Simulation Awareness: Assessing Performance with Limited Simulation Instrumentation

Randy Jensen, Sowmya Ramachandran, PhD,
Eric Domeshek, PhD, Kathy Tang
Stottler Henke Associates, Inc.
San Mateo, CA
jensen@shai.com, sowmya@shai.com,
domeshek@shai.com, ktang@shai.com

John Marsh
Comtech TCS
Pensacola, FL
john.marsh@comtechtel.com

ABSTRACT

Experts in troubleshooting are skilled at identifying important diagnostic cues and making justified inferences about problems and their causes. In a training setting, students can be assessed for the same troubleshooting skills, as long as there is clarity about the cues students use as a basis for their decisions. In a simulation-based intelligent tutoring system (ITS) where assessment is automated, this means the simulation must be transparent enough to afford an accurate picture of the cues that have been revealed to the student in the environment, in order to validate or invalidate the student's decisions. But not all simulations can be affordably instrumented to provide the desired level of transparency. This paper describes a domain modeling and assessment approach designed to accommodate reasoning with partial knowledge in cases where there are practical limits on simulation instrumentation. The assessment approach is applied in an ITS for training information technology troubleshooting, with a free-play simulation using virtual machines to reproduce a realistic network of computers. From an experiential point of view, this is ideal for giving trainees the opportunity to perform in a realistic operational environment. However, only a subset of simulation events in the virtual machines can be feasibly collected by instrumentation, and in many cases it is only practical to monitor either student actions, or their results, but not both. The paper describes the modeling and assessment approach in this context, with examples where reductions in simulation instrumentation were achievable. We discuss the applicability of this approach for other domains and its limitations, as well as the methods used for model authoring.

ABOUT THE AUTHORS

Randy Jensen is a group manager at Stottler Henke Associates, Inc., working in training systems since 1993. His research areas include adaptive training, distributed learning, game-based training, behavior modeling, and natural language processing. He has led projects to develop Intelligent Tutoring Systems and automated after action review tools for the Army, Air Force, Navy, and Marines. Recent work includes a model-based performance assessment capability for training troubleshooting skills in an intelligent tutor for the U.S. Navy. He also recently led the development of a game-based trainer for small unit tactical decision-making at the U.S. Military Academy at West Point. Mr. Jensen holds a B.S. with honors in symbolic systems from Stanford University.

Dr. Sowmya Ramachandran is a research scientist at Stottler Henke Associates, where her research focuses on the application of AI and Machine Learning to improve education and training. She leads research and development of intelligent tutoring systems (ITSs) and ITS authoring tools for a diverse range of military and civilian domains. Dr. Ramachandran headed the development of ReadInsight, an intelligent tutor for teaching reading comprehension skills to adult English speakers. She also led the development of a tutor for training Tactical Action Officers in the Navy. This system uses natural language processing technologies to assess and train TAOs and is currently in operational use at the Surface Warfare Officers School. She has recently led the development of an ITS for training U.S. Navy Information Systems Technicians in troubleshooting and maintenance skills. Dr. Ramachandran holds a Ph.D. from The University of Texas at Austin. For her dissertation, she developed a novel machine learning technique for constructing Bayesian Network models from data.

Dr. Eric Domeshek is an AI project manager at Stottler Henke Associates, Inc., where he leads and supports projects applying AI technology to problems in training and decision support. He has worked on a wide range of ITSs and related training, education, and simulation environments spanning applications to military tactics, medical diagnosis, engineering systems management, business decision-making, and historical analysis. He is particularly interested in exploration of Socratic tutoring techniques and the development of authoring tools. He recently led work on the authoring tools for the ITADS ITS in development for the U.S. Navy. Dr. Domeshek received his Ph.D. in Computer Science from Yale University, focused on case-based reasoning. For his dissertation, he developed representations of decision rationale for social situations, intended to support case retrieval; this included extensive representations of characters' relationships, traits, and motivational structures. He served as research faculty at the Georgia Institute of Technology College of Computing where he contributed to the development of a line of case-based design aids. He was also an assistant professor at Northwestern University, developing goal-based scenario training systems at the Institute for the Learning Sciences.

Kathy Tang is a project manager at Stottler Henke Associates, Inc., where she works on a variety of artificial intelligence related projects. She has interests in a diverse range of research areas including intelligent tutoring systems, vision systems, and scheduling systems. She is a major contributor to the development efforts of the tutor for the ITADS ITS for the U.S. Navy, leading the performance assessment engine implementation. Ms. Tang holds a M.S. in Electrical Engineering, with a focus in Machine Learning and Computer Vision from Stanford University.

John Marsh is a U.S. Navy IT subject matter expert at Comtech TCS (formerly TCS), who specializes in cyber range development. He also serves as an Adjunct Professor in the Computer Science Department at Pensacola State College. Mr. Marsh is a PhD candidate in the College of Engineering and Computing at Nova Southeastern University. He is interested in cyber security and privacy. More specifically, his work examines privacy concerns related to social media usage.

Simulation Awareness: Assessing Performance with Limited Simulation Instrumentation

Randy Jensen, Sowmya Ramachandran, PhD,
Eric Domeshek, PhD, Kathy Tang
Stottler Henke Associates, Inc.
San Mateo, CA
jensen@shai.com, sowmya@shai.com,
domeshek@shai.com, ktang@shai.com

John Marsh
Comtech TCS
Pensacola, FL
john.marsh@comtechtel.com

INTRODUCTION

Experts in troubleshooting are skilled at identifying important diagnostic cues and making justified inferences about problems and their causes. In a training setting, students can be assessed for the same troubleshooting skills, as long as there is clarity about the cues students use as a basis for their decisions. But the question of whether an inference is justified or not can be entirely dependent on what information the student has uncovered. A student may assert a hypothesis that is factually correct but that remains unjustified until certain supporting facts have been revealed. In a simulation-based intelligent tutoring system (ITS) where assessment is automated, this means the simulation must be transparent enough to afford an accurate picture of the cues that have been revealed to the student in the environment, in order to validate or invalidate the student's decisions. Depending on the domain, these cues may either be the direct results of student actions, or events unfolding in the virtual environment independent of student actions. But not all simulations can be affordably instrumented to provide the desired level of transparency. This paper describes a domain modeling and assessment approach designed to accommodate reasoning with partial knowledge in cases where there are practical limits on simulation instrumentation.

The assessment approach is applied in an ITS for training information technology troubleshooting, with a free-play simulation using virtual machines to reproduce a realistic network of computers. From an experiential point of view, this is ideal for giving trainees the opportunity to perform in a realistic operational environment. However, the nature of the virtual machines and the range of possible actions makes it prohibitively expensive to instrument the simulation with collection mechanisms to support a complete record of student actions and their results, as well as to develop a model with which to interpret and assess performance. While the effort to develop the instrumentation and modeling for any single action or result event may be small, the challenge arises when quantities potentially number in the hundreds, with an aggregate effect on the total modeling and instrumentation cost. Thus in this approach, the assessment model and inference mechanisms are structured to monitor either student actions, or their results, without requiring both. The paper describes the modeling and assessment approach in the context of the troubleshooting ITS, with examples where reductions in simulation instrumentation were achievable. This approach has the potential to benefit other domains by limiting the cost of instrumentation and modeling. We discuss applicability for other domains and limitations, as well as the methods used for model authoring.

BACKGROUND: SIMULATION AWARENESS CHALLENGES

Often in tutors for unstructured troubleshooting or problem-solving domains, a custom simulation or scenario player environment is constructed that exhibits controlled behavior based on a process model. The process model has a dual role, not only informing assessment of student decisions but also driving simulation behavior. This guarantees a parallel between the assessment model and the simulation behavior, which also makes it possible to explain connections between simulation cues and valid decisions (Wenger, 1987). The Sherlock tutor (Lesgold, Lajoie, Bunzo, & Eggan, 1992) is a commonly cited successful example of this strategy, used to train Air Force technicians in the diagnosis of possible faults in electronics boards. The Sherlock process model accommodates multiple solution paths by decomposing domain knowledge into individual skills such as the performance of specific diagnostic actions, as opposed to tracing predefined expert solution paths. With the process model tightly integrated with the simulation to the extent that it is literally the origin of simulation responses to diagnostic actions, it is unnecessary to collect the responses back from the simulation since the model is the source of this knowledge. In a similar manner, other simulation-based tutors are often constructed to leverage a close relationship with the simulation. However, it is more difficult to implement ITSs with independent, operational virtual environments, partly due to the challenge of building sufficient instrumentation for assessment. In such cases, the assessment model may parallel the simulation's exhibited behavior, but it requires sufficient data collection fidelity from the simulation to maintain the parallel state – that is, to maintain awareness of student actions and simulation behavior.

This paper describes a domain modeling and assessment approach in the context of a sample training domain of IT troubleshooting skills for Navy shipboard helpdesk personnel. The approach is applied in a research project called ITADS (Intelligent Tutoring Authoring and Delivery System), which aims to advance general practices of ITS authoring, starting with the IT troubleshooting application. The assessment modeling methods used in the ITADS project illustrate general concepts that may apply to a range of simulation-based training domains where an independent simulation must be sufficiently instrumented to provide awareness to an assessment model.

At the highest level, the assessment model structure resembles the PARI cognitive task analysis technique (Hall, Gott, & Pokorny, 1995). PARI codifies four cognitive elements of problem-solving or complex decision-making activities: Precursor – Action – Result – Interpretation. As a knowledge engineering formalism, PARI is used to structure interviews with experts and novices, where the representation of a solution path for a given problem is broken down using these four elements. For the purposes of an assessment model, roughly the same components apply, although the Precursor component is omitted because it can be challenging to factor into assessment with a simulation-based trainer. The Precursor concept is intended to account for a decision-maker's intention before taking an Action. One case where the Precursor concept could be applied would be a dialog-based trainer, where it is common to elicit intentions before actions, as the training method itself affords control over a relatively small, fixed set of actions available to the student. But in a free-play simulation-based trainer, and particularly a situated tutor where the objective is to allow trainees to perform in an environment resembling the real operational world, it may be impossible to elicit intentions by stealthy means, and too disruptive to query for them overtly.

Both the PARI Precursor component and Interpretation component relate to the student's mental model. Often in the troubleshooting loop, the intention Precursor for a planned action is informed by the operator's Interpretation from a prior Action and Result. So while there may be practical reasons to allow students to perform actions in a simulation without declaring intentions (i.e., to omit the Precursor component), the Interpretation component is a critical assessment factor in decision-making domains and cannot be overlooked. In a troubleshooting domain, Interpretations are essentially possible diagnoses that must be explicitly declared by the student if an assessment model is to determine if they are justified and correct. Thus the assessment model developed for ITADS focuses primarily on applying the remaining three PARI elements: Actions, Results, and Interpretations. Figure 1 below depicts how these constructs are adapted for an assessment model.

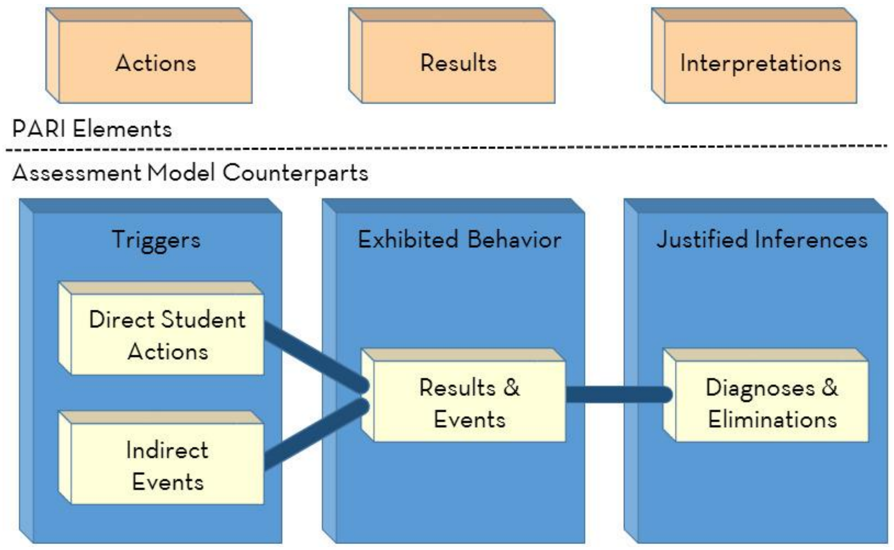

Figure 1. Organizational Constructs for Decision-Making Assessment

For scenario-driven simulation-based training, the mapping of working ITADS assessment model elements to PARI concepts is discussed in more detail in Table 1 below.

Table 1. Model Constructs Mapped to PARI

Element	Description	Examples from IT Troubleshooting
Triggers	• Comparable to PARI Actions, but Triggers include Indirect Events. • Collected from a simulator. • Direct Student Actions are overt actions performed by a student in the simulator. • Indirect Events are independent of the student's actions, such as a scripted event.	• Direct Student Action o Student double-clicks on a network folder to open a shared resource • Indirect Event o Scheduled backup on a workstation is initiated (but this event is invisible to the student)
Exhibited Behavior	• Comparable to PARI Results. • Collected from a simulator. • Observable changes in behavior in a simulator, as a result of Triggers.	• Exhibited Behavior from a Direct Student Action o Network folder contents are displayed (result of double-clicking on the folder) • Exhibited Behavior from an Indirect Event o Warning popup from a backup operation that failed to connect with the backup server (result of the Indirect Event of a scheduled backup)
Justified Inferences	• Comparable to PARI Interpretations. • For a troubleshooting domain, inferences are represented as problem-related hypotheses to be expressed by students and traced in an assessment model. • Diagnoses and Eliminations are student inferences that hypotheses are true or false (respectively).	After a Trigger of double-clicking to open a network folder, and the resulting Exhibited Behavior of the simulator displaying the folder contents… • Diagnosis o Assert that a file is missing from the folder • Elimination o Rule out the hypothesis that the folder cannot be reached due to connectivity problems

Application of these constructs in an assessment model for simulation-based training requires three primary tasks:

- **Simulation instrumentation**. When events corresponding to Triggers and Exhibited Behavior occur in the simulation, they must be conveyed in a form that can be recognized by the model.
- **Simulation modeling**. The model must contain a representational structure accommodating the space of possible Triggers and Exhibited Behavior that are observable and meaningful in the simulation.
- **Inference modeling**. The relationships between all cues (Triggers and Exhibited Behavior) and Justified Inferences must also be modeled, to assess student inferences compared to what the model deems correct.

As an illustration of how these tasks combine to form a workable assessment model, consider the objectives for modeling a "ping" command, a common diagnostic tool for the IT troubleshooter. A network device may be pinged by address (e.g., "ping 192.168.1.22") or by name (e.g., "ping workstation1"). Although these two ways of pinging a device may both produce the same results in terms of success or failure, they may reveal different information about the network configuration. The result text from pinging a computer by name also reveals the address, but the reverse is not true – pinging a computer by address does not reveal the name. So in a training setting, the ping command itself is a Trigger, and more specifically a Direct Student Action. The result of the command, in the form of output text in the command line interface, is the Exhibited Behavior in the simulation. The knowledge of how to interpret the output text, or more generally the knowledge about what is revealed by the combined Trigger action and Behavior result, is modeled as Justified Inferences. If a ping is successful, then it can be inferred that there is no connectivity problem between the source and target. And if a ping by name is successful, then an association between the named network device and its address can also be inferred. Since these are the justified inferences that an expert would make, the objective is to mirror such inferences during runtime, and compare student inferences against them to ultimately assess their troubleshooting decisions. This is illustrated with an abstracted example of the intended flow for diagnostic exercises in ITADS, shown in Figure 2 below.

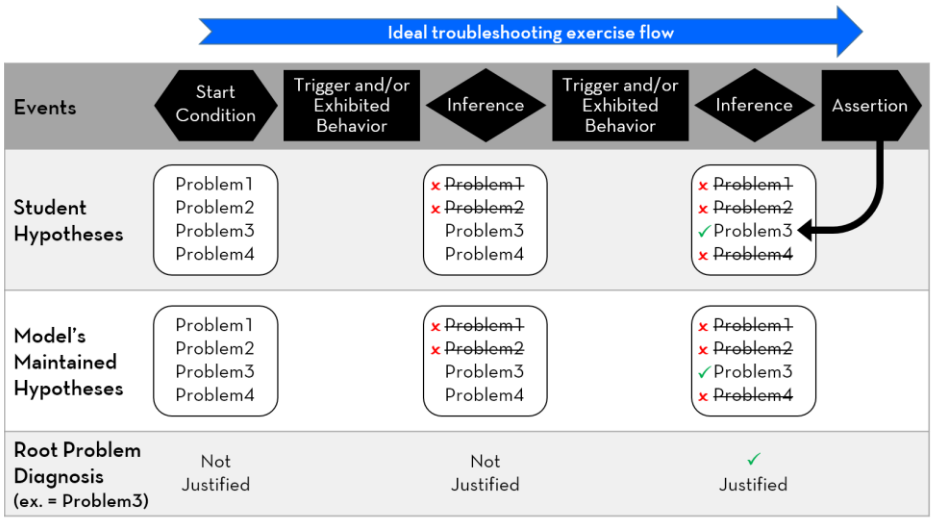
Ideal troubleshooting exercise flow
Events
Start Condition
Trigger and/or Exhibited Behavior
Inference
Trigger and/or Exhibited Behavior
Inference
Assertion
Student Hypotheses
Problem1 Problem2 Problem3 Problem4
~~Problem1~~ ~~Problem2~~ Problem3 Problem4
~~Problem1~~ ~~Problem2~~ ✓Problem3 ~~Problem4~~
Model's Maintained Hypotheses
Problem1 Problem2 Problem3 Problem4
~~Problem1~~ ~~Problem2~~ Problem3 Problem4
~~Problem1~~ ~~Problem2~~ ✓Problem3 ~~Problem4~~
Root Problem Diagnosis (ex. = Problem3)
Not Justified
Not Justified
✓ Justified

Figure 2. Flow within Troubleshooting Exercises

With a given start condition established by a symptom or trouble report, both the student and the assessment model have respective sets of active diagnostic hypotheses, in this case expressed as possible IT problems. Any new diagnostic action and/or its result (i.e., Trigger and/or Exhibited Behavior) yields the possibility of new inferences that may reduce the hypothesis space, and the assessment model automatically propagates its own inferences. As students simultaneously make inferences in parallel which may or may not be correct, they are requested to express when they believe hypotheses can be eliminated or asserted. This allows the assessment model to identify differences between its own inferences and the student's expressed inferences. The loop of actions, results, and inferences continues until the model determines that a diagnostic assertion of the root problem is justified.

Thus a high level of simulation awareness is required for the comparisons with the model's hypotheses, which implies that an effective assessment model requires knowledge of Triggers and Exhibited Behavior as they happen. However, it can be prohibitively expensive to develop exhaustive simulation instrumentation and assessment model logic for all possible Triggers and Exhibited Behavior in a free-play simulation. But for any specific action-result pair, advance knowledge of the scenario provides the ability to predict the valid inferences from either. This means that Triggers and Exhibited Behavior do not both need to be modeled as a pair; as long as at least one is modeled, the choice of one or the other can be based on instrumentation cost. This is illustrated in the following sections, with an introduction to the working elements of the assessment model and examples of its functionality in runtime.

ASSESSMENT MODEL

Table 2 below outlines the major functional object types utilized in the assessment model.

Table 2. Major Functional Object Types

Object Type	Description	IT Troubleshooting Example
Activities	Actions or events that can be collected from a simulation	Entry of a ping command
Problems	Diagnostic conditions to be identified in a troubleshooting domain, which may be the causes of a certain symptom	A disconnected network cable on a workstation
Findings	Pieces of information that may be revealed while troubleshooting, without directly supporting or refuting a Problem diagnosis	Network address of a workstation

Assessment Model Objects: Activities

Activity objects correspond to individual evidentiary events that can be collected from a simulation. An Activity can be evidence of a Trigger (Direct Student Action or Indirect Event) or Exhibited Behavior (Results and Events).

In a thoroughly instrumented simulation, a single student action in the interface could produce several distinct Activities, including an event for the original student input action, a separate event for its result, and potentially other forms of data about the same event if multiple collection methods are used. Each implemented Activity object involves two components: the collection mechanism that records when the associated action or event occurs in the simulation, and the model logic that represents the inferences (if any) that apply when the Activity occurs.

One of the aims of the assessment model design is specifically to reduce the space of Activity objects that must be collected and modeled. In the virtual machines used to simulate the IT environment for ITADS, some Activities present unique challenges, but the dominant cost concern is for the overall instrumented quantity to be manageable.

Assessment Model Objects: Problems

Problems are faults that generally require fixes, and they correspond to the diagnostic hypotheses that can be inferred in a troubleshooting domain (Justified Inferences). For more general purpose applications, the Problem object type could be broadened to represent any conditions that must be identified or factored into decisions. However, in the IT troubleshooting domain where the assessment model has been initially applied, Problems are specifically faults in an IT network environment. The Problem objects in the model serve several purposes:

- **Diagnostic hypotheses**. The Problem objects provide a common semantic basis for designating diagnostic hypotheses that can be either confirmed or ruled out, enabling the comparison between student inferences and model inferences.
- **Ground truth**. The assessment model has advance knowledge of the ground truth of the actual root Problem occurring in a scenario. This is used in the assessment of student inferences (for example, if the student attempts to rule out a Problem object that is actually still valid, or even the root Problem in the scenario). More significantly, knowledge of the ground truth facilitates the predictive capability to make inferences with incomplete simulation instrumentation, from either Triggers or Exhibited Behavior alone. In concept there could be more than one ground truth Problem in a scenario, but in the ITADS implementation the virtual environment is always configured in each scenario to contain exactly one fault.
- **Learning objectives**. The collection of Problem objects also directly plays a role in the organization of learning objectives for the student model and instructional decisions. In a tutor constructed to train troubleshooting skills, competencies relating to making correct diagnostic inferences are structured and expressed in terms of the Problem objects. The meaning of a Problem object construed as learning objective encompasses the range of decisions relating to that Problem, which the student demonstrates in the process of converging on a diagnosis in a scenario.

Assessment Model Objects: Findings

Findings are intermediate forms of evidence that do not directly relate to Problems, but provide information for future diagnostic or procedural actions. For example, when a student performs an action to obtain a network address for a workstation, the resulting revealed evidence is a Finding. In the model, the student's action is identified as an Activity, which contains a data structure specifying the Finding that is revealed by this instance of the Activity. On its own, that may not reduce the set of possible Problems, but combined with other information it may help to reach a diagnosis. Using the ping example, consider three sequences where a student at machine SOURCE01 is trying to determine connectivity to machine TARGET01 at network address 192.168.1.22. Suppose that TARGET01 has no connectivity problem, so the ping command succeeds in each sequence, shown in Table 3 below.

Table 3. Comparison of Example Sequences for Applying Findings

	Action	Result	Justified Inference
Sequence 1	1. "ping TARGET01"	Ping succeeds	TARGET01 has connectivity
Sequence 2	1. Check network map	Finding: TARGET01 is at 192.168.1.22	
	2. "ping 192.168.1.22"	Ping succeeds	192.168.1.22 has connectivity TARGET01 has connectivity
Sequence 3	1. "ping 192.168.1.22"	Ping succeeds	192.168.1.22 has connectivity
	2. Check network map	Finding: TARGET01 is at 192.168.1.22	TARGET01 has connectivity

The Finding about the network address for TARGET01 is only intermediate evidence for the inferences about connectivity. In Sequence 1, the Finding isn't used at all, because the action determines TARGET01 connectivity directly by name without using the address. In Sequences 2 and 3, the ping command is executed with the network address, so the connectivity inference depends on when the Finding is revealed, establishing the association between TARGET01 and its address. Whether the Finding comes before the diagnostic ping action (Sequence 2) or after (Sequence 3), the inference about TARGET01 only becomes justified when the ping result is combined with the Finding. In terms of diagnostic actions, clearly Sequence 1 seems like the preferable choice, closest to what an expert would do. But the assessment model must make appropriate justified inferences while tracking any plausible sequence that a student might follow, since the virtual machines offer a free-play environment.

Findings are not collected from a simulation, but are artifacts in the assessment model reflecting knowledge that the student can gain from the simulation. The values of Findings defined with a scenario are included with the model's advance knowledge about an exercise, along with the ground truth Problem in a scenario. The assessment model tracks when named Findings are revealed during runtime, which makes them a key conduit for more complex linkages between Problem inferences and multiple combined Activity objects, as in the ping example above.

Assessment Model in Runtime

A student completes the diagnosis portion of a troubleshooting scenario by correctly asserting the ground truth Problem after the assertion is considered *justified* and *complete* in the runtime model:

- A Problem assertion is *justified* when sufficient Activities have been performed to eliminate all competing Problems other than the ground truth Problem (in single-fault scenarios). This can be accomplished as a result of series of inferences that either rule out Problems until only one is left, or directly prove a singular Problem to the exclusion of any competing hypotheses.
- A Problem assertion is *complete* when any Findings required by the Problem have been revealed. The concept of completeness arises because of conditions where a Problem can be the only remaining hypothesis by process of elimination, but no direct evidence has been explored about the Problem itself.

Activity objects in the model are linked to sets of associated Problems, where each set represents a group of hypotheses that will be shown to be possible or impossible when the Activity occurs in the scenario. When the runtime model receives an incoming Activity notification, the inferential results are a function of the Activity's Problem sets, the root Problem, and the runtime model's active Problem hypotheses, as shown in Figure 3.

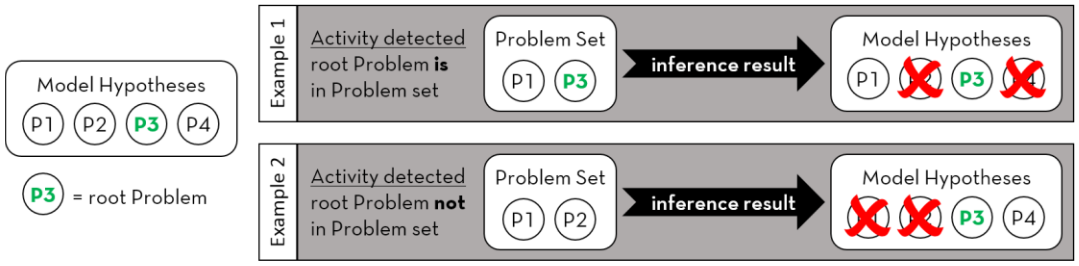

Figure 3. Examples of Inferences Based on Incoming Activity Objects and their Problem Sets

In Figure 3 above, Example 1 shows a case where the incoming Activity object's Problem set contains the root Problem. When the Activity occurs, this means that evidence is given that the root Problem and all other Problem hypotheses in the set are possible. In a single-fault scenario this also means that all other model hypotheses can be eliminated, so an intersection operation reduces the runtime model's Problem hypotheses accordingly. In Example 2, the root Problem is not contained in the Activity object's Problem set. This means that the Activity produces evidence that all Problems in the set are impossible, so they can be ruled out of the model's Problem hypotheses.

At the beginning of an exercise, troubleshooting notionally starts with a blank canvas where any hypothesis in the model could be the root Problem. But in runtime the assessment model immediately reduces its hypothesis space based on the symptom information represented in the exercise scenario. Students are not aware of the internal state changes in the model’s sets of active Problem hypotheses and Findings during a scenario. In fact, a student may fail to make the same inferences from the initial reported symptom, and may start a scenario with a different or larger set of Problems in mind than those in the runtime model. Distractors may also be presented to the student as possible hypotheses, though they were never considered relevant in the model at all. These are all intended to produce training challenges where students must think about the problem space and demonstrate their own inferences.

Students express hypotheses in a tabular user interface where their operations on Problem objects indicate inferential states: inactive (never added to the list), active (present in the list, but not asserted or ruled out), ruled out from the list, or asserted. During a scenario, any student elimination or assertion of a Problem hypothesis is only allowed when the model has determined that it is a justified inference based on Activities performed. An unjustified inference triggers generic feedback about justification, which is independent of whether the inference is actually correct in terms of ground truth. This is partly to prevent random assertions of what might be the correct Problem before taking diagnostic actions. The diagnostic portion of the exercise can only be concluded after sufficient diagnostic Activities have been performed to support a justified and complete assertion of the root Problem.

ASSESSMENT WITH LIMITED SIMULATION INSTRUMENTATION

The concept applied in the assessment model is that while the full set of cues encountered by students includes both Triggers and Exhibited Behavior, they do not both need to be modeled as a pair. In terms of data collection, the Activity objects are the mechanism for conveying both kinds of runtime simulation information to the assessment model, which then produces Justified Inferences referencing Problem objects. However, each Activity requires simulation instrumentation, and collectively they pose a major development task.

Different simulations provide different facilities for instrumentation to collect actions and events, and sometimes there are multiple avenues for collection such as different data protocols and object models. For example, the virtual machines used in ITADS to simulate a network of computers can be instrumented in a variety of ways from low-level atomic actions (e.g., mouseclicks and keystrokes), to application and window polling, to operating system events. A collection approach focusing on low-level atomic actions produces large numbers of reports, and hence threatens information overload; each such report imposes a processing burden to extract meaningful events from independent actions that are themselves meaningless without context. Returning to the example of the ping command, suppose that a processing mechanism is constructed to detect a student ping command assembled from atomic actions. This requires collecting the clicks and keystrokes needed to open the command prompt, the series of character keystrokes (with possible revisions from backspaces, interjections from clicking in other windows, etc.) to compose a command like “ping 192.168.1.1” and finally detecting the <enter> keypress to execute. Such a mechanism would collect this series of atomic actions from the user input stream, and aggregate them into meaningful higher-level student actions. But this still only accounts for the Activities that are Direct Student Actions. In order to capture Exhibited Behavior for the ping results, a completely different collection mechanism would be required: command line text output relayed to an assessment component and parsed to determine what was revealed to the student as a result of these actions. In that sense, for this domain, the low-level granular actions alone have limited value for building the kind of simulation awareness that the assessment model seeks.

A contrasting approach focusing on more targeted high-level event-based collection rules presents a more general strategy because it can be used to capture either / both Direct Student Actions or Exhibited Behavior from the simulation. This approach aims to detect specific pre-defined events in the virtual environment. In the IT domain, this involves developing rules that extract operating system events from the virtual machines, which requires knowledge of the virtual environment’s event models to detect and extract events into a parameterized format that can be consumed by an assessment engine. Returning to the example of the ping command, a low-level collection approach would aggregate and detect a command composed in a series of atomic keystrokes. In contrast, the more high-level collection approach interfaces with the operating system to detect the event of a command being issued, packaged together with the textual content of the command. Although a higher-level approach simplifies the collection by requiring less aggregation, each rule still requires custom development. This led to the observation that it is often redundant to develop rules for both actions and results when the assessment model has advance knowledge of the root Problem and Finding values in the Scenario.

Explicitly Modeling Triggers with Implicit Exhibited Behavior

While a complete model of the cognitive elements of troubleshooting includes the full path from actions to results to inferences from the decision-maker's perspective, an assessment model can be constructed with direct linkages from actions to inferences if the results can be deterministically predicted (Figure 4). This is how knowledge of the root Problem and Finding values in the scenario is used with the runtime inferences from Activities.

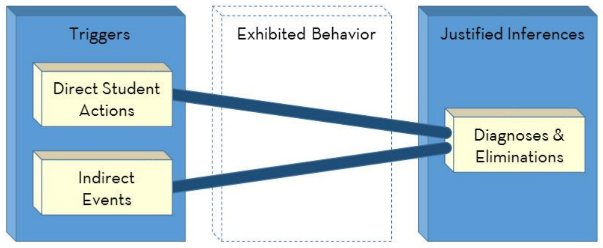

Figure 4. Implicit Exhibited Behavior

Consider the ping command once more. The Direct Student Action of entering the ping command produces Exhibited Behavior (command line output text), which must be interpreted to determine if this yields any Justified Inferences. Both the input and output would be implemented with Activity objects collecting separate information from the simulation. However, since the runtime model has knowledge of the root Problem, the input action alone is sufficient to predict what inferences will be justified, without needing to collect or even model the resulting output as a separate Activity. There are several possible results from a ping command, which reveal different information:

- Reply from <destination address> (4 times, with additional information)
- Request timed out (4 times)
- No reply from <destination address>
- <destination address> is unreachable

However, it isn't necessary to model these four different results by defining four additional Activities reflecting Exhibited Behavior in the form of the output command line text. The Activity object contains sets of Problems that establish the direct linkage to Justified Inferences without tracking the result or even modeling the possible contents of the results. Focusing on one Problem set as an example, consider an Activity created to model the ping command performed at SOURCE with destination TARGET, defined with a Problem set containing the following information.

Prerequisite: TARGET is the workstation with the reported symptom

$$\text{Problem Set: } \left\{ \begin{array}{l} \text{local connectivity problems at TARGET} \\ \text{local connectivity problems at SOURCE} \\ \text{network connectivity problems} \end{array} \right\}$$

If the prerequisite condition is met, then the Problem set is evaluated. To illustrate how a single Problem set is processed, instead of stepping through the four kinds of ping results, the following two cases reduce the possible outcomes to just ping success vs. failure, in the context of an Activity object containing only this one Problem set.

- **Case 1:** the ping fails. This means that one of the Problems must exist, and therefore the root Problem in the scenario must be contained in this Problem set. The inference outcome (in a single fault scenario) is that all other active hypotheses can be eliminated, but these three Problems remain.
- **Case 2:** the ping succeeds. This means that none of the connectivity Problems in this set could exist, since that would directly contradict the fact of the successful result. In this case, the inference operation will find that the root Problem is not contained in this Problem set. The inference outcome is that the entire Problem set can be ruled out in the runtime model.

Essentially, the inference mechanism in the assessment model is making use of the deterministic behavior of the simulation to apply the deduction that if a Trigger occurs, then something will be revealed to the operator that will support certain inferences. The inference information is represented in the Trigger Activity, but the content of what will be revealed is not represented. As long as the simulation is deterministic, it is not necessary to additionally collect Activities to confirm *what* is revealed to the operator in the form of Exhibited Behavior.

Out of 73 Activities implemented with the ITADS application, 21 model only the Trigger side of an action-result pair. In many of these cases, a single Trigger could produce several different kinds of Exhibited Behavior, which means that the implementation ratio would be greater than one-to-one. If Activities were constructed to model both Triggers and Exhibited Behavior, then the resulting total number of Activities for these 21 cases would more than double. This proportional increase is especially evident in the cases of operator actions relating to command line inputs, which often tend to have multiple possible kinds of output results. As noted earlier, it's not necessarily difficult to build additional Activity objects for multiple different command line text output events. The prohibitive factor is from the aggregate cost of instrumenting a simulation to collect both elements for all cues that are relevant to student inferences. Thus, the approach described here presents an opportunity for savings.

Explicitly Modeling Exhibited Behavior with Implicit Triggers

A similar simplification can be made in the reverse, when it may be preferable to collect and model Activities for results in the form of Exhibited Behavior, and omit the Activities relating to Triggers (Figure 5). One simple case of this in the IT troubleshooting domain involves the diagnostic action of opening a folder. An operator might perform this action as a quick check for permissions problems, to see if a file is present, or to see if a network folder is accessible.

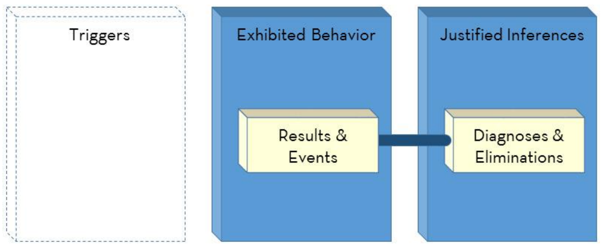

Figure 5. Implicit Triggers

There may be many ways to open a folder in the virtual operating system environment, but in this case the matter of how this result occurred, from what precipitating action (e.g., double-clicking on the folder from a parent folder), is of little interest for diagnostic purposes. The significant inferences come from the knowledge that the operator saw the opened folder. So regardless of the precipitating actions from the student, the simulation event of the folder being opened is a natural choice to model as an Activity reflecting Exhibited Behavior. This result then establishes a direct connection to Justified Inferences, leaving the chain of precipitating actions implicit.

Of the 73 total Activities in the ITADS implementation, 52 Activities model only the Exhibited Behavior side of an action-result pair. In the same fashion as the 21 Activities implemented for the Trigger side, many of these cases could have been initiated by one of several possible Triggers. Often the Exhibited Behavior is an operating system state change or event that could have been brought about in numerous ways in the user interface. For example, there are many Direct Student Actions that could lead to the Exhibited Behavior of a folder being opened. So if the assessment model were structured in a way that required Activity objects modeling both sides of any operator action-result pair, then again the resulting total number of Activities for these 52 cases would more than double.

CONCLUSIONS

The assessment model was tested with the broader operational ITADS tutoring system as part of a training effectiveness validation study where ten students ran twenty-two exercise scenarios over a ten-day period. The model worked correctly in all scenario runs and accurately tracked justified inferences throughout all students' different troubleshooting paths. For example, no condition occurred where a student's actions should have justified an inference that was still deemed unjustified by the model. So from a training utility perspective, the gains in reduced instrumentation cost were effectively realized while meeting the training goals for automated performance assessment. In the spectrum of inferential reasoning tasks, IT troubleshooting probably lies somewhere between abstract reasoning activities like the Wason selection task, and activities that have a more intuitive component like enforcing social rules. As IT practitioners gain expertise, they most likely develop a familiarity that makes troubleshooting decisions more intuitive and less abstract. However, the assessment model does not rely on any particular level of abstractness in the inferential reasoning activity, so ultimately the nature of the context or the level of an individual's expertise and intuition are not factors in the applicability. The application conditions where this assessment modeling approach is best suited include:

- Troubleshooting or complex decision-making where actions and results can be deterministically linked, and where, therefore, the simplification of explicitly modeling only one of the two will suffice. The greatest utility most likely comes in applications where actions can have different results in different conditions, and where results may have been precipitated by any of several different actions.

- Situated training applications using independent simulations where the assessment model has no direct knowledge of simulation events and cues other than through instrumentation and data collection.
- Unstructured problem-solving in free-play scenarios where multiple paths to a solution can be appropriate, and where it is not practical for an assessment model to enforce coarse predefined expert solutions.

There are also potential limitations in the strategy of using this assessment modeling approach to reduce simulation instrumentation requirements. First, when a practical choice is made to model Exhibited Behavior explicitly and leave Triggers implicit, a disadvantage can be the lack of insight into how or why a student took a certain action. While the primary need for simulation awareness may be satisfied in terms of relating cues to Justified Inferences, the omitted Trigger information (such as Direct Student Actions) may be desired for feedback purposes in some applications. In the IT troubleshooting domain, there is little need for this distinction. However, other domains may have conditions where modeling the distinction between actions and results is more useful.

A second limitation is the single fault assumption in the initial implementation of the assessment model. In order to eliminate this assumption for conditions where a scenario may contain more than one fault, most of the assessment model's runtime mechanics remain the same, except for one inference operation. When a student finds definitive evidence confirming a particular Problem, this would not justify eliminating competing hypotheses. Therefore in scenarios where multiple faults are possible, all Problems must be either directly confirmed or directly refuted.

Finally, the embodiment of inference information in the assessment model's Activity objects implies that a certain logical thinking style and close attention to detail is necessary in the authoring process, which can be a challenge in any development undertaking. But ultimately this also amounts to another reason it is valuable to minimize the quantity of Activity objects that must be implemented to model the domain, to reduce the burden on authors managing model elements and their interactions.

The model-driven structure of the assessment approach helps to minimize the amount of scenario-specific authoring required to create tutored exercises. By creating a model that represents the major components and linkages required for decision-making assessment, the variations from one scenario to another essentially amount to different paths through the same modeled space. Scenario authoring does not require developing new model artifacts and inference logic, but rather only requires specifying the attributes that shape the path – the root Problem, symptom, and scenario-specific Finding values. An authoring tool supports the creation of new scenarios in this manner by subject matter experts (SMEs), as well as the development and extension of the core assessment model elements (Activity objects, Problem objects, and Finding types) by more technically oriented scripters. For example, with the creation of a new Activity object, the mechanics of specifying the logical details—using authoring interface elements like tables, menus, and picklists to enter Problem sets and other content—is not particularly complex. However, the task of specifying the abstract relationships between new and existing model objects can require scripter-level logic that some SMEs may find challenging. Initial feedback from instructors and SMEs has been that one of the most likely common methods for developing new scenarios will be creating variations of existing examples that use the assessment model.

In the ITADS application, the total of 73 Activity objects in the assessment model is obviously highly preferable to an alternative of 200 or more which would have been required to model both sides of every action-result pair. Given that the ratios between actions and results are often several-to-one, the assessment model does not require the author to uniformly instrument one or the other, but rather to make the optimal choice on a case-by-case basis. The choice may be a single action with several possible results, or a single result that may be precipitated by several possible actions. The reductions in the net total are valuable not only for the implementation phase, but also for testing, tracking, and verifying the inferences made by the model, using real scenarios that students will encounter.

ACKNOWLEDGEMENTS

The concepts described in this paper were developed in a collaboration between Stottler Henke Associates, Inc. and ComTech Telecommunications Corp., under the ITADS research effort which was contracted and administered by the Naval Air Warfare Center Training Systems Division, Orlando, FL for the Center for Information Dominance, Pensacola, FL. The authors are grateful to the entire team of sponsors and contractors for supporting this effort.

REFERENCES

Hall, E. P., Gott, S. P., & Pokorny, R. A. (1995) *A procedural guide to cognitive task analysis: The PARI methodology*. Brooks Air Force Base. TX: Air Force Human Resources Laboratory.

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

Wenger E. (1987), Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge. Los Altos, CA: Morgan Kaufmann Publishers.