# Design Recommendations to Support Automated Explanation and Tutoring

*Dave Gomboc*
*Steve Solomon*
*Mark G. Core*
*H. Chad Lane*
*Michael van Lent*

Institute for Creative Technologies
University of Southern California
13274 Fiji Way
Marina del Rey, CA, USA 90292

Keywords:
Automated Explanation, Automated Tutoring, Behavior, Computer-Generated Forces, Simulation

**ABSTRACT:** *The after-action review is an essential component of military training exercises. The use of constructive simulations for training poses a challenge when conducting such reviews, because behavior models are typically designed to simulate satisfactorially, without explicit concern for the interrogation of synthetic entities afterward. Ideally, users could obtain knowledge about not only the choices made by a simulator's behavior models, but also the rationale for those choices. This requires a rich representation of behavioral knowledge within the software system.*

*We have integrated our explainable AI system with behavior models and log information from two simulation systems. Selecting examples from these simulators, we identify areas for improvement to facilitate the automation of explanation and tutoring.*

## 1 Introduction

The after-action review ("AAR") is an essential part of any military training exercise, be it live training or training via simulation. Reviews of exercises conducted by way of constructive simulation[1] pose a particular challenge. Specifically, the introduction of synthetic entities, often called computer-generated forces ("CGFs"), controlled by a simulator's artificial intelligence ("AI") system, impedes the process of understanding what happened, why it happened, and how trainees could do better. U.S. Army Field Manual 25-101 on "Battle Focused Training" states:

*"The OPFOR[2] can provide valuable feedback on the training based on observations from their perspectives. …the OPFOR can provide healthy insights on:*

- *OPFOR doctrine and plans*
- *The unit's actions.*
- *OPFOR reactions to what the unit did."*

What if the OPFOR are either fully or in part computer-generated? Current best practice for gathering their perspectives, feedback, and insights is limited to an analysis of the simulation log files by a technical expert. Similarly, when validating CGF behaviors, it can be difficult for a subject matter expert ("SME") to fully comprehend and judge those behaviors based only upon examining logs and observing CGFs in a plan-view display.[3]

A fully-capable explanation system is a valuable tool not only for AARs and CGF validation, but also for debugging, causality analysis, and automated tutoring. To support this wide range of applications, explanations are required that go beyond just presenting the available logged information. The generation of more satisfying explanation requires additional information be logged, plus access to the underlying behavior models that drive CGF behavior.

With such extended logs, an AI system can answer who-, what-, where-, and when-style questions. The behavior models are necessary to answer the how- and why-style questions that are sometimes the most useful. Natural language generation is a valuable component of our explainable AI ("XAI") system, for it allows the system to answer questions from trainees, instructors, and SMEs in

[1] Constructive simulations are those that involve simulated people operating simulated systems.

[2] OPFOR stands for the opposing forces.

[3] A plan view is a two-dimensional overhead view.

easily understandable English, and supports a dialogue-style interaction with the user.

The XAI system described herein is able to interface with two simulators: Full Spectrum Command ("FSC"), and the OneSAF Objective System[4] ("OOS"). Additionally, an earlier XAI implementation of FSC was done at our institution under our project leader (van Lent et al., 2004). Our experiences in developing these systems has provided us with insights into the benefits of designing simulation environments and AI systems with explanation in mind, rather than attempting to add explanation capabilities after the system has already been built.

In particular, we recommend that simulators intended for use in explanation and tutoring contexts provide external visibility of the logic that underlies all behaviors, and that this be provided in a declarative form that specifies the goals of synthetic entities, and the preconditions and end effects of their actions. Detailed reporting of the internal state of the simulator throughout execution is also necessary. Furthermore, this reporting should extend to user interactions with the simulator in tutoring contexts.

We begin by describing our current XAI system. We then discuss the necessary extensions made to the behavior representations in OOS to support explanation. Section four describes the effects of simulation logging code on explanation. We then discuss additional issues introduced by the use of XAI for tutoring, and conclude by summarizing our recommendations.

## 2 XAI Proof-of-Concept System Overview

Our XAI system, depicted in Figure 1, requires three kinds of information from the simulator. Firstly, the definitions of all behaviors that CGFs may perform must be available. "Patrol Area", "Ambush", and "Clear Building" are three examples of such tasks from FSC.

The second kind of required information describes the current scenario. This category includes the initial state of the simulated world, the rules of engagement, the types and amounts of personnel and equipment that will be employed, and the initial assignments of CGFs to tasks by the simulation user.

The third kind is the dynamic state of the simulated world. This includes, for example, the location and status of all synthetic agents, manipulable objects, and the dynamic state of the world environment. Items such as the location of soldiers may change extremely frequently and do not necessarily follow easily-formulated trajectories, so instead of having perfect information at our disposal, we require only that it be sampled at regular intervals.

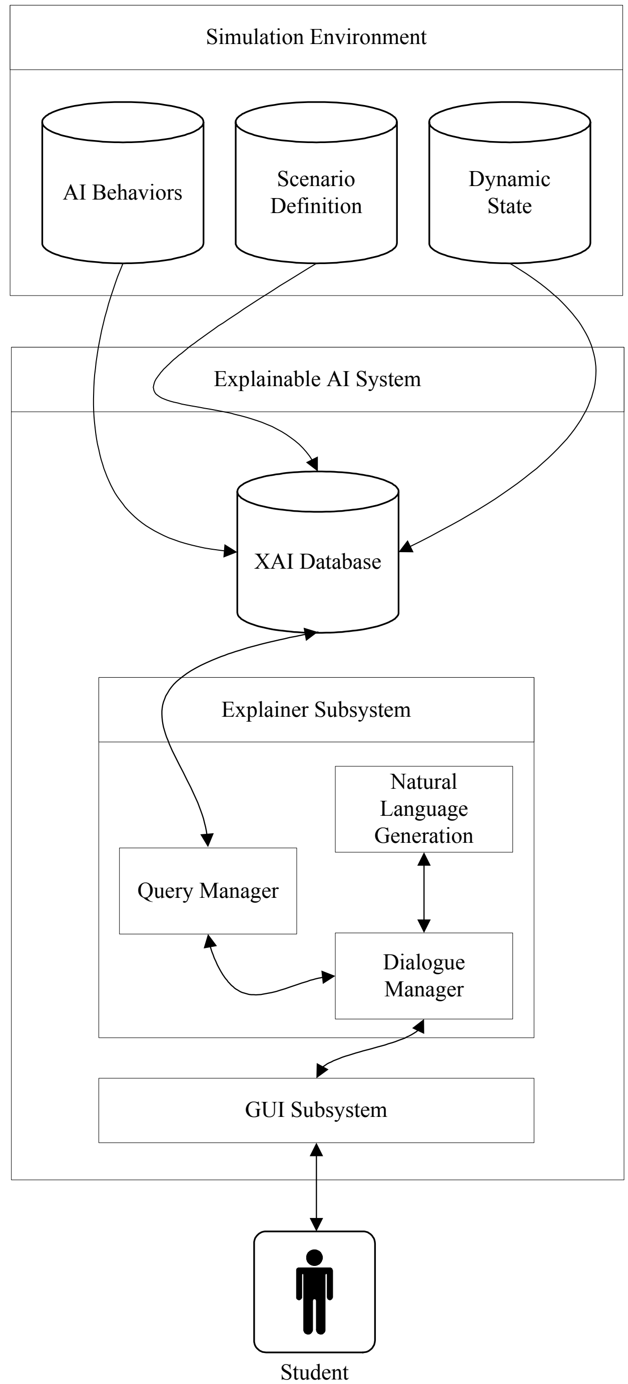


**Figure 1: XAI System Architecture Diagram**

The simulator log often does not provide all the required information. We extracted what we could from the log into the XAI system's relational database, and in certain cases, manually augmented that data.

[4] We used the Block B release of OOS, which was the latest available version at the time.

At program start, the AAR arbitrarily begins with the first CGF. When beginning to speak with a CGF, the 'interesting' time point for that CGF are identified. For example, our OOS version deems all task start, mid-point, and completion times as interesting, as well as all times when the entity fired a weapon. The time point menu on our graphical user interface ("GUI") makes these points of interest available to the user.

The dialogue manager component generates a menu of questions based upon the CGF being interviewed and the current time point of interest. Some questions concern the state of entities ("What is your health?"), while others deal with task information ("What is your unit's task?"; "How do you execute your task?"). Most questions may be asked at any time, but certain questions are context-dependent. For instance, one may ask "What are you shooting at?" only when speaking to an entity about a moment at which it fired its weapon.

The database is queried on demand as the user interacts with the application; the natural language generator component uses domain-specific XSLT templates to transform query results into English responses. The dialogue manager records prior questions, answers, and other state information about the conversations that take place, enabling context-sensitive responses that reduce the frequency of repetitious dialogue. Together, the query manager, dialogue manager, and natural language generator constitute the explainer subsystem.

The dialogue manager sends all required information via XML to the stateless GUI servlet, which in turn produces HTML that is rendered by the user's web browser. Figure 3, on the following page, is a screenshot of our HTML-based XAI proof-of-concept GUI. The top frames contain header and simulator information. The session's dialogue history is available mid-screen; a vertical scroll bar appears when necessary so that the entire dialogue may be consulted at any point. The bottom frame provides menus by which the user may guide the dialogue. The servlet submits the user's input to the explainer, causing the system to prepare a new response, which begins the cycle anew.

## 3 Behavior Simulation

Behaviors in OOS are classified either as composite or as primitive. Composite behaviors are constructed by visually specifying procedural expressions that reference other behaviors; these are represented in XML. Primitive behaviors are directly coded in Java, and are considered to be atomic by the behavior composition system. A thin XML wrapping is provided for these, so that they may be referenced by OOS's visual behavior specification tool.

Primitive behaviors are always assigned to individual entities. Composite behaviors may also be assigned to individuals or to groups of entities, as appropriate. Figure 2 shows the composition hierarchy for the "Fire Team Clear Room" behavior, a task used by fire teams in the OOS scenario for our XAI proof-of-concept system. Regarding the primitive behaviors: "Entity Move on Route" is performed by every CGF of the fire team; "Throw Grenade and Wait for Detonation" is assigned to the grenade thrower of the clear room task; "IC[5] Enter and Clear Room" is also assigned to each CGF.

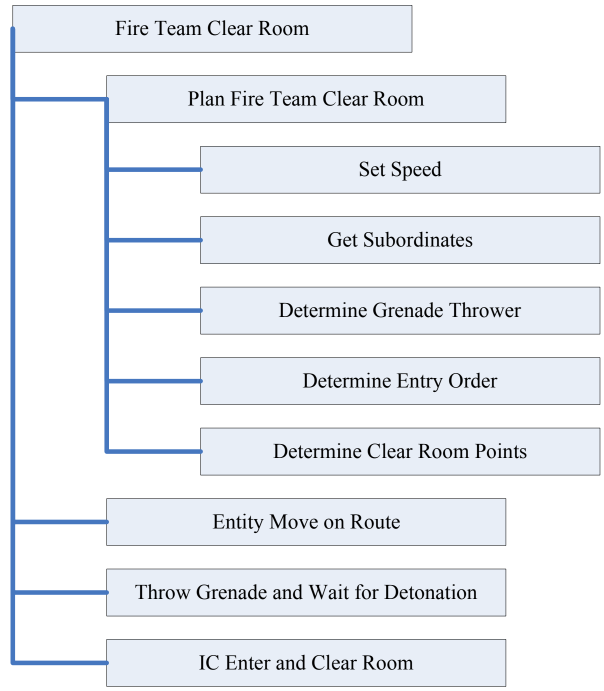


**Figure 2: OOS Behavior Hierarchy for Fire Team Clear Room**

The composite behavior "Plan Fire Team Clear Room" consists of several primitive behaviors. In contrast to non-planning behaviors, this behavior does not affect the simulation world. Rather, it binds variables within the scope of the main behavior that will be used in the task's further execution.

ICT has enjoyed access to early versions of OOS, an advantage in that we have visibility into the development process. This allows us to provide feedback to OOS developers on ways to improve the suitability of OOS for automated explanation of behaviors. It is also a disadvantage in that we must rely upon a behavior architecture that is not fully implemented. Here, we discuss three areas where we have encountered challenges in the explanation of behaviors with OOS Block B.

---

[5] IC stands for an individual combatant.

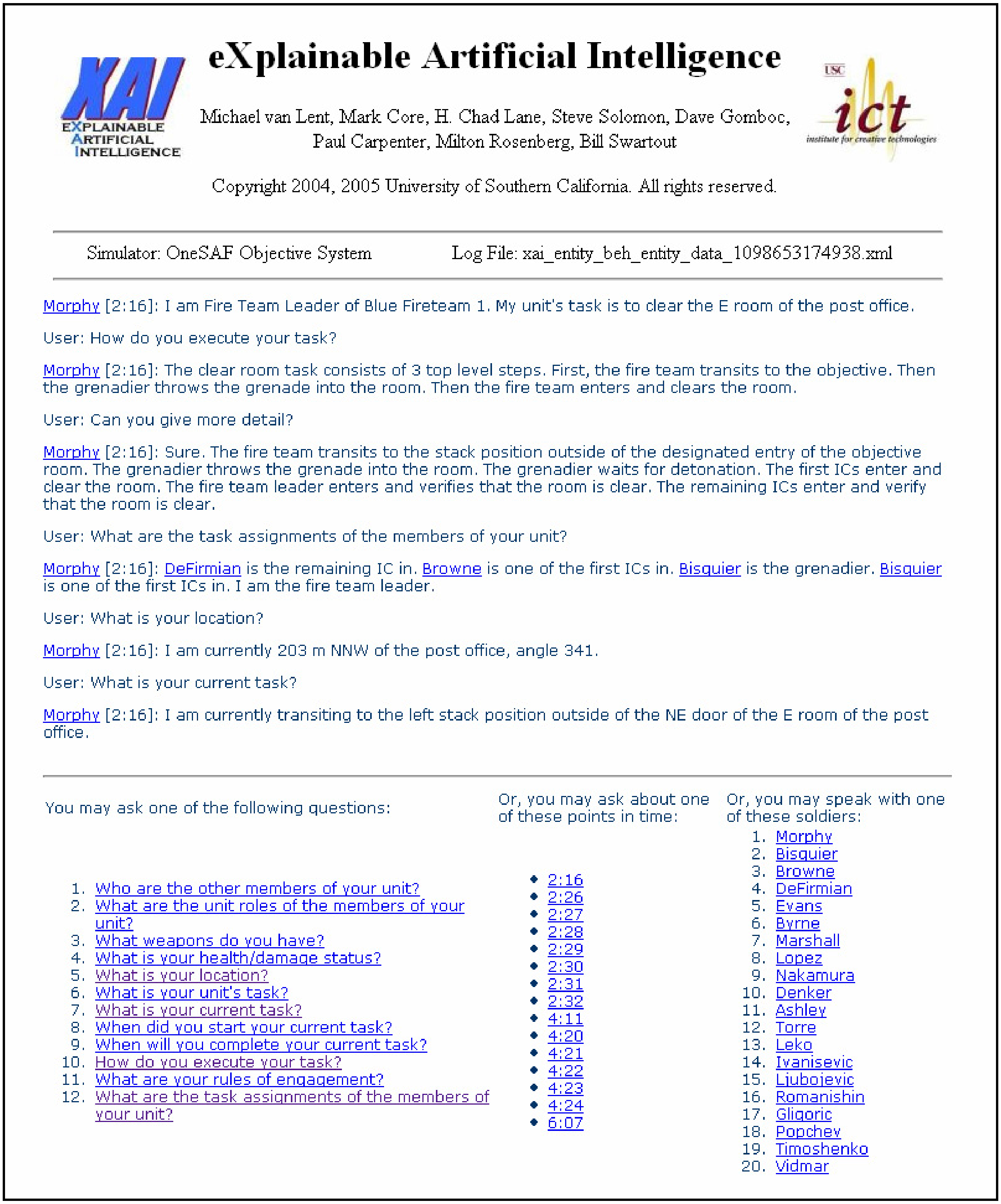


**Figure 3: XAI for OOS Proof-of-Concept Graphical User Interface**

### 3.1 Absence of Declarative Information

While the behavior generation system in OOS represents a significant advance over previous simulation architectures, it does not include every feature necessary for explainable AI. In OOS, like many other CGF architectures, behaviors are derived from the goals of the simulation user. These high-level goals are not directly encoded into the behavior model. Similarly, declarative preconditions are not fully specified, as required by explanation. Consequently, it is not possible to reason about the long-term motivations of an entity.

Furthermore, while alternative actions may be specified by using conditional expressions within the behavior composer, all alternatives must be completely foreseen by the behavior designer. This limits the ability of an XAI system to answer "what-if" questions; it would be preferable if the behavior model could explore unforeseen alternatives.

How can reasoning directly about preconditions, goals, and alternative actions that satisfy goals allow for more effective explanation? Consider the example of a fire team clearing a room. Once the fire team is positioned outside the room, the grenadier throws a grenade into the room to suppress enemy fire before the team enters the room. Currently, XAI cannot answer the question "Why did you throw a grenade into the room?" because there is no representation in the parent composite behavior of the goal or motivation for the grenade toss.

XAI also cannot reason about alternative behaviors that could satisfy the goal of suppressing enemy fire, because the parent behavior makes no options available. Generic questions about alternatives, e.g. "What else could you have done to suppress enemy fire?" are therefore not supported. Finally, it is also difficult for XAI to answer "Why not?" questions, e.g. "Why didn't you use non-direct fire?", because preconditions are implicit in the procedural representation of the behaviors, whereas XAI requires explicit representation in a declarative form.

### 3.2 Opacity of Primitive Behaviors

As previously mentioned, composite behaviors are provided in an XML format. This format is high-level, descriptive, and is easily understood. Consequently, our XAI system is able to answer questions such as "How do you perform your task?", when the question is asked of a composite task.

In contrast, primitive behaviors are developed in Java, and, at least in the Block B version, did not contain metadata. Consequently, our XAI system cannot know the preconditions and end effects of actions.

Currently, some of the most interesting actions for explanation, particularly primitive behaviors that perform planning behaviors, cannot be reasoned about by the XAI system. For instance, in the "Fire Team Clear Room" task, we would like to be able to answer the question "Why are Morphy and Bisguier the first ICs in?", but to answer this question, the XAI system would have to comprehend the Java implementation of the behavior.

### 3.3 Multiple Behavior Sources

In addition to the behavior execution engine of OOS that executes behaviors assigned to units in the task execution matrix, OOS also uses behavior agents that are not tied to specific orders. Rather, these agents are composite attributes of entities that are capable of performing reactive actions based upon the current state of the simulation world as perceived by the CGF.

For example, the behavior agent that controls an entity's weapon will automatically aim and then fire the weapon whenever the CGF holding the weapon perceives an enemy soldier, the rules of engagement permit firing, and the entity has ammunition. Thus, the entity's actions are not necessarily the consequence of a task that it is performing (other than the rules of engagement which may be specified as part of the task).

Behavior agents are implemented similarly to primitive behaviors: procedurally, and without hooks to show why specific actions are performed. We do not doubt there are valid software design considerations that suggested this division of handling behavior. We merely remark that the difficulty of explanation increases when not all sources of behavior reflect intent by the entity.

## 4 Simulator Record-keeping

In the previous section, we discussed the challenges of supporting explanation in the behavior representation and generation components of a simulation. However, it is not enough to represent the necessary information; it must also be made available to the XAI system. While this could be done at run-time given sufficient support for introspection, our focus is on the use of the XAI system as an AAR tool, so we frame this as an issue of information logging.

First, we list several deficiencies with logging that we identified while connecting simulators to our XAI system. Then, we contrast the information acquired from OOS and from FSC, and describe the different sets of questions our XAI system can answer as a consequence.

### 4.1 Logging Issues

Here we selectively discuss concrete logging issues, particularly in cases where an attempt has been made to store the information, but what has been stored is, for our purposes, incomplete.

One interesting facet of FSC is that its source code contains a considerable degree of debug information –

almost all of it disabled. Much of that information would have been valuable to an enhanced XAI system.

Also, recall that the original XAI implementation of FSC was done at our institution under our project leader: in this respect, we are learning from our own mistakes.

#### 4.1.1 Task Execution Matrix

The step-by-step plans of the forces under the control of the trainee are recorded in the task execution matrix. Such information is required to ask virtually any question relating to the planning of the operation. This information is of course modeled in FSC, but it is not logged. Therefore, we have had to go back and modify the source code to log this data.

#### 4.1.2 Event-Driven Architecture

During the AAR, we may wish to know the route that an entity took to a target, and how far along that route it was at an arbitrary simulation time point. However, FSC takes an event-driven approach to logging, rather than recording log information at fixed time intervals. Therefore, we usually cannot even know where an entity is, unless an event such as firing a weapon or a shift to a new task takes place at that point in time.

#### 4.1.3 Composite Behaviors

Block B of OOS does not log dynamic state information regarding composite behaviors that are being executed. Therefore, it is impossible to ascertain which composite behavior is executing at arbitrary simulation time points.

To answer questions such as "What is your unit's current task?", we manually recorded the active mission phases, plus their start and end times, then used this information in conjunction with the task execution matrix. However, this method is not correct when the scenario does not execute according to the original plan, for instance, if a unit is forced to withdraw.

#### 4.1.4 Variable Bindings

Block B of OOS does not store the variable bindings of behaviors.[6] For example, in the composite behavior "Fire Team Clear Room", the unit task roles include "Team Lead", "Grenade Thrower", "First ICs In", and "Remaining ICs In". These variables are bound at runtime by child planning behaviors, but are not logged, necessitating that we observe a simulation run and manually add this information.

[6] Newer versions of OOS do store this information. This may also be true of other items we have listed.

#### 4.1.5 Pathfinding

Neither FSC nor OOS log pathfinding details, whether for units or individual entities. Therefore, the XAI system cannot answer questions about the planned routes of travel. However, even if this information were logged, it is doubtful that the XAI system could provide compelling explanations. Variations of the A* search algorithm are typically used to compute paths; this algorithm is not similar to human methods of tackling the problem.

#### 4.1.6 Inscrutableness of Log Format

Figure 4 contains a small sampling of the text version of the XAI log emitted by FSC. During the corresponding 0.002 seconds of the program run, a task object was created, then updated four times. Conceptually, though, the task was merely being defined. Tracking the implementation too closely would lead to the misinterpretation that the CGF repeatedly changed its mind about what the task ought to encompass. However, once the simulation is underway, it might well be that a CGF would exhibit such indecision, so distinguishing between the two cases could be problematic.

```
3 63233968.000000 3 Assign Roles 4 0 0 -1 0 0 0 0
3 63234296.000000 3 Assign Roles 4 0 0 -1 0 0 0 0
3 63234652.000000 3 Assign Roles 4 0 0 -1 1 90052
0 0 1 50006
3 63235092.000000 3 Assign Roles 4 0 0 -1 2 90052
90053 0 0 1 50006
3 63235548.000000 3 Assign Roles 4 0 0 -1 3 90052
90053 90054 0 0 1 50006
3 63236052.000000 3 Assign Roles 4 0 0 -1 4 90052
90053 90054 90055 0 0 1 50006
```

**Figure 4: Sample from FSC XAI Text Log**

After attempting to extend the original XAI logging, we found it more useful to write additional, but separate code that logs the information we were interested in for our current XAI system. This newer log format uses XML so that the data description is implicit in the log itself.

### 4.2 Answerable Questions

Figure 5 provides a chart relating the information made available to the XAI system with the questions that it is therefore able to answer. The lack of exposure of underlying behavior reasoning is reflected in the total absence of why-style questions.

The information made available to XAI is not the same as the information modelled by the simulator. For instance, FSC models stance information, but as we chose not to log this, we were unable to answer the two posture questions. Also, we hand-augmented the OOS unit task information, as previously discussed.

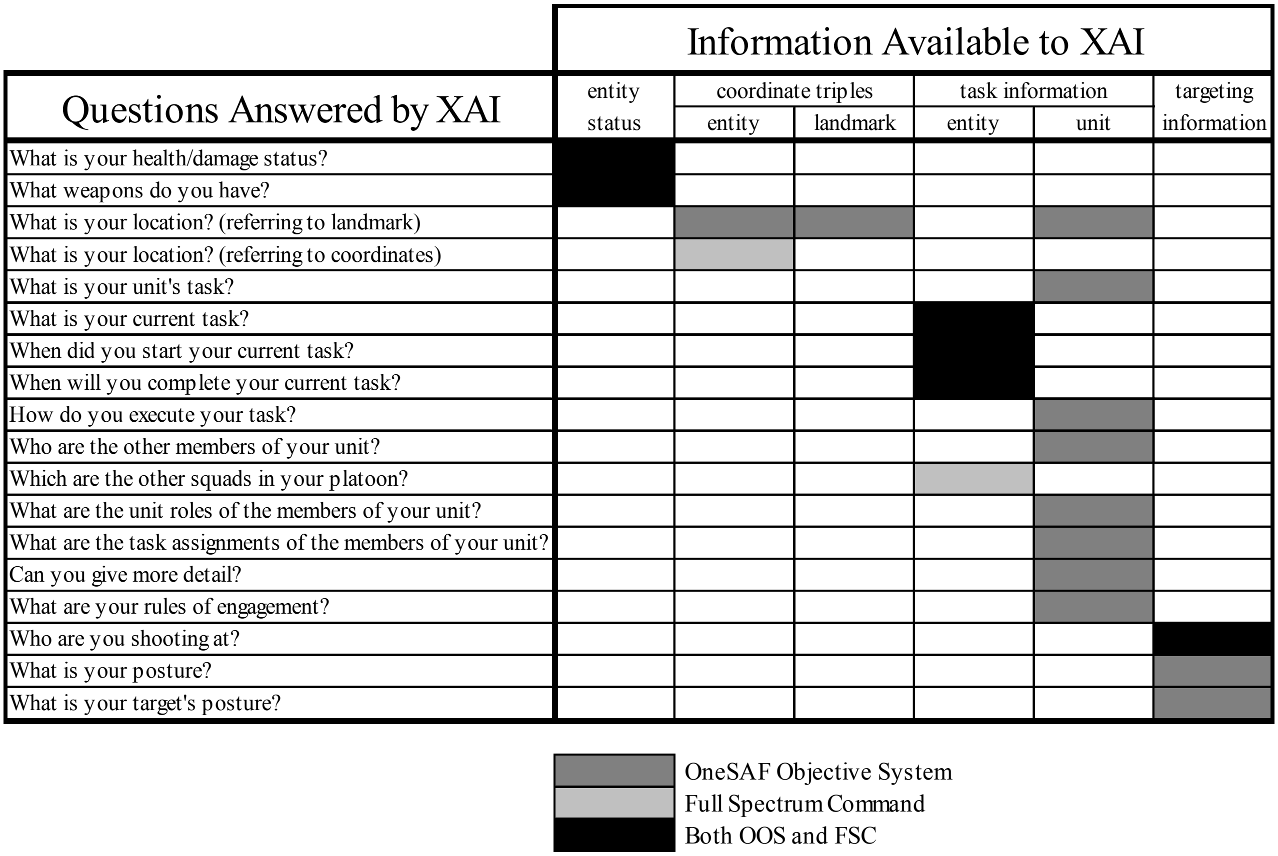


**Figure 5: Comparison of Information Flow Into and Out from the XAI System**

Many questions are available to the user when using simulation data from OOS, but not FSC, because OOS provides unit task information in a readily-usable form. In contrast, while FSC represents platoons and squads to the user, its internal code refers to task-oriented elements that typically do not coincide with U.S. Army units.

The differing treatment of the question "What is your location?" between XAI for OOS and XAI for FSC is also of interest. In the case of FSC, the response is rudimentary: the XAI system provides the (x, y, z) coordinate triple of the entity. In the case of OOS, information about landmarks and the task objective is available, so we can offer a more useful response that expresses the distance and angle of the soldier from their target, as shown in Figure 3.

## 5 Logging for Automated Tutoring

The presence of or desire for an automated (a.k.a. intelligent) tutor is a significant factor when deciding what information to record. Interestingly, many of the lessons learned in the construction of an explainable AI system also have implications for intelligent tutoring.

Intuitively, teachers should have the ability to explain material in a way that makes sense to the student. Additionally, good teachers do not quietly solve problems on the board, they *engage* their students. They pause to ask questions, connect their activities to principles, and elicit answers from students rather than just giving them away (Leinhardt and Schwartz, 1997). An automated tutor for a training simulator likely must also be able to similarly engage the student.

Behavior models consisting of only procedural information and logs containing only the raw events of a simulation are insufficient for intelligent tutoring: knowledge that is critical for a tutor to have is not present in such models. To produce "tutoring-friendly" logs, several suggestions derived from our work appear below. Before discussing them, however, it is important to note one important historical example of the general problem of retroactively augmenting an AI system for pedagogical purposes (Clancey, 1984).

### 5.1 Background

GUIDON, a tutoring system built on top of the medical diagnostic expert system MYCIN, attempts to teach students the rules in its knowledge base through limited dialogue. Using a record of the inferences made by the MYCIN reasoning engine, GUIDON walks the student through each rule application until a final diagnosis is reached. Although it was able interact effectively with students on individual problems, there was no representation of general skills involved with medical diagnosis. In addition, no higher level organization of the rules was made available by MYCIN. The knowledge base was essentially authored, debugged, and applied for the *sole* purpose of producing diagnoses. With no global organization on the set of rules, GUIDON was not able to help students synthesize or organize what appeared to them to be a large, jumbled body of knowledge (Clancey, 1984).

The key lesson from GUIDON was that if pedagogical goals are in the horizon for an AI system, its knowledge base should be built with these goals in mind from the outset. Not doing so risks overlooking important knowledge representation and pedagogical issues. When intent exists for a simulator to be used for pedagogical purposes, similar organizational measures ought to be taken when authoring its AI behaviors.

The tutoring component of our research is nascent, so we are not yet able to propose specific changes to behavior representations beyond those described above in support of explanation, but we do explore some additional considerations in a general manner below.

### 5.2 Recording Rationale

To provide an explanation of actions taken in a traditional expert system, it is necessary to record the chain of inferences that led to an action or conclusion. Performing an explanation from a log alone implies that not only should it include events, but also the "thinking" that occurred between events. Just as an explainer requires such inference chains, so does a tutor. In addition to selected operators and facts used during the inferences, novices often need explanations as to why certain operators did not apply, and reminding of what facts are true at certain times during a simulation to help the student understand why some inferences can or cannot be made. The suggestion for log files, then, is to record in great detail, behavior selections and the details of their application or non-application. Armed with such knowledge, a tutor would be able to give negative or positive feedback about student suggestions and answers, lessening any need to persistently run re-simulations in the background.

### 5.3 Alternative Outcomes and Negative Evidence

Tutorial decision making often involves events that *did not happen* during a problem solving episode (McArthur et al., 1990). In addition, representing events that provide *negative* evidence against some problem solving path or critical decision is also very valuable in teaching domain knowledge effectively (Suthers et al., 2001). Knowing how different decisions by a user in a simulation may have played out could help a tutor defend a suggested alternative to the student's chosen path. Similarly, pointing out observations that the student should have made during a simulation (as negative evidence) has pedagogical benefits for teaching decision making.

Certainly, logging all possible alternative outcomes is not feasible, nor is re-simulating everything on-the-fly during an AAR. We suggest pre-compiling common novice mistakes and problems encountered during specific scenarios, and identifying sets of critical mistakes. Also, the simulator can compute and log alternative outcomes as processor availability permits. An automated tutor can exploit even a single instance of such.

### 5.4 Usage Data

Many tutoring systems model the student's evolving knowledge of the domain, learning characteristics, and skill using the interface. This assists the tutor when deciding which tutoring strategies to employ and when formulating appropriate feedback messages. When tutoring is based on a log of previous use of a simulation, as is the case with OOS and FSC described above, it is important this log include as much information about the user's state as possible to initialize a student model for use during the tutored AAR. For example, knowing if an execution matrix was correct upon its initial design, but changed to a sub-optimal plan for some reason would enable the tutor to remark that the student's first impression was correct. Depending on how advanced the tutorial model is, a discussion of the importance of following one's instincts could even follow.

A second example of logging user usage data is to record all interface activities (e.g. menu selections, typed input, button clicks) and the timings thereof. This permits the detection of *floundering*, the almost random exploration in response to an impasse. Knowing that a student has struggled in this way, and what events during the simulation preceded, or possibly caused such a response can be of great help in building a more accurate student model.

## 6 Planning for Explanation and Tutoring

External visibility of the logic that underlies synthetic entity behavior, plus detailed reporting of internal simulation state, are necessary to enable an XAI system to answer a full range of AAR questions. Detailed reporting of user interaction with the user interface will contribute to achieving tutoring objectives.

Specific recommendations for "XAI-friendly" logs include:

- Employ a rich behavior model within the simulator, and make it visible to external software. Include declarative representations of goals, preconditions, end effects, and conditional and repetition constructs within tasks.

- Produce a high-fidelity log file.
    - Include all scenario information, including the initial state of the world, and all predefined orders to entities.
    - Distinguish between reactive behaviors, planned behaviors, and orders.
    - Include details regarding the satisfied conditions that caused reactive behaviors to occur.
    - Record the specific subgoals that planned behaviors are intended to fulfill; include details of the satisfied conditions that cause them to be aborted.
    - Log changes to state variables that appear in behavior preconditions and end effects.
    - Delimit meta-events in the log, i.e., indicate the set of actions that together denote an action at a higher level of abstraction.
- Use a self-documenting log file format that is easy to comprehend by both humans and machines.
- Include the consequences of what-if scenarios, as processor time permits.

If there are multiple ways to execute a behavior, the XAI system needs to know why one approach was chosen over another. If a behavior consists of a series of repeated actions, the XAI system needs to know why the repetition ended.

These design considerations are crucial not only for why-style questions, but also for queries about how to perform behaviors and queries about entity state. If the simulator does not provide task decompositions, or the log file does not contain the relevant data about entity state, then such queries cannot be answered.

As discussed in section five, the inclusion of automated tutoring introduces additional considerations. Events that did not happen may be more important than those that did, for instance, the trainee failed to provide adequate troop cover. For a particular domain, subject matter experts need to identify these common mistakes so that simulation builders can design logging capabilities to capture them.

The trainee's interaction with the simulator is not limited to the orders given to CGFs. Even if perfect orders are given, ongoing activities such as monitoring the progress of troops under the user's command are also important. As a first step, simulation designers should log the trainee's interaction with the user interface; future work would involve recognizing user activities, such as when the user observes the progress of synthetic entities under the user's command, plan recognition, and real-time pedagogical support.

# 7 Conclusion

Explainable AI and intelligent tutoring have been active topics of research in the AI community. Both technologies have a great deal of promise to increase the effectiveness of constructive simulations that include complex behavior models as training tools. However, in order to support automated explanation and tutoring, simulation and behavior model developers must take these features into account early in the development process. Adding the necessary features retroactively is not feasible without extensive system revision.

We hope that the recommendations presented here, as well as previous research on the topic (Swartout, Paris and Moore, 1994), will be of benefit to the modeling and simulation community as they design and develop new simulation systems and behavior architectures for automated explanation and tutoring applications.


# Acknowledgements

We thank the members of the OOS and FSC development teams for assisting us by answering technical questions about their simulation systems. We thank RAND's Arroyo Center for providing access to an SME to address our subject-matter questions. We also thank the anonymous reviewers for their constructive comments.

This research was sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM). Statements and opinions expressed herein do not necessarily reflect positions or policies of the U.S. Government; no endorsement should be inferred.


# References


Clancey, W. (1984). Methodology for Building an Intelligent Tutoring System. In W. Kintsch, P. Polson, & J. Miller (Eds.), *Methods and Tactics in Cognitive Science*, (pp. 51-83). Hillsdale: Lawrence Erlbaum Associates.

Courtemanche, A. and Wittman, R. (2002). OneSAF: A Product-Line Approach for a Next-Generation CGF. In *Proceedings of the Eleventh SIW Conference on Computer-Generated Forces and Behavioral Representations*, 349-361.

Leinhardt, G. and Schwartz, B. (1997). Seeing the Problem: An Explanation from Polya. *Cognition and Instruction*, vol. 15(3), 395-534.

McArthur, D., Statz, S., and Zmuidzinas, M. (1990). Tutoring Techniques in Algebra. *Cognition and Instruction*, vol. 7(3), 197-244.

Swartout, W., Paris, C., and Moore, J. (1994). Design for Explainable Expert Systems. *IEEE Expert*, 6(3), 58-64.

Swartout, W. and Moore, J. (1993). Explanation in Second-Generation Expert Systems. In J. David,

J. Krivine, and R. Simmons (Eds.), *Second Generation Expert Systems*, (pp. 543-585). New York: Springer-Verlag.

Suthers, D., Connelly, J., Lesgold, A., Paolucci, M., Toth, E., Toth, J., and Weiner, A. (2001). Representational and Advisory Guidance for Students Learning Scientific Inquiry. In Forbus, K. and Feltovich, P. (Eds.), *Smart Machines in Education: the Coming Revolution in Educational Technology*, (pp. 7-35). Menlo Park, Calif.: AAAI/MIT Press.

van Lent, M., Fisher, W., and Mancuso, M. (2004). An Explainable Artificial Intelligence System for Small-unit Tactical Behavior. In *Proceedings of the Sixteenth Conference on Innovative Applications Innovative Applications of Artificial Intelligence (IAAI-04)*, 900-907.

## Author Biographies

DAVE GOMBOC is a Research Programmer/Analyst at the ICT. He received a M.Sc. in Computing Science from the University of Alberta.

STEVE SOLOMON is a Research Programmer/ Analyst at the ICT. He received an M.A. in Linguistics from the University of California, Los Angeles.

MARK G. CORE is a Research Scientist at the ICT. He received a Ph.D. in Computer Science from the University of Rochester. His current research focuses on dialogue management and natural language generation technology for the XAI project.

H. CHAD LANE is a Research Scientist at the ICT. He received a Ph.D. in Computer Science from the University of Pittsburgh, specializing in artificial intelligence and tutoring systems. His current research involves the use of XAI to drive tutoring and enhance the reflective activities of student users.

MICHAEL VAN LENT is a Project Leader at the ICT and Research Assistant Professor of Computer Science at USC. He received a Ph.D. in Computer Science from the University of Michigan. His current research focuses on explainable AI, adaptive forces, and combining game and simulation technologies.