Characteristics of a Multi-User Tutoring Architecture

Stephen B. Gilbert  
*Iowa State University*, gilbert@iastate.edu

Eliot H. Winer  
*Iowa State University*, ewiner@iastate.edu

Joseph Holub  
*Iowa State University*, dholub3@iastate.edu

Trevor Richardson  
*Iowa State University*

Michael C. Dorneich  
*Iowa State University*, dorneich@iastate.edu

See next page for additional authors

Follow this and additional works at: [http://lib.dr.iastate.edu/imse_conf](http://lib.dr.iastate.edu/imse_conf)  
Part of the [Ergonomics Commons](http://lib.dr.iastate.edu/imse_conf), [Industrial Engineering Commons](http://lib.dr.iastate.edu/imse_conf), and the [Operational Research Commons](http://lib.dr.iastate.edu/imse_conf)

Recommended Citation  
[http://lib.dr.iastate.edu/imse_conf/40](http://lib.dr.iastate.edu/imse_conf/40)

This Conference Proceeding is brought to you for free and open access by the Industrial and Manufacturing Systems Engineering at Iowa State University Digital Repository. It has been accepted for inclusion in Industrial and Manufacturing Systems Engineering Conference Proceedings and Posters by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Authors
Stephen B. Gilbert, Eliot H. Winer, Joseph Holub, Trevor Richardson, Michael C. Dorneich, and Michael Hoffman

This conference proceeding is available at Iowa State University Digital Repository: http://lib.dr.iastate.edu/imse_conf/40
Characteristics of a Multi-User Tutoring Architecture

Stephen Gilbert¹, Eliot Winer¹, Joseph Holub¹, Trevor Richardson¹, Michael Dorneich¹, Michael Hoffman²
Iowa State University¹, Dignitas Technologies²

INTRODUCTION

Intelligent tutor systems have been quite successful in instruction of individuals (Koedinger, Anderson, Hadley, & Mark, 1997; Ritter, Kulikowich, Lei, McGuire, & Morgan, 2007; Vanlehn, et al., 2005), but multiple challenges exist when attempting to tutor a team. Sottilare, Holden, Brawner, and Goldberg (2011) describe some of the architectural challenges of team tutoring at a high level in terms of functional requirements. In this paper we describe specific challenges in terms of implementing a team architecture within the Generalized Intelligent Framework for Tutoring (GIFT), including simultaneous startup and synchronization with distributed team members, maintaining state of multiple users, and timing feedback for teams and individuals appropriately.

Illustrative Example: The Recon Task

To provide an example that drives functional requirements for simple team tutoring, we present the Recon Task. In this reconnaissance mission, a team of two soldiers (Alpha Team, made up of Alice and Bob) is responsible for conducting surveillance over respective sectors of a specific area. Each soldier has three responsibilities, or subtasks, in this Recon Task: 1) identify opposing forces (OPFOR) within the sector, 2) report to the teammate if an OPFOR is moving into the teammate's sector, and 3) acknowledge the alert if teammate reports an incoming OPFOR. This task is described in further detail as a broad experimental test bed for teams by Bonner et al. (2015). See Figure 1. We implemented the Recon Task scenario in VBS2, since that game engine was compatible with the current GIFT release. For ease of implementation, all three subtasks are accomplished by typing individual keyboard keys, e.g. "Type Q to identify an OPFOR in your sector."

Figure 1: The Recon Task. Alpha Team members Alice and Bob in blue must scan their sectors for opposing forces (diamonds) and alert each other if one is moving into the partner's sector. Civilians are distracters.
To provide team tutoring for the Recon Task, we would like the team tutor to be able to offer feedback both to each individual and to the entire team, depending on the dynamics of its performance. For simplicity's sake, let us assume initially that feedback is given in real-time based on errors in any of the three subtasks, rather than other varieties of feedback, such as real-time feedback that praises good performance, prompts that remind members of required actions ahead of time, or summative feedback in an after-action review. This approach leads to the possible feedback messages shown in Table 1. We assume that these feedback messages will be given to individuals by GIFT, and that each team member is logged into an individual computer running VBS2 and GIFT. We will assume that the decision of whether to address the feedback to an individual or to the team is simple: if both individuals' performances merit the same feedback within a very close time window, the feedback is addressed to "Alpha Team."

Let us also assume that our GIFT team tutor will maintain learner models for Alice, Bob, and the Alpha Team as a whole based on performance on these subtasks, essentially keeping score. In Table 1, to illustrate that calculation methods of team performance can vary, the team performance for Subtask 1 is the average of the team members' performance, while for Subtasks 2 and 3, the team performance is the minimum of the both team members' performance.

Table 1: Feedback for each subtask of the Recon Task, as well as sample performance scores for Alice, Bob, and the Alpha Team as a whole. Each performance column can be considered to be a simple learner model.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify OPFOR</td>
<td>Alice/Bob, identify OPFOR as quickly as possible.</td>
<td>Alpha Team, identify OPFOR as quickly as possible.</td>
<td>80%</td>
<td>50%</td>
<td>65%</td>
</tr>
<tr>
<td>2. Alert of incoming.</td>
<td>Alice/Bob, communicate crossings promptly.</td>
<td>Alpha Team, communicate crossings promptly.</td>
<td>40%</td>
<td>60%</td>
<td>40%</td>
</tr>
<tr>
<td>3. Acknowledge alert.</td>
<td>Alice/Bob, acknowledge all alerts.</td>
<td>Alpha Team, acknowledge all alerts.</td>
<td>100%</td>
<td>90%</td>
<td>90%</td>
</tr>
</tbody>
</table>

This scenario now provides sufficient requirements that we can describe the technical architecture and its corresponding challenges. Before discussing our architecture, we briefly describe previous efforts at team tutoring to explore whether previous architectures could support a task like this.
OTHER EFFORTS IN TEAM TUTORING

In the 1990s, the Advanced Embedded Training System (AETS) was designed to facilitate team-based training a Naval Air Defense Team (Zachary, et al., 1999). AETS featured instruction for individual operators based on tracking keystrokes, speech, and eye movements and a comparison of operator behavior with expected behavior. It updated student skill models throughout the simulation. The AETS also monitored individual and team performance and provided a dashboard of relevant information to human trainers, but did not have an architecture in place itself to offer automated team feedback. Another team led by Zachary (Zachary, Santarelli, Lyons, Bergondy, & Johnston, 2001; Zachary, Weiland, Scolaro, Scolaro, & Santarelli, 2002) later created SCOTT, a system for operational team training that used synthesized teammates. However, this system also did not offer automated team feedback.

Marsella and Johnson (1998) did offer feedback regarding team behavior using PuppetMaster, an agent that monitored activities throughout a large scale simulation environment and aggregating feedback and guidance for a human instructor. The researchers in this case decided that it was impossible to do classic intelligent tutor knowledge tracing (inferring an agent's plan based on known possible plans to a goal), so instead they tracked agents' behaviors to see whether they aligned with current goals. Their agent-based approach may be a useful inspiration for our own architecture, though they concluded with a desire for a method of integrating a model of an individual agent's behavior with a model of team behavior.

Rickel and Johnson (1999) used avatars in a virtual environment for training. Their virtual instructor, Steve (Soar Training Expert for Virtual Environments) had originally been developed for individual training, but in the 1999 research was adapted for team training. The authors describe similar requirements to our team tutoring task. A Steve agent must be able to track multiple other entities in the virtual environment (other Steve agents or people) and direct communication to appropriate entities for team coordination. Steve agents were implemented using Soar (Laird, Newell, & Rosenbloom, 1987) and thus had modules for perception, cognition, and motor control. The perception module monitors the full simulation state (including other entities). To enable Steve's task-based plans and goals to accommodate team training, the researchers made the task descriptions more modular, enabling steps to be done in variable order when possible, and thus be done in parallel or in a non-specified order by different team members. Task steps were mapped to roles that other team members could take on. By focusing on a hierarchical list of task steps, the agents could have a basic do-everything-myself plan but regularly check the state of the simulation to see what is already done and what remains. The researchers were also able to use this task-based architecture to enable two virtual instruction agents to speak with each other.

Nair, Tambe, Marsella and Raines (2004) explored team behavior by creating agents that analyzed team behavior in a soccer context. Their automatic team analyst, ISAAC, much like our architecture, contained multiple models of behavior, a model for individual agents, a model for multiple agents (e.g., two soccer players), and a model for the entire team. The multiple agent model focused on recognizing specific patterns of soccer play among small numbers of players. This multiple agent model might also be a possible inspiration for our own team architecture.
TEAM ARCHITECTURE

To create a tutor for the team-based Recon Task, the architecture first requires a representation of each participant's system: a computer running the simulation (VBS2) and a tutor client that can 1) monitor what the participants are doing and 2) give the participants tutor feedback when needed. That tutor client in our case is the GIFT Gateway Module, running as a local server on each participant's computer, and displaying its feedback for each participant in a webpage. See Figure 2. The Gateway Module also uses two plugins to 1) translate VBS2 DIS messages into a format for GIFT and 2) exchange commands with VBS2 via its API.

![Figure 2: The architecture of each team member's computer, running VBS2 and the GIFT Gateway module.](image)

Because the Recon Task requires evaluation of both team and individual performance, some information must be stored in common across all participants in a separate layer. In specifying this common layer, we consider two types of information: information updates that are used short-term (e.g., DIS packets, simulation events, and learner actions) and long-term stored information (e.g., individuals' skills demonstrated over time, and the set of conditions and feedback that are given based on performance in the task domain). Because of the short-term updates and longer-term storage that must be shared across team members to enable this tutor, the architecture requires both a method of communicating update messages and a method of maintaining information over time. In our implementation, the primary GIFT server provides the common layer for some of both. GIFT uses a third-party messaging system called ActiveMQ for communication updates, and the long-term storage occurs within GIFT's Domain Module and its Learner Module.

There are two kinds of short-term information updates that need processing. ActiveMQ is used within GIFT to communicate within the modules, and in particular, to pass learners' actions to the modules. If the updates are of interest to a module, e.g., if the message is about Alice identifying an OPFOR, the tutor will process it. Higher fidelity information that needs to be communicated at higher frequency between players, e.g., game engine state changes such as "Alice just moved to x, y, z, so draw her at new position a, b, c on Bob's screen," are handled by the VBS2 multi-player module. It is worth noting that if another client simulation were used for team tutoring that did not have a similar multi-player module to handle high fidelity synchronization, this architecture would need to change somewhat.

Two separate modules, the Domain Module and Learner Module, store information longer-term. The Domain Module contains preprogrammed feedback that the learners will receive, along with the conditions that trigger those feedback messages. Those conditions are passed to the Pedagogical Module.
during initialization of the scenario for execution during run-time. The Learner Module contains accumulated skill ratings for the learners and the team, and its information is updated frequently based on team performance. See Figure 3.

In our example, the Learner Module stores the information in the three blue columns at the far right of Table 1, the performance scores for Alice, Bob, and Alpha Team for the three subtasks within the Recon Task. The Domain Module stores the feedback that is shown in the light orange second and third columns. The conditions for the subtask "Identify OPFOR," for example, include "A learner who identifies an OPFOR after more than 10 seconds of its appearing on screen is graded Below Expectation" and "A learner who identifies sooner that 5 seconds after its appearance is Above Expectation." The Domain Module also contains conditions for the team, e.g., for the Acknowledge Alert subtask, a condition might be "If both players perform Below Expectation, then the team is graded Below Expectation." The Domain Module also references the evaluation algorithms for assessment of different learning goals, sometimes called check functions, which are typically java classes customized based on the type of assessment needed for a specific scenario. In our Recon Task, for example, the check functions evaluate whether OPFORs have been identified on time, whether alerts of incoming OPFOR have been given and when, and whether alerts have been acknowledged. While the Domain Module steadily receives update messages and sends back replies according to the knowledge stored within it, its knowledge does not change within a given scenario. In a more complex tutor, the feedback messages themselves might be variable, dynamically adapting to the actions of the learner.

To walk through a typical communication flow within our Recon Task example (the "tutor loop"), the learner takes an action within VBS2, and VBS2 sends the game state to the Gateway. The Gateway translates the VBS2 game state to a GIFT message and sends it to the ActiveMQ bus. Now see the dotted line arrows and numbers in Figure 3. The Domain Module retrieves the message, processes it, and when appropriate, outputs an assessment of a learner's performance for the Learner Module (1). The Learner Module compares the new performance with the current performance, and if the learner state changes, it passes a message to the Pedagogical Module (2) via the message bus. That module decides whether a pedagogical intervention is warranted. If yes, it passes a message to the Domain Module (3) via the message bus. The Domain Module then determines the particular tactic for implementing feedback, e.g., displaying a message in the GIFT webpage or in VBS2 itself, and passes that along the message bus to the Gateway.
Figure 3: The GIFT team architecture that supports the Recon Task. The steps of the "tutor loop" are numbered.

What Makes This a Team Tutor Architecture?

The architecture described so far is not significantly dissimilar from a typical intelligent tutoring architecture in which there is a learner client interface, a tutor running on a separate server, and a translation module in the middle that sends learner actions to the tutor and receives feedback from the tutor to display to the user in the client (Cheikes, et al., 1999; Ritter & Koedinger, 1996). However, two main challenges arise in setting up the team tutor that must still be answered. 1) How is the startup of the scenario handled with multiple users? 2) How does the tutor manage feedback to individuals vs. feedback to the team?

Management of Multi-user Startup

Currently in GIFT, and in the architecture described above so far, multiple users are supported, but each with his or her own GIFT session. The sessions are independent and not aware of each other, however, so that there is no communication across sessions to promote team activities. To link the individual participants' sessions, a team session is required for synchronization at startup. This team session identifies participants and sets up the communication channels so that individuals can receive feedback specific to them, and they can all receive team feedback if appropriate. The solution we have implemented for this is a GIFT Network Lobby. Just as in other multiplayer games, in which users login and then wait for other players to join before entering the game, learners choose a team-based scenario within GIFT, choose a role they will play in the scenario, and then wait for other roles to be filled before
being able to launch. In our particular implementation of the team tutor for the Recon Task, this architecture leads to the startup procedure described in Table 2.

Table 2: Sequences of actions by two players at startup that illustrate synchronization using the GIFT Network Lobby. Alice starts first.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launches VBS2.</td>
<td>Launches VBS2.</td>
</tr>
<tr>
<td>(First launch becomes the master VBS2 node.)</td>
<td></td>
</tr>
<tr>
<td>Launches GIFT webpage &amp; logs in.</td>
<td>Launches GIFT webpage &amp; logs in.</td>
</tr>
<tr>
<td>Chooses Recon Task and Player 1 role.</td>
<td>Chooses Recon Task and Player 2 role.</td>
</tr>
<tr>
<td>(Enters GIFT Network Lobby; waits for Player 2.)</td>
<td></td>
</tr>
</tbody>
</table>

All players present; GIFT Network Lobby launches VBS2 Recon Task Scenario via VBS2 plugin.

Management of Feedback for Both Team and Individuals

This element of team tutoring is the most complex; it can be difficult to coordinate the mechanisms offering feedback to the team and to the individual so that they do not conflict or overlap. In traditional individual tutor, there is one player receiving feedback and one tutor generating it. This conflict can happen on the back end, within the tutor, if the tutor's conditions indicate that a player currently merits feedback both as an individual and as a team member. This issue of prioritizing multiple feedback messages to give can happen within an individual tutor as well, but that issue can be resolved with priorities assigned to the conditions, done by Le, Menzel, and Pinkwart. (2009) among others. In the constraint-based tutor, ASPIRE, Mitrovic et al. (2009) organized conditions by domain concept to address this issue. However, if the components in a team tutor that give feedback are created as semi-autonomous processes, e.g., a feedback agent for the individual, and a separate feedback agent for the team, then simple prioritization does not work as well; the challenge becomes more of a coordination issue.

This issue of multiple conflicting messages can also arise on the front end, in the learner's user interface. If the tutor is allowed to give the learner multiple messages, a decision must be made as to how they will be timed and visually arrayed or played back so that one message does not get upstaged by the other(s).

In our case, because part of our goal was to extend GIFT to accommodate team tutoring, we used its existing Domain Knowledge File (DKF) format. A DKF file typically stores the conditions and feedback for an individual tutor. One approach we considered was to use one DKF for the individual tutoring, and one DKF file for the team tutoring. However, since our requirements for the Recon Task include having the individual feedback include the participant's name (e.g., "Alice, identify..."), and current DKF feedback statements cannot be customized, we settled on using three DKFs, one for Alice, one for Bob, and one for the team. This implementation has led to some duplication of code, in that the same conditions and feedback are repeated in Alice's DFK and Bob's DFK. Also, because some of the
conditions (but not all) overlap with the Team DKF, there is additional code redundancy. For example, if we want to check whether Alice is acknowledging alerts, we do that in her DKF. We would also have similar condition code in Bob's DKF to measure whether he is acknowledging alerts. Finally, we might have a slight variation on that condition in the Team DKF to measure whether the team as a whole acknowledging alerts. This approach made us realize how helpful it would be in the longer term to have an architecture that allowed domain knowledge inheritance or nesting of files. E.g., there could be a single individual DKF class that spawns customized instances of individual DKFs for each learner. Additionally, the Team DKF could allow inclusion of or referencing of conditions and feedback from the individual DKF class to eliminate that redundancy.

This current implementation also has the issue that because the three DKF files are independent, it is not possible to create conditions that depend on the actions of multiple players (or Boolean combinations of conditions). For example, I could not write a condition like, "If Alice is doing well at identification, AND Bob is not, then tell Alice..." Also, because the DKF files are independent, an individual learner may receive multiple messages, one from the individual DKF and one from the team DKF. In our current implementation we do not have a method of prioritizing these according to pedagogy or type of condition. We could implement a system-wide rule such as, "If there are feedback messages from both team and individual DKFs, give the team one." Currently our implementation allows both messages to appear on screen.

One potential approach to addressing the above issues might be to have scripting language statements allowed in both conditions and feedback. If the feedback message could be written, "{name of learner}, identify..." and the message format sent by ActiveMQ included the learner's ID, then multiple individual DKFs would not be needed. To address multiple Boolean conditions, an author might write a condition that translates to "If {learner 1} is doing well at identification, AND {any other learner} is not, then tell {learner 1} ..." Writing conditions that depend on time and learner history could also be facilitated by writing a condition that includes relative comparisons instead of absolute ones, such as, "If performance level of {any learner} has improved more than one level in the past 5 minutes, then..." Such a scripting language could be called GIFTscript, and be based on the principles of Applescript (2007) and Tutorscript (Blessing, Gilbert, & Ritter, 2006), a language used with Cognitive Tutors at Carnegie Learning, Inc.

**CONCLUSIONS**

When tutoring teams, we are primarily interested in two scenarios: teams that are co-located and synchronous, and those that are distributed and asynchronous. In its current state, multiple GIFT instances can be connected via a local network. By connecting trainees via a multiplayer training application, we can simulate the co-located, synchronous scenario. However, limitations exist within the current architecture: it is difficult to provide the complex tutoring to both teams and individual members that a human coach would provide because the GIFT DKF models (one for each team member and one for the team) remain static and independent of each other. This paper describes an initial attempt to create an architecture to support team tutoring. While we have been successful in doing so, an understanding of the lack of scalability of this architecture and its required simplicity will inform future development of a more robust future team tutoring architecture within GIFT.
ACKNOWLEDGEMENTS

The research described herein has been sponsored by the U.S. Army Research Laboratory - Human Research & Engineering Directorate (ARL-HRED). Statements and opinions expressed in this paper do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

REFERENCES


**ABOUT THE AUTHORS**

**Stephen Gilbert, Ph.D.** is an associate director of the Virtual Reality Applications Center and assistant professor of Industrial and Manufacturing Systems Engineering at Iowa State University. His background includes cognitive science, engineering, and Human Computer Interaction. He is PI on the team tutoring project with ARL HRED and also co-led the development of a reconfigurable mixed-reality training environment for the warfighter. Dr. Gilbert has over 10 years’ experience developing intelligent tutoring systems.

**Eliot Winer, Ph.D.** is an associate director of the Virtual Reality Applications Center (VRAC), associate professor of mechanical engineering, and a faculty affiliate of the human computer interaction (HCI) graduate program at Iowa State University. He has integrated four virtual and three live environments in a simultaneous capability demonstration for the Air Force Office of Scientific Research and has co-led the development of a next-generation mixed-reality virtual and constructive training environment for ARL HRED. Dr. Winer has over 15 years' experience with virtual reality, computer graphics, and simulation technologies.

**Joseph Holub, MS** is a graduate research assistant at the Virtual Reality Applications Center (VRAC) at Iowa State University where he is finishing his Ph.D. in human computer interaction (HCI) and computer engineering. He has worked on projects for path planning of unmanned aerial vehicles, live virtual constructive training of dismounted soldier, and visualization of large data using contextual self-organizing maps. Currently, he is working on building tools for augmented reality assembly in manufacturing as well as the GIFT team training architecture. His Ph.D. research is on visualizing functional imaging data on multiple hardware platforms.

**Trevor Richardson, MS** is a graduate research assistant at the Virtual Reality Applications Center (VRAC) at Iowa State University where he is finishing his Ph.D. in human computer interaction (HCI) and computer engineering. He has worked on augmented reality assembly in manufacturing as well as the GIFT team training architecture. His Ph.D. research is on optimization techniques using contextual self-organizing maps.

**Michael Dorneich, Ph.D.** is associate professor of Industrial and Manufacturing Systems Engineering and a faculty affiliate of the human computer interaction (HCI) graduate program at Iowa State University. Dr. Dorneich's research interests focus on creating joint human-machine systems that enable people to be effective in the complex and often stressful environments found in aviation, robotic, learning, and space applications. Dr. Dorneich has over 19 years’ experience developing adaptive systems which can provide assistance tailored to the user's current cognitive state, situation, and environment.

**Michael Hoffman, MS** is a Senior Software Engineer at Dignitas Technologies with a M.S. in Computer Science from the University of Central Florida and over 10 years of experience in software development. He has worked on various efforts in the fields of modeling and simulation as well as integrating numerous software and hardware systems such as third party simulations and sensors. Michael is the lead on ARL’s Generalized Intelligent Framework for Tutoring (GIFT) project.