the resources and facts associated with the part. The browser is then invoked recursively on each of the part's subparts.

Review. The student can normally only view new material in a lock-step fashion, using the Next button, but may review previously presented material in any order. Parts can be reviewed by selecting either their locators or their entries in the Outline pane. The materials for an individual part can be reviewed by selecting entries in the Contents pane.

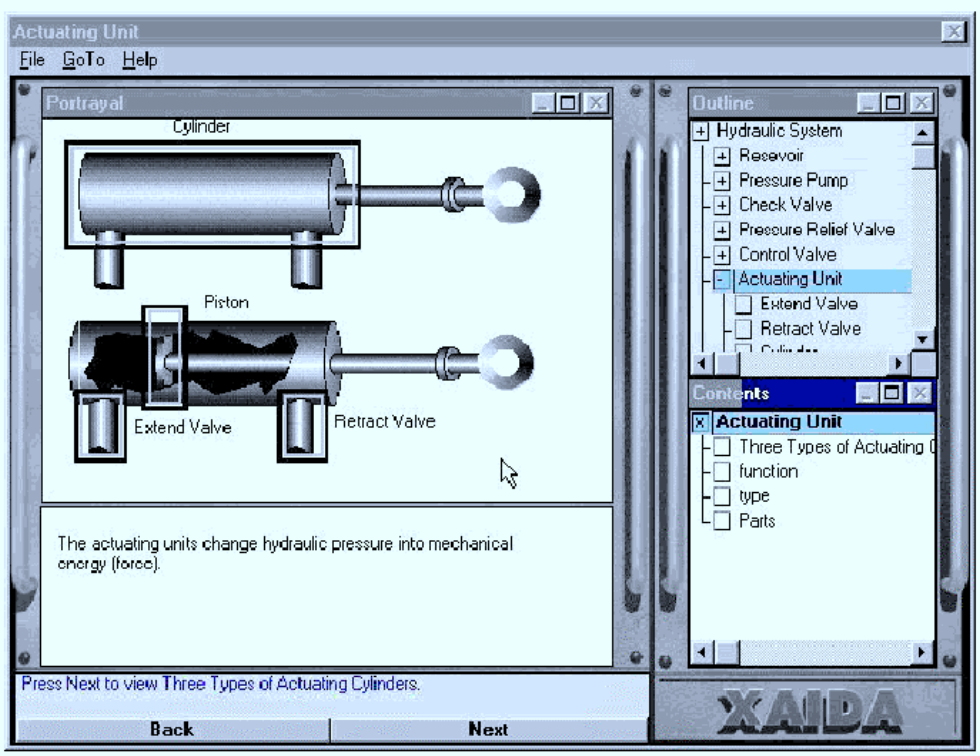

Figure 2. Physical Characteristics Browser

Practice

The student is given practice on fact recall after coverage of all of the subparts of any part with subparts. The scope of the practice includes all the facts for all previously presented parts. XAIDA's practice capability is an Intelligent Tutoring System (ITS) that helps students learn the facts about a device under study and its parts. This ITS consists of four, now classic, components: an expert module, a student model, an instructional module, and a student interface (Polson & Richardson, 1988; Wenger, 1987). They are described as follows:

The expert module. The expert module of the ITS consists of the semantic network of facts that have been presented to the student. The structure of this network is described above in connection with the knowledge representation for physical characteristics.

The student model. The student model has two parts: a marking of the semantic network that indicates the state of mastery of each fact, and a list of misconceptions: "facts" not in the semantic network that the student has exhibited in her performance. The student model is updated whenever the student answers a question generated by the instructional module. Each question can address any number of facts, and the student's answer can exhibit mastery of a fact (correct) or failure of mastery (error). Facts are either mastered or unmastered. A fact is deemed to have been mastered if the student generates two (or some other criterial number of) correct responses with no intervening errors. Conversely, two errors in a row puts the fact back in the unmastered state. In addition, a student's answer can exhibit a misconception. A misconception is recorded when a student answers a question about an attribute with a part and a value that are not so related in the expert model. For example, if the student responds that the

capacity of the crew oxygen system is 15 liters, when it is actually 25 liters, the misconception (crew oxygen system, capacity, 15 liters) is formed. Misconceptions are removed from the model whenever the student passes up an opportunity to exhibit the misconception. For example, the misconception (crew oxygen system, capacity, 15 liters) is removed if the student correctly states that it has a capacity of 25 liters, or even if the student incorrectly states that it has a capacity of 30 liters. In the latter case, the misconception (crew oxygen system, capacity, 15 liters) would be removed, but a new misconception (crew oxygen system, capacity, 30 liters) would be added.

The instructional module. The instructional module repeatedly generates questions about facts. It then allows the student to answer each question, and provides feedback on the responses.

Each question is formed by configuring one of 11 exercise schemata. One of these schemata in action is illustrated in Figure 3. Most schemata, like the one in Figure 3, address more than one fact, and it is even possible for an exercise not to address the target fact used in the exercise's generation.

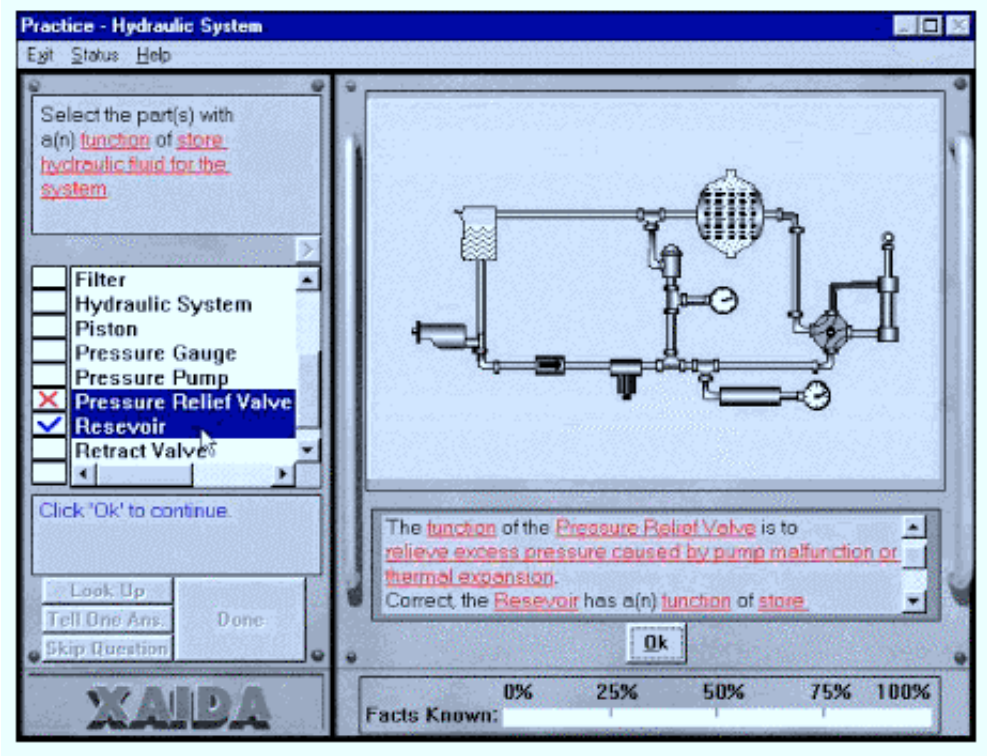

Figure 3. Physical Characteristics Exercise

Some schemata ask questions about parts and attributes (e.g., Enter the function(s) of the Pressure Gauge). Schemata such as these are configured with a particular part-attribute combination. Other schemata ask questions about attributes and values (e.g., Enter the part(s) with a function of indicates pressure). Schemata such as these are configured with a particular attribute-value combination.

Question generation is a two step process. First, a part-attribute pair or an attribute-value pair is randomly chosen with a bias towards pairs that address a large number of unknown facts or misconceptions. Second, a schema is randomly chosen from among those that fit the selected pair. The pair selected for the exercise in Figure 3, for example was (function, store hydraulic fluid for the system). This pair was a likely choice to the extent that the student knew none of the parts that store hydraulic fluid and to the extent that he had misconceptions about parts that store hydraulic fluid. The schema, "Select part(s) with a(n) attribute of value," was chosen because it is configured with an attribute-value configuration. Another, equally likely choice was "Enter part(s) with an attribute of value."

Feedback is quite detailed. First, each of the student's responses is reviewed. The student is told whether or not the response is correct. In the case of incorrect answers, the student is given related information about the response (see Figure 3). After reviewing the student's responses, XAIDA shows any correct answers that she may have failed to provide.

The student interface. The student interface is one that supports practice and lookup of question answers. Students are able to search the course materials using the browser and return to the question when they have found an answer. Available to students having difficulty is a "Tell One" button that supplies a single, randomly chosen, correct answer to the question or, if there are no more answers, so informs the student. Also available is the option to move on to another question without answering the one presented.

Any of three conditions serve to terminate a practice session: when a fixed percentage of the facts have been mastered, when a fixed amount of time has elapsed, or when a fixed number of questions have been asked.

Instructional Development

Instructional Development in XAIDA is supported by a program called Develop. Although Develop and Deliver are distinct programs, the former's WYSIWYG approach (described below) is one that gives the developer a good feel for the instruction offered by her lesson. In fact, through features such as selective preview (also described below) Develop offers a better view of the lesson than that afforded by running it with Deliver.

XAIDA takes a What-You-See-Is-What-You-Get (WYSIWYG) approach to knowledge acquisition. That is, the developer works mainly by editing the same displays that are used by Deliver during instruction. This approach obviates the need for separate preview capabilities and for extensive off-line planning of lesson structure. Developers always have before them the instruction that will be delivered to students, and the development process, by and large, follows the same course as does instruction.

The main difference between Develop and Deliver are the needs, in the former, to be able to edit or add to a knowledge base, to designate instructional materials (names, descriptions, resources, etc.), and to selectively preview instruction. These functions are accommodated in XAIDA by menu selections or selections from a tool palette.

Develop is not completely WYSIWYG in that forms are provided for a few authoring functions. Forms are provided for editing facts, templates, instructional parameters, and other data of this sort. Facts, for example, are constructed and edited using a special purpose fact editor. The fact editor allows the developer to construct pools of attributes and values and to assemble facts from these pools.

Develop also allows developers to selectively preview and revise the declarative and question templates for a given attribute. The need for selective preview arises from the generative nature of XAIDA'a approach, particularly in the practice module. Although a developer can get a good sense of the overall operation of the practice module simply by running her lesson in Deliver, she cannot use Deliver to examine the operation of particular exercise types with particular facts. Thus Develop offers the capability to view the operation of the practice module on a fact and an exercise type of the developer's choosing. The developer may also specify which question schemata apply to each attribute and modify the templates used to express facts.

When we field tested lessons with parts having more than one fact per attribute, some users complained that XAIDA's tendency to present each fact as a separate statement was awkward and repetitive. For example, when presenting the facts (Crew Drain Valve, connected, Buildup and Vent Valve) and (Crew Drain Valve, connected, Crew Converter), XAIDA will use two separate statements, e.g.,

The Crew Drain Valve is connected to the Buildup and Vent Valve.
The Crew Drain Valve is connected to the Crew Converter.

We therefore gave developers the option to suppress the presentation of facts for a given attribute. Facts for an invisible attribute are available for practice questioning, but are not presented in the browser. Developers can, and as a rule should, then present that factual information in a more aesthetically pleasing manner elsewhere in the lesson, such as in a bitmap or video resource.

Other instructional parameters are also under control of the developer. These include the phrasing of virtually all text messages to the student and the use of certain system generated facts, namely those involving part-of and location attributes in exercises. Developers may also specify the number of times a student must correctly answer a question about a fact before XAIDA considers the fact to be known (default = 2); criteria for exiting a practice session (default = 100% of presented facts are known); and whether practice sessions should occur every time a student finishes viewing all the sub-parts of a part (the default) or only at the end of a lesson.

XAIDA and Other Physical Characteristics Tutors

XAIDA provides a more thoroughgoing instructional treatment of physical characteristics information than any other knowledge-based system for maintenance instruction we know of (Hsieh, 1997). Most other ITSs for maintenance instruction have had simulation as their centerpiece; physical characteristics instruction, if available at all, is a lagniappe. RIDES, for example, is a powerful and well-designed system for developing equipment simulations which includes a facility for easy development of various types of exercises. However, these exercises are non-adaptive, and their inclusion with the meat of the instruction (i.e., manipulation and exploration of the simulation) is left to the discretion of the developer.1

Merrill's (ID_2 Research Group,1995) Electronic Trainer (ET) is the only other system we know of for which factual information is bread-and-butter. ET also generates questions from developer input. However, XAIDA's knowledge representation is more fine-grained and open-ended than ET's, so that XAIDA can ask about more kinds of information (anything that can be expressed as a part-attribute-value triplet) and can use a greater variety of questioning schema (eleven for XAIDA versus three for ET). In addition, XAIDA provides adaptive practice (albeit simple).

The teaching of associative knowledge has not been a hot area for ITS R&D, possibly because it is considered to be well mapped out. However, the need to teach factual information is clear and widespread, in maintenance training and in many other domains as well. An easy-to-use authoring tool for associative knowledge would certainly both fill a gap and be very handy to have around, as our own experiences fielding XAIDA have borne out (see the section below, Research with XAIDA).

THEORY OF OPERATION

The Theory of Operation shell provides instruction in the theory governing a device's behavior. The main goal of this instruction is that of teaching students to reason about the behavior of a device. In particular, it teaches students how to use rules to infer unknown characteristics of a device's behavior from known antecedents of those characteristics. Examples of such inferences include (a) inferring the position of an hydraulic actuator from the pressures on each side of the actuator's piston, (b) inferring the output of a binary circuit from the circuit's inputs, and (c) inferring whether or not an automobile will start from knowledge of the factors (fuel, battery condition, etc.) needed for the operation. Inferences such as these are important in both the normal operation of the device and in successful troubleshooting. The many uses of such knowledge are described in Kieras (1988).

The most critical issue in teaching theory of operation is the choice of an ontology or epistemology in which theories of operation can be cast. There are two major considerations in this choice: power and accessibility. Power refers to the ontology's ability to support reasoning

about a wide range of systems. For example, representing systems in terms of multiple, simultaneous constraints, such as circuit laws, is a powerful and widely used basis for scientific reasoning. Accessibility refers to ease of use of the ontology by both students and developers. Thus, constraint-based systems, in spite of their power, are not very accessible in the sense used here. The typical technician or student in technical training does not solve problems or explain device behavior using systems of simultaneous constraints. Rather, they tend to offer explanations in terms of causes and effects. For example, in a digital logic circuit, the inputs to a gate are often viewed as causing the output to take on a certain value. Causal models have important limitations in dealing with many physical phenomena, but, judging from their ubiquity in technical training, they have much to offer in the way of accessibility. Because accessibility is a prime goal in the design of XAIDA, we have adapted a causal reasoning scheme to teach theory of operation. This choice gives us the opportunity to study the strengths and weaknesses of causal reasoning in an environment where the causes and effects are given explicit, formal representations.

Knowledge Representation: Causal Models

XAIDA uses a simple cause-and-effect scheme to represent a device's theory of operation. There are two components to this scheme. Variables represent the principles governing the device's operation, and Cases represent pedagogically salient applications of those principles. We refer to a set of variables (along with their interdependencies) and cases as a *causal model*, or more simply a *model*. This use of the term should not be confused with the term "student model."

Variables

A device's theory of operation can be understood in terms of a number of variables, each of which takes on a value taken from a fixed set of values. The values of some variables are determined by the device's operator (e.g., a switch or dial setting), a state of nature (e.g., ambient temperature or wind velocity), or some other external agent (e.g., a generated signal). We refer to these as setting-based variables. The values of variables that are not settings are functions of other variables. Because these functions are usually cast as systems of rules, we refer to their variables as rule-based variables. Rule-based variables correspond to indicators such as meters and lamps and to intermediate variables such as signal strength and hydraulic pressure. To give the reader a better sense of this scheme, Table 2 lists the variables governing the simple hydraulic circuit shown in Figure 4.

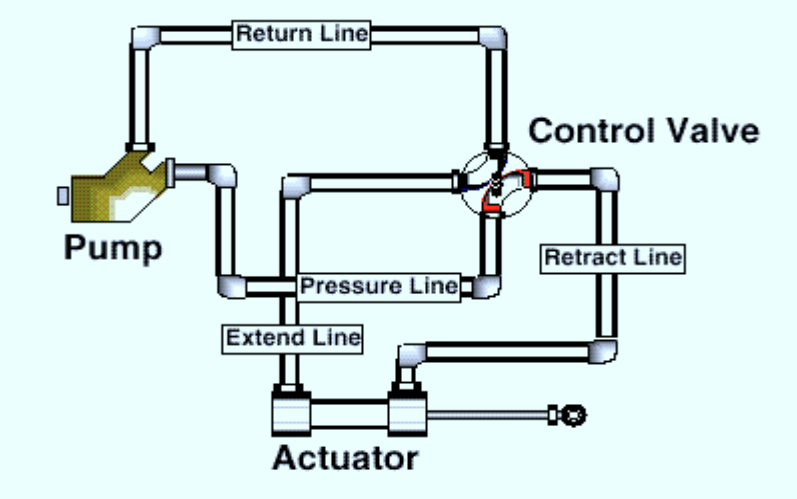

Figure 4. Simple Hydraulic Circuit.

Note that the scheme described here is not a completely general causal reasoning scheme. In particular, it does not allow for sequential or time based effects. That is, a variable's value is not determined by its value or the value of any other variable at some previous time. In addition, feedback is not permitted the system. A variable cannot be a function of its own value either directly or indirectly.

Table 2. Variables for a Simple Hydraulic Circuit

Variable Type	Variable Name	Possible Values/Rules
Setting	Power	off, on
	Control Valve Position	extend, retract
Rule-Based	Return Line Pressure	1 3 psi[1]
	Pressure Line Pressure	2 If Power = off then 0 psi
		3 If Power = on then 3000 psi
	Retract Line Pressure	4 If Control Valve Position = retract and Pressure Line Pressure = 0 psi, then 0 psi.
		5 If Control Valve Position = retract and Pressure Line Pressure = 3000 psi, then 3000 psi.
		6 If Control Valve Position = extend then 3 psi.
	Extend Line Pressure	7 If Control Valve Position = retract, then 3 psi.
		8 If Control Valve Position = extend and Pressure Line Pressure = 0 psi, then 0 psi.
		9 If Control Valve Position = extend and Pressure Line Pressure = 3000 psi, then 3000 psi.
	Actuator Position	10 If Return Line Pressure ≥ Extend Line Pressure then retracted
		11 If Extend Line Pressure > Return Line Pressure then extended

Cases

XAIDA provides instruction on theory of operation in the context of particular combinations of settings called cases. The set of cases to be used in instruction is provided by the developer. Ideally, XAIDA should generate the cases, but such a capability is not feasible for both theoretical and practical reasons. Hence XAIDA requires the developer to select a pertinent set of cases and, by way of support, audits that set to ensure complete coverage of all rules.

Each case has two parts: a set of initial settings and a set of actions. The initial settings are those that characterize the device in some beginning state. The actions specify the settings that bring the device to the state addressed by the case. For example, to explain how the simple circuit in Figure 4 behaves when the pump is turned on and the control valve is in the retract position, we would define a case in which the initial settings are Power = off, Control Valve Setting = retract and which has the actions Power = on, Control Valve Setting = retract.

XAIDA elaborates each case into a sequence of steps. The first steps correspond to the actions. The remaining steps are the results of applying the model's rules to determine the values of rule-based variables. For example, the case described above would have the following steps.

[1] Rules are written in the form "v" to indicate that the rule's variable always takes on the value v, or in the form "if e then v" to indicate that the rule's variable takes on the value v whenever the logical expression e evaluates to true.

1. Power = on (Action).
2. Control Valve Setting = retract (Action).
3. Return Line Pressure = 3 psi (Rule 1).
4. Pressure Line Pressure = 3000 psi (Rule 3).
5. Retract Line Pressure = 3000 psi (Rule 5).
6. Extend Line Pressure = 3 psi (Rule 7).
7. Actuator Position = retracted (Rule 10).

The steps are ordered to reflect the order of rule application using a forward reasoning scheme.

Normally, a model will have several cases: one that illustrates the device's basic function and others that, taken together, exercise all of its rules.

Instructional Materials

As is the case with semantic networks, XAIDA relies on certain instructional materials to convey the content of causal models to students and to allow students to communicate with XAIDA about these models.

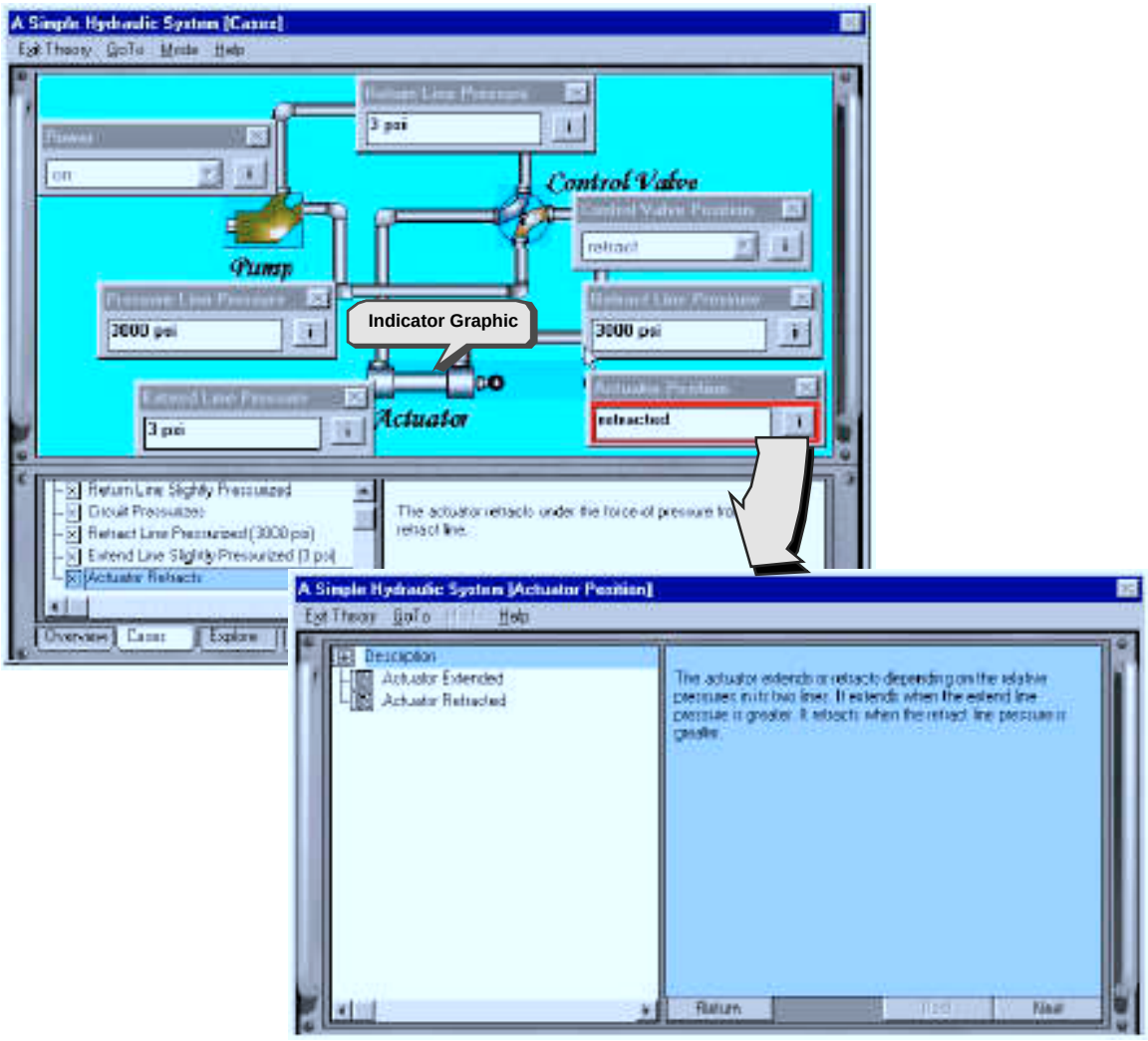

Figure 5. Theory of Operation Browser (Upper Panel) and Variable Inspector (Lower Panel).

Each model, variable, setting value, rule, case and case step has a name, a description, and a resource list. In addition, each model has a background upon which upon which rule and value indicator graphics are overlaid. These indicator graphics are bitmaps associated with each setting value and each rule. An indicator graphic depicts any visible manifestation of the variable value (see Figure 5).

With the foregoing description of XAIDA's representation of maintenance knowledge and materials for the presentation of that knowledge, we turn to the instructional procedures that XAIDA uses to convey this knowledge to students.

Instructional Delivery

As is the case with the Physical Characteristics shell, the Theory of Operation shell has two modules: a browser, which presents the theory of operation of a device, and a practice module, which provides practice in making inferences. Available as part of both modules is a variable inspector that students can use to look up rules and other information related to variables.

The Variable Inspector

The student interface to the Theory of Operation shell is shown in Figure 5. Note that each variable has a panel displaying its name, a momentary value, and an "i" button. This button brings up a Variable Inspector with detailed information on the variable, namely, its description, its resources, and its rules.

The Theory of Operation Browser

The Theory Browser, like its Physical Characteristics counterpart, introduces each model with an overview. It then provides a detailed presentation of each case. The theory browser also has an exploratory capability that allows students to construct their own cases and a review capability like that of the Physical Characteristics browser.

Overview. Instruction in a device's theory of operation begins with an overview in which the student is shown the model's description, its background, and panels used to present information on variables. As the student allows the mouse to hover over a panel, a description of the variable appears in a text pane.

Presentation. Presentation of the model proceeds on a case-wise basis. The presentation of each case begins with the case description. At the developer's option, a list of initial conditions can be appended to this description. Each step in the case is then presented. If the step involves the application of a rule, then XAIDA presents the rule description as the step is discussed (see Figure 5).

Exploratory mode. Exploratory mode allows the student to construct his own cases and view the behavior of the model in those cases. Construction of a case is a three step process. First, the student chooses values for each of the settings using pull-down menus in the variable panels. Second, XAIDA "runs" the model by recursively applying rules to determine the values of rule-based variables. Finally, all of the steps in the case are presented in the same way as the steps of a developer-defined case is presented.

Review. As is the case with the Physical Characteristics browser, students can review previously presented material but normally cannot skip ahead in the curriculum. These review capabilities can be invoked through the tabbed interface shown in Figure 5 or by selecting items in the list of topics.

Practice

The Practice phase consists of a sequence of exercises that require students to infer the values of rule-based variables using the rules that govern those variables (see Figure 6). Practice is administered by a miniature ITS of the same sort used in the Practice phase of the Physical

Characteristics shell. As with the Physical Characteristics shell, this ITS can be discussed in terms of its components, namely, an expert module, a student model, an instructional module, and a student interface.

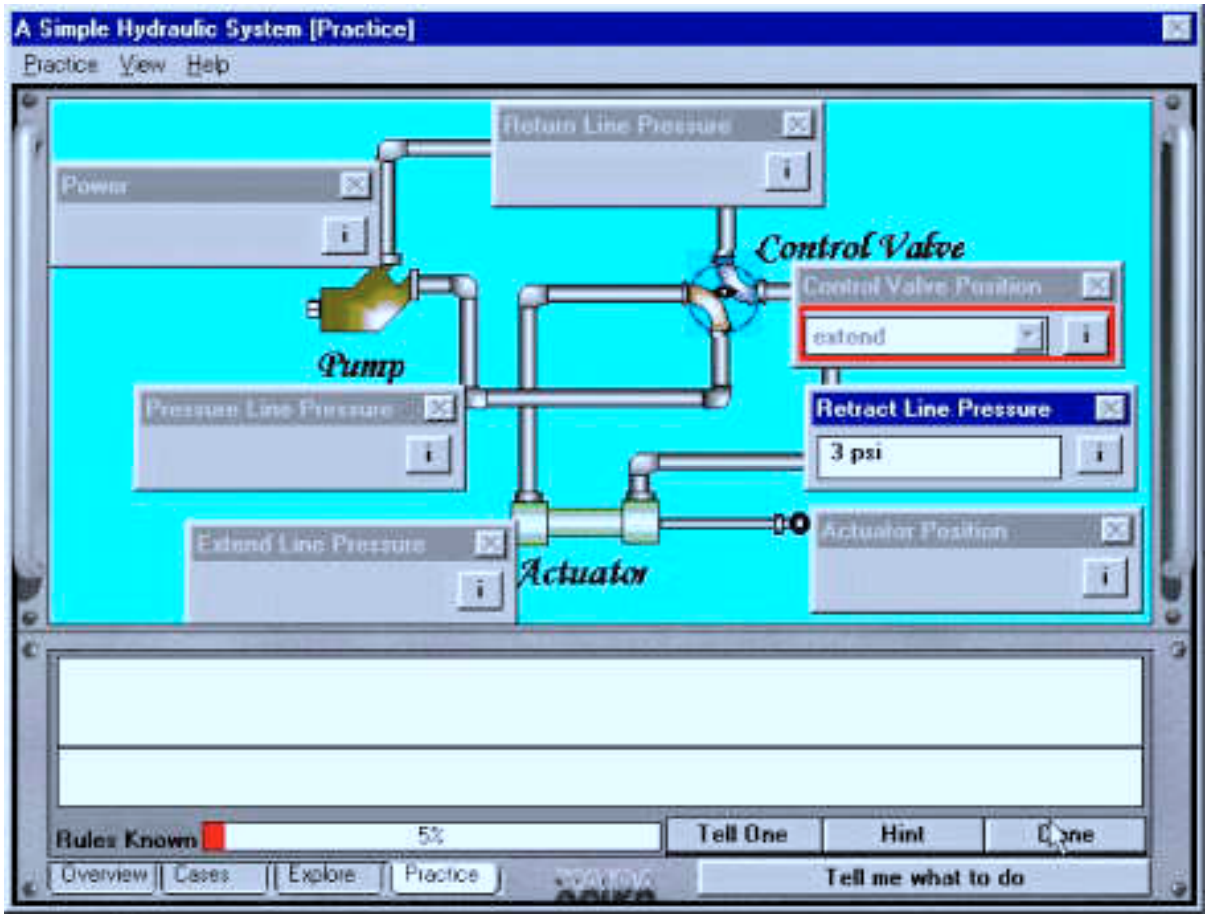

Figure 6. Theory of Operation Exercise. (Note: The student's task is to enter the correct value (3 psi) in the Retract Line Pressure panel.)

The expert module. The expert module of the ITS consists of the collections of variables and cases described above. All of the expert module's inference knowledge is pre-compiled in the cases. In particular, the expert module has two basic capabilities. Given a rule, it can select a case in which that rule is applied, and it can apply rules to determine variable values in any case. As it turns out, these two capabilities are all that is needed for effective tutoring.

The student model. The student model for Theory of Operation is rule-based in much the same way that the student model for Physical Characteristics is fact-based. Each rule in the model is deemed to be mastered, partially mastered, or unmastered, and its state is determined after each opportunity to apply the rule. An opportunity arises in the course of an exercise when the student is asked to supply the value of the rule's variable under conditions where the rule applies.

All rules start in the unmastered state. A single successful application of the rule moves it to the partially mastered state. A second successful application moves it to the mastered state. A failure to correctly apply the rule moves it to the unmastered state.

The instructional module. The instructional module uses a four-step process to construct an exercise:

1. A target rule is selected using a random mechanism that is biased towards unmastered and partially mastered rules.

2. A case is randomly chosen from among all those where the rule applies.

3. An exercise type is selected. Three types of varying complexity are available. Selection of the type is biased by the state of the target rule so that more complex exercises are chosen for rules in more advanced states of mastery. Figure 6 shows an exercise of the lowest complexity.

4. A response mode is randomly selected. Two modes are available: selection of values from a pull-down menu or keyboard entry of values.

Feedback is provided when the student indicates that he is "Done" with the problem (see Figure 6). The system reviews each of the variables whose values were to be supplied by the student. The student is shown the correct value of the variable and the rule that applies. If the student supplied an incorrect answer, this fact is called to his attention before presenting the correct answer.

The student interface. As with the Physical Characteristics shell, considerable support is provided to students working Theory of Operation exercises. The options available to students are roughly the same as those available in the Physical Characteristics shell. The hint option in Theory of Operation exercises randomly chooses a rule that applies to the exercise and directs the student's attention to that rule.

Any of four conditions terminate a practice session. The student can request a termination. A preset time limit can be exceeded. A preset limit on the number of exercises can be exceeded. The student can master a preset percentage of the rules. The student can return for more practice even after a termination criterion has been exceeded.

Lesson Development

Theory of Operation lessons require the specification of three types of knowledge: models, variables, and cases. Developers can enter basic data for a model using a WYSIWYG editor similar to that used for Physical Characteristics. This editor allows the developer to enter or edit a model's basic data: its name, description, resources, variables, and cases. Variables can be added and edited using the Variable Inspector described above. In its editing mode, this inspector provides access to the variable's name and description. The variable's variables or rules are edited using a form-based rule editor. Part of this editor is a structure-directed editor for constructing the conditions and actions that make up each rule.

XAIDA and Other Theory of Operation Tutors

Theory of operation instruction has been of some concern to the training community. Traditional instruction on these topics focused on problem-solving skills since predicting the behavior of a device or system can be viewed as a problem-solving exercise. Hence, it is not unusual for technical training courses to devote considerable attention to basic scientific and engineering principles such as theory of electrical circuits or fundamental hydraulics. Worth noting is that the traditional approach to problem-solving of this sort is that of traditional scientific reasoning based on simultaneous constraints such as circuit laws. Students are taught to predict system behavior in much the same way that college students are instructed in introductory science courses. The reasoning skills needed to solve these academic problems are often not those needed to solve technical problems in equipment. In addition, they are not very scaleable. It is within most individuals' reach to solve circuit problems involving a few components, but not within most people's reach to solve the same sorts of problems in circuits with hundreds of components.

Responding partly to a concern that students acquire only rote problem solving skills instead of deep understanding, partly to technological opportunities, and partly to the concerns mentioned above, more recent approaches to theory of operation training are based on experiential methods and on simulation in particular. This trend is reflected in development systems such as RIDES (Munro, Pizzini, Towne, Wogulis, & Coller, 1994), RAPID (Emultek, 1995), and PowerSim (Byrknes & Myrtveit, 1997), that support the development of complex

system simulations. These systems take a hands-off approach to theory-of-operation training. They allow the user to explore system behavior, but do not, in general, make explicit the strategies and procedures needed to reason about this behavior. Nonetheless, simulation-based training is viewed as a useful, and often necessary part of technical training. Worth mention in connection with simulation-based approaches is the difficulty of constructing simulations, even with the advanced development systems mentioned above. PowerSim simulations for example may require the construction of thousands of processes, each of which must be individually configured. RIDES simulation development requires mastery of a powerful but complex programming language for modeling the behavior of components. Furthermore, some critical simulation capabilities are next-to-impossible in all of the simulation packages that we have surveyed. Assembly, disassembly, and other gross structural changes, for example, are difficult to simulate.

XAIDA's approach has the spirit of traditional problem-solving approaches but the content of simulation-based approaches. That is, it addresses theory of operation as a problem-solving process and makes explicit the principles to be applied in the process. Those principles are not, however, the constraints used to reason scientifically about device behavior, but rather rules that embody reasoning in terms of causes and effects. It is our feeling that these causal reasoning schemes simplify the problem-solving process, thereby easing both its acquisition and application.

One other system, Merrill's (1997) Instructional Simulator, addresses theory-of-operation training as does XAIDA. The Instructional Simulator models systems in terms of causal rules and makes those rules available to students in the course of instruction. It also features a number of instructional methods, each of which draws upon the same knowledge base of variables and rules. The main difference between XAIDA and the Instructional Simulator is that the latter offers considerably more flexibility in constructing a simulation. The developer, however, pays the price for this flexibility in terms of ease of use and learnability.

PROCEDURES

The need to teach procedures to maintenance technicians is obvious. Less obvious is what technicians need to know about a procedure and how that knowledge should be conveyed. The approach we have taken in the design of the Procedures shell is derived primarily from observations of classroom instructors in action (under the assumption that since the classroom teaching "medium" has survived for many centuries, there is likely some merit to it).

In these observations, we noticed instructors would typically give an introduction, then demonstrate each step in a procedure while talking about it. Types of things they would talk about included reasons for performing the step, tips for doing it right, how to know it has been performed correctly, potential problems to watch out for, and how to recover from those problems.2 Typically also, if equipment was available, they would then have students perform the procedure for themselves, and assist as necessary.

The idea of providing ancillary information about steps while teaching them makes sense to us because we believe there is much that goes on in the brain when performing a procedure--more than just the mechanics and sequence of executing the steps (which is the typical focus in other systems for teaching procedures). One of our authors had a epiphany of this sort while performing a simple procedure using XAIDA itself. We decided to design the Procedures shell to accommodate this type of information.

Knowledge Representation: Semantic Networks

Like Physical Characteristics, the Procedures shell uses a semantic network to represent knowledge about a procedure. Each node in the network represents a step, except the root node, which is the procedure itself. Associated with each step node are a name, a description, an illustration, an action-object-tool-and-background combination, a position in the procedure,

and any number of user-defined facts, multimedia resources, and problems (i.e., undesirable events which may occur while performing a step or procedure).

Procedures are cast as semantic networks, not unlike those for devices in the Physical Characteristics shell. A fragment of the network for a procedure is shown in Figure 7. Note that the network is, for the most part, a decomposition of the procedure into a sequence of steps. Also associated with the procedure are problems that may arise in the course of the procedure and any other notes that the developer deems pertinent, for example, the schedule note in Figure 7.

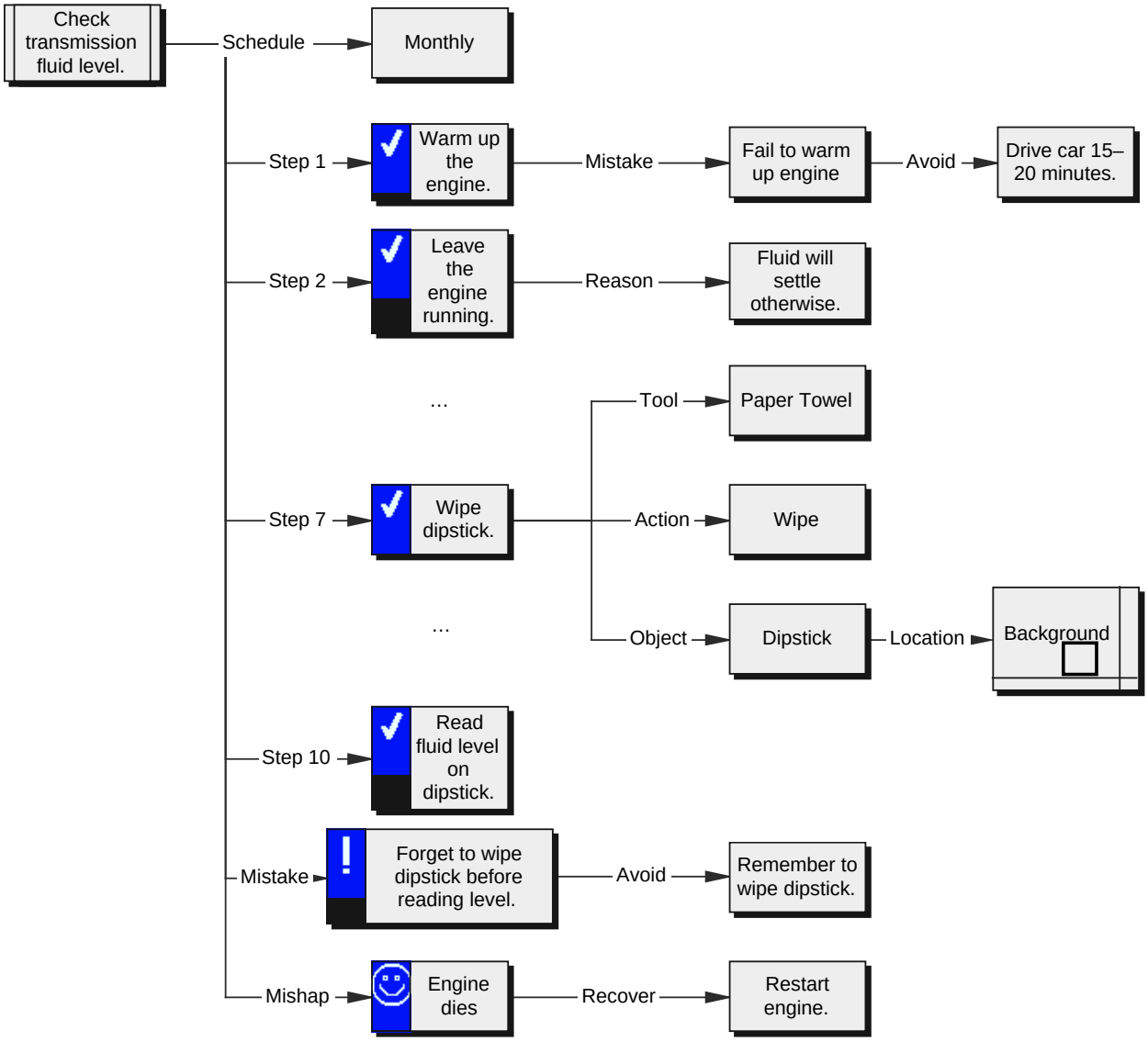

Figure 7. Fragment of an XAIDA Procedure Knowledge Representation.

Steps

The information associated with each step, like that associated with the main procedure consists of problems and attributes. In addition, a schema is associated with each step that designates the tool, if any, used in carrying out the step, the action required of the technician, and the object that the step acts upon. Any number of actions can be associated with an object. Each object has a location on a developer-designated graphic called the background. Backgrounds can have several objects and can be associated with more than one step. The tool-action-object schema is used to allow the student to practice executing the step and recalling its components.

Notes

Just as developers can add facts to physical-characteristics semantic networks, so can they add notes to a procedure or any of the procedure's steps. The "schedule" and "reason" notes in Figure 7 are examples of notes. Developers can designate notes as either optional or obligatory. The former can be skipped by the student and are not addressed in practice. The latter must be viewed by the student and are addressed in practice.

Problems

Developers can describe problems that arise in the course of the procedure or in the course of particular steps. There are two types of problems. Mistakes are problems that arise as the result of "technician error." Developers can associate with mistakes information on how to avoid mistakes and how to recover from them. Mishaps are problems that arise by accident, hence avoidance information is not pertinent to mishaps. Developers can, however, provide recovery information about particular mishaps.

A Focus on Simple Procedures

It should be noted that a Procedures network can only accommodate a single procedure at this time. That is, subprocedures must be fleshed out in separate networks (which in turn must be presented in separate Procedures lessons). This differs from the Physical Characteristics shell, in which a single lesson may present a system with many levels of subparts. However, in the future we hope to allow developers to link in other Procedures lessons as resources from within a Procedures lesson.

Also worth noting is that the shell does not address two central issues in the representation of procedures, namely, cognitive operations and branching. We have purposely bypassed these two issues, partly because they are not central to most maintenance procedures and partly because they have received extensive treatment both in the instructional literature and the artificial intelligence literature. We do, however, explicitly address some important aspects of cognition and branching that have not been given extensive attention, namely, recognition of problems that arise during execution of procedures and steps to recover from these problems.

Instructional Materials

Instructional materials for procedures training are of several sorts.

Virtually all elements of a procedure knowledge base have names, descriptions, and resources, as do elements of the knowledge structures for the two shells discussed above.

Illustrations are used to show the student the procedure as a whole, each of its steps, problems, avoidance procedures, notes, and recovery procedures. These illustrations can be text, graphics, video, or a slide show. Players are provided for the latter two methods.

Portrayals are used to give the student practice with a procedure. A portrayal consists of a background graphic along with locators for a number of objects. The locators can be labeled, and, when selected present a menu of actions available for the object. A multilevel capability allows students to zoom in and out through multiple backgrounds when a procedure addresses a particularly complex system.

Because procedure training is often conducted in the context of technical documentation, procedures, steps, and problems can have hypertext links (html anchors) to electronic documentation so that students can consult the documentation during training.

Instructional Delivery

Like XAIDA's other three shells, Procedures incorporates a browser and practice.

Browser

The browser presents information in sections. There are separate sections for the introduction, an overview of the procedure, presentation of each step in the procedure, and problems associated with the procedure as a whole. Students may explore material from any section in the browser at any time, or simply keep pressing Next for a sequential presentation.

The Introduction (Figure 8) consists of an instructional objective and any number of developer-specified resources. Material in the Introduction is not used in generating practice questions.

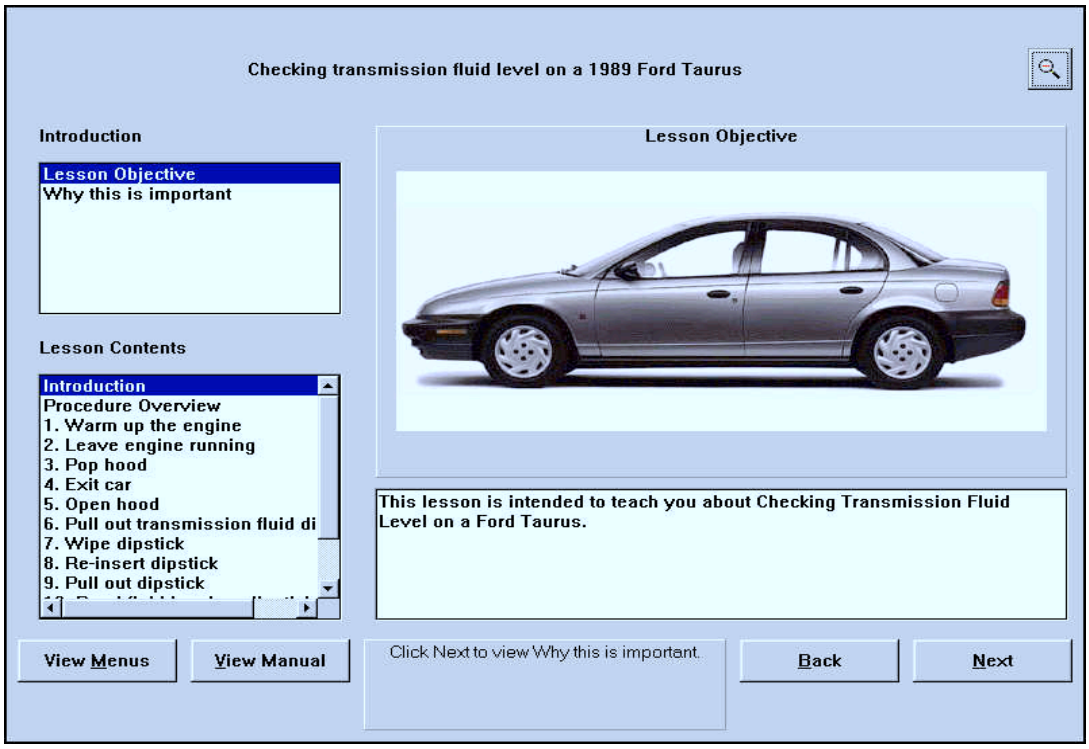

Figure 8. Procedure Introduction.

The Procedure Overview (Figure 9) presents the name and description of the procedure, any associated facts and resources, and a procedure overview. If the developer does not specify a procedure overview, XAIDA will by default create one by concatenating the illustrations from all the steps.

The Step Presentation (Figure 10) sections present material associated with each step. If any mistakes or mishaps have been associated with a step, they are presented immediately after that step--mistakes first, then mishaps.

The Procedure Problems section presents mistakes and mishaps associated with the procedure as a whole. These are problems which either affect the performance of the whole procedure, or which can't be associated with any particular step. Again, mistakes are presented first, followed by mishaps.

Practice

The Procedures shell's design calls for 19 types of practice questions, similar to those employed in the two shells described above. One of these types is illustrated in Figure 11. As in the other shells, the practice environment is fully supported with hints, a lookup capability and detailed feedback. Unlike the other shells, the design of the Procedures shell calls for practice during presentation and in three other contexts as well.3

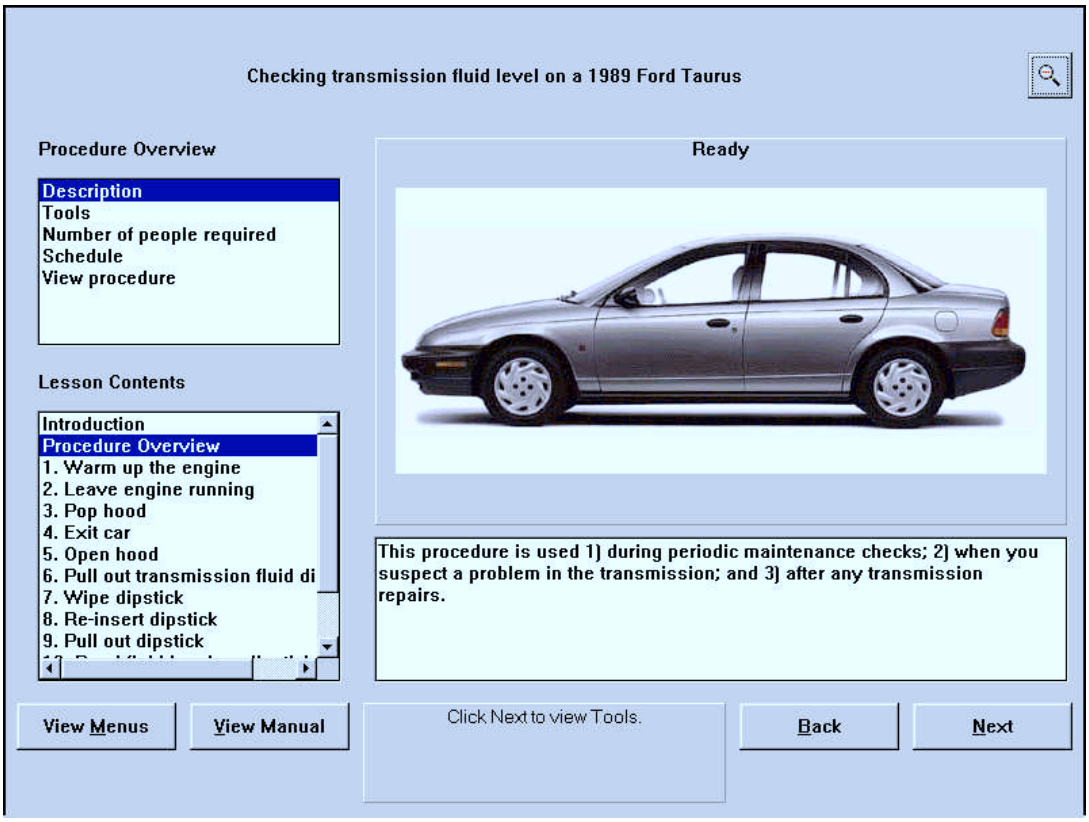

Figure 9. Procedure Overview.

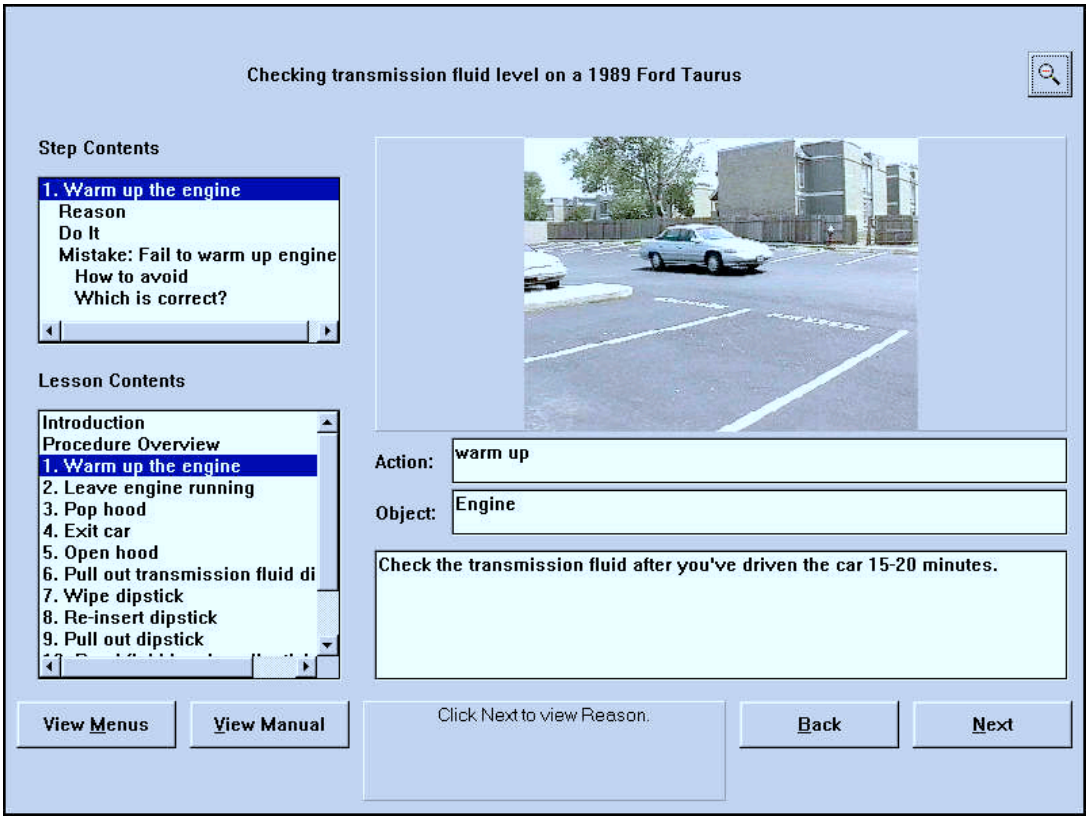

Figure 10. Step Presentation.

Embedded practice. The purpose of embedded practice is to intermingle practice with the presentation of the steps and main procedure, under the assumption that immediate practice makes learning more interactive and facilitates recall of the material just presented. There are three occasions where an embedded practice item will occur. After the presentation of a step, but before presentation of its associated mistakes or mishaps, XAIDA asks the student to select the appropriate tool-action-object combination for that step (see Figure 11). After the presentation of a mistake/mishap and its how-to-avoid/normal condition, XAIDA shows two pictures and asks the student to select which depicts a correct/normal condition. Finally, after the presentation of how to recover from a mistake or mishap, XAIDA asks a text multiple choice question about how to recover.

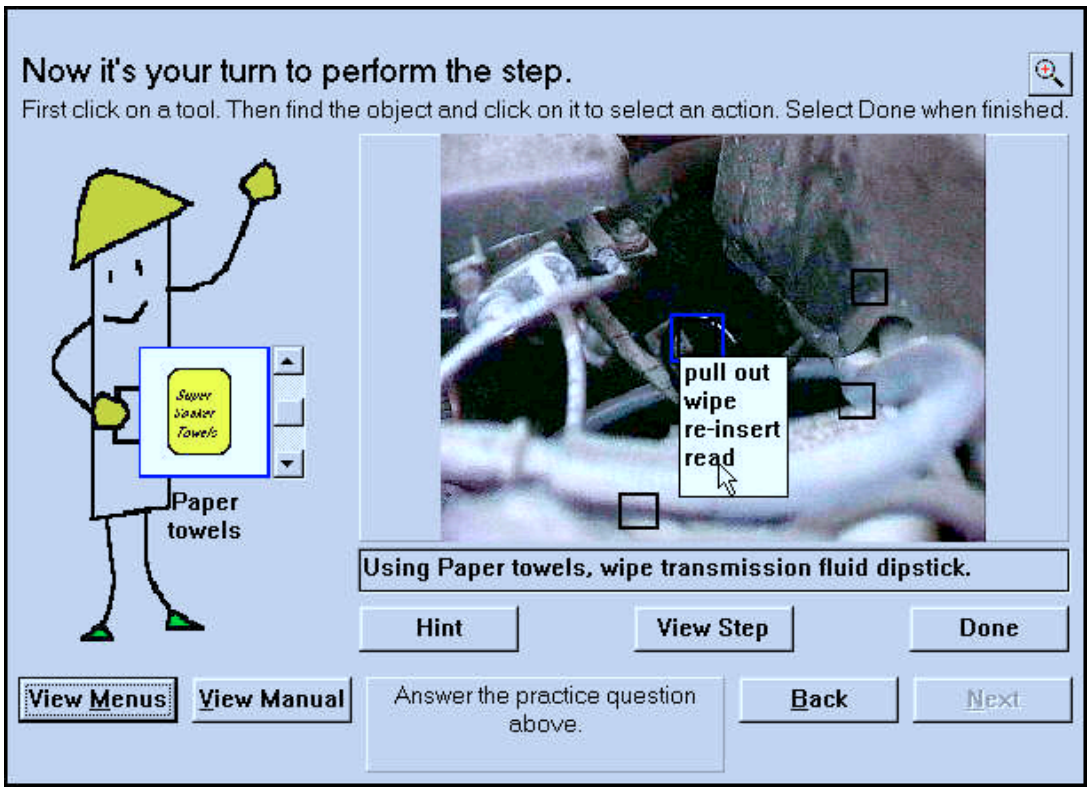

Figure 11. Do-Step Practice Display.

Review practice. Review practice incorporates a technique human instructors sometimes use to help students go over recently presented material. Its purpose is to reinforce both the sequence of the procedure and the things to think while doing the procedure. It achieves this by asking questions about the material just presented in a "tell me what I told you" fashion. For example, in reviewing a step and an associated mistake, it would ask:

<given an illustration> What step is this?
Pick the reason(s) for fill reservoir to half-way mark.
Do this step.
What mistakes are associated with fill reservoir to half-way mark?
What problem is this?
Why is this wrong (or abnormal)?
<Given problem label> How would you avoid this problem?
<Given problem illustration> How would you recover from this problem?

For the sake of variety, the phrasing and sequence of questions may vary within review practice for a given step; however, the procedure or step the questions are based on will always be the same. For example, when asking about the reason for a step, XAIDA might in one

instance have the student refer to the name of the step and in another instance a picture of that step.

Procedure practice. The purpose of procedure practice is to allow concentrated practice on performing the steps of a procedure in order. The student is asked do-it questions (see Figure 11) for each step in succession. Feedback is provided after each incorrectly performed step. The student is allowed to continue the procedure after making an error; but at the conclusion of the procedure, if he has made greater than some criterion number of errors (default = 0 errors), he is required to repeat the procedure from the beginning.

Extended practice. Retrieval demands are not always sequential. Therefore it is necessary to provide practice under non-sequential conditions. The purpose of extended practice is to reinforce the student's knowledge of procedure-related information by providing non-sequential practice. As in Physical Characteristics, XAIDA utilizes an algorithm to select a set of triplets to ask about. All triplets are eligible for questioning, but the algorithm causes XAIDA to tend to ask about material the student has not mastered yet. However, unlike Physical Characteristics practice, which jumps from question to question with no regard for conversational flow, Procedures at least attempts to ask questions in related groups--e.g., it will ask about several questions in a row concerning a selected step or attribute.

Lesson Development

There is no Develop software available for Procedures. However, it would be very much like Physical Characteristics Develop in having mostly WYSIWYG editing with a few forms as needed. We have also explored the idea of using wizards for initial data input, and the WYSIWYG screens for further editing and/or tweaking.

XAIDA and Other Procedures Tutors

In the introduction to this Procedures section, we stated that XAIDA's approach to teaching maintenance procedures is based on our observations of the kinds of things classroom instructors say and do. We have identified two other computer-based approaches to teaching procedures.

The first is to demonstrate a procedure, and then with varying degrees of instructional support, allow the student to practice the procedure using a graphical simulation of the equipment. This could probably be characterized as a classical computer-based approach to teaching procedures, and is found in systems dedicated to teaching a particular subject (e.g., Orey et al., 1995), as well as in authoring tools. A exemplary authoring tool which uses this approach is RIDES (Munro et al., 1994). In RIDES, the developer's main tasks are to create the graphical simulation and define the sequence and degree of support.

The second approach, used by the Instructional Simulator (Merrill, 1997), is similar to the first in that the system teaches a student the steps in a procedure and gradually fades instructional support until the student can do the procedure independently. The primary difference is in the manner in which the instruction is created. In the Instructional Simulator, procedures are derived from a developer-defined qualitative model of how the system works. After the model has been defined, a developer or student simply specifies a goal and a set of initial conditions, and the system automatically generates both the procedure and the instruction for the procedure. This is a very easy way to generate instruction as long as the steps the student must perform match the model.

The primary difference between the approach used by XAIDA's Procedures shell and those used by these other systems is that XAIDA addresses not only correct sequence and performance of steps in a procedure, but also other types of knowledge related to step performance, such as reasons, mistakes, and mishaps. We view this approach as a first step towards incorporating notions based on theories of action (Miller, Galanter, & Pribram, 1960; Norman, 1981) into instructional practices. These theories call our attention to the ubiquity of human error in even the simplest of performances, to the loose coupling of intention and action,

and to the pervasive role of feedback in training and performance. Learning a procedure is not just a matter of learning the sequence of steps; it also involves to a large extent, how to accomplish what is intended at each step.

TROUBLESHOOTING

Troubleshooting, by reputation if not in fact, is one of the most critical objectives in maintenance training. The task itself is easily characterized. A system consists of a number of components, at most one4 of which may be faulted. Clues to a fault's identity or location are available through a number of tests; the results of each test are determined by the fault state of the system. Furthermore, each component in the system can be replaced or repaired. A cost can be assigned to each possible test, repair, or replacement. Troubleshooting is the practice of testing, repairing, and replacing components in such a way as to eliminate any fault. Troubleshooting performance can be evaluated in terms of the total cost of the operation and its effectiveness in eliminating any faults.

Troubleshooting skills appear to be as complex as the task itself appears to be simple, and training requirements vary widely. As a gross oversimplification, we can say that effective troubleshooting calls for a space-splitting approach, that is, a sequence of observations that narrow the fault down to successively smaller regions of the system. The keys to successful troubleshooting are choice of a maximally informative test at each point and accurate interpretation of the test's results. Troubleshooters sometimes rely on system-specific troubleshooting guides or procedures to lead them to the faulted component. At other times they analyze the functional dependencies among components to determine the best test and interpret its results.

XAIDA is concerned with system-specific troubleshooting procedures, those normally found in technical documentation. An examination of training and technical manuals presents a clear picture of these procedures.

First, virtually all of them rely implicitly on space splitting. They are typically presented to readers in the form of unannotated decision trees, but an analysis of these trees reveals sequences of decisions which confine the possible fault location to increasingly smaller subregions of the system.

Second, the structure of the troubleshooting procedure is almost never made explicit in its documentation. In particular, the documentation discusses neither what region or module a test addresses or what can be concluded from the test's outcome. XAIDA teaches students how to reach these implicit conclusions as they follow the troubleshooting procedure. The rationale for this choice of an instructional goal is treated below, but is best understood in the light of XAIDA's training methods.

Knowledge Representation: Fault Trees

As is mentioned above, XAIDA's troubleshooting training addresses both the procedural aspects of troubleshooting, that is, how to select and sequence tests, and the reasoning processes that should be employed during the troubleshooting procedure, that is, what conclusions can be drawn from each test. This knowledge is captured in a structure called a fault tree. Figure 12 exhibits one such tree, adapted from Halff (1990b), illustrating an abbreviated troubleshooting procedure for a no-start condition in a T-38A aircraft.

Nodes in the structure represent regions of the system. Consistent with current practice, we refer to a fault tree's terminals as components, to its root as the system, and to non-terminal nodes as modules. The troubleshooting procedure calls for successively testing modules and submodules until the fault is localized to a component. The component is then repaired (or replaced).

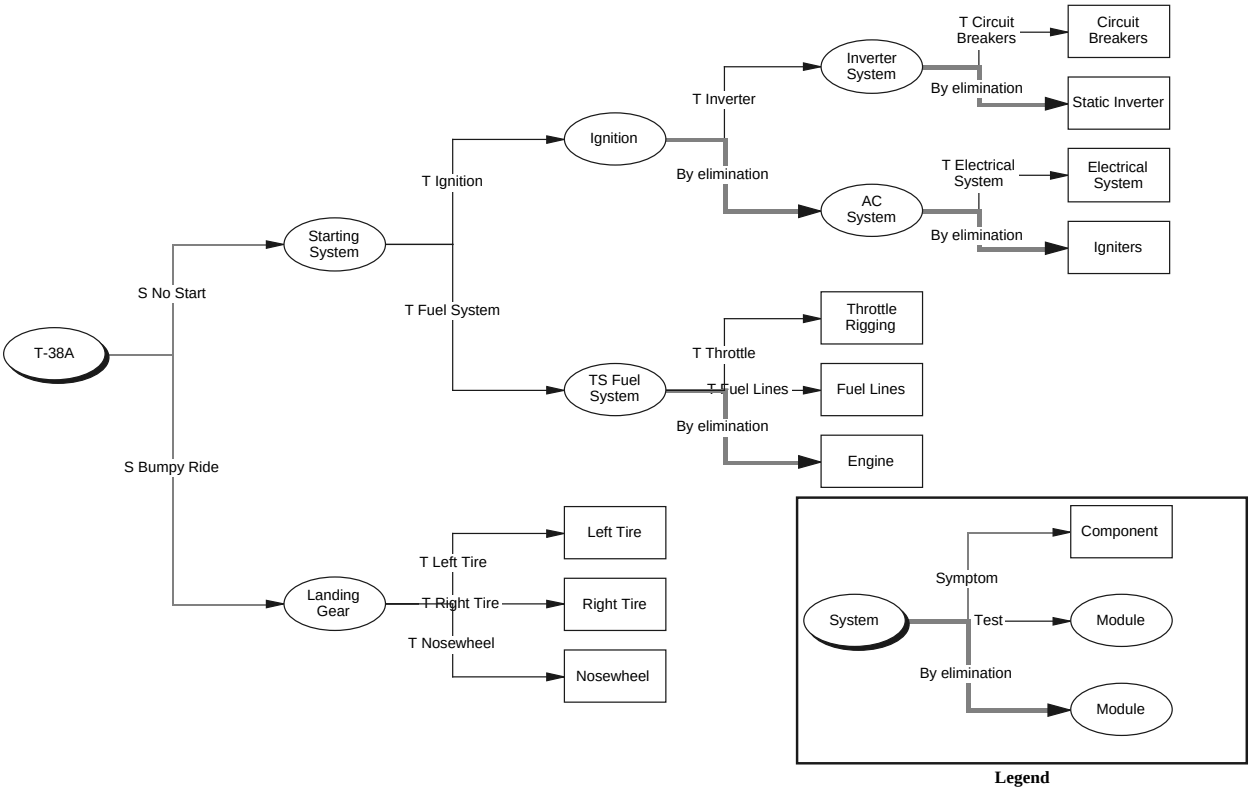

Figure 12. Fault Tree for T-38A.

Links represent tests of the nodes to which they point. Associated with each test are two outcomes, one indicating that the target node (module or component) is functioning normally, the other indicating that the node is malfunctioning. A special last resort link implicates its target when all other possibilities have been eliminated.

The fault tree encodes a troubleshooting procedure for the system. Implementing the procedure is a matter of traversing the tree using the following recursive procedure.

To troubleshoot Region M,
    if M is a component then repair/replace it;
    otherwise,
        test each of its subregions until you find one (M') that fails or that is implicated by elimination (a last resort);
        then troubleshoot M'.

Note that the system can reach an impasse if a module fails its test, but all submodules pass theirs. This happens quite often in real systems. XAIDA precludes this kind of behavior by ensuring that a test fails if and only if a component of the module tested is faulted. Also note that XAIDA enforces a specific testing order for submodules. This constraint is consistent with existing maintenance practices and can be justified on economic grounds. In this regard, note that only one submodule of a module can be implicated by elimination, and that this module comes last in the testing order.

Some links in the fault tree are distinguished as symptoms. A symptom is a direct manifestation of a system malfunction, for example, a failure of the system to start. Symptoms function as tests because they carry diagnostic information about possible faults, but they also serve as entry points to the troubleshooting procedure since they represent occasions for troubleshooting. Indeed, every fault has a single (but not unique) symptom.

Related to the notion of a symptom is that of a scenario. A scenario is a particular manifestation of a symptom. It may be a trouble report, a video showing how the system

malfunctions, or other stimuli of this sort. Any number of scenarios can be associated with each symptom. XAIDA randomly chooses one at the beginning of each exercise.

Instructional Materials

Troubleshooting lessons contain an introduction that is virtually identical to that used by the physical characteristics shell. In the former however, the lesson outline is replaced by a graphical depiction of the fault tree.

A variety of materials are associated with elements of the fault tree. Each module, component, fault, test, and repair has a name and a description. Modules and components can have graphical portrayals and links to Physical Characteristics lessons. Tests and repairs can have links to Procedure lessons. Associated with each test is a description of the normal outcome, a description of the failure outcome, and resources illustrating each of these two outcomes. Certain materials can also be associated with scenarios. For example, a graphic or video might be used to exhibit a malfunctioning system, or text might be used to simulate a filed trouble report or complaint.

Instructional Delivery

XAIDA's Troubleshooting shell, like the shells discussed above, consists of a browser and a practice environment. The browser is presented as a Troubleshooting Guide as shown in Figure 13. The Troubleshooting Guide presents a number of different types of information, but its main function is to exhibit the regions and subregions of the system and the test used to check each module and component. Figure 13 shows that the Starting System module of the T-38A aircraft consists of Ignition and Fuel modules.5 These submodules are listed in the order tested and the testing procedure for each one is shown as well.

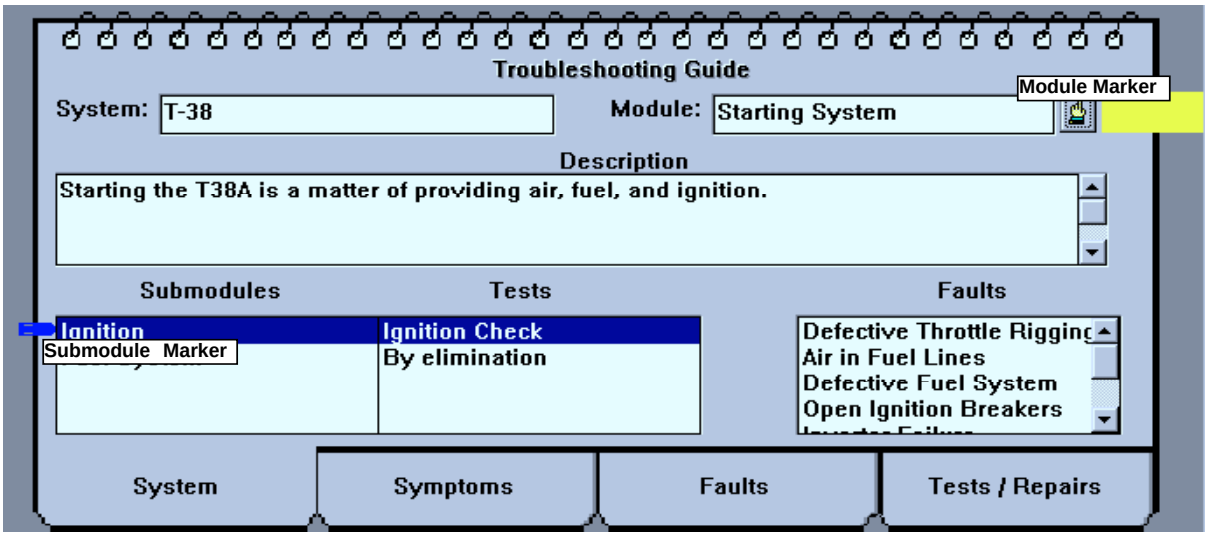

Figure 13. Troubleshooting Guide. (Note the "sticky note" marking the module being investigated and the tab marking the submodule under investigation.)

The practice environment is somewhat more complex. It consists of the Troubleshooting Guide, a simulation of the system called the Stopgap Simulation System (SSS), depicted in Figure 14, and a record of the troubleshooting procedure cast as a "Service Order" (Figure 15).

The SSS, depicted in Figure 14, allows the student to select tests or repairs and to view the results of these actions. In particular, the student can choose to check any symptom, perform any test, or repair any component. The results of the chosen action are then displayed to the student. The SSS determines the outcome of an action by consulting the fault tree and the fault state of the simulated system. A symptom is manifest whenever one of its associated faults is

present. A component test fails if and only if the component is faulted. A module test fails if and only if one of the module's components is faulted. Repairs always succeed in removing any fault from the replaced component.

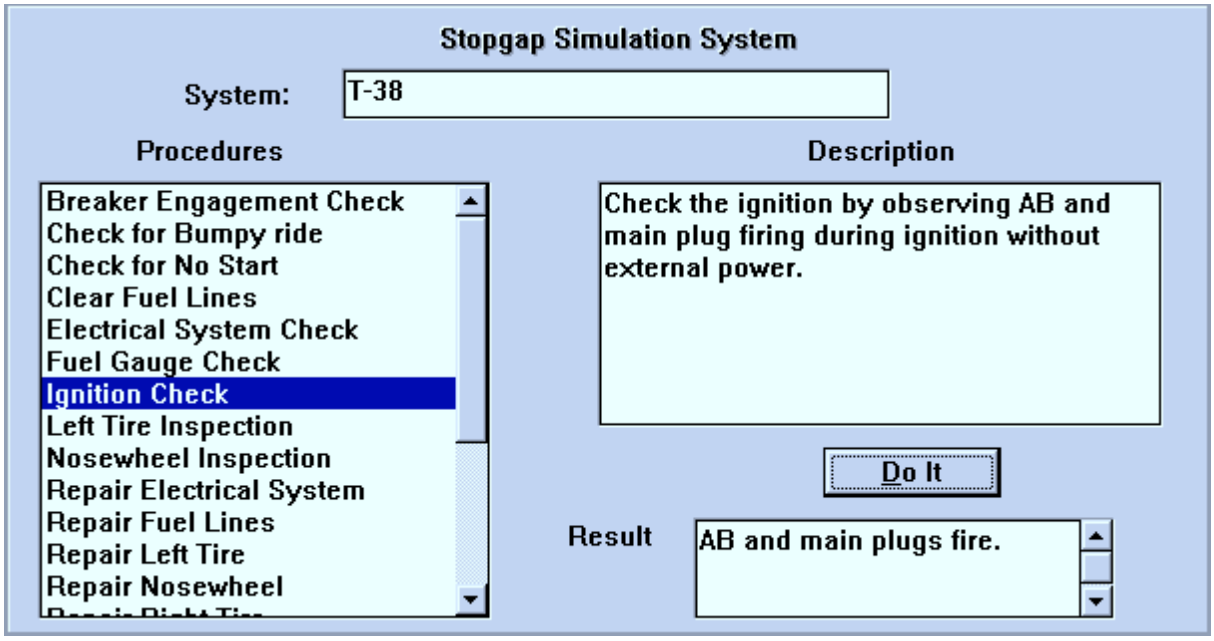

Figure 14. Stopgap Simulation System.

The Service Order, depicted in Figure 15, contains a description of the symptom derived from the chosen scenario. More importantly, it records in a sequence of notations, all observations and the conclusions drawn therefrom. Each notation has three parts: the observation itself, derived from the output of the SSS, the module or component tested, and the implication (faulted or normal) of the observation about the condition of the tested module. The student assembles the notation using a menu-based editor (shown in Figure 15) listing only the possible observations, modules/components, and implications.

XAIDA's instructional procedures for troubleshooting are based on practice in using the procedure represented by a fault tree. That is, XAIDA dynamically constructs a sequence of exercises, each of which requires the student to troubleshoot a system with a particular fault. Understanding the procedure is a matter of understanding how systems and faults are selected (curriculum construction) and how a particular exercise is conducted (practice).

Curriculum Construction

Troubleshooting curriculum construction in XAIDA is based on notions elucidated in Van Lehn's (1987) Step Theory and Reigeluth and Stein's (1983) Elaboration Theory. The driving notion is that the curriculum is divided into units that we call levels, each of which introduces a new variant of the procedure. In our case, each level introduces a new fault. Practice at each level consists of exercises on the new fault and on faults introduced at previous levels. Thus, for the fault tree shown in Figure 12, Level 1 introduces the student to the circuit-breaker component and its fault, Level 2 adds the static inverter, and so on.

It is also worth mention that, consistent with step/elaboration theory, the knowledge base is pruned to the student's current level. In particular, the material in the Troubleshooting Guide addresses only the faults introduced at that level and the regions of the system that contain those faults. Likewise, the SSS allows only those actions that pertain to the pruned region. Thus, the student's view of the system becomes more complex as he learns to troubleshoot more faults.

The exercises provided at each level are based on a student model similar to those employed in the other shells. In particular, the first exercise at any level addresses the fault introduced at that level. The second exercise addresses the fault introduced at the previous level. Exercises beyond the second are chosen randomly from those at or below the current level and in a fashion that is biased by a student model.

The student model tracks competence in two types of skills, that of choosing the correct procedure to test or repair a module or component, and that of noting (on the Service Order) what can be concluded from an observation. The system notes a student's weaknesses in each of these individual skills and tends to select those exercises that address the most weaknesses. The student normally moves on to the next level when he or she exhibits no weaknesses at the current level.

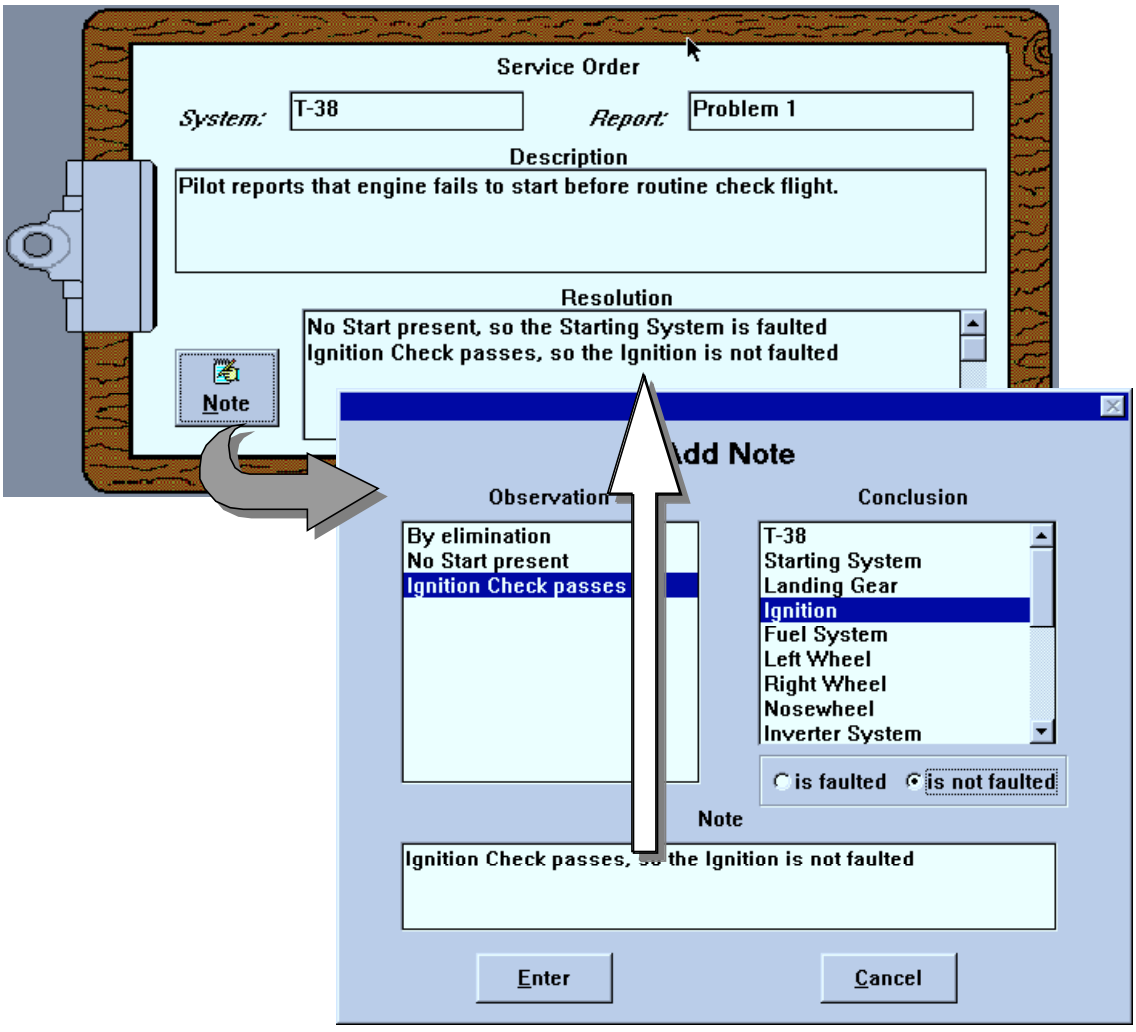

Figure 15. Service Order.

Practice

The responsibilities of the practice module are to provide an environment for support of troubleshooting practice, provide for guidance in the form of a tutor, and track performance for student-modeling purposes.

Practice environment. The practice environment, as is mentioned above, consists mainly of the Troubleshooting Guide, the SSS, and the Service Order. A Navigator is also provided to provide for the transitions between levels and exercises, and to allow for student-initiated review exercises.

Students complete each exercise using a three-step cycle.

1. Use the Troubleshooting Guide (Figure 13) to determine the next action (test or repair), marking the Guide to indicate the module known to be faulted and the submodule under test.
2. Use the SSS (Figure 14) to carry out the required action and observe the result.
3. Note the observation and what can be concluded from it on the Service Order (Figure 15), using the conclusion to guide the next cycle.

Note that students are required to keep detailed notes on the troubleshooting process. This reflects a general principle, called the show-work principle by Van Lehn (1987) that requires instruction to make explicit the normally implicit cognitive steps in a procedure. It also reflects our concern for teaching students to make inferences about possible malfunctions; these inferences are almost never made explicit in troubleshooting guides. The show-work requirement also simplifies the task of a model-tracing tutor that guides practice.

Guidance. A model-tracing tutor (Anderson, Corbett, Fincham, Hoffman, & Pelletier, 1992) is a form of intelligent tutoring system that tracks a student's progress during an exercise in such a way that it can offer context-specific advice when needed, intervene when students pursue nonproductive actions, and intervene when students manifest certain common bugs.

At their most sophisticated, model-tracing tutors operate in an exploratory environment and exercise considerable sophistication in inferring student plans or bugs. XAIDA's troubleshooting tutor has none of this sophistication. It operates in a procedure-learning environment, where students are never allowed to stray from the target procedure, and has no inference powers since students are required to make explicit the results of cognitive operations. Hence, the tutor knows precisely what needs to be done at each step in the procedure. If the student asks for advice, the tutor responds with a discussion of the next step (see Figure 16). If the students attempts an incorrect action, the tutor also responds with a discussion of the correct action. In this regard, the only actions checked for correctness are moving a marker in the Troubleshooting Guide, attempting an action in the SSS, and adding a note to the Service Order. The tutor's advice is context specific; it is formed by filling a template specific to the required action with information from the process' position in the fault tree.

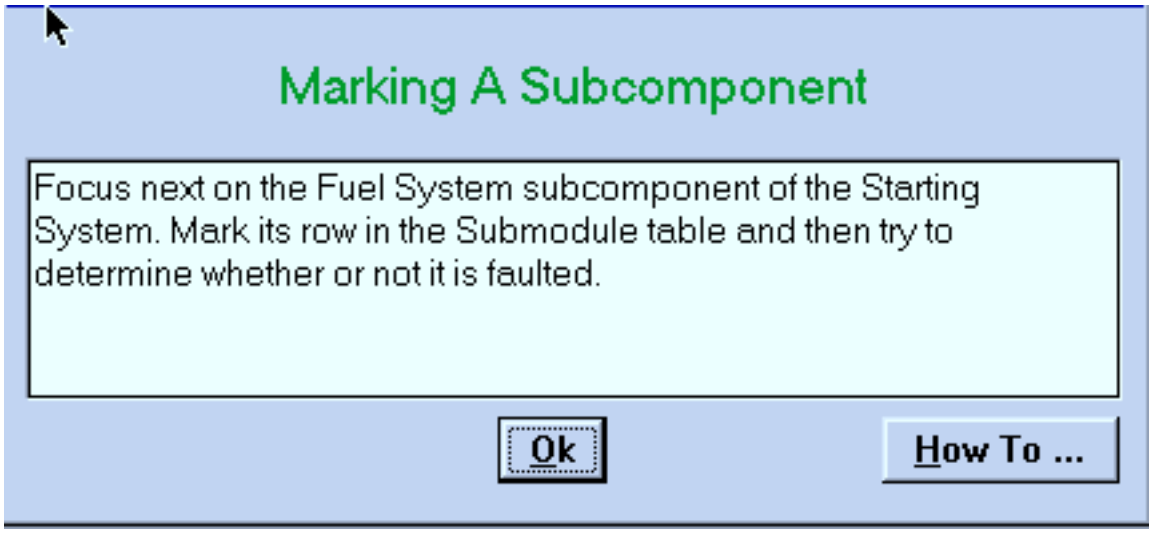

Figure 16. Example of Model-Tracing Tutor's Advice.

Performance Tracking. The practice module must also be alert to weaknesses in the skills tracked by the student model. This function is implemented by noting those occasions where the student chooses an incorrect action from the SSS and those occasions where the student

tries to enter an incorrect conclusion in the Service Order. The former type of error is taken as a weakness in selecting a test or repair for the component or module under consideration. The latter type of error is taken as a weakness in making the correct inference in response to the most recent SSS observation.

Unlike the other XAIDA shells, the student model in the Troubleshooting shell puts the student in an initial state of mastery of all skills. This decision reflects the fact that the student has before him (in the Troubleshooting Guide) all of the data needed to make the correct move at each point. The reader may well ask what, if initial mastery is presumed, does the troubleshooting shell aim to teach. Our less than precise answer to this question is that the training should (a) instill in the student the discipline of making appropriate inferences during troubleshooting, (b) promote memorization of the material needed to make these inferences, and (c) remediate any particular weaknesses in the student's inference skills. Only the last goal (c) is specifically tracked by the student model.

Instructional Development

As is mentioned above, we have not worked out the authoring procedures for the Troubleshooting shell. The shape of such procedures however is clear. Apart from the Introduction, the knowledge required for troubleshooting can be captured via three coordinated editors. Developers would "write" the Troubleshooting Guide, thereby specifying the structure of the fault tree along with material for the elements in this structure (chiefly modules, branches, and tests). A secondary capability would allow developers to edit the SSS in order to specify the consequences of tests, repairs and replacements. Finally, a scenario editor based on the Service Order would allow developers to specify a collection of scenarios manifesting the possible symptoms of each malfunction.

XAIDA and Other Troubleshooting Tutors

Most informed readers will, at this point, be asking themselves how XAIDA's approach to troubleshooting compares to other troubleshooting tutors including MITT/MITTWriter (Norton, Wiederholt, & Johnson, 1991; Wiederholt, 1991), RIDES (Munro et al., 1994), and Sherlock (Lesgold, Lajoie, Bunzo, & Eggan, 1992). Put simply, these other tutors are simulation-based and focused on problem-solving skills, whereas XAIDA's Troubleshooting shell is procedure-based and focused on pattern-recognition skills.

Simulation-Based vs. Procedure-Based Training

Perhaps the popular and effective training principle of all time is "Practice makes perfect," and the most straightforward implementation of this principle in troubleshooting is to provide students with a simulations of a variety of faults and let them have at it. Although there is no doubt that this approach is effective, there are (at least) two good reasons for rejecting it as a panacea.

First, simulations are difficult to construct, particularly when one must simulate not only normally functioning equipment but also all possible fault states. Consider the three systems mentioned above. Sherlock's simulations were constructed on the basis of a lengthy, expensive, and effortful series of in-depth structured interviews with experts. Sherlock's simulations reflect nothing more than those experts' consensus concerning all possible contingencies in the system. RIDES uses a more principled approach to simulation and is considerably more tractable than Sherlock's development process. Still, a RIDES simulation of any complexity requires considerable expertise on RIDES' simulation techniques, and may be only distantly related to the principles governing the function of the device. Only MITTWriter comes close to a system that is approachable by novice developers, and its simulations are little more than those provided by XAIDA's Theory of Operation shell.

Our survey of these and other simulation systems such as RAPID (Emultek, 1997) and PowerSim (Byrknes & Myrtveit, 1997), have led us to the conclusion that simulation, although a seductive and powerful methodology, is not the sort that is now easily accessible to SMEs. We therefore took the step of trivializing the simulation aspects of XAIDA's Troubleshooting shell to a simple, table-driven mechanism (the SSS), one whose tables are small enough to be easily constructed in the course of defining a fault tree. The developer that demands realism has some recourse by using multimedia or other interactive programs (even a simulation) as resources for the SSS.

Second, troubleshooting in the real world is largely procedure driven. Virtually every complex piece of equipment comes with a troubleshooting guide. Students are often required to employ the procedures in the guide, and it is likely that experts adopt these procedures to their own use. But the procedures in troubleshooting guides in general have two major deficiencies. First, they contain virtually no explanation of their rationale, that is, of the reasons for undertaking any particular troubleshooting action or the conclusions supported by any particular action. Second, they are, for the most part, incomplete and offer no assistance in the many cases where they fail. As the consequence of these two deficiencies, a naive user of the procedure will often find herself at an impasse without a clue of what has been discovered or how to adapt the procedure to novel conditions. XAIDA offers a partial solution; it teaches technicians what can be learned in the course of executing a troubleshooting procedure. XAIDA's solution, of course, is also incomplete in that it provides no practice in dealing with a troubleshooting procedure's failures. Ideal would be a combination of procedure-based and simulation-based training, one in which the simulation-based training directly addresses failures of troubleshooting procedures and takes account of progress made while following the procedure.

Problem Solving Skills and Pattern Matching

Others (e.g., Rouse & Hunt, 1984) have pointed out that skilled troubleshooting has two components: a problem-solving component used to deal with novel faults in unfamiliar equipment and a pattern-recognition component used for familiar faults. The former is far less efficient than the latter, but is often deemed to be more critical, partly because it is more prone to failure, and partly because it is more interesting. RIDES and MITT and, to a lesser degree, Sherlock, are all concerned with problem-solving skills. The choice is natural. These skills are important, and computers, after all, are very good at solving troubleshooting problems. All three systems have their own troubleshooting experts that can dole out troubleshooting advice on demand for any circumstance. (It may or may not matter that the experts are so good that their methods are well beyond the mastery of any human troubleshooter.)

XAIDA is more concerned with pattern-recognition skills, and consciously so. There are three reasons for this focus. First, pattern-driven troubleshooting practices are both ubiquitous and efficient. We wanted to make up for some of the neglect that they have suffered in the training literature. Second, XAIDA gains its economic advantage as a system-specific trainer whereas troubleshooting problem-solving techniques are, by their nature, system-independent. Put differently, XAIDA is meant to be used by SMEs to develop training in the particular methods that pertain to particular systems, hence the focus on ease of use and development efficiency. Those interested in teaching general problem-solving techniques for troubleshooting would be better advised to attack the problem at a level of generality greater than that of an individual system. Otherwise they risk training people to use inefficient, problem-solving methods in the very cases where more efficient pattern-recognition methods apply. Finally, we felt more comfortable with a tutoring approach in which the expert model actually employs the same approach that students are expected to master. The brute-force experts of simulation-based approaches do not satisfy this constraint.

CONCLUSION

We conclude with a brief summary of XAIDA research, a description of where XAIDA stands in relation to other ITS authoring tools, and directions for future research and development.

Research with XAIDA

As mentioned in the introduction, XAIDA's main objective is to allow subject matter experts to produce effective interactive courseware with minimal training and effort. To meet these objectives, the design of XAIDA has been continually informed by a program of empirical research involving developers, students, and the organizations that will ultimately use the product. The following sections summarize the studies and their findings.

Studies Involving Students

There have been thirteen studies of students taking XAIDA lessons (see Appendix 1). These studies have investigated not only XAIDA's overall instructional effectiveness, but also the effectiveness of various elements of XAIDA's instructional presentation, such as the browser and the adaptive practice; the relative benefits of taking an XAIDA lesson individually or within a team of two; and the benefits of audio. These studies were also important in helping us refine XAIDA's Deliver interface.

Our results point to three conclusions. First, XAIDA is successful in almost any context at promoting mastery of typical performance standards. Second, if students are intrinsically motivated to master the material, XAIDA promotes the development of the deeper cognitive structures typical of experts in the subject. Third, these deeper structures do not appear to emerge if students are not intrinsically motivated to master the material.

Studies of Developers

Because major design goals for XAIDA are that it be easy to use and easy to learn to use, there have been eight field studies of developers using XAIDA. Typically, these studies have taken place in the context of a three day training class for developers. Issues of interest have included usability of the software (both in general, and for specific interface elements), attitudes toward XAIDA and instructional technology in general, changes in developer's knowledge structures after learning about XAIDA, the uses of XAIDA in classrooms and other less than ideal settings, the impact of XAIDA on the structure of instructional development teams, and the influence of XAIDA on Air Force organizational plans and policy (Wenzel & Dirnberger, 1997; Wenzel et al., 1997). A brief summary of the eight field studies can be found in Appendix 2. The types of data gathered are listed below.

4. Comments
5. Attitudinal - expectations and impressions - administered at six different points throughout training
6. Journal files - record of user actions collected automatically by XAIDA
7. Behavioral - ability to perform selected development tasks
8. Self-reported computer skills - administered pre- and post- training
9. Self-reported XAIDA skills - administered pre- and post- training
10. Self-reported usability/understandability of XAIDA materials - administered after training to assess ease of understanding and use of various development features, interfaces, and support materials.
11. Knowledge structures - as measured using Pathfinder scaling methodology (Schvaneveldt, 1990); administered after training.

12. Self-reported organizational information - perceptions of various factors related to organizational acceptance of new technology

The findings can be summarized as follows. First, we have found that developers can indeed create XAIDA lessons quickly and easily. In fact, seven of the studies involving students (listed in Appendix 1) utilized lessons created by XAIDA training alumni (Wenzel et al., 1997; Wenzel & Ellis, 1997). Our informal estimates of the time to develop a 1–2 hour lesson is 3–4 days including training. One young high-school student serving an internship at USAF Armstrong Laboratory developed a lesson after two hours of training and with only 24 hours of working on her own.

Second, although XAIDA was designed to be used by the maintenance training community, it (the Physical Characteristics shell, in particular) has also proven to be quite versatile—and in demand—for teaching certain topics within other domains, including medicine, algebra, computer literacy, military customs and courtesies, and biology (see Appendix 1 and 2 for a more detailed listing of lesson topics). In fact, one developer, who wound up teaching the material in her XAIDA lesson on parabolas on a day when XAIDA was not available, reported that her lecture was successful because she had adopted the same approach she used in her XAIDA lesson! We believe this demonstrates 1) the ubiquitous nature of factual knowledge; and 2) that there is a high demand for an easy-to-use authoring tool for this type of knowledge.

Third, novice developers tend to want to implement their existing lesson plans. It can be difficult for them to grasp that their main task is not to teach the way they have always taught, but to express their knowledge using XAIDA's knowledge representation formalisms. This might not be an issue for a conventional authoring package such as Authorware, but since XAIDA is knowledge based, it gets maximum leverage (in terms of instructional power and development efficiency) when users work within the types of knowledge representation it offers. In fact, once developers catch on to the XAIDA way of thinking, they can often implement much of their original lesson plan; but they have to learn to reformulate their knowledge first (one developer compared the experience to wearing tight underwear).

XAIDA's Place in the Domain of Knowledge-Based Authoring Tools

In this section we try to show where XAIDA stands in the context of other ITS developmental research, particularly those listed in Table 3. In this discussion we point out some of the design problems that authoring tool developers must address, the approaches that have been taken to dealing with these problems, and the advantages and disadvantages of the various approaches. We conclude each section with some comments about XAIDA.

Table 3. Selected ITSs and ITS Development Systems

ITS/ITS Development System	References
Electronic Trainer	ID_2 Research Group, 1996
Eon	Murray, 1996
Goal-Based Scenario Builder	Bell & Kedar, 1995; Korcuska & Bell, 1995
GTE	Van Marcke & Vedelaar, 1995; Van Marcke, 1992
MITTWriter	Wiederholt, 1991; Johnson & Norton, 1992
REDEEM	Major, 1995
RIDES	Munro, Pizzini, Towne, Wogulis, & Coller, 1994

We structure this discussion in terms of Wenger's (1987) four components of knowledge communication in an ITS: domain expertise, student modeling, pedagogical expertise, and student interface. Since XAIDA is an ITS authoring tool, we will address its approaches to developer and student interfaces separately.

Domain Expertise

At a minimum, the domain module of an ITS should contain the knowledge needed to teach the domain. The issues are 1) what knowledge is needed; and 2) how to represent it. A good knowledge representation scheme will not only contain the needed knowledge, but also make it accessible for questioning and student modeling.

Generating a sufficiently rich and instructionally useful knowledge representation for a given ITS application is a difficult task. Teams of subject matter experts, knowledge engineers, and programmers must often work together on this problem. Deciding how knowledge should be represented becomes even more difficult for designers of ITS authoring tools, because the knowledge to be represented and the people that will be entering that knowledge are both unknowns. Some approaches that have been taken to this problem include the following:

1. Assume a knowledge engineer will be available. Eon is a suite of ITS tools intended ultimately to be used on three levels. At the top, or meta-level, a knowledge engineer helps a subject matter expert or teacher to use the tools to represent the desired knowledge. This approach allows ITSs to be developed more quickly, while still being tailored to the needs of a domain. The disadvantages are first, that a knowledge engineer must be available to help develop any new application. Second, as with any higher level software tool, there is some loss in representational power.

2. Represent as little as possible. The Goal-Based Scenario Builder and other similar tools in development at the Institute for the Learning Sciences are intended to capture teaching architectures in forms that ordinary classroom teachers can use. Examples of teaching architectures include designs for learning-by-doing through investigation (Bell & Kedar, 1995) and for making decisions based on incomplete or conflicting information (Korcuska, Herman, & Jona, 1996). To quote Bell & Kedar (1995), "the designer essentially 'fills in the blanks' of the sample interface". The advantages are that first, this approach captures and preserves several very creative instructional and interface designs. Second, with relatively little effort, non-programmers can utilize these designs to create attractive CBT. However, from a domain expertise point-of-view, the knowledge that is captured is essentially intended to fill the needs of a specific instructional approach, and cannot be used to reason with or be manipulated for other instructional purposes.

3. Maintain instructionally useful knowledge in a knowledge base; link in the rest. This approach is used in a number of authoring tools, including REDEEM, the Electronic Trainer, and XAIDA. These systems vary in the amount of knowledge they represent, and how they represent it. In REDEEM, the knowledge resides on randomly accessible pages of a Toolbook lesson. The Electronic Trainer uses a knowledge base which contains such information as captions for multimedia files, names and sequences of steps in procedures, and examples and non-examples of concepts; but other types of information must be linked in. XAIDA has four separate, rather detailed knowledge bases for physical characteristics, theory of operation, procedural, and troubleshooting knowledge. The advantage of this approach is that to the extent that authors are able to develop a rich domain knowledge base, they also lay the groundwork for the student model and pedagogical strategies. The disadvantage is that there will probably always be some instructionally useful knowledge that cannot be expressed in the available knowledge representation schemes.

4. Put the knowledge in a simulation. This approach is used in simulation tools such as MITT Writer and RIDES. The advantage of this approach is that in domains when simulation is applicable (e.g., maintenance; console operations), simulation is a very natural way to learn. The disadvantage is that simulation is not always the most appropriate teaching strategy for a given area. In fact, as a simulation becomes more

sophisticated and effective as a simulation, it becomes more opaque and less effective as a repository for instructionally useful knowledge.

For XAIDA, we have chosen to handle the representational problem with a combination of approaches: First, we have declared that XAIDA exists primarily for maintenance training, and focused on representing the four types of knowledge mentioned previously. This limits the scope of what we have to represent. Second, we use different knowledge representations for different purposes—e.g., semantic nets for physical characteristics, causal reasoning for theory of operation, and so on. This allows us to present more and different kinds of knowledge, and more importantly, gives us the granularity we need for detailed questioning. Third, we allow the author to link in any multimedia files she wants, so that she can say whatever she is not able to describe adequately using XAIDA's formalisms.

Student Model

The student model of an ITS should keep track of what the student knows at any given time. The primary purpose of the student model is to allow the system to adapt instruction to the student's ongoing needs. Issues to consider in developing a scheme for student modeling include what to model and how to use it. Some approaches that have been taken to student modeling in ITS authoring shell development include the following:

1. Don't do it. Often limited funding or research objectives preclude development of student models. For example on projects where the emphasis is on pedagogy, such as the Electronic Trainer or the Goal-Based Scenario Builder, student modeling does not appear to be an emphasis. This approach side-steps both the problems and the potential benefits of student modeling.
2. Student modeling is possible, but not automatic; and the decision about what to model is relatively open. RIDES, for example, includes a powerful authoring language which could be used to develop a student model and adapt instruction accordingly. At a slightly higher level, Eon provides tools which allow the author to specify how the student model will work, with a set of defaults. GTE appears to use some form of student modeling, it also appears to require deliberate effort on the part of the author.
3. Student modeling is not automatic and the ability to decide what to model is limited, but it is easy to do. REDEEM has a very approachable developer interface (consisting of questions and a sliding scale for responses) for student modeling. However, the author can only make the decisions the system allows her to make.
4. Student modeling is automatic, but the model is simple. XAIDA keeps track of students' mastery of the information in its knowledge bases based on their performance on practice questions. XAIDA adapts training by its tendency to ask questions about material it thinks the student has not mastered.

XAIDA's student model is simple, in that it only monitors students' performance on practice questions, but we believe it provides good bang for the buck.

Pedagogical Expertise

The pedagogical module represents an ITS's approach to teaching a particular domain. A good pedagogical module uses sound instructional strategies to achieve valid instructional goals. However, since comparisons of the soundness of any particular tool's instructional approach can be controversial, we will sidestep the issue here, other than to say that most tool developers want to somehow minimize the likelihood that their tool will be used to develop poor instruction. The problem then becomes how to codify pedagogical expertise in such a way that authors can create good instruction in spite of themselves. The following are some approaches that have been taken to this problem:

1. Develop instructional-strategy-specific tools. The Goal-Based Scenario Builder, which is based on Schank's concept of teaching architectures, is an example of such a tool. We have already noted that this approach has the advantages of being able to capture some instructional strategies and being relatively easy to use (just give the system what it asks for!). Because the theory behind the use of these tools is still being worked out, it is difficult to say for sure what the disadvantages are. However, one potential disadvantage is that these may become tools in search of appropriate instructional applications. It is even possible that an author might use such a tool simply because it is available and produces neat-looking CBT, at the expense of teaching other important types of knowledge for which there is no tool available.

2. Develop generic tools. Programmatically, this approach usually takes the form of a separate module containing generic teaching rules. GTE and Eon are examples of systems which utilize separate pedagogical modules. The advantage is that the pedagogy in these modules can be easily inspected and revised. Also, in theory at least, these modules can be used and reused for many different ITS applications. The disadvantage is that sorting through and codifying all potentially relevant pedagogical expertise is a monumental task, and people may still not agree with the pedagogy selected in the end.

3. Use built-in teaching strategies. This approach is used in Electronic Trainer and XAIDA. Programmatically, it means that the system traverses the knowledge base to teach certain knowledge types in specific and predetermined ways. The advantage is easier implementation. Also, if a rich knowledge base is available, it is not difficult to construct alternative teaching strategies, or even a meta-scheme for choosing among alternative teaching strategies. This approach may be less elegant and more application specific than having a separate pedagogical expertise module, but for now at least, it is probably more achievable.

It is still possible to build poor instruction with XAIDA, particularly if the author does not take full advantage of its capabilities, if she does not construct the knowledge base well, or if she provides deficient instructional materials (names, descriptions, resources, etc.). However, we feel these difficulties are surmountable with training and documentation that address these areas. We could also do more in terms of implementing different teaching strategies.

Student Interface

The ideal student interface should be authentic, intuitive, and instructionally complete. Approaches that have been taken include:

1. Simulation. Simulations can provide a realistic, intuitive interface, including realistic feedback. Superior packages, such as RIDES, allow the student to experiment with settings to see what happens. Simulations, however, can be difficult to develop and instructionally opaque.

2. Task-based environment. Goal-Based Scenario Builder provides an environment where the student is presented with a problem, performs tests to gather information, and makes a decision based on the available information. The advantage is the instructional task has an authentic look and feel. The disadvantage is that such tools often do not teach other potentially relevant information, such as enabling or nice-to-know knowledge. Also, there does not appear to be a provision for changing the instructional approach or information if the student has a problem.

3. Traditional CBT. XAIDA's learning environment is neither task-authentic nor flashy. However, we have attempted to provide the right types of information, and the ability to access that information easily at any time. The sequence in which information is initially presented to the student is modeled roughly after the sequence in which

information is presented in a typical classroom. The operation of the interface is designed to be as obvious as possible.

Based on testing, we have redesigned the student interface once and have not had any problems since. Students can learn with it.

Developer Interface

The developer interface should allow an author to design effective instruction in a minimal amount of time. There are two underlying issues to worry about in constructing this interface. The first is that it should help the author to develop an attractive and effective screen layout for the student. The second is that it should acquire the needed domain knowledge. Approaches to these issues include the following:

1. Student interface is minimally configurable. This is the approach used in XAIDA, the Electronic Trainer, Goal-Based Scenario Builder, and MITT Writer. The author essentially fills in pre-determined blank spaces. The advantage is that the author no longer has to worry about screen design issues. The disadvantage is that authors will sometimes insert information which "looks funny" in the one-size-fits-all format.
2. Student interface is highly configurable. This approach is used in Eon and RIDES. RIDES has an easier task, because it is only used to design simulations. The advantage is that the author has more control. The disadvantage is that novice authors may have more control than they should have.
3. Forms-based development interface. This approach is used in MITT Writer, the Electronic Trainer, and Goal-Based Scenario Builder. The advantage is that the developer can be told exactly what to put in. The disadvantage in some cases is that she may not know why she is putting something in or where it is going to appear until she opens some other program to view the lesson.
4. Icon-based development interface. This approach is used in Eon, and resembles that used in commercial authoring software such as Authorware and Icon Author. The advantage is that the author can get a big picture of the flow of the lesson. The disadvantage is that as with a forms-based interface, she often has to run some other program to actually see what the student sees.
5. Object-based development interface. This approach is used in RIDES. For simulation development, this is a good approach. Nonetheless, our experience with RIDES and a number of other simulation development systems is that the effort and expertise required to develop simulations are very steep functions of the simulation's sophistication.
6. WYSIWYG development interface. This approach, in combination with a few forms, is used in XAIDA. The advantage is that the author sees what the student sees as she authors it. The disadvantage, for knowledge representation purposes, is that novice authors sometimes fail to concentrate on building the knowledge structure, and instead get distracted by all the other tasks they need to perform to make a complete lesson. Future versions of XAIDA may incorporate a combination approach in which the author is first prompted (in a wizard-like fashion) to provide the basic components of the knowledge structure; and later views and edits the lesson using the WYSIWYG interface.

Opportunities for Enhancement of XAIDA

The version of XAIDA described in this paper, although useful for many purposes, is an interim version. A complete picture of the system can only be had with an understanding of the capabilities that could not be accommodated in this version. This last section describes our

ideas for enhancements. Some of these ideas may be incorporated into future versions. Most will fall by the wayside either because of resource limitations or because they prove to be less than useful upon closer scrutiny.

We see opportunities for enhancements in three main areas: in knowledge representation, in increased flexibility in instructional strategy, and in the use of simulation.

The knowledge representations employed in the current versions of the shells are impoverished in many ways. The Physical Characteristics shell, for example, does not take advantage of natural constraints on many relationships such as one-one mappings. More importantly, no provision is made for instruction on the taxonomic structure of devices. Future versions should incorporate knowledge of class structure. We also recognize the many limitations of our current causal reasoning scheme for teaching theory of operation. We hope in future versions to introduce more flexibility into this scheme by providing for more complex rules and for sequential effects.

We are uncomfortable with the pattern of interaction in the Physical Characteristics shell, since it is characterized by long periods of presentation followed by equally long or longer periods of practice. Future versions should implement an approach in which presentation and practice exercises are intermingled in a dynamic and adaptive fashion. We have already begun to explore some of our ideas in this regard in the newer shells, such as Procedures.

We also see the need to explore alternative curriculum models. The model used in Version 5 is based on the part structure of the system and is driven by the Physical Characteristics shell. Alternative curriculum models could be based on knowledge structures employed by the other three shells.

Equipment simulation is a central issue for us. It is difficult to conceive of effective computer-based maintenance training that does not somehow provide access to a simulation of the equipment under instruction. One way of dealing with this problem is to replace the static graphic backgrounds employed in Version 5 with active simulations of the equipment.

References

Anderson, J. R., Corbett, A.T., Fincham, J.M., Hoffman, D., & Pelletier, R. (1992). General principles for an intelligent tutoring system architecture. In J.W. Regian & V.J. Shute (Eds.), *Cognitive approaches to automated instruction* (pp. 81-106). Hillsdale, NJ: Erlbaum.

Bell, B. & Kedar, S. (1995). When less is more: Supporting authoring and interface building via special-purpose task models. In J. Greer (Ed.), *Artificial Intelligence in Education, 1995: Proceedings of AI-ED 95 - 7th World Conference on Artificial Intelligence in Education,* Washington, DC (pp. 533-540). Charlottesville, VA: Association for the Advancement of Computing in Education.

Byrknes, A. & Myrtveit, M. (1997). *Learning dynamic modeling*. Goteberg, Sweden: Powersim Press.

Carbonell, A. M. (1970). AI in CAI. An artificial intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11, 190–202.

Casaus, M.G., Gibson, E.G., Wenzel, B.M., & Halff, H.M. (1997). *Effectiveness of adaptive practice in the Experimental Advanced Instructional Design Advisor (XAIDA)*. Unpublished manuscript.

Collins, A. M., & Loftus, E. F. (1975). A spreading-activation theory of semantic processing. *Psychological Review*, 82(6), 407-428.

Collins, A. M., & Quillian, M. R. (1969). How to make a language user. In E. Tulving and W. Donaldson (Eds.), *Organization and memory.* New York, New York: Academic Press.

Emultek Corporation (1995) *Rapid*. (Computer Software).

Halff, H. M. (1990a). *Automating maintenance training.* Arlington, VA: Halff Resources, Inc.

Halff, H.M. (1990b) *Teaching troubleshooting procedures.* Arlington, VA: Halff Resources, Inc.

Halff, H. M. (1993). Prospects for automating instructional design. In J. M. Spector, M. C. Polson, & D. J. Muraida (Eds.), *Automating instructional design: Concepts and issues* (pp. 67–132). Englewood Cliffs, NJ: Educational Technology Publications.

Hickey, A. E., Spector, J. M., & Muraida, D. J. (1991). *Design specifications for the advanced instructional design advisor (AIDA)* (Vols. 1 & 2) (AL-TR-1991-0085). Brooks AFB, TX: Technical Report for the Air Force Armstrong Laboratory, Human Resources Directorate.

Horwitz, C.D., Shute, V.J., & Fleming, J.L. (1997). Creating an adaptive training system: Integration of the SMART student model in a RIDES tutor. In C.L. Redfield (Ed.), *Intelligent tutoring system authoring tools: Papers from the 1997 AAAI Fall Symposium* (Technical Report FS-97-01, pp. 32-38). Menlo Park, CA: AAAI Press.

Hsieh, P. (1997). *A review of knowledge-based systems for maintenance training.* Unpublished manuscript.

ID_2 Research Group (1995). *The Electronic Trainer* [On-line]. http://www.coe.usu.edu/it/id2/

Kieras, D. E. (1988). What mental model should be taught: Choosing instructional content for complex engineered systems. In J. Psotka, L. D. Massy, and S. A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned* (pp. 85-112). Hillsdale, NJ: Lawrence Erlbaum Associates.

Korcuska, M. & Bell, B. (1995, April) *The Goal-Based Scenario Builder: Experiences with novice instructional designers.* Paper presented at the annual meeting of the American Educational Research Association, San Francisco, CA.

Korcuska, M., Herman, J. & Jona, M. (1996). *Evidence-Based Reporting* [On-line]. Available: http://www.ils.nwu.edu/~korcuska/publications.html

Lesgold, A. M., Lajoie, S. P., Bunzo, M., & Eggan, G. (1992). SHERLOCK: A coached practice environment for an electronics troubleshooting job. In J. Larkin & R. Chabay (Eds.),*Computer assisted instruction and intelligent tutoring systems: Shared issues and complementary approaches* (pp. 201-238). Hillsdale, NJ: Lawrence Erlbaum Associates.

Major, N. (1995). REDEEM: Creating reusable intelligent courseware. In J. Greer (Ed.), *Artificial Intelligence in Education, 1995: Proceedings of AI-ED 95 - 7th World Conference on Artificial Intelligence in Education,* Washington, DC (pp. 75-82). Charlottesville, VA: Association for the Advancement of Computing in Education.

Merrill, M. D. (1993). An integrated model for automating instructional design and delivery. In J. M. Spector, M. C. Polson, and D. J. Muraida (Eds.) *Automating instructional design: Concepts and issues* (pp. 147-190). Englewood Cliffs, NJ: Educational Technology Publications.

Merrill, M.D. (1997). *Instructional transaction theory (ITT): Instructional design based on knowledge objects* [On-line]. Available: http://www.coe.usu.edu/it/id2/

Miller, G. A., Galanter, E., & Pribram, K. (1960). *Plans and the structure of behavior*. New York: Holt, Rinehart & Winston.

Munro, A., Pizzini, Q. A., Towne, D. M., Wogulis, J. L., & Coller, L. D. (1994). *Authoring procedural training by direct manipulation* (Working Paper WP94-3). Redondo Beach, CA: University of Southern California - Behavioral Technology Laboratories.

Murray, T. (1996). Having it all, maybe: Design tradeoffs in ITS authoring tools. In C. Frasson, G. Gauthier, & A. Lesgold (Eds.), *Intelligent tutoring systems: Third international conference proceedings, ITS 96* (pp. 93-101). Berlin: Springer-Verlag.

Norman, D. A. (1981). Categorization of action slips. *Psychological Review*, 88, 1–15.

Norton, J. E., Wiederholt, B. J., & Johnson, W. B. (1991). Microcomputer Intelligence for Technical Training (MITT): The evolution of an intelligent tutoring system. In *Proceedings of the Contributed Sessions - 1991 Conference on Intelligent Computer-Aided Training*, Houston, TX, Volume 1 (pp. 1-16).

Orey, M., Zhao, R., Kelly, M., Fan, H., Park, J., Harvey, E., Hansen, S., & Keenan, R. (1995). *Intelligent simulation training for general purpose user signal equipment* [On-line]. Available at: http://lpsl.coe.uga.edu/

Polson, M. C., & Richardson, J. J. (1988). *Foundations of intelligent tutoring systems*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Reigeluth, C.M., & Stein, F.S. (1983). The elaboration theory of instruction. In C.M. Reigeluth (Ed.), *Instructional design theories and models: An overview of their current status.* Hillsdale, N.J: Erlbaum Associates.

Richardson, K.H., Wenzel, B.M., Halff, H.M., & Gibson, E.G. (1996). *The transfer of mental models in computer-based training.* Poster presented at the Eighth Annual Association for Research in Memory, Attention, Decision-making, Imagery, Language, Learning, and Organizational Perception Conference, University of Texas, Austin, Texas.

Rouse, W. B., & Hunt, R. M. (1984). Human problem solving in fault diagnosis tasks. In W. B. Rouse (Ed.), *Advances in man-machine system research* (Vol 1) (pp. 195–222). Greenwich, CT: JAI Press.

Schvaneveldt, R. W. (Ed.). (1990). *Pathfinder associative networks: Studies in knowledge organization.* Norwood, NJ: Ablex.

Spector, J. M. (1990). *Designing and developing an advanced instructional design advisor*. (AFHRL-TP-90-52). Brooks AFB: TX: Technical Paper for the Training Systems Division of the Air Force Human Resources Laboratory.

Spector, J. M., Polson, M. C., & Muraida, D. J., Eds. (1993). *Automating instructional design: Concepts and issues.* Englewood Cliffs, NJ: Educational Technology Publications.

Van Lehn, K. (1987). Learning one subprocedure per lesson. *AI Journal*, 32, 1-40.

Van Marcke, K. (1992). Instructional expertise. In C. Frasson, G. Gauthier, & G. I. McCalla (Eds.), *Intelligent tutoring systems* (pp. 234-243). Berlin: Springer-Verlag.

Van Marcke, K. & Vedelaar, H. (1995). Learner adaptivity in generic instructional strategies. In J. Greer (Ed.), *Artificial Intelligence in Education, 1995: Proceedings of AI-ED 95 - 7th World Conference on Artificial Intelligence in Education*, Washington, DC (pp. 323-333). Charlottesville, VA: Association for the Advancement of Computing in Education.

Wenger, E. (1987). *Artificial intelligence and tutoring systems*. Los Altos, CA: Morgan Kaufmann.

Wenzel, B.M., Dirnberger, M.T., Fox, H., Hearne, R., Keeney, C., Licklider, J., Reyna, H., Roberts, J.D., Strebeck, N., & Halff, H.M. (1997). *Field tests of an interactive courseware development system.* Unpublished manuscript.

Wenzel, B. M. and Dirnberger, M. T. (1997). Evaluating a courseware generation system designed for subject matter experts. Unpublished manuscript.

Wenzel, B. M., & Ellis, R. (1997). *Evaluation of the instructional effectiveness of interactive multimedia distributed courseware for emergency medical technician training*. Unpublished manuscript.

Wenzel, B.M., Richardson, K.H., & Gibson, E.G. (1996). *Assessing the instructional effectiveness of the Experimental Advanced Instructional Design Advisor (XAIDA).* Unpublished manuscript.

Wiederholt, B. (1991). MITT Writer: An authoring system for developing intelligent tutors for complex technical domains. In *Proceedings of the Contributed Sessions - 1991 Conference on Intelligent Computer-Aided Training*, Houston, TX, Volume 1 (pp. 17-30).

Author Note

Patricia Y. Hsieh, Training Technology Division; Henry M. Halff, Training Technology Division; Carol L. Redfield, Training Technology Division.

The research reported in this paper was conducted for the US Air Force Armstrong Laboratory, Human Resource Center under Contract Contract F41624-93-F5002 to the Mei Technology Corporation. The opinions expressed in this article are those of the authors and do not necessarily reflect those of the US Air Force. Thanks are due to our colleagues on the XAIDA Research and Development Team: Charles Balog, Mike Casaus, Dan Christinaz, Tom Chudanov, Mary Dirnberger, Elizabeth Gibson, Bill Hawks, Kevin Richardson, Darryl Song, Bill Walsh, and Brenda Wenzel.

Correspondence concerning this article should be sent to Henry M. Halff, Mei Technology Corporation, 8930 Fourwinds Drive, Suite 450, San Antonio, TX 78239. Electronic mail may be sent via Internet to henry@meitx.com.

Footnotes

1. It should be noted that a developer who is familiar with RIDES authoring language could add an adaptive exercise capability (see Horwitz, Shute, and Fleming, 1997). But our main point is that XAIDA, unlike RIDES and other simulation-based systems, automatically generates adaptive practice on physical characteristics.
2. It is interesting to note, by the way, that although instructors often mention this type of information during instruction, they seldom test it. It is as if they believe such information is useful for learning and recalling the procedure, but not important enough to assess. XAIDA does administer practice on this type of knowledge, however.
3. The current prototype implements three question types embedded in presentation. The other types and contexts are not supported at this time.
4. In practice, a system can be multiply faulted, but troubleshooting training almost universally restricts its training to single-fault cases. XAIDA adheres to this single-fault constraint.
5. In real aircraft, air is also required.

Appendix 1. Studies of Students

Study	Wenzel, Richardson, & Gibson (1996)	
Lesson Topic	C-141 crew oxygen system	XAIDA Version 5.04
Issues Investigated	Gains in declarative knowledge and changes in knowledge organization and structure resulting from an XAIDA lesson. Also, instructional effects of practice.	
Participants	11 high school seniors	
Data	Written performance test scores and Pathfinder.	
Findings	Significant 217% score increase post instructional presentation. Near significant 27% score increase post practice. No relationship found between students' Pathfinder networks as compared to C-141 maintenance instructor/expert. No change in students' Pathfinder networks as a result of XAIDA lesson.	
Study	Richardson, Wenzel, Halff, & Gibson (1996)	
Lesson Topic	C-141 crew oxygen system and bleed air system (2 lessons)	XAIDA Version 5.04
Issues Investigated	Comparison on written test with untrained controls. Changes in Pathfinder patterns as the result of training as a function of lesson content and presence of advance organizer.	
Participants	71 undergraduate students	
Data	Written performance test scores and Pathfinder (pre and post)	
Findings	Significant 52% gain in performance over untrained controls. No consistent indications of training effects on Pathfinder measures.	
Study	Wenzel, Richardson, & Gibson (1996)	
Lesson Topic	C-141 crew oxygen system	XAIDA Version 5.04
Issues Investigated	Gains in declarative knowledge and changes in knowledge organization and structure resulting from an XAIDA lesson (no control group)	
Participants	11 USAF C-141 maintenance trainees	
Data	Written performance test scores and Pathfinder (pre and post)	
Findings	Significant 73% test score increase. Correlations between student and expert 1) Pathfinder networks and 2) relatedness ratings before and after lesson increased.	

Study	Wenzel, Richardson, & Gibson (1996)	
Lesson Topic	Anatomy of the auditory system	XAIDA Version 5.04
Issues Investigated	Gains in declarative knowledge and changes in knowledge organization and structure resulting from an XAIDA lesson (no control group)	
Participants	7 university introductory psychology students	
Data	Written performance test scores and Pathfinder (pre and post)	
Findings	Significant 526% test score increase. Correlations between student and expert 1) Pathfinder networks and 2) relatedness ratings before and after lesson increased.	
Study	Wenzel, Dirnberger, Fox, Hearne, Licklider, Keeney, Reyna, Roberts, Strebeck, & Halff (1997)	
Lesson Topic	Parabolas and Quadratic Equations	XAIDA Version 5.1a
Issues Investigated	Lesson effectiveness (no control group)	
Participants	9 community college algebra students	
Data	Written performance test scores (pre and post)	
Findings	Significant score increase of 56%	
Study	Wenzel, Dirnberger, Fox, Hearne, Licklider, Keeney, Reyna, Roberts, Strebeck, & Halff (1997)	
Lesson Topic	Personal computer motherboard	XAIDA Version 5.1a
Issues Investigated	Effectiveness of learning in dyads vs. as individuals	
Participants	21 community college computer hardware students	
Data	Written performance test scores (pre and post); self-rating of parabola knowledge; opinion survey	
Findings	Significant score increase of 167% in both treatments; no difference in performance between pairs and individuals	
Study	Wenzel, Dirnberger, Fox, Hearne, Licklider, Keeney, Reyna, Roberts, Strebeck, & Halff (1997)	
Lesson Topic	Drillwell recirculation system	XAIDA Version 5.1a
Issues Investigated	Effectiveness of XAIDA vs. lecture	
Participants	24 continuing education students	
Data	Knowledge tests; knowledge self-ratings; computer comfort ratings	
Findings	Significant increases in performance test (47%), knowledge self-ratings, and computer comfort ratings. No differences between XAIDA and lecture conditions.	

Study	Wenzel, Dirnberger, Fox, Hearne, Licklider, Keeney, Reyna, Roberts, Strebeck, & Halff (1997)	
Lesson Topic	Oxyacetylene welding equipment	XAIDA Version 5.1a
Issues Investigated	Lesson effectiveness (no control group)	
Participants	10 community college pipe-fitting students	
Data	Written performance test scores (pre and post); lesson evaluation questionnaire	
Findings	Significant 113% score increase on performance test; overall very positive response to the lesson itself.	
Study	Wenzel, Dirnberger, Fox, Hearne, Licklider, Keeney, Reyna, Roberts, Strebeck, & Halff (1997)	
Lesson Topic	Computer keyboard	XAIDA Version 5.06
Issues Investigated	Lesson effectiveness; control group engaged in non-relevant activity	
Participants	16 community college microcomputer students	
Data	Written performance test and reaction time measure	
Findings	XAIDA lesson group scored significantly higher on written test than control group and had significant 104% score increase; control group had no significant score increase. XAIDA group also on average slightly faster and more accurate than control group on reaction time test.	
Study	Wenzel, Dirnberger, Fox, Hearne, Licklider, Keeney, Reyna, Roberts, Strebeck, & Halff (1997)	
Lesson Topic	Computer literacy	XAIDA Version 5.1a
Issues Investigated	Instructional effectiveness of audio in courseware. Compared audio with text, audio-only, and text only.	
Participants	32 community college computer-literacy students	
Data	Written performance test scores; learning activities and instructional methods preference surveys	
Findings	Significant score increase of 47% in all treatments; text-audio combination was superior to text alone and audio alone. The last two were equally effective.	

Study	Casaus, Gibson, Wenzel, & Halff (1997)	
Lesson Topic	The brain	XAIDA Version 5.1a
Issues Investigated	Comparison of adaptive and non-adaptive practice (yoked control group)	
Participants	20 undergraduate psychology students	
Data	Multiple-choice performance measures administered at three points in time: pre-lesson; post-lesson, pre-practice; and post-practice.	
Findings	Significant score increases from pre-lesson to post-practice for both groups. No increase in score due to adaptive practice.	
Study	Casaus, Gibson, Wenzel, & Halff (1997)	
Lesson Topic	The brain	XAIDA Version 5.1a
Issues Investigated	Comparison of adaptive and non-adaptive practice (yoked control group)	
Participants	12 undergraduate psychology students	
Data	Multiple-choice performance measures administered at three times: pre-lesson; post-lesson, pre-practice; and post-practice.	
Findings	Significant score increase in both groups from pre-lesson to post-practice. Adaptive group scored significantly higher than non-adaptive group ($p<.001$). Some, but not statistically significant, increase in score due to adaptive practice.	
Study	Wenzel & Ellis (1997)	
Lesson Topic	National Emergency Response Guidebook	XAIDA Version 5.1b
Issues Investigated	Instructional effects on knowledge and performance of physical characteristics and theory of operation training	
Participants	53 USAF military and civilian medical personnel	
Data	Written tests and timed performance tests taken throughout training	
Findings	Significant 366% gain on written tests. Significant gains in time and accuracy on timed performance tests.	

Appendix 2. Studies of Developers

Users	XAIDA version	Lesson topics	Data gathered[a]
8 USAF military and civilian personnel, various specialties (Brooks AFB)	5.04	Nerve cell; hydraulic system	A, B, C, D, E, F
25 USAF military and civilian personnel, primarily aircraft maintenance (362nd TTS, Sheppard AFB)	5.04	F-16 egress system; munitions; time compliance technical order (TCTO) form; chip detector; winch; speed brake; hydraulic system; statistical hypothesis; tools; Core Automated Maintenance System (CAMS) data hierarchy; emergency power unit; AF scheduling form 2407; airframe; flight controls; weight arm movement; cargo restraint; basic aerodynamics; engine starting system; personal computer	A, B, C, D, E, F
4 USAF military reserve and civilian personnel, primarily paralegals	5.06	Various AF 3070 forms (charge sheet, non-judicial proceedings, Federal Tort Claims Act)	A, B, C
2 USAF military personnel, various specialties (Travis AFB)	5.1a1	Human body	A, C
8 Community college instructors and support personnel, various content areas	5.06 (Dev.); 5.1a1 (Del.)	Quadratic equations (parabola); micrometer; motherboard; keyboard; personal computer; oxyacetylene equipment; drill well circulation system	A, B, C, D, E, F
7 USAF military and civilian personnel, primarily in basic military training (Lackland AFB)	5.1a2	Proper military attire; military customs and courtesies	A, B
17 USAF military and civilian medical personnel (383rd TRS, Sheppard AFB)	5.1a2	Cardiac assessment; EKG machine; cover of 6-part folder; occupational skill levels; gastrointestinal endoscope; slint; soft tissue wounds; surgery tray; developing plans of instruction; blood pressure; transportation; medical terminology; skeletal system; surgical attire; heart parts; emergency response guidebook structure and use; IV setup	A, B, C, D, E, F
5 National Imaging and Mapping Agency (NIMA) and Defense Language Institute (DLI) personnel	5.1b	Map duplication (printing process); NIMA catalog; parts of speech, F-16 cockpit	A, B, C, D

[a]Key: A = Comments; B = Attitudinal; C = Journal files; D = Behavioral; E = Self-reported computer skills; F= Self-reported XAIDA skills; G = Self-reported usability/understandability of XAIDA development features, interfaces, and support materials; H= Knowledge structures (Pathfinder scores); I = Self-reported organizational factors related to acceptance of new technology.