The LEAP Authoring Tool: Supporting complex courseware authoring through reuse, rapid prototyping, and interactive visualizations

Randall Sparks, Scott Dooley, Lori Meiskey, Rick Blumenthal

International Journal of Artificial Intelligence in Education (1998), 10, 75-97

The LEAP Authoring Tool: Supporting complex courseware authoring through reuse, rapid prototyping, and interactive visualizations

Randall Sparks, Scott Dooley, Lori Meiskey, Rick Blumenthal
U S WEST Advanced Technologies, 4001 Discovery Dr., Boulder, CO 80303
randall@uswest.com, sdooley@uswest.com, lmeiske@gv.uswest.com, rblumen@gv.uswest.com

1. INTRODUCTION

An important goal of current work in computer-based learning environments is to develop systems that combine the richness and effectiveness of an individually crafted intelligent tutoring system (ITS) with the generality and flexibility of a computer-assisted instruction (CAI) authoring tool. Our effort to achieve this goal is demonstrated in the Learn, Explore and Practice (LEAP^{sm}) ITS shell and its courseware development component, The LEAP Authoring Tool (LAT).[1] The LAT was developed for use by non-programmer subject matter experts to create courseware for use in the LEAP system. In this paper we will provide a brief description of LEAP and then describe the goals, design, implementation, and evaluation of the LAT, and discuss the process we followed in the development of the LAT and some of the lessons we have learned along the way.

1.1. Overview of LEAP

LEAP is designed around the tenet that the learning of a complex task is best achieved in a rich "apprenticeship environment", in which skills and knowledge are acquired through realistic exercises conducted in a realistic context (Collins, Brown & Newman, 1989; Lave & Wenger, 1991; Lesgold et al., 1992). The primary goals of the LEAP and LAT development efforts were to: (1) create this type of apprenticeship environment for end users learning a certain type of complex task and (2) support low-cost authoring of high-quality (effective) courseware by subject matter experts without computer programming experience.

The specific task undertaken by the LEAP development team was to build a system to teach customer contact employees (CCEs), such as customer service representatives, the knowledge and skills they need to respond effectively to customer requests and problems. This includes knowledge of company products and services, appropriate conversational style, consultative sales skills, the ability to find and use customer information and enter work orders, etc. LEAP simulates CCEs' work environment, in which they are expected to carry on a dialogue with customers over the telephone while simultaneously interacting with database systems for entering service orders and obtaining information about customers, their accounts, etc.

The LEAP training environment is shown in Figure 1. At the top left is the LEAP Practice Commands window. This is where the trainee interacts with LEAP as a tutoring system, performing such actions as selecting a practice mode, reviewing the history of a practiced conversation, getting help on how to use the system, and quitting a practice session. The Set Practice Mode button allows the trainee to select one of three practice modes: (1) observe, (2) practice, and (3) focused practice. In observe mode, the trainee observes LEAP as it simulates an expert CCE handling a conversation using audio recordings, animated data entry into the database system simulators, etc. In practice mode, trainees practice performing the actions of a CCE within a simulated conversation. That is, the role of the customer is simulated by LEAP using audio recordings, while the trainee records his or her own replies and uses the database system simulators to perform appropriate actions on behalf of the customer. Coaching is

available at any time from the system in the form of hints, explanations of appropriate actions, and observable simulations of appropriate actions (as in observe mode). In focused practice mode, LEAP uses its model of the student's progress to alternate between observe and practice modes, as well as skipping certain parts of the conversation, in order to focus practice on those actions on which the particular trainee appears to need the most work.

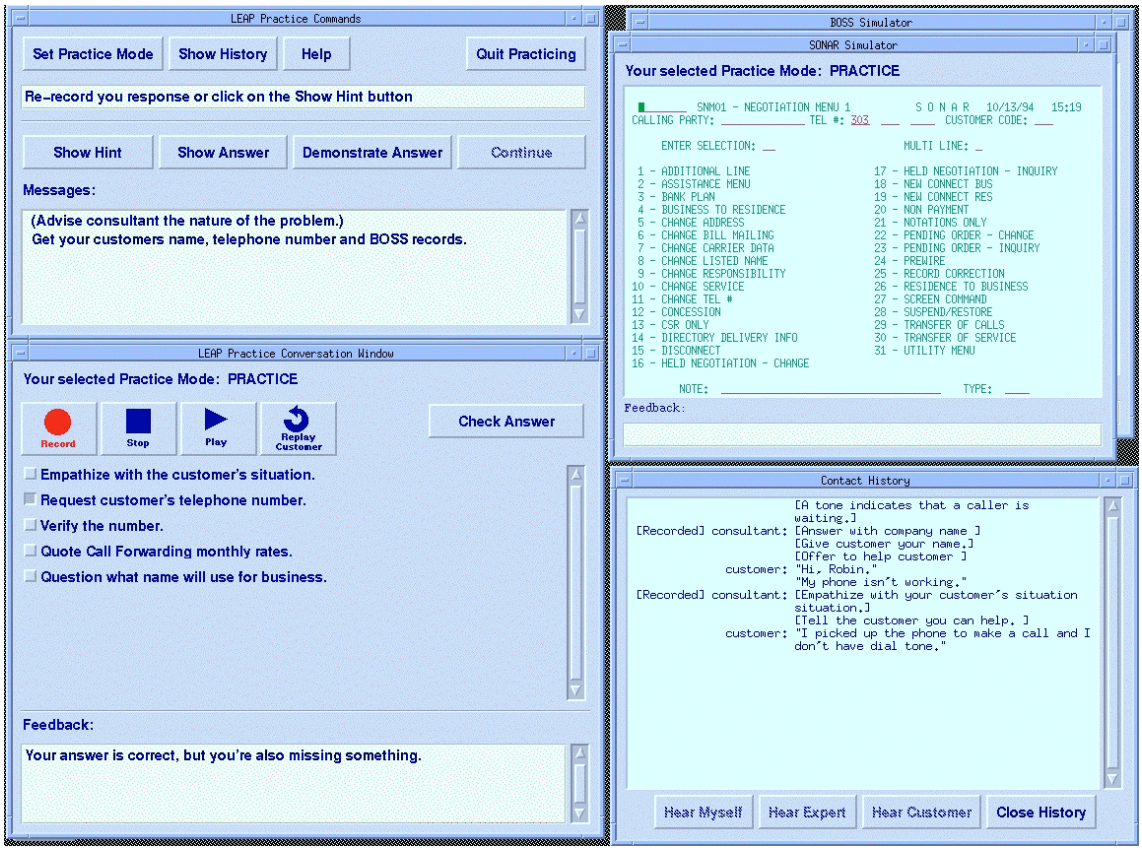

Figure 1. The LEAP Practice Environment.

In addition, the Practice Commands window is where the trainee can receive certain types of coaching and other guidance from LEAP. Below the first row of buttons in the Practice Commands window is a prompt telling the trainee how to proceed through the conversation, such as clicking the Continue button, recording a response to the recorded statement by the simulated customer. Below the prompt are buttons that allow the trainee to request that LEAP show a hint on how to proceed, show the best answer, demonstrate the best answer in the current situation, or, when appropriate, continue to the next step in the conversation. The Continue button most commonly results in the playing of the next statement by the simulated customer. The space below these buttons is used to display the requested hints, answers, etc.

Below the Practice Commands window is the Practice Conversation window. This window contains tape recorder-like controls that allow the trainee to re-play the last customer statement as well as record and listen to his or her own response. In practice mode, trainees are also asked to indicate what action or actions they are taking at this step in the conversation using a multiple-choice mechanism, and to check their answer against the courseware author's definition of appropriate actions. This is accomplished by clicking the Check Answer button, with feedback appearing at the bottom of the Practice Conversation window.

At the top right of the practice environment are two database application simulators. These allow the same operations as the real systems they simulate, but can also demonstrate user interactions with the databases by highlighting relevant regions of the screen and showing an animation of appropriate data being entered. The simulators also provide immediate feedback on trainee actions, such as indicating that a certain action is correct, not appropriate at this time, etc.

At the bottom right of the practice environment is the Contact History window. Trainees can open this window in order to review what has occurred so far in the practice conversation. The history displays the actions for each turn in the conversation. Trainees can highlight a turn in the conversation by clicking on it and then click one of the buttons below in order to hear the customer's statement, his or her own recorded response, and the pre-recorded expert CCE response. This allows trainees to compare how they actually sound to the recorded expert CCE.

LEAP also provides: (1) an Instruction Manager that selects and recommends an appropriate scenario (simulated conversation) for a learning session; (2) coaching assistance to guide learners as they are engaged in the scenario; (3) an opportunity for learners to review and receive feedback on their performance at the completion of the scenario; and (4) conclusions about the skill levels the learner has achieved in the various areas of the domain. The LEAP ITS is described in detail in Bloom et al. (1995).

LEAP was designed as an ITS shell, but one centered around the task of interacting with customers and computer systems in a telephone-based dialogue.[2] The content of the dialogues, the domain courseware, is independent of the learning environment. This allows the rich learning environment of LEAP to be used for learning a relatively wide variety of particular skills and knowledge, even while focused on the general task of interacting with customers and related computer applications.

1.2. LEAP Courseware

The courseware component of a LEAP tutoring application combines features from the domain model and expert model components of a traditional ITS, although the knowledge is less explicit than in most ITSs. The courseware consists of a set of learning modules. Each learning module is essentially a conversation grammar (a kind of task model), augmented with domain and expert behavior information. Representing the knowledge in this way allows authors to concentrate on concrete examples of behavior that the learner will be coached toward. The authors do not have to build an abstract set of rules or other expert model representations.

A LEAP conversation grammar, in its simplest form, is a collection of separate conversations, each consisting primarily of a series of conversational turns taken by the CCE and the simulated customer. When a learner uses LEAP, the role of the CCE is played by either the learner or the simulated expert CCE, depending on whether practice or observation is the emphasis of the current learning mode. The LEAP "coach" may also interact with the learner during his or her turn. Audio recordings are used to simulate the customer and the expert CCE, and for learners to record and listen to themselves. Pertinent database systems are simulated and their use linked to particular steps in the LEAP conversation grammar. This allows the learner to practice performing actions such as entering and retrieving data and then receiving feedback on his or her performance, or to observe animated sequences showing the simulated expert CCE performing these steps.

The transcript in Figure 2 below shows an example portion of a conversation in LEAP. The CCE's statements are preceded with the label "Rep" (for customer service representative). The CCE is talking on the phone with a customer, whose statements are similarly indicated.

Each turn in the conversation consists of a set of actions that one of the actors performs. For the learner, the author can specify constraints pertaining to the ordering, co-occurrence, and optionality of the actions in each turn. More flexibility can be introduced by adding branches to the conversations, thus introducing multiple solution paths. The example conversation in Figure 2 shows an example of 2-way branching. The author can also abstract high-level sub-tasks from the conversations, which we refer to as "subdialogues" (e.g., opening the conversation). Subdialogues can be shared by different conversations, thus allowing authors to efficiently reuse material.

In addition to the conversation grammar, a LEAP learning module requires a topic hierarchy. The topic hierarchy and the associations specified between topics and individual learner actions help define the domain. These allow LEAP to determine learner skill levels for each topic, based on the learner's exposure to and performance of the actions associated with that topic.

CONVERSATION BEGINS

Rep: U S WEST Communications. This is Sean.
How can I help you?

Customer: Hi. My call forwarding isn't working.

Rep: I'm sorry to hear that.
Can I have your telephone number?

Customer: Sure, it's 303, 111, 2345.

Rep: That was 303, 111, 2345?
[Rep. types number into application]
Is this Mr. Anderson?

Customer: Yes.

BRANCH 1	BRANCH 2
Rep: I think I see what the problem is.	**Rep**: Could you please hold while I check into the problem?
	Customer: Sure.
	Rep: [Retrieves customer service record.] [Notices the problem.] Mr. Anderson?
	Customer: Yes.
	Rep: I think I've found the problem.

BRANCHES COME BACK TOGETHER

Customer: Oh, great!

[The Rep explains the problem and negotiates a solution acceptable to the customer.]

Figure 2. A conversation with two branches.

Other components of a learning module include the hints that the author provides for each learner action and the expert CCE examples (demonstrations using text, audio, and screen animation) for each action. The various components of LEAP courseware--especially the conversation grammar and the power afforded by its flexibility--combine to make the author's task of creating one or more learning modules potentially very complex. In the following sections we discuss the ways in which we have attempted to support the author in accomplishing this task through the LAT and what we consider to be some of our success, failures, and lessons we have learned along the way.

2. DESIGN OF THE LEAP AUTHORING TOOL

As described above, the goal of providing cost-effective, high-quality training for the types of skills needed by CCEs led us to the rich, apprenticeship learning environment of LEAP. However, the richness and complexity of this environment poses a significant challenge for the authoring process. Authors must create courseware that includes potentially complex conversation grammars, a description of all the individual actions that the actors in the conversation may take, a hierarchy of topics and links from the actions to these topics, hints for each step of the conversation, audio recordings and textual representations of customer and expert CCE actions, etc. In order to meet the requirement of cost-effectiveness without sacrificing the quality of training provided by a rich environment, we knew it would be necessary to provide courseware authors with a sophisticated, easy-to-use authoring tool to support and guide the authoring process.

We identified this requirement for the LAT on the basis of our experience working with the three subject matter experts initially assigned to the LEAP project: one expert CCE and two CCE training experts. These experts had to develop courseware for LEAP while it was under development using basic text editing tools. Their requirements for quality training led us to the rich, relatively realistic learning environment of LEAP and its associated complexities in the structure of the courseware. Their struggles to work effectively with these complexities and their suggestions on what would be most helpful to them led us to the particular design goals we set for the LAT and the ways in which we addressed these goals during its development.

2.1. Design Goals

The design of the LAT was guided by two types of constraints. First, the design had to accommodate the authoring of the particular structures and objects required for a LEAP courseware module, i.e., topics, actions, transitions between steps in a conversation, definition of subdialogue, hints, etc. (We will discuss these objects further in the discussion below.) Second, the design was also constrained by the expected skill set of the targeted authoring community, namely subject matter experts (i.e., expert CCEs and CCE trainers). These constraints led us to set the following goals for the design of the authoring tool: (1) ease of use, (2) rapid prototyping and iterative development, (3) reuse of courseware components, (4) multiple representations and input mechanisms, and (5) interactive visualizations of the courseware structure and content.

2.1.1. Ease of Use

Although it may have become somewhat commonplace to include ease-of-use as a design goal, one lesson we learned during our development effort was that it was extremely important to maintain an emphasis on ease-of-use throughout the design and development process. The possibility of including new, powerful features constantly had to be weighed against the need to maintain simplicity for the majority of use cases. For us, ease-of-use meant both easy to learn (new users from our targeted author community should be able to learn to use the tool to build runnable courseware within a couple of days) and easy and efficient for experienced users to maintain and extend an existing body of courseware.

2.1.2. Rapid Prototyping and Iterative Development

Based on the experiences of the subject matter experts involved in the development of LEAP and its initial body of courseware (developed without the LAT), it was apparent that authors would benefit substantially from the ability to see, right from the beginning, how the conversations they were authoring would actually be experienced by learners. This meant that they needed to "run" the courseware before it was complete, i.e., sketch out and run a skeletal, prototype version of their courseware in LEAP, then iteratively refine and complete it. The LAT was designed to support this style of development through the inclusion of several features,

including usable default values for unspecified information, full integration with run-time LEAP so that changes can immediately be tested, an agenda mechanism to help authors keep track of any incomplete authoring tasks, and the provision of several alternative methods for entering courseware information (see section 2.1.4 on multiple representations below).

2.1.3. Reuse

To minimize the cost of developing new courseware, we made reuse of courseware components a key design goal for the LAT. Because there are many similarities among different conversations, both within a courseware module for a given domain and across modules, there is an opportunity to reuse portions of existing conversations to build new ones. For example, most, if not all conversations in a given body of courseware may begin with the same initial opening or greeting. In the initial set of LEAP courseware that our subject matter experts developed, all conversations begin with the telephone ringing. This is followed by the CCE answering by stating the company name and their own name, and then offering to help the customer. (See the example in Figure 2 above.) The authors took advantage of this by defining this segment of conversation as a reusable chunk, or "subdialogue", to be used in all the conversations in this courseware module.

Re-using parts of conversations helps authors by: (1) reducing the amount of work involved in creating a new set of conversations, (2) reducing the maintenance effort required for updates such as changes in prices, policies, or procedures, and (3) making it easier to maintain consistency across a large number of conversations.

However, one lesson that we learned through our effort to support reuse was that it is relatively easy to underestimate the costs of adding this sort of "power" feature to a complex system such as an ITS authoring tool. We knew that reuse had the potential to save authors of relatively large bodies of courseware a great deal of effort in maintaining and extending their LEAP grammars. Yet, at the same time, supporting reuse may have considerable costs in terms of system complexity and ease-of-use.

In our case, supporting reuse required additional complexity, both in (1) the LEAP grammar representations and processing mechanisms and (2) the LAT user interface. The technical challenges this posed for the LEAP grammar itself proved to be manageable. Furthermore, we feel that we were able to leverage the capabilities of good GUI design to develop relatively sophisticated tools for helping authors handle the complexities of reuse. However, the experience we have had with users suggests that they may not be attracted to the advantages of reuse enough to be willing to deal with its associated complexities, at least not to the extent that we had anticipated. Reuse in the LAT thus constitutes one case in which power can be traded off against ease-of-use. It appears in this case that we may have underestimated the impact that adding this type of power to LEAP and the LAT would have on ease-of-use for authors. On the other hand, our experiences with LEAP courseware authors have been quite limited so far. We have not yet seen the LAT used extensively enough for authors to have faced the need to maintain a relatively large body of courseware, which is the use case for which reuse offers the most significant advantage.

2.1.4. Multiple Representations and Input Mechanisms

Our experiences with the subject matter experts who served as courseware authors during the development of LEAP led us to the conclusion that different authors would have different individual authoring styles and different starting points for the authoring process. For example, some authors may be adapting existing training materials while others are starting from scratch. For this reason, we felt it was important to design the LAT to accommodate some of these different approaches to courseware development.

One significant difference in authoring approaches is that some authors prefer to work in a "bottom-up" manner and others in a "top-down" manner. Those who work in a bottom-up fashion begin by entering a "script" of a conversation. After the conversation has been scripted, they then divide it up into relevant subdialogues and, perhaps, elaborate this into several distinct

conversations. Authors who work top-down begin by creating a conversation in terms of its high-level major subdialogues (such as "Greeting the customer", "Determining the customer's need", "Providing a response", and "Closing the call"). These authors then move to a more detailed level by defining the structure of each of these subdialogues, and continue in this fashion until each individual step of each subdialogue has been defined.

Another way that the LAT accommodates different authoring techniques is that a script of a conversation may initially be prepared in a word processor or text editor. This script can then be read into the LAT as a structured text file and converted into a valid internal LEAP representation of a conversation. After importing the script, the author can test the conversation in LEAP and continue to develop it.

Finally, the LAT also accommodates different authoring styles by providing two distinct primary visualizations of the courseware content. These are discussed in the next section.

2.1.5. Interactive visualizations

As mentioned above, some authors may write new courseware by first creating scripts of conversations, then elaborating them into complete courseware modules. We also expect that all authors will find it helpful on occasion to be able to view a transcript-like representation of a particular conversation, i.e., a listing of what is said in each turn by each actor in the conversation, unencumbered by branching, hints, topics, associated audio files and other information associated with every step of the conversation. This is helpful, for instance, in determining that the flow of the defined conversation is coherent and natural. For these reasons, the LAT includes a "script editor" that provides such a view of a conversation and allows editing of the conversation within this view. The script editor is described in more detail below in section 3.1. As mentioned above, other authors prefer to work primarily in a top-down fashion, defining a hierarchy of subdialogues before fleshing out the details. We also expect all authors to find it helpful to be able to view the subdialogue structure of their conversations. This is helpful, for example, in understanding branches in conversations or which conversations share a particular subdialogue. For these reasons, the LAT includes a "subdialogue graph editor" that provides a node-and-link representation of the structure of a subdialogue and also allows editing from within this view. This editor is also discussed in more detail below in section 3.2. In addition to the script and subdialogue views, a third visualization is provided for the transitions between nodes in the conversation grammar. The nodes in the grammar represent the state of a conversation being run in LEAP. That is, at any given moment, the current state of the simulated conversation is defined by the currently active node(s). The transitions define groups of actions available at a given node, i.e., that are available to the learner at any given point in the conversation. Transitions are potentially complex, as they may include many different combinations of operators for defining ordering, optionality, co-occurrence, and other relationships among available actions. The transition editor and the structure of transitions are discussed below in section 3.3.

3. USING THE LEAP AUTHORING TOOL

In this section, we will describe how the LAT is used for the creation of LEAP courseware, including descriptions and examples of the use of the script, subdialogue graph, and transition editors and other editing tools and how these tools address the design goals mentioned above.

3.1. Scripting Conversations with the Script Editor

The script editor is a text-based tool that provides a visualization of a single path of conversation through a LEAP grammar. We refer to this visualization as a "script view" or simply a "script". This visualization is analogous to the script of a play, although there are a number of additional elements in the display that allow the conversation to be edited in the ways

required for creating a complete LEAP grammar. The script view provides a clear and concise description of who is performing which actions, speaking which parts of the dialogue, and the order (or the multiple possible orders) in which these actions may take place.

One primary purpose of the script editor is to allow a user to quickly and easily create a single conversation path (i.e., a script). To do this, a LEAP author (a subject matter expert) enters a script much as a playwrite might create a play using a word processor. Using the script editor, the author can simply type in the text of the dialogue that takes place among the "players". However, because the script is not going to be performed by human actors, but rather executed by the LEAP tutoring system, it is important to provide the author with some guidance as to how to create a script that can run successfully in LEAP and meet the training goals for the courseware being developed.

Let's start with an example. We assume a use case in which the author is initially creating a new conversation and begin our discussion just after the author has entered the script editor. The author has previously informed the LAT that we are creating a new conversation, so the script editor is initialized to display a single first node of the conversation. The node appears as a box, surrounded above and below by "subdialogue bars". These indicate which subdialogue the current sequence of nodes is contained within. Figure 3 shows this first node (after two additional nodes have been created) as the top rectangular box, labeled with the name JAIED-DEMO-MAIN-BEGIN.

In the upper right-hand corner of the node box is an editable text field labeled "Node" in which the author can specify the name of the node, JAIED-DEMO-MAIN-BEGIN in this example. To the left of the node name is a pop-up menu that allows the author to select the actor who is responsible for performing the actions associated with this node. In this example, the "Phone" actor is selected. Because a telephone conversation typically starts with the phone ringing, this is the default configuration with which the LAT begins a script. Other actors in the conversation may include the trainee, the other person involved in the dialogue, and a couple of special cases, such as the "coach", an actor that can be used by authors to provide direct coaching of the learner at any point in simulated conversation. In our example, the trainee appears as the CCE (the customer service representative, or "Rep", actor), as this is the role being tutored. The other human participant being scripted is the "Customer" role.

Below the actor and node name fields are specifications for one or more rows of actions contained within the node. The first node in Figure 3 has only one row with a single action for ringing the telephone. The pop-up menu on the left side of the actions specification allows the author to select the type of action being specified, in this case a "Situation" action. To the right of the selected action type is a text field. The purpose of this text depends on the type of action. This area is most frequently used to specify the spoken text of "Talk" actions.

At this point, we have a script that consists of the telephone ringing. To continue, the author adds a new node by selecting "New Node" from the Edit menu or pressing a key. The script editor adds a new node to the display and populates it with defaults for its name, actor, and action type. The default information is based on heuristic knowledge of how the conversation flows. In this case, the ringing of the telephone is typically followed by the Rep answering the phone. Thus, the new node shows the Rep as the actor and a Talk action in which the Rep will say something to the caller, such as "Hello".

With these defaults, the author only has to type in the text associated with the first Talk action. In our example, the author has chosen to have the Rep answer the phone by saying "U S WEST Communications. This is Sean. How can I help you." and to represent this as a sequence of three actions. The Rep's entire statement could be represented as a single Talk action. However, for the purpose of relating actions to instructional topics and for detailed modeling of learner performance, it is typically useful to divide statements of this type into several separate actions. So, after entering the text "U S WEST Communications" for the first Talk action, the author presses a key to create another action, enters "This is Sean", and repeats this for the third action. The result is the Rep node (the second rectangular box) shown in Figure 3, containing a sequence of the three "Talk" actions that make up the Rep's initial turn in the conversation.

Next, the author creates another new node. This time, the actor supplied by the script editor defaults to "Customer", as the calling party is expected to say something next. The author may now fill in the spoken text for this turn in the dialogue, as above, or edit the actor, action type, or other information about the node. In Figure 3, the author has the customer respond by saying "Hi. My call forwarding isn't working." Creating a conversation in the script editor proceeds in this way--much like writing the script of a play, but with all of the additional editing capabilities needed to create the tutorial courseware close at hand.

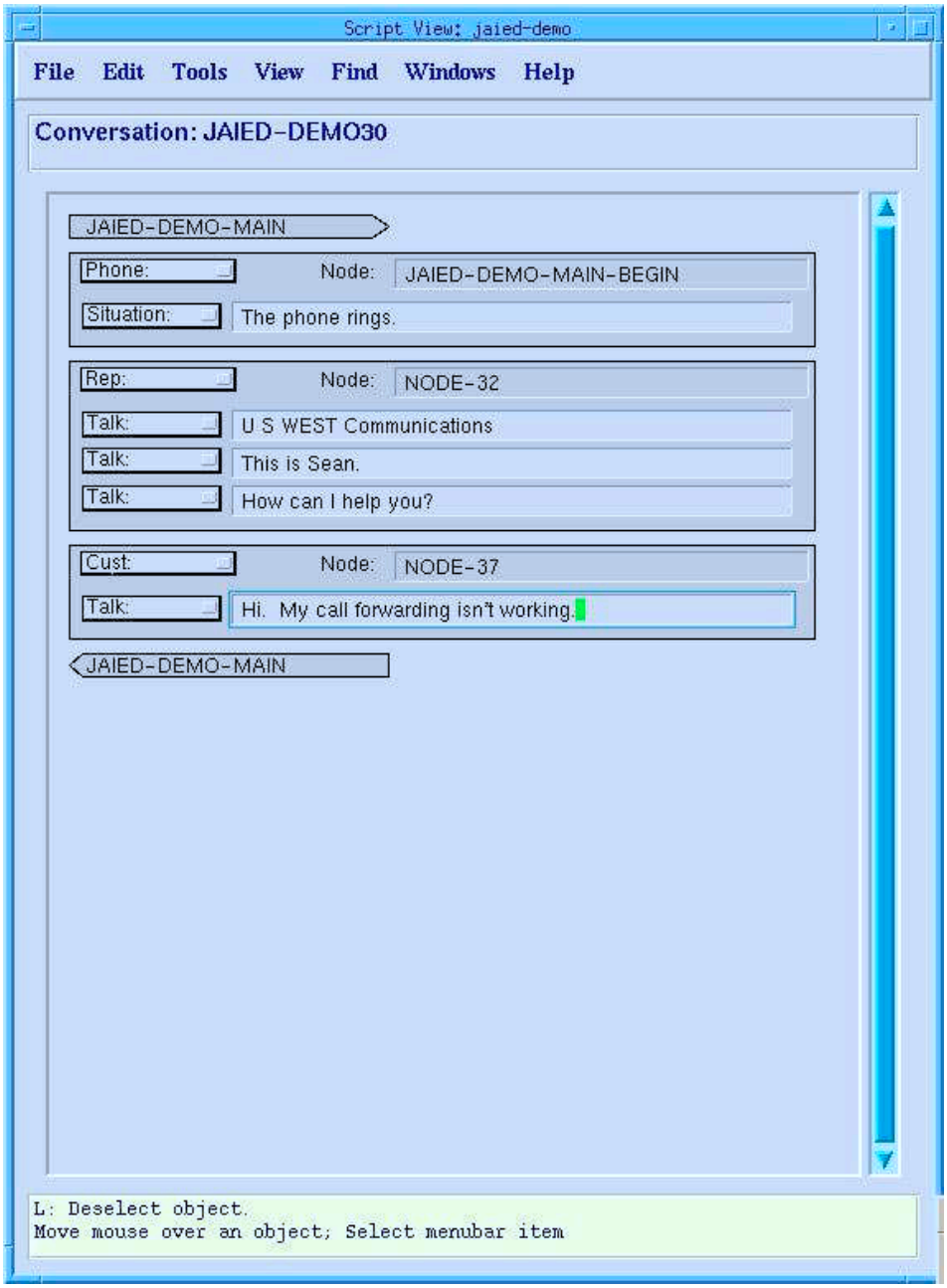

Figure 3. The Script Editor, showing the beginning of a conversation.

Of course, the default information provided will not always be what the author intends. The script editor provides several mechanisms for allowing the author to easily change the default information. The actor and action menus can be used to specify a new selection, in which case the LAT keeps track of the implications of these changes behind the scenes. In addition, the script editor allows the user to invoke other components of the LAT. Each of the visual objects in the script editor corresponds to a LEAP grammar object. The appropriate editor for each of

these objects can easily be invoked from within the script editor by selecting the object with the mouse and invoking an edit command.

Before continuing with our example, we describe two additional aspects of the visualization presented by the script editor: branching and subdialogues. LEAP conversation grammars are partitioned by authors into "subdialogues", each representing a relatively coherent chunk of the larger dialogue centered on a particular local topic. Knowing which subdialogues contain a particular node plays a vital role in the development of the grammars. The script editor displays this information with "subdialogue bars" that mark the beginning and end of subdialogues within the conversation. For example, as shown in Figure 4, the conversation begins with the JAIED-DEMO-MAIN subdialogue, indicated by the labeled subdialogue bar pointing to the right. After the node representing the telephone ringing, the embedded subdialogue OPEN-CONTACT-SUBD begins with the OPEN-CONTACT-START node. It ends four nodes later, as indicated by the labeled subdialogue bar pointing to the left.

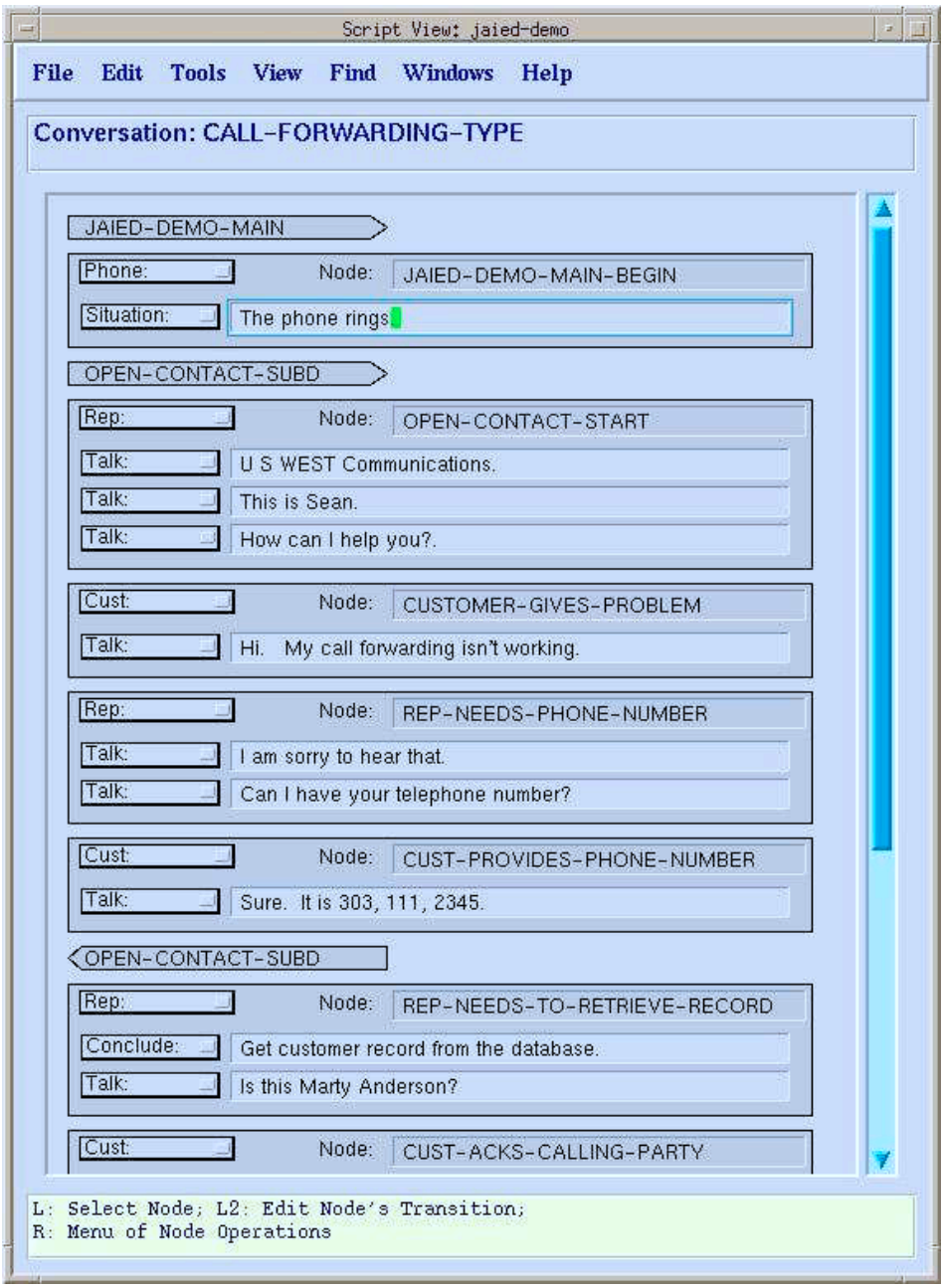

Figure 4. The Script Editor, showing an embedded subdialogue.

Branching represents places in the grammar where two conversational paths diverge. This may represent either a choice of two different possible paths for the learner to follow within a single conversation or a point in which two or more conversations that had been sharing a sequence of nodes now diverge. Because branching may be difficult for authors to keep track of, it is important to represent branching information visually. The presence of branches at a node is depicted by visual "tabs" on the bottom of the node, as shown on the node REP-INVESTIGATES-OR-UNDERSTANDS in Figure 4. The author can click on one of these tabs to select which branch of the conversation is displayed below the tab in the editor. An alternative view shows the conversation and its various branches as a graph, as shown in Figure 5.

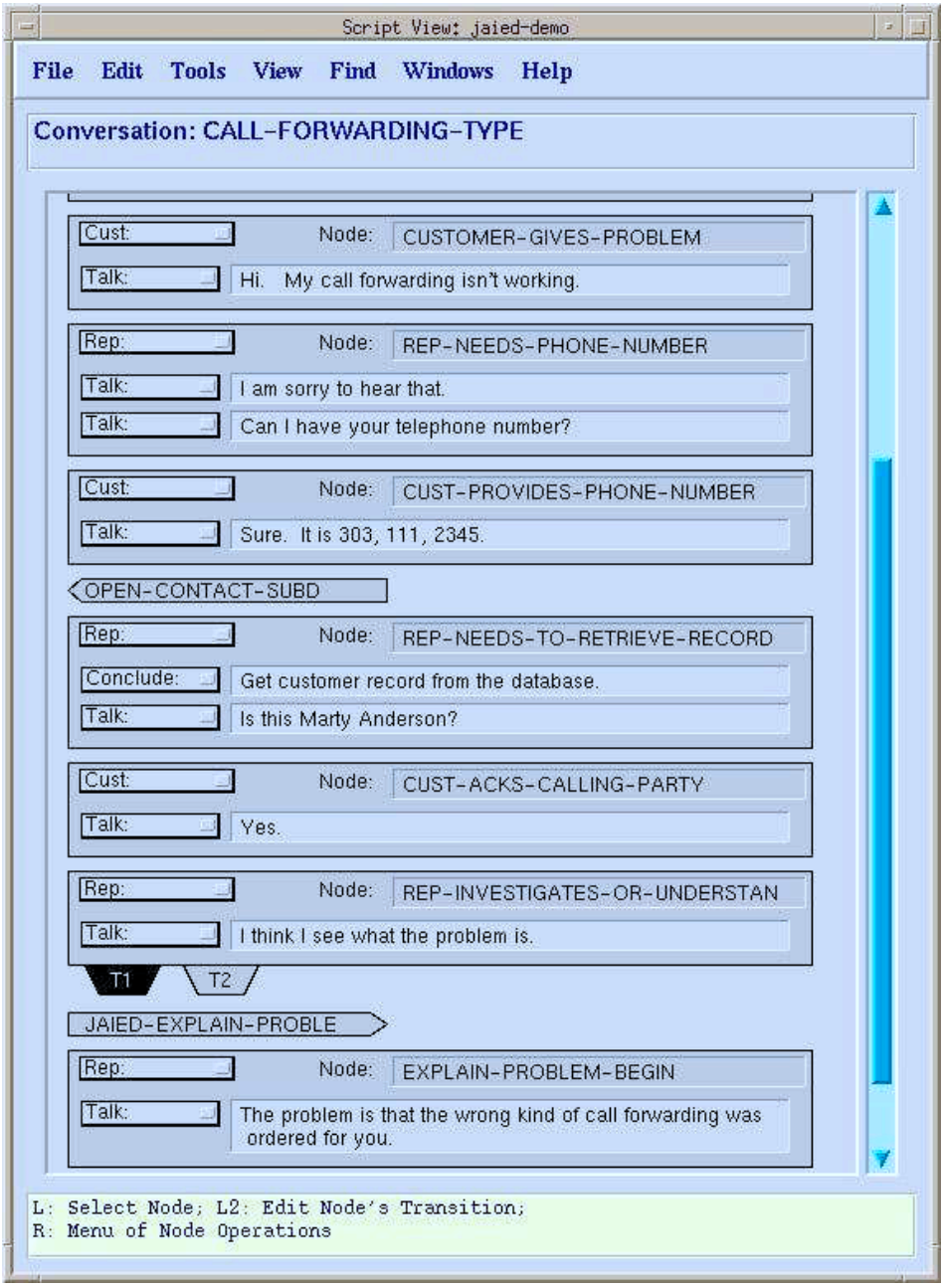

Figure 5. The Script Editor, showing a branching node.

Returning to our example conversation as shown in Figure 4, the author has now created several nodes and their associated actions by entering the text of these actions and allowing other information to default to the values supplied by the LAT. At this point, the author may

wish to invoke the LAT's ability to run the grammar under development in LEAP to see how the learner will experience the courseware as defined up to this point. This can be done seamlessly from within the LAT, because the LAT includes full LEAP run-time functionality and maintains an executable representation of the grammar objects being edited at all times. Of course, if grammar elements such as hints or recorded audio have not yet been authored, these will not be present when the grammar is run. For example, you would only see the written text for spoken actions rather than hearing them. By using useful default and place-holder information, the LAT allows authors to work with courseware that is runnable under LEAP right from the beginning of the authoring process. In our user testing, this proved to be a useful feature for the authoring process, as it allows an iterative write-and-test development cycle that lends itself to rapid prototyping of new courseware.

3.2. Subdialogue Graph Editor

The subdialogue graph editor is the LAT component best suited for giving the author an overview of the courseware being working on. Figure 6 shows how the editor looks for the simple conversation in our example. The graph shown represents the top level subdialogue for the set of conversations in the conversation grammar.

Each of the smaller rectangles (with labels in all capitals) in the figure represents a node, and the larger, compound rectangles represent transitions between nodes. As previously noted, a transition is the set of actions to be performed by the learner, simulated customer, or some other agent to move from one point in time (a node) to the next. In addition to a simple set of actions, a transition can consist, instead, of a set of one or more subdialogues. This is how nesting of subdialogues is accomplished. In our example, the last transition on the left-most branch in the figure represents a single subdialogue, as indicated by the label "Transition: Subdialogue". The name of the subdialogue is shown as JAIED-EXPLAIN-PROBLEM.

In addition to displaying a graphical representation for a dialogue, the subdialogue graph editor also allows the creation of nodes and transitions. This done by highlighting the appropriate node and choosing menu items to add or remove transitions from it. Nodes and transitions are created with basic, default information that can then be modified.

The subdialogue graph editor is a structured graph editor. As items are added and deleted from the subdialogue graph, the other items automatically reposition themselves appropriately. This ensures that the graph maintains readability and removes this burden from the user. In future versions we would like to add the ability for the author to reposition items manually if he or she desires. This would require keeping positioning information so that the user's version of a graph could be restored as well as adding the mechanism to handle dragging items with the mouse.

In addition to creating and deleting new graph elements, the author can get to any of the appropriate editors for a component by highlighting it and either double-clicking or choosing an appropriate menu item. For example, from a transition component the author can go to the transition editor (see the next section) to see the ordering restrictions on the actions, or he or she can go directly to an editor for a component of the transition. This could include an action editor, a new subdialogue graph, a subdialogue information editor, or a conversation information editor. Additionally, by double-clicking a node or by choosing the appropriate menu item, the author can edit the name and other information for any node in the graph.

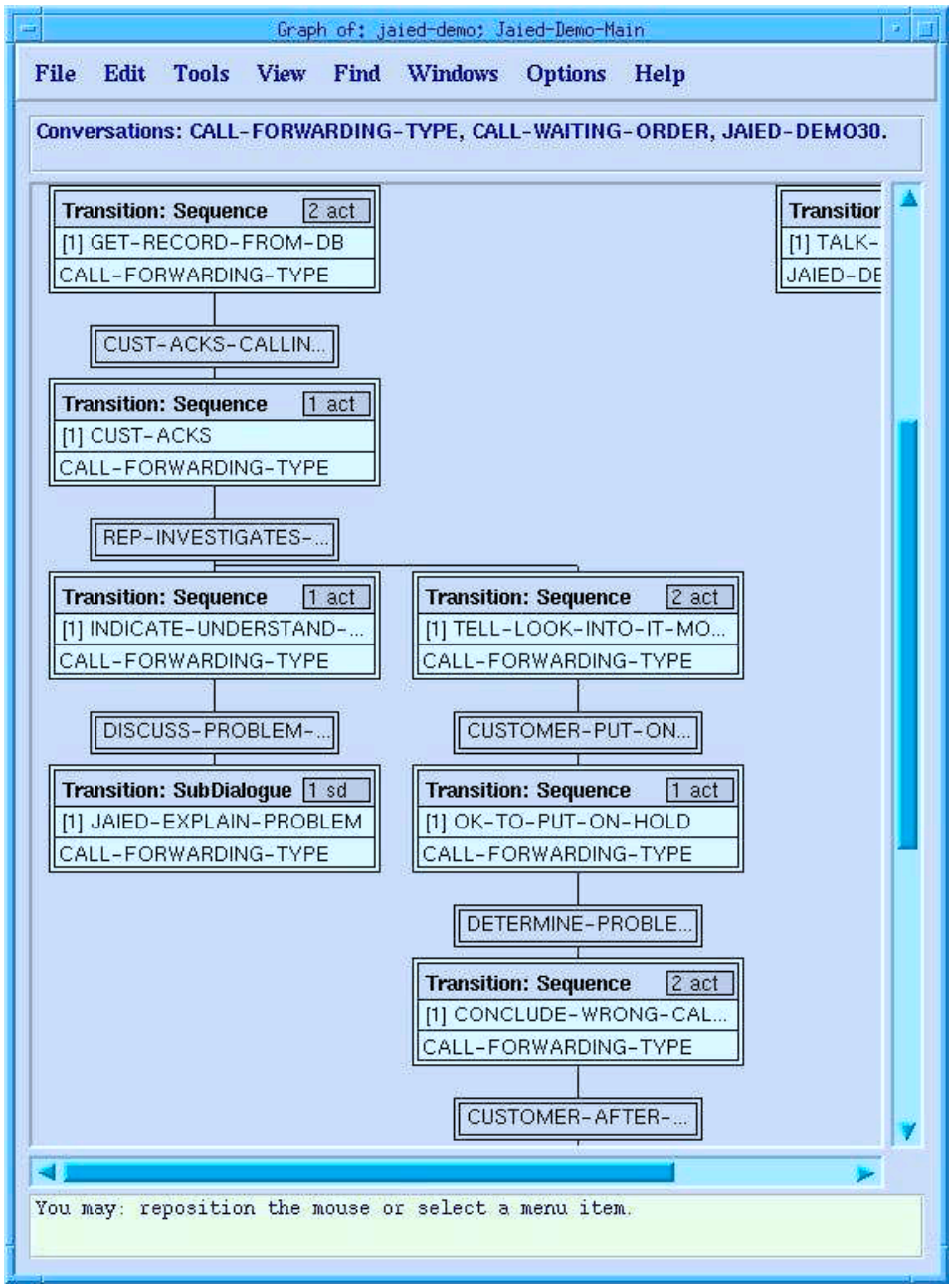

Figure 6. The Subdialogue Graph Editor.

3.3. Transition Editor

The LEAP grammar formalism allows for a great deal of flexibility in how a conversation may unfold. For example, at a given point in a conversation, the author may allow the learner to follow only a single path or may define multiple acceptable paths by which the conversation may move forward. This feature allows learners using LEAP to experience relatively realistic simulated conversational interactions. However, the price to be paid for this flexibility is a corresponding degree of complexity in the LEAP grammar. The transition editor was designed to provide authors with a powerful, directly manipulable visualization of the transitions between the nodes in the grammar. The progression of a conversation from one step to the next is defined in a LEAP grammar in "transitions" between nodes. Transitions are LEAP grammar objects that define, for any given node, which nodes can come next and what actions can be performed to

reach each of these possible next nodes. They are displayed in the transition editor in a way that quickly conveys which actions are available at a given node and the relationships among them. The relationships that may pertain among actions are those of ordering, co-occurrence, and optionality.

In some situations, actions must be taken in a certain order. For example, the actions Give-Company-Name, Give-Self-Name, and Ask-How-Can-Help in the OPEN-CONTACT-START node shown in Figure 7 are to be performed in this order. This is specified by combining the actions with the "Sequence" operator, as shown in the figure.

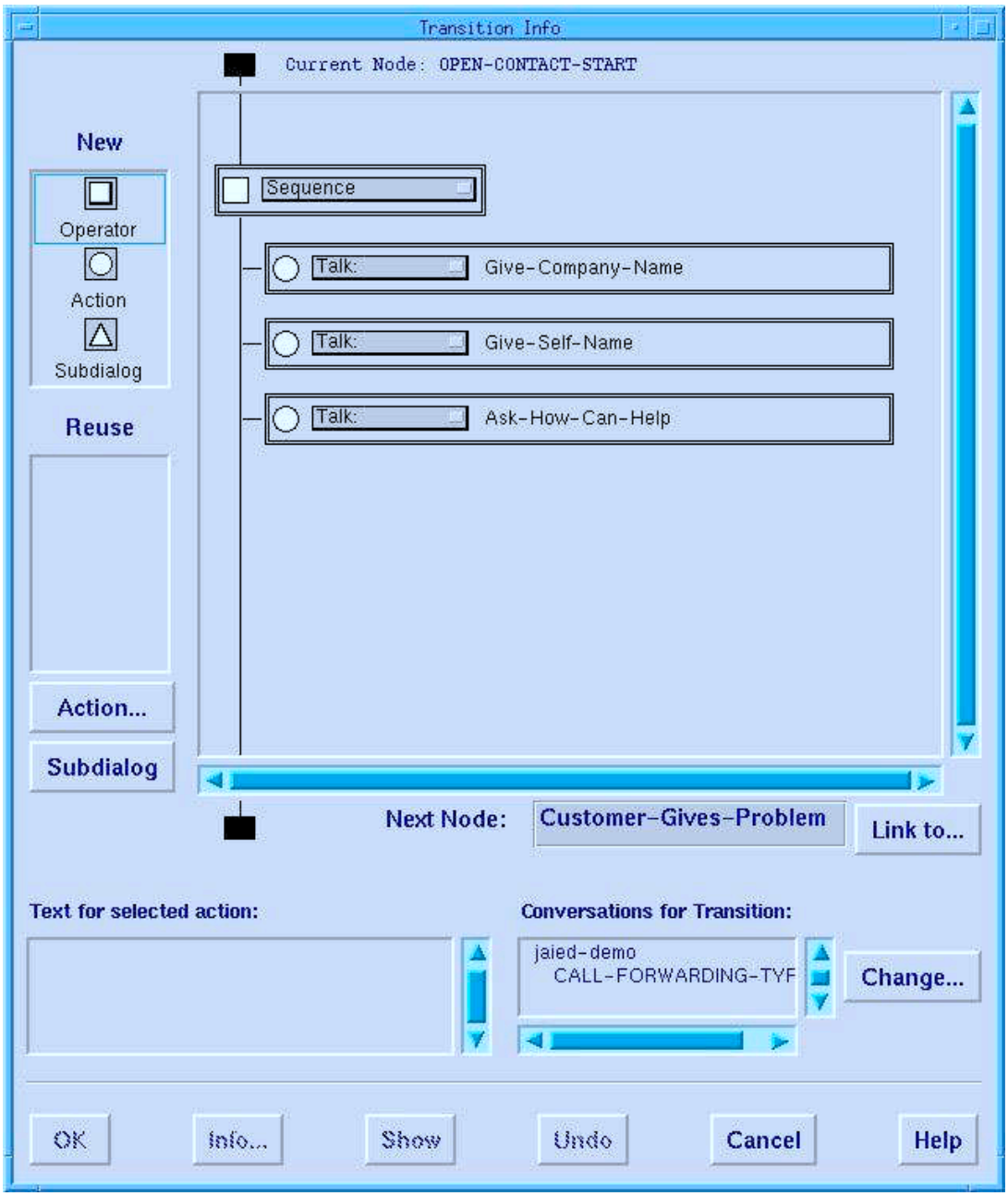

Figure 7. The Transition Editor.

In other situations, it may be appropriate to perform one action or another, but not both. In this case, there is a co-occurrence restriction between the two actions. This can be specified in the transition editor with the "xor" (exclusive or) operator. Finally, there are situations in which some actions are required while others are optional, which can be specified by other operators. Still other transition operators are used for combining subdialogues, rather than actions.

As shown in Figure 7, the main component of the transition editor is a large panel that displays graphically the current specification of the transition. A "flow line" runs down the left side of this panel that indicates how the transition links the node above to the node below. The visual objects inside this panel represent actions, subdialogues, and the transition operators used to combine them. The actions or subdialogues combined by an operator are referred to as its operands.

Most operators appear as short rectangles in the display and are marked with a small square icon. Operands are more elongated and are marked with with a circle (for actions) or a triangle (for subdialogues. In Figure 7, the operator is "Sequence" and the action objects "Give-Company-Name", "Give-Self-Name", and "Ask-How-Can-Help" are its operands. Both operators and operands have associated pop-up menus that allow an author to quickly change their type.

As a transition is essentially a recursive specification of operators and operands, only one operator is allowed on the flow line at the left of the panel. All other operands and operators are nested within this top-level operator. Authors may use the mouse to drag and drop operator and operand objects to specify the structure of the transition.

On the left side of the transition editor is a palette from which the author my create new transition objects. Existing grammar objects can be reused in the current transition by selecting them from a list, then dragging and dropping them at the desired location in the transition being edited.

3.4. The Agenda

Although the conversation in the example we have been using can be run directly in LEAP as it is, the courseware is not complete and suitable for tutoring purposes until several additional authoring steps have been completed. Most obviously, perhaps, authors will want to create audio recordings for all spoken actions. After these recordings have been made and saved, these audio files need to be associated with their corresponding actions. The current version of the LAT does not provide its own facilities for recording audio, so authors must use other tools to create the audio files, then associate them with actions using the LAT.

Authors will also want to engage in the other authoring steps, such as defining topics and associating actions with them, grouping sequences of conversation nodes into subdialogues, creating appropriate hints for each action and subdialogue, replacing the default names for nodes and actions with more helpful names, defining branches in conversations, etc. To help authors keep track of what authoring activities remain to be done to have a complete (as opposed to minimally runnable) LEAP grammar definition, the LAT provides authors with an agenda mechanism.

Complete LEAP grammars are complex, and in building all but the simplest conversations, it is difficult to keep track of all relevant aspects of the current state of the grammar. To help authors keep track of where they are in the authoring process, the agenda mechanism keeps track of the state of the grammar in relation to the authoring process. The various LAT editors communicate with the agenda in a manner that allows the agenda to watch authors as they construct the grammar. At any time, the author can call upon the agenda to provide assistance as to what remains to be completed.

The agenda contains a model of what information is needed to complete a LEAP grammar. By observing the author's activities, the agenda knows what information, if any, is still required before the grammar is complete. There are two categories of information that must be complete. The first includes missing information that is required for the grammar to run properly in LEAP. Without first supplying this information, the courseware may not run as the user expects. For example, a conversation might reach a point where it abruptly ends (i.e., the graph representing the grammar has unconnected pieces).

The second category consists of information for which the LAT has supplied default values. For example, the LAT keeps track of whether a recorded audio file has been specified for Talk actions. LEAP will display the text associated with the action if this action is executed

without an associated recording, but the recordings are needed for effective use of LEAP for tutoring.

3.5. Refining and Completing LEAP Grammars with the LAT Editors

To fully specify a LEAP grammar and make it ready for learners to use, an author needs to use additional form-style editors for each of the LEAP grammar objects. These editors allow the author to provide additional information not needed for initial prototyping but important to a fully functioning LEAP grammar. Six such editors are included in LAT; one each for modules, conversations, subdialogues, nodes, actions, and topics.

3.5.1. Node Editor

Figure 8 shows the node editor with a node representing a state in which the learner is expected either to recognize the customer's problem or to ask the customer to wait on hold for a short time while he/she figures out the problem. This information, as described in the comment field of the node, indicates the types of transitions the author has created that can be taken to move to the next node. For more information on transitions and the transition editor see section 3.3.

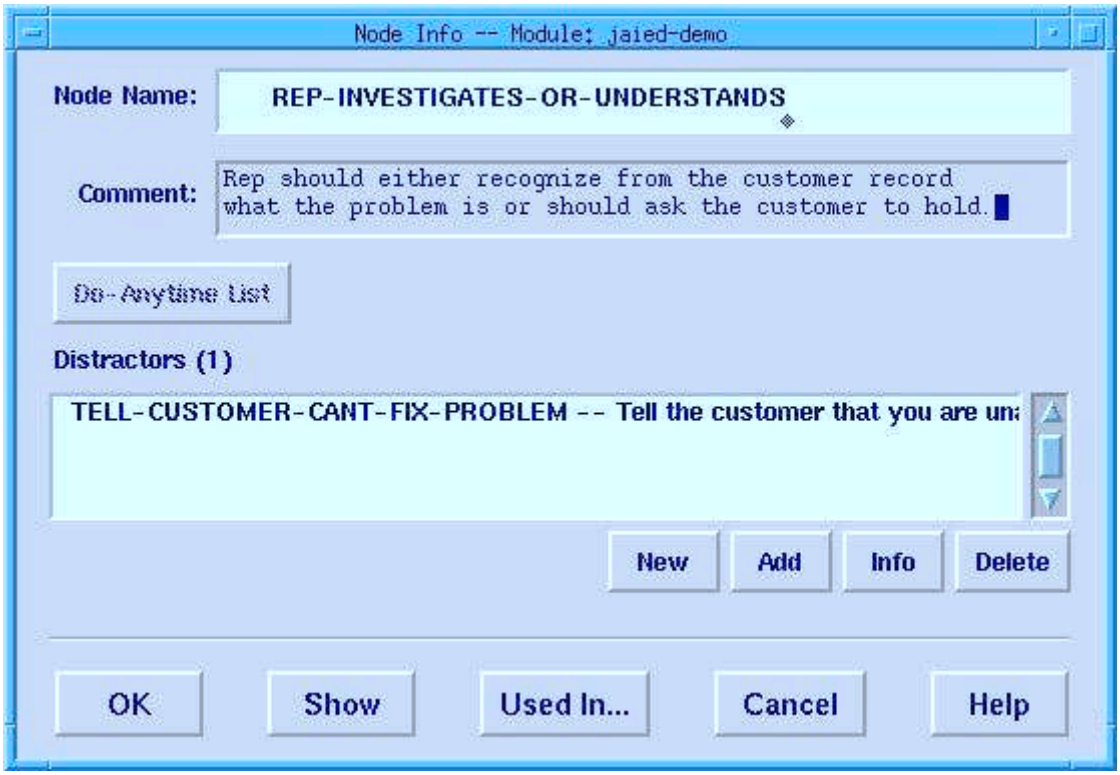

Figure 8. The Node Editor.

One optional field that an author may specify from the node editor is the distractor field. Distractors are used in LEAP in much the same way as they are used in multiple-choice tests. They are included with the set of correct steps for a given situation (node) for a learner to choose from. LEAP distractors are actually action instances, and the author can create actions specifically as distractors for a given node, choose existing actions (that are correct in other contexts) as distractors for a node, or let LEAP choose the distractors to be used for a node.

3.5.2. Action Editor

The most complicated object type, in terms of the number of fields, is the LEAP action. Actions of various types can be created. "Talk" actions, such as saying hello to the customer and the customer telling the learner that she has a problem with her service, are the most common type of action. Figure 9 shows the action editor for a talk action. Another action type is the "Conclude" action. These are used by an author when the learner does not need to perform any observable action, but rather needs to make an appropriate inference based on the current situation.

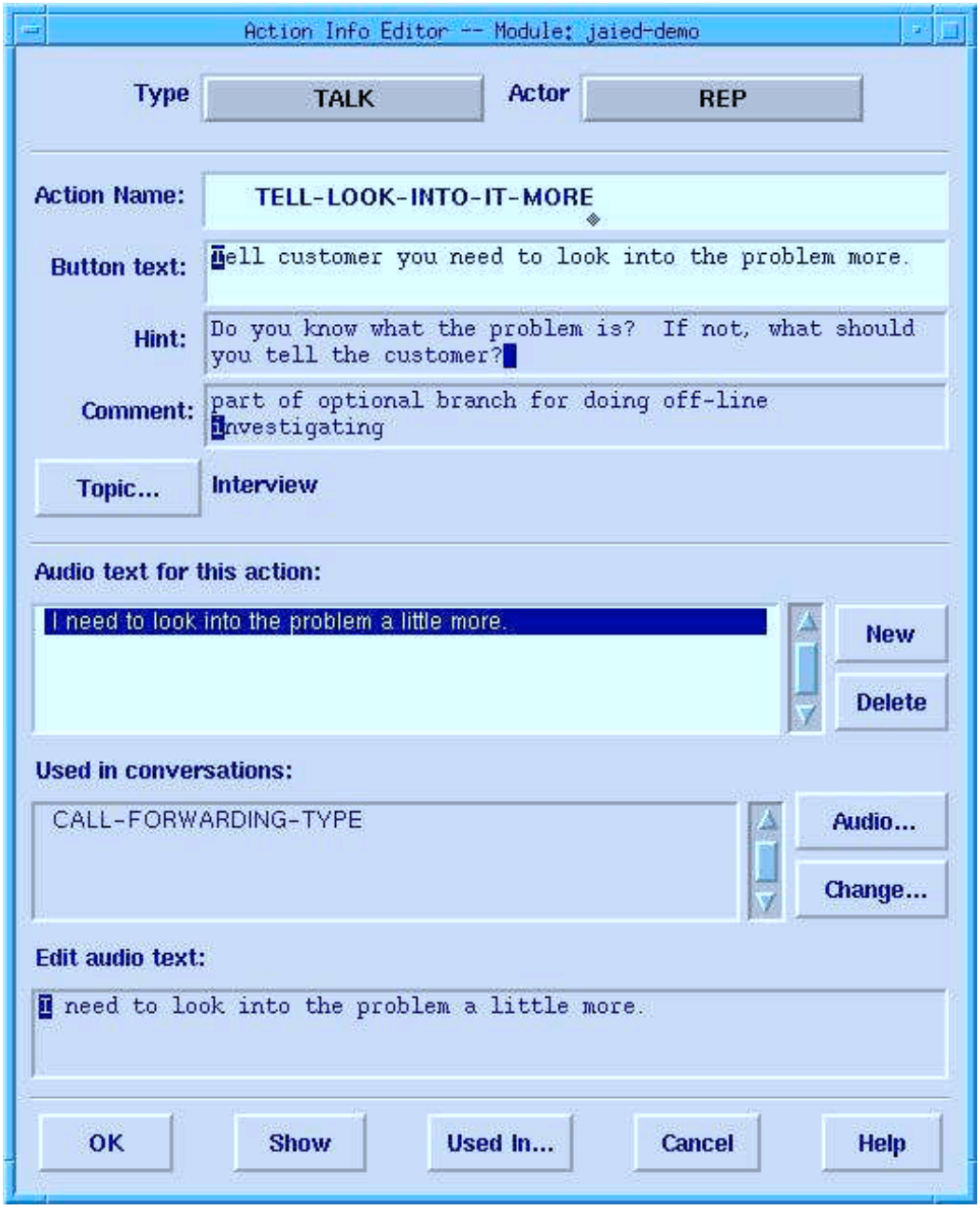

Figure 9. The Action Editor.

For actions in which the learner (referred to in Figure 9 as "Rep") is the actor, the author needs to specify the "button text". This text should briefly describe the action. It will appear for the learner using LEAP next to a checkbox in a list of choices of steps that may be appropriate for taking in a given situation. An optional but recommended learner action field is the hint.

When the learner requests a hint in a specific situation, the hint field values for the correct actions for that situation are displayed. Another important field for learner actions is the topic field. By associating an action with a topic, the author allows the LEAP tutoring module a way to group actions and determine mastery levels for topics by analyzing the learner's performance on the actions associated with the topic.

Even with the fields discussed above fully specified, an action created in the action editor is still generic in the sense that it is independent of any particular conversation. In other words, it is not yet really being used. What remains to be done is the application of the action to one or more conversations. The author does this by specifying the audio text (transcript) for talk actions, or a detailed description for other action types, for each specific conversation in which the action is to be included. Actually, this is exactly what is done by an author when he/she uses the script editor to type in what is to be said or done in a particular conversation. The action editor also provides a way to create or edit the audio text. It also provides a concise description of the applications of an action in specific conversations, whereas the script editor shows the action's audio text for just one conversation (although, as described earlier, it provides an excellent mechanism for creating and checking the flow of that conversation.) Figure 9 shows that the action TELL-LOOK-INTO-IT-MORE is used in just one conversation: CALL-FORWARDING-TYPE.

3.5.3. Other LAT Editors

The topic hierarchy editor, shown in Figure 10, lets the author specify the topics that the learner needs to master and their hierarchical relationship. It is used to specify the topic name and a short description of the topic which is displayed to the learner each time a learner highlights the topic, which is the first step to practicing a conversation focusing on the topic.

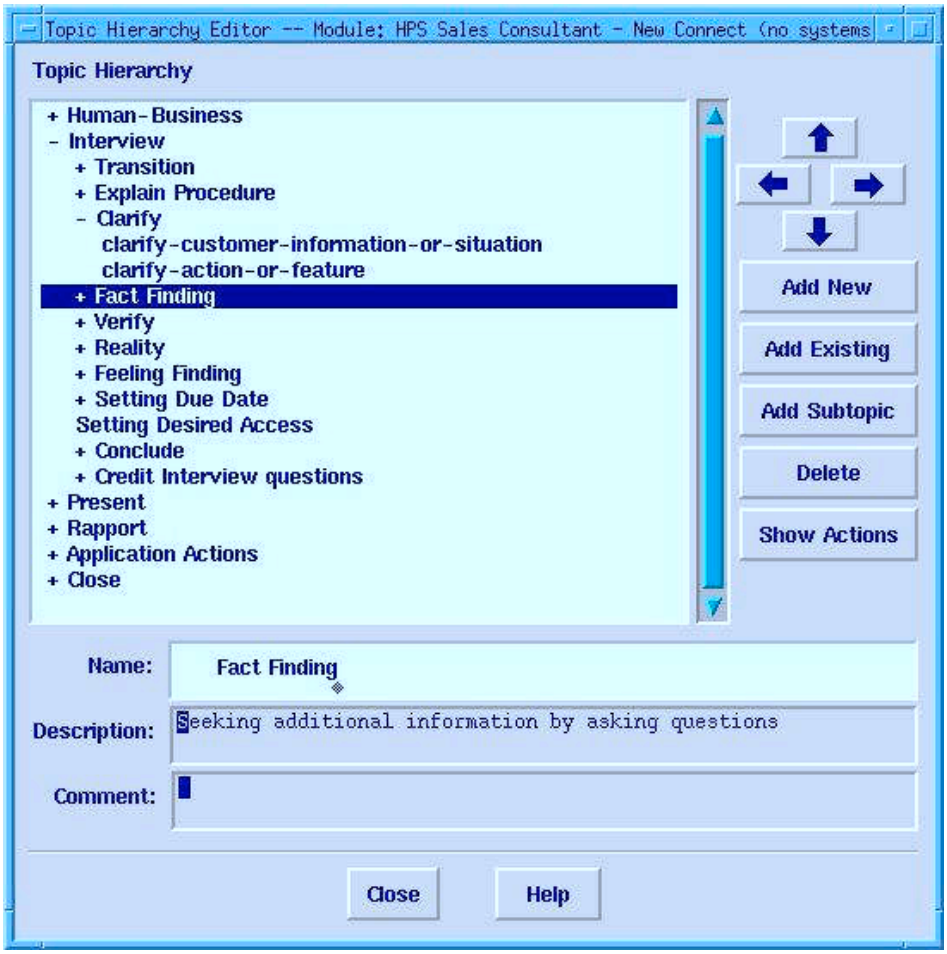

Figure 10. The Topic Hierarchy Editor.

Other secondary editors allow editing of such information as the author's estimated complexity of a conversation (used as a factor in suggesting conversations to practice by the LEAP tutoring module) and the description of a learning module that the user sees when he or she is choosing which module to practice.

4. DESIGN PROCESS

The LAT was built using a collaborative design process that included potential users, subject matter experts (in the initially targeted courseware domain of consultative skills), human factors experts, and programmers. (See Muller et al. 1995). Because the purpose of the LAT is to support the authoring of courseware to be run as a tutoring session in LEAP, the design of the LAT began with a specification of the functionality required to build working LEAP grammars. This part of the design process primarily involved the LEAP developers and took a relatively short amount of time. The bulk of the design process involved the specification of the user interface for the LAT. This phase involved numerous design meetings; building user interface mock-ups and paper simulations; and testing a small set of users with these simulated systems.

4.1. Use Cases for the LAT

The interface design phase began with a determination of a set of expected "use cases" for the tool. These use cases included such information as what background an author likely would have (in using computers, developing training materials, building other LEAP courseware, etc.), what an author's starting point would be (e.g., whether other types of training materials for the targeted domain were available), and what individual preferences and working styles should be considered. We used these use cases to determine the structure and content of the tools various components and the relationships among them, to recommend methods for use of the tool, and to test the tool's ability to support the required authoring activities.

We outlined three general use cases for the tool along the two dimensions of (1) the author's experience in using the tool and (2) the author's preferred approach to building courseware. The first use case involved a novice LEAP courseware author. We assumed that a novice would probably start by building scripts, then introduce more complex structures as he or she progressed. This would allow the novice author to get started without having to learn too much about the use of the tool and its more complex features. We also expected it to be easier for most authors start with the more familiar representation formalism of a script of a conversation.

A second use case involved an author maintaining existing courseware. The tool must support authors in the tasks of revising, repairing, and extending the courseware content. This use case required search tools and good visualizations so that the author could easily find the piece of courseware that needed to be changed and edit it.

A third use case involved the reuse of courseware objects to build new courseware or to extend existing courseware. Because building good courseware is a time-consuming effort, the tool was designed to support the reuse of courseware components for building new material. This use case also required good search tools and the ability to insert courseware components designed for other modules into the current module.

The design process we followed turned out to be a mixture of a sequential design-code-test approach and a more iterative design-as-you-code approach. Design team members tended to fall into one of two groups, depending of which approach they favored. One group wanted to follow a more traditional design process, where the design was documented up-front, then the tools were coded to this design and then tested. The benefit of this approach is that the design is thought out ahead of time and documented for all members to agree on and work towards.

The other group of design team members wanted to work out the design as various components were implemented. The motivation for this approach was that it was difficult and time-consuming to map out all of the aspects of and interactions among different tool

components in detail before beginning any implementation. In addition, designs sometimes led to dead ends that were only found when a use case was tried on an implemented system component. Once a component had been implemented, users could try it out. This typically exposed problems in the existing design and suggested possible improvements that could then be made. On the other hand, in some cases, the implementor's understanding of the goals for a component did not completely match that of the other team members.

There was also some tension between the two groups over control of the design. The developers could control the design either through the design meetings or by coding the tool in a particular way. Most of the developers favored a "design as you go" approach. The user interface experts, on the other hand, could only control the design by having a formal design document that was followed. Therefore, this group favored the design-code-test approach. The actual design occurred through a combination of both methods. The design of the main components of the tool was documented and followed, whereas details of the tool and some subsidiary components were designed as they were coded.

One lesson learned from our design process was that the diversity of backgrounds on the design team both aided and complicated the design process. We often found it difficult to find common ground, and some members of the team felt less empowered than others. (The programmers had the power to actually produce the interfaces, and thus had the most power). Our desire to support multiple authoring styles also caused difficulties, since different members of the team focused on different styles or use cases, sometimes producing conflicts over the importance of specific user interface strategies. However, the development of the LAT involved a complex set of goals and many different system components, and it was difficult for any one person to keep all of the goals and constraints in consideration and in balance. Multiple perspectives on the design kept us honest and struggling, so that we didn't forget or minimize any particular authoring style or use case. In the end, we found that our particular set of compromises between the two basic design strategies worked pretty well, though it did involve a considerable degree of frustration for different team members at different times.

4.2. Design Strategies

Throughout the design process, the team used analogies to existing systems and tools whenever possible. This aided our thinking about the LAT, and we expected at least some of these analogies to be helpful for new users of the LAT, as well. For example, in order to visualize how the script editor should work, we used two analogies: word processors and scripts for plays. Although LEAP grammars are networks of interconnected subdialogues, the result of engaging in a LEAP session is a linear sequence of interactions. In order for authors to test their courseware, they needed to be able to see the conversations that resulted from the courseware. These conversations are very similar to the script of a play. In a printed play script, the lines each character says are shown, preceded by the character's name. Other stage directions are marked in the script in the order in which they need to occur. Since scripts are a relatively well-known way to represent dialogues, we thought it would be useful to present LEAP conversations in this way. However, authors also need to be able to edit these scripts. Therefore, we combined the structure of a script with the editing capabilities of a word processor (e.g., cutting and pasting text), which are also familiar to our target users.

Another way we tried to leverage users' familiarity with other tools was to include a facility for importing scripts composed as simple texts. Users can write a script in a word processor, save it as a simple text file, then import it for use in LEAP. As long as the file follows the simple formatting convention of lines of text preceded by the actor's name, the LAT will read the script file and transform it into the data structures required by LEAP. A simple, working LEAP grammar can be created this way, although authors will always want to use the LAT further to complete the conversation with hints, topics, audio, etc.

Similarly, we hoped to leverage authors' potential familiarity with graphics programs to enhance the usability of the graphical components of the LAT, especially the subdialogue graph and transition editors. Although our direct experience with authors during the design of the LAT was limited to three potential users, our conclusion is that most of our targeted author

community would not have a great deal of familiarity with the kinds of graphical tools that would best transfer to using the LAT. For example, the graphical class browsers often included in object-oriented programming environments have a number of similarities to the LAT's editors, but we expect few of our targeted authors to be familiar with this or similar types of graphical editor. Rather, many of them may be more familiar with such graphics applications as draw and paint programs. These, however, bear only a superficial resemblance to the LAT's graphical components. (For example, typical drawing programs give the user a great deal of control over the layout and appearance of objects, whereas the LAT's graphical editors impose a much greater degree of constraint over the appearance of a given structure--mostly, we believe, for the benefit of typical authors.)

So, although in retrospect we don't believe the LAT design has been able to leverage users' prior familiarity with similar graphical programs to the extent we had hoped, the graphical representations used have still proved successful in their goal of enabling authors to conceptualize and manipulate the complex structures of LEAP grammar representations. The three authors who have both written LEAP courseware by hand and used the LAT all found the LAT's graphical representations superior to the alternative textual representations and were able to learn the LAT graphical conventions and manipulations relatively easily.

5. EVALUATION OF THE LEAP AUTHORING TOOL

After determining the main components that would be needed for the LAT, based on our chosen set of use cases, we conducted participatory design sessions. Participants included several representatives of the actual community of potential LEAP authors, a CCE training expert, as well as the LAT software developers and human factors experts. These sessions resulted in both paper and interactive computer-based mockups of the various LAT user interface components. These designs were then tested with potential authors and several others. Based on the results of these tests, we made appropriate revisions to the design and began development of the LAT software.

After an initial (alpha-level) version of the LAT was built, we conducted a focused user test with one novice author. This evaluation focused on the following questions: (1) Can the user create a simple module easily?, (2) Does the flow of use of the tools seem "natural" to the author?, (3) Is the method of presenting the grammar (script, graph, and information windows) clear and helpful?, (4) Are the concepts of conversations, subdialogues, transitions, and reuse clear?, and (5) Did the user have any trouble finding and using specific windows, commands, or operations?

After completing less than one day on instruction and a tutorial, the author was easily able to build a simple module. The author found the tool easy to work with and was able to use all of the components of the LAT successfully and produced a working body of courseware within three days (Meiskey, Root, & Somers, 1996).

We attribute the successful results of this user test (and of the design of the tool) to these factors. First, our method for training the novice user was to move from building simple to complex courseware. The tool was specifically designed to accommodate this transition. Second, in the design of the LAT, we had used analogies to other tools that were familiar to the user, including word processors and graphics programs. In fact, the user was annoyed when the tool didn't follow some of the conventions of other tools (e.g., she couldn't drag and drop objects in the graph editor).

The LAT was also used by two other authors for maintaining the large body of courseware initially developed in parallel with the LEAP system for training CCEs. These authors found the LAT to be a great improvement over their previous method for developing courseware: editing grammar files as text, an extremely difficult and error-prone method. The authors were able to take a large LEAP grammar and create several new ones from it by reusing subdialogues, adding new "coaching" actions, and making other relatively minor changes.

6. SUMMARY & CONCLUSION

LEAP is an ITS that provides a rich, apprenticeship environment for people learning CCE skills. The LEAP Authoring Tool provides LEAP courseware authors with a powerful, yet relatively easy-to-use tool for developing the kind of high-quality and sophisticated courseware needed for such a rich learning environment in a cost-effective and, we think, enjoyable manner. The LAT was designed to achieve this through a number of features that support rapid prototyping of new courseware material, reuse of existing material, and multiple, interactive visualizations for creating and manipulating courseware structures and components.

One valuable lesson that we learned through the experience of designing and developing LEAP and the LAT is it is important to maintain a focus on ease-of-use throughout the design and development process. In the end, we felt that we had probably erred somewhat in the direction of including power features that would be of use primarily to more experienced users because this also added complexity and therefore some difficulty for less experienced users. We believe we missed the the best possible balance between power and ease-of-use primarily because we over-estimated the level of experience that would be achieved by typical LEAP authors, but also because the development effort was somewhat experimental in nature, and we felt it important to explore new possibilities and approaches to solving the challenges of authoring complex courseware.

We believe that much of the success of the LAT design was due to the fact that we had the participation of appropriate subject matter experts (potential authors) and human factors experts from the beginning of the project. The requirements and recommendations of our initial LEAP courseware authors provided direction at the key decision points in the design of the LAT. We also learned a great deal about how to design some of the LAT components simply by watching the authors struggle with the task of writing courseware. In this sense, it was quite valuable to have the working LEAP system with (hand-coded) courseware already running before we began the design of the LAT. The requirements of the existing LEAP system and the experiences of the original courseware authors provided a great deal of valuable constraint on the design of the LAT.

It also proved important to use a participatory design process that captured the input and expertise of the different categories of team members: subject matter experts (in our case, both expert CCE trainers and expert CCEs themselves), human factors and user interface experts, and software developers. Many conflicts among team members and their points of view had to be resolved to achieve the best results. We also profited from an iterative design process that included early user evaluations of our designs (e.g., using paper or computerized mockups).

Finally, we believe that our experience developing the LAT has also taught us the importance of (1) providing authoring tool users with multiple ways of accomplishing the primary tasks involved in courseware development, (2) providing multiple, well-designed, and manipulable visualizations of courseware structure, (3) accommodating different authors' individual preferences, work styles, and starting points for authoring, and (4) supporting rapid, incremental courseware development that allows authors to test what they have created so far throughout the development process.

The LEAP system has so far been used by a few hundred learners, and user evaluations have confirmed its value as a tool for training CCEs (Bloom, Linton, & Bell 1993). However, it has not yet been deployed as widely as we would like and, as a result, the LAT has not yet been widely used. This is due in part to the fact that the requirements for the delivery of LEAP training have changed. Efforts are currently under way to deliver LEAP training over a corporate intranet.

REFERENCES

Bloom, C. P., Linton, F., & Bell, B. (1993). *An Evaluation of the Learn, Explore and Practice Intelligent Tutoring Systems Platform*. U S WEST Advanced Technologies, Technical Report AT-09_10-002817-00.01.

Bloom, C. P., Bell, B., Meiskey, L., Sparks, R., Dooley, S., & Linton, F. (1995). *Putting intelligent tutoring systems technology into practice: A study in technology extension and transfer*. Machine Mediated Learning, 5, 13-41.

Blumenthal, R., Meiskey, L., Dooley, S., Sparks, R. (1996). *Reducing Development Costs With Intelligent Tutoring System Shells*. Position Paper for ITS'96 Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs, Montreal, June 10, 1996.

Collins, A., Brown, J. S., & Newman, S. E. (1989). *Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics*. In L. B. Resnick (Ed.), Knowing, learning, and instruction: Essays in honor of Robert Glaser (pp. 453-494). Hillsdale NJ: Erlbaum.

Dooley, S. A., Meiskey, L., Blumenthal, R., Sparks, R. (1995). *Developing Usable Intelligent Tutoring System Shells*. Workshop on Authoring Shells for Intelligent Tutoring Systems, 7th World Conference on Artificial Intelligence in Education (AI-ED '95).

Lave, J., and Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge: Cambridge University Press.

Lesgold, A., Lajoie, S., 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. Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 201-238.

Meiskey, L., Root, R., Somers, P. (1996). *LEAP Authoring Tool User Test Phase 1 Results*. U S WEST Advanced Technologies, Technical Report No. T-09_04-005638-01.00.

Muller, M. J., McClard A., Bell B., Dooley, S., Meiskey L., Meskill, J. A., Sparks R., & Tellam, D. (1995). *Validating an extension to participatory heuristic evaluation: Quality of work and quality of work life.* Proceedings of the 1995 ACM Conference on Human Factors in Computing Systems (CHI '95), 115-116. Denver, Colorado.

NOTES

[1] LEAP and the LAT were developed at U S WEST Advanced Technologies in Boulder, CO. LEAP is a service mark of U S WEST.

[2] Although most of the training situations targeted by LEAP that we know of involve telephone-based conversations, we expect that LEAP would also be appropriate for training face-to-face interactive tasks, as well. For example, one inquiry that we have received regarding a possible use for LEAP was for preparing legal testimony.