Team Training with Simulated Teammates

Emilio Remolina, Jian Li
Stottler Henke Associates, Inc.
San Mateo, California
{remolina,li}@stottlerhenke.com

Alan E. Johnston
NASA Marshall Space Flight Center
Huntsville, Alabama
Alan.E.Johnston@nasa.gov

ABSTRACT

Joint team simulations are usually used to allow a team to practice working together. For example, team training simulations at the Payload Operations Center (POC) at NASA Marshall Space Flight Center (MSFC) are held in order to train a POC console position. Typically, these training simulations require instructors to help students operate the control displays, monitor and evaluate trainee's performance and provide help and instructional feedback to students. Qualified POC operators play the role of other teammates: these teammates are given a script outlining the different interactions they will have with the trainee and other teammates and a time line for the actions they should take. Joint team simulations are however scarce and expensive.

In this paper we present CITTP (Computerized Individual Trainer for Team Performance), an intelligent tutoring system (ITS) framework for individuals to rehearse their team related tasks using computer based simulations. CITTP is used as a cost effective training tool to complement team integration exercises. CITTP concentrates in defining three key elements that a team ITS must have: (i) authoring tools to define training scenarios; (ii) intelligent simulated teammates; and (iii) spoken natural language capabilities that allow simulated teammates to interact with the trainee. We illustrate CITTP when used to train a Payload Rack Officer at MSFC's POC. A training scenario requires the trainee to apply some NASA procedures involving coordinated action with teammates. CITTP provides real time feedback, evaluates the trainee performance, executes actions on behalf of the student, and in most cases coaches the trainee whenever he makes mistakes. CITTP's simulated teammates respond to trainee interactions by providing required information or asking the trainee for missing information. Simulated teammates also interact among themselves and change the state of the simulation by executing appropriate actions according to NASA procedures.

ABOUT THE AUTHORS

Emilio Remolina is an Artificial Intelligence research scientist at Stottler Henke Associates, Inc. He received his Ph.D. in Computer Science, specializing in cognitive robotics from the University of Texas at Austin in 2001. His graduate work focused on map building, whereby an autonomous robot combines sensory information and actions it performs in order to build and localize in a map of its environment. Dr. Remolina's research interests include intelligent tutoring systems, planning, simulation and common sense reasoning.

Jian Li is an Artificial Intelligence software engineer at Stottler Henke Associates, Inc. He received his Master's Degree in Computer Science from the University of San Francisco in 2000. Mr. Li's field of expertise is software design and development with a concentration in intelligent tutoring systems, simulation, training systems, and graphical user interfaces.

Alan E. Johnston is team lead of the NASA training organization responsible for crew and ground support personnel training for the International Space Station (ISS) payload program. He received his undergraduate and masters degrees from the University of Tennessee, and has been working at the NASA Marshall Space Flight Center in Huntsville, Alabama for 22 years. His experience includes 14 years of space flight payload program training and operations in the Spacelab, MIR, and ISS programs. As team lead of ISS payload training, Mr. Johnston is responsible for ensuring the astronaut crews and Payload Operations Center ground support personnel are adequately trained to support payload operations aboard the ISS.

Team Training with Simulated Teammates

Emilio Remolina, Jian Li
Stottler Henke Associates Inc.
San Mateo, California
{remolina,li}@stottlerhenke.com

Alan E. Johnston
NASA Marshall Space Flight Center
Huntsville, Alabama
Alan.E.Johnston@nasa.gov

INTRODUCTION

Although individual expertise contributes to a great extent to team performance, other factors contribute significantly as well - factors such as a shared awareness of the overall goals or tasks, communication skills and protocols, familiarity with other team members, their expertise and competency, and knowledge of the roles and responsibilities of each team member. It is then important to train individuals not only on their particular task but in the context of the team where those tasks are to be applied. Joint team simulations are usually used to allow a team to practice working together. However, it is expensive or impossible to perform these exercises in actual situations, with the actual equipment, software, and personnel, not to mention the required instructors to evaluate the students' actions, in addition to the necessary facilities.

Using simulation-based training systems to allow individuals to rehearse their team related tasks offers a cost effective means to complement and to make more effective the scarce and expensive team integration exercises. However, improvements are needed to fully exploit the training opportunities made possible by these technologies. In particular, (i) methods and tools are needed which enable instructors and subject matter experts to create an intelligent tutoring system (ITS) easily, without programming, to monitor and assess the student's actions in complex, dynamic situations; (ii) methods are also needed to create simulated teammates that behave as humans will do when confronting the conditions posed by a training scenario; and (iii) simulated teammates with spoken dialog capabilities are needed so the trainee using the ITS can operate as he would do in a real situation. Next, we discuss these needs in turn.

Traditional question and answer CBT systems cannot differentiate between active and inert knowledge, thus missing whether individuals can apply their knowledge to make correct decisions in operational circumstances. Most ITSs allow the trainee to practice its skills on real scenarios. However, traditional approaches to the development of ITSs are hampered by the knowledge acquisition bottleneck - the need to construct an explicit expert mental model. A practical ITS solution should define means for easily authoring scenarios by non-programmer SMEs who have no detailed understanding of the ITS system.

In order to rehearse team exercises, other team members are needed. For simple scenarios where at any time the student can only perform a limited number of actions, it is possible to code in seemingly intelligent simulated entities. A simple solution requires an almost explicit enumeration of the different ways a scenario can evolve. This solution, however, is impractical for a large team (e.g., mission operations team) and simulation environment where participants have multiple goals, multiple ways to achieve these goals, and changes in the environment are due to events other than a participant's actions (e.g., a teammate's action).

The training value of a simulation-based ITS depends ultimately on the user's ability to feel immersed in the instructional scenario that it presents. That means being able to operate the training application without focusing conscious attention on it as a software system. For team training, it is important to have a mixed-initiative spoken dialog interaction between the trainee and the simulated teammates. This natural mode of interaction allows trainees users to mentally engage with simulated teammates, while keeping their eyes, hands, and focus of attention on the training exercise and its representation of the situation. These spoken dialog facilities in turn impose further requirements in the creation of simulated teammates that now should be able to understand the trainee utterances and act accordingly.

In this paper we describe CITTP (Computerized Individual Trainer for Task Performance), a framework for developing simulation-based intelligent tutoring systems to train individuals on procedural tasks requiring team coordination. CITTP focuses on extending existing ITS technologies in the following three aspects: (i) it supports the definition of training scenarios which are more complex and dynamic than those supported by previous tutoring systems which presume that all changes in the simulation are caused by the student, carried out under normal operating conditions; (ii) it provides the means to define intelligent simulated teammates; and (iii) it allows the student to communicate with the simulated teammates using a natural language interface.

RELATED WORK

The idea of training with simulated teammates is not new. In particular, the NAVAIR's Synthetic Cognition for Operational Team Training (SCOTT) effort has as objective "to apply advanced instructional strategies and training technology to help aviators practice team skills in a deployed simulation environment using realistic models of human behavior as simulated adversaries, teammates and instructors. The result will be a cost-effective and deployable system to enhance and maintain crucial aviation skills in fleet operators. The product of this effort will be a prototype of an intelligent, stand-alone training system for deployed and forward deployed aviation teams so individuals and teams can practice crucial advanced team skills. The prototype will integrate: (i) a capability for simulating teammates in a realistic mission simulation; and (ii) capabilities for automated scenario generation, performance measurement and diagnostic feedback. The prototype will incorporate human performance modeling techniques and will leverage on advances in eye tracking and voice recognition." (see web site for the Research and Development from NAVAIR Orlando Training Systems Division).

Different approaches and representation languages have been used to model the behavior of simulated teammates. For example, [Zachary et al. 2001; Scolaro et al. 2002] illustrate how "iGEN-based synthetic entities use speech interactions to work with the human trainees and each other. This allows the trainee to practice specific tasks and teamwork skills. One synthetic entity also acts as an instructor and provides situation feedback and detailed data for an after-action review". The iGEN cognitive engine is an implementation of a broader framework for modeling human information processing described in the research literature under the name COGNET.

The work described in this paper illustrates how to build simulated teammates and tutors for training of procedural tasks that require team coordination. It mainly uses and enhances two technologies: (i) it uses the Task Tutor Toolkit (T3) language [Ong and Noneman, 2000] to model the trainee procedures; and (ii) it uses Behavior Transition Networks (BTNs) to model simulated teammates. T3 and BTNs, in our opinion, provide expressive representation languages that yet can be used by subject matter experts to create scenario based training ITSs. In particular, as illustrated next, we show how such languages can be used to model speech interactions with simulated teammates.

CITTP FUNCTIONAL DESCRIPTION

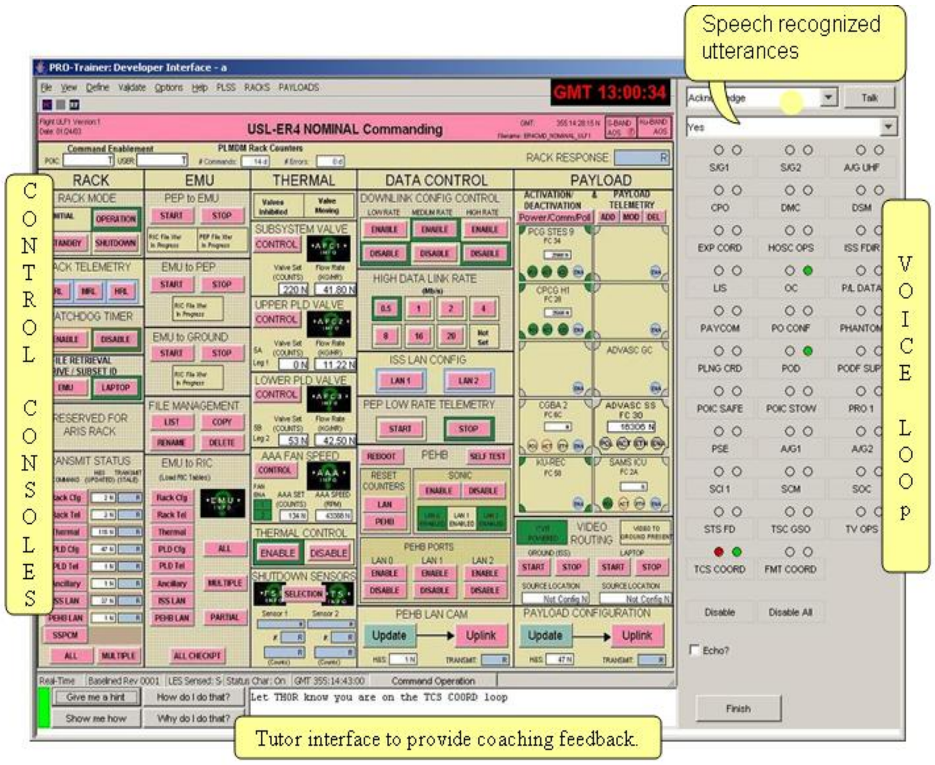
In a tutoring session, the trainee is presented with a scenario describing a concrete situation where some procedures are to be applied. These procedures will require the trainee to contact simulated teammates, which he will do using a spoken natural language interface. The ITS monitors the trainee's actions and evaluates his performance in terms of the principles associated with the different procedures that should have been applied. In addition to monitoring student actions, the tutor monitors the state of the simulation, and dynamically determines the expected behavior of the trainee. Finally, the tutor can execute actions on behalf of the student and coach the trainee whenever he performs actions that are not expected according to the procedures that apply in the scenario.

1. Coordinate with MCC-H THOR 30 minutes before activity.
2. Inform POD of the activation 5 minutes before activity.
3. Ask CPO enable for commanding.
4. Disable thermal control on Express Rack.
5. Adjust EXPRESS Subsystem Valve full Open.
6. Adjust the RFCA to accommodate the rack required flow plus the payload required flow.
7. Once the RFCA flow has stabilized out and the THOR authorizes it, the payload leg valve can be opened to its required setting.
8. Once the flow has stabilized through the payload leg valve, the PRO can close the subsystem valve to its nominal setting.

Figure 1. Simplified outline of the NASA procedure to activate a water cooled express subrack payload.

Activating an ExPRESS Rack payload scenario

We illustrate CITTP when used to train a Payload Rack Officer (PRO) at the Payload Operations Center at Marshall Space Flight Center (MSFC). The PRO should activate an express rack payload following the procedure in Figure 1. Such payloads usually support some scientific experiment at the International Space Station. A procedure requires the execution of tasks within time constraints (see steps 1 and 2 in the figure) and the coordination with several teammates, including: the Thermal Operations and Resource Officer (THOR), the Command and Payload Multiplexer/Demultiplexer Officer (CPO), the Payload Operations Director (POD), the Payload Designer (PD), the Operations Controller



Let THOR know you are on the TCS COORD loop

Figure 2. Instructional interface for training a PRO. The interface provides a simulator of the POC's consoles, a voice loop simulator, an interface to ask for help from the tutor, and a box to display the recognized utterances.

(OC) and the flight director (FLIGHT). Note that Figure 1 shows the procedure as a sequence of steps. However, in practice some steps can be executed in parallel and there are different methods to fulfill the purpose of each step.

When used for simulation-based training, a NASA's procedure description has to be augmented with the pragmatics and human interactions that they suppose. For example, (i) to communicate with a teammate, the PRO should know in which voice loop to contact the teammate, connect to that loop, announce itself on the loop, and wait for the teammate to acknowledge the PRO on the loop; (ii) the PRO should announce any uplink command on the FMT coordination loop (see step 6 in Figure 1); (iii) the PRO should check that the activation was successful by contacting the Payload Developer (PD) to check that they receive telemetry.

Simulator

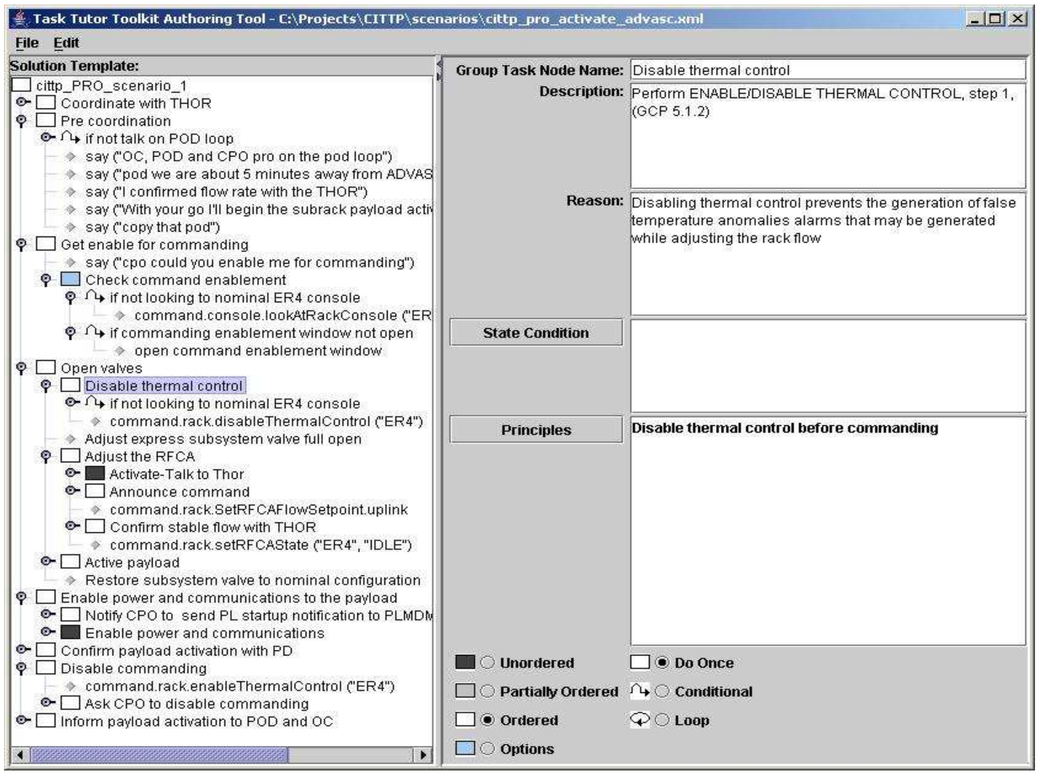
Figure 2 shows the instructional interface used to support the PRO training. The simulator has interactive controls representing those on the actual consoles used by the PROs at MSFC. The PRO interacts with these controls to check the state of the rack or to command the rack. The simulator includes a voice loop that is used by the student to communicate with teammates. The student must monitor conversations happening in more than one voice loop. To communicate with other teammates, the student speaks in a microphone. The recognized utterance is shown to the student, and if the student agrees with the utterance he will press the talk button. The simulator will then send the utterance to the appropriate teammates (those teammates listening in the loop the student is connected to talk). This extra step of pressing the "talk button" is needed because the system does not recognize all possible utterances the student says or because the system could fail to recognize a valid utterance. [1]

Authoring student procedures

During a training scenario the student is expected to apply some NASA procedures. These procedures are modeled as an ordered constrained set of tasks. We used the Task Tutor Toolkit (T3) language [Ong and Noneman, 2000] to define these procedures. This language provides the following representation constructs:

- **Hierarchical decomposition**: a task can be described in terms of subtasks. For example, to open a valve, three subtasks are required: disable the thermal control, adjust the RFCA and activate the payload (see Figure 3).

[1] We used Microsoft Speech recognition engine under the "Command and Control" mode. It has a 90% recognition rate in our system.


Task Tutor Toolkit Authoring Tool - C:\Projects\CITTP\scenarios\cittp_pro_activate_advasc.xml
File
Edit
State Condition
Principles

Solution Template:

```
cittp_PRO_scenario_1
  Coordinate with THOR
  Pre coordination
    if not talk on POD loop
      say ("OC, POD and CPO pro on the pod loop")
      say ("pod we are about 5 minutes away from ADVAS
      say ("I confirmed flow rate with the THOR")
      say ("With your go I'll begin the subrack payload acti
      say ("copy that pod")
  Get enable for commanding
    say ("cpo could you enable me for commanding")
    Check command enablement
      if not looking to nominal ER4 console
        command.console.lookAtRackConsole ("ER
      if commanding enablement window not open
        open command enablement window
  Open valves
    Disable thermal control
      if not looking to nominal ER4 console
      command.rack.disableThermalControl ("ER4")
    Adjust express subsystem valve full open
    Adjust the RFCA
      Activate-Talk to Thor
      Announce command
      command.rack.SetRFCAFlowSetpoint.uplink
      Confirm stable flow with THOR
      command.rack.setRFCAState ("ER4", "IDLE")
    Active payload
    Restore subsystem valve to nominal configuration
  Enable power and communications to the payload
    Notify CPO to  send PL startup notification to PLMDM
    Enable power and communications
  Confirm payload activation with PD
  Disable commanding
    command.rack.enableThermalControl ("ER4")
    Ask CPO to disable commanding
  Inform payload activation to POD and OC
```

Group Task Node Name: Disable thermal control

Description: Perform ENABLE/DISABLE THERMAL CONTROL, step 1, (GCP 5.1.2)

Reason: Disabling thermal control prevents the generation of false temperature anomalies alarms that may be generated while adjusting the rack flow

Principles: **Disable thermal control before commanding**

■ ○ Unordered	□ [✓ ● Do Once]
▨ ○ Partially Ordered	○ Conditional
□ [✓ ● Ordered]	○ Loop
▨ ○ Options	

Figure 3. Representation of the activate payload procedure in Figure 1. The left panel shows the hierarchical decomposition of the procedure. Tasks in the procedure consist of subtasks organized by order constraints (unordered, partially ordered, ordered) and conditions for their execution (optional, do once, do if, loop). Basic tasks represent actions to be done using the simulator (e.g., say an utterance, look at a particular console, disable thermal control for a rack). The right panel defines task information used for coaching.

- **Partially ordered tasks**: order constraints among subtasks define a partial order. For example, before up linking the command to set the payload's flow setpoint, the student should do (in any order) both (i) activate the RFCA control and (ii) inform the THOR of the activation.
- **Conditional tasks**: some tasks should not be done if certain conditions are met. For example, if a valve is already at the desired setpoint the student should not modify the setpoint for the valve.
- **Disjunctive tasks methods**: a task might be carried on in different ways. For example, the PRO can verify that he is enabled for commanding by looking at the USL-ER4 Commanding console or by looking at the command enablement window for the rack.
- **Loops**: a task may be executed repeatedly until some condition is satisfied. For example, a teammate must be contacted until he responds.

Each task in a procedure is annotated with the feedback that should be given to the student: **hints** on what to do (e.g., perform enable/disable thermal control step), **how** to do it (e.g., go to the USL-ER4 Nominal console, press the disable button under the thermal control sub panel), and **why** to do the action (e.g., disabling thermal control prevents the generation of false temperature anomalies alarms).

The tutor can also demonstrate to the student how to perform a task (see Figure 4). For this purpose, *the tutor is represented as any other teammate in the system*. As explained later, simulated teammates' behaviors are modeled as Behavior Transition Networks (BTNs).

Authoring student dialogs

Student's interventions on dialogs are described in terms of the primitive action *say(utterance)*, where *utterance* is a set of strings covering all the possible sentences the student is expected to say at a particular time. Each string represents a *pattern of sentences* that the student can pronounce. For example, the string "we … adjust the rack flow" will match both sentences "we are calling to adjust the rack flow" and "we will adjust the rack flow".

The author of the scenario defines the logic of a dialog using ordered constraints, conditional tasks and state constraints as illustrated next:

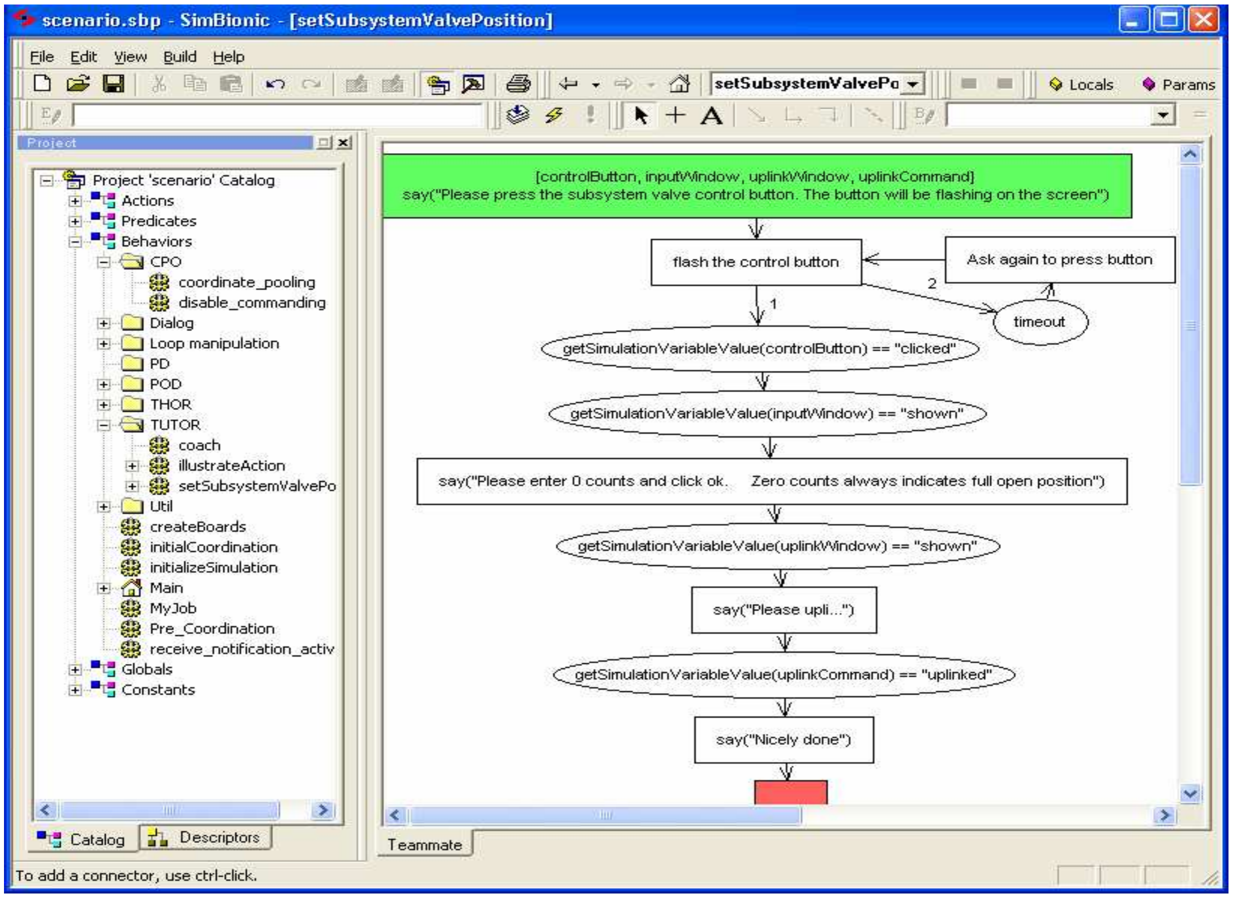

scenario.sbp - SimBionic - [setSubsystemValvePosition]
File Edit View Build Help
setSubsystemValvePo
Locals
Params
Project
Project 'scenario' Catalog
Actions
Predicates
Behaviors
CPO
coordinate_pooling
disable_commanding
Dialog
Loop manipulation
PD
POD
THOR
TUTOR
coach
illustrateAction
setSubsystemValvePo
Util
createBoards
initialCoordination
initializeSimulation
Main
MyJob
Pre_Coordination
receive_notification_activ
Globals
Constants
Catalog
Descriptors
[controlButton, inputWindow, uplinkWindow, uplinkCommand]
say("Please press the subsystem valve control button. The button will be flashing on the screen")
flash the control button
Ask again to press button
2
1
timeout
getSimulationVariableValue(controlButton) == "clicked"
getSimulationVariableValue(inputWindow) == "shown"
say("Please enter 0 counts and click ok. Zero counts always indicates full open position")
getSimulationVariableValue(uplinkWindow) == "shown"
say("Please upli...")
getSimulationVariableValue(uplinkCommand) == "uplinked"
say("Nicely done")
Teammate
To add a connector, use ctrl-click.

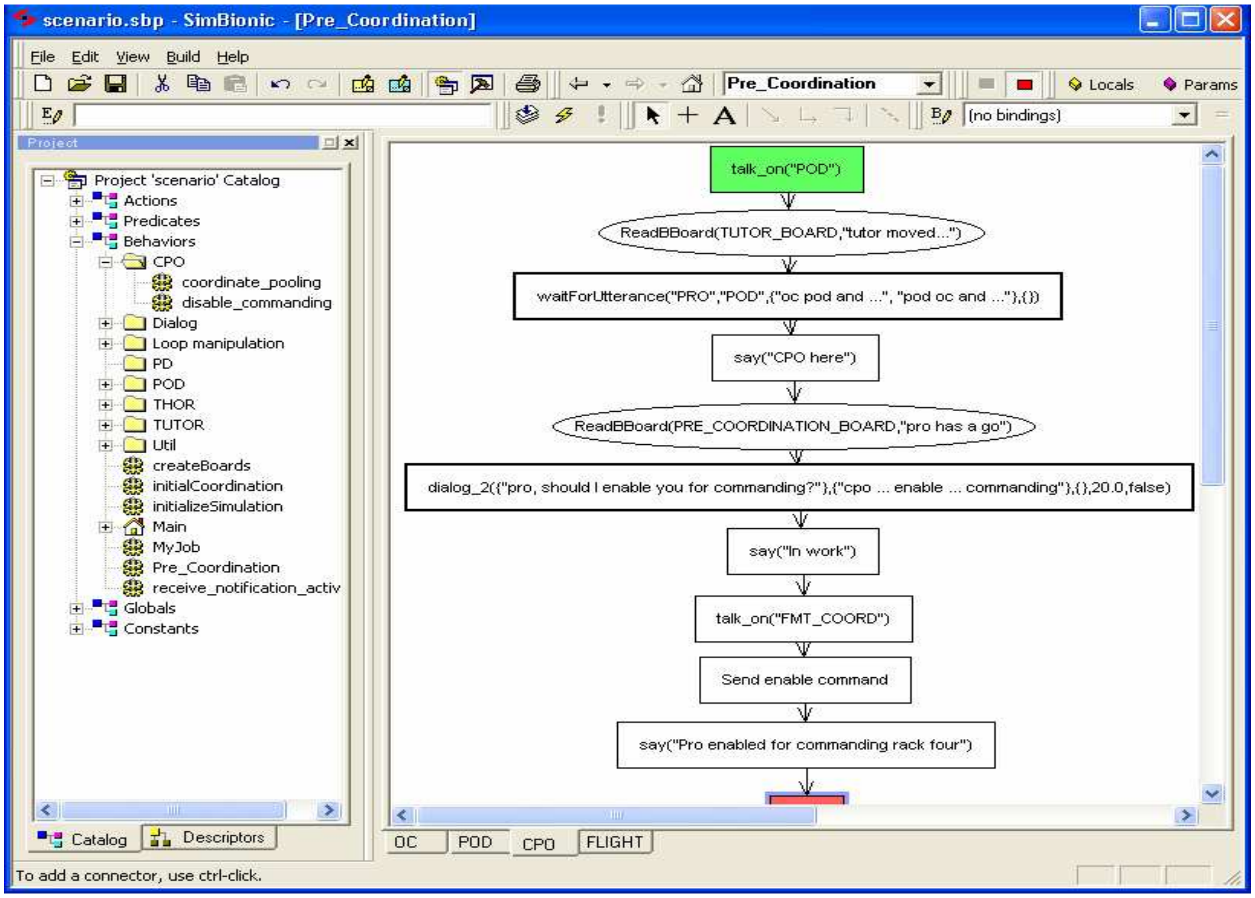
Figure 4. Definition of the tutor's behavior showing the student how to open a valve to full open. The tutor flashes the subsystem valve control button on the screen and asks the student to click on it. The tutor waits for the student to click the control button (a timeout exists to repeat the instruction if the student does not do as told). After the student clicks on the control button, the tutor waits for the counts input window to appear and then asks the student to input zero counts, checks the student do so, asks the student to uplink the command, and verifies that the student do so.

- T*ask order constraints* define the order of expected student interventions in a dialog. For example, it is possible to represent that the PRO should tell the THOR the rack temperature and flow rate before asking for his authorization to proceed with the activation. The PRO may mention the temperature and the flow rate in any order.
- *Conditional tasks* can be used to decide the course of the dialog based on conditions derived from a teammate's communication and simulation state variables. For example, if the rack is already enabled for commanding, then the PRO should not contact the CPO asking to enable him.
- *State constraints* are used to coordinate interactions with teammates. For example, the THOR will authorize the payload activation and the PRO should acknowledge it. The tutor will check the common sense fact that the PRO should acknowledge the authorization only after he actually hears it. In this case, the action *say("copy ...")*, used to acknowledge the authorization, has associated the state constraint "*lastUtterance.THOR == go*".

Authoring simulated teammates

The system does not differentiate between the student and simulated teammates, and both obey the same rules when interacting with the simulator. Moreover, simulated teammates do not know whether they are interacting with other simulated teammates. This way, the definition of simulated teammates has as much detail as the procedure the trainee will follow during the scenario. For example, for the simulated POD to contact the simulated FLIGHT it will have to join the FLIGHT loop to talk and announce itself on that loop.

The behaviors of simulated teammates are modeled using Behavior Transition Networks (**BTN**s). A BTN is a kind of state machine where states are behaviors and state transitions represent the conditions under which an agent will stop executing one behavior and start executing another behavior. Basic behaviors (those that cannot be described in terms of other behaviors) are called *actions*. Unlike state machines, BTNs provide the following constructs proper of programming languages: (i) they are hierarchical, (ii) have variables



```
talk_on("POD")
ReadBBoard(TUTOR_BOARD,"tutor moved...")
waitForUtterance("PRO","POD",{"oc pod and ...", "pod oc and ..."},{})
say("CPO here")
ReadBBoard(PRE_COORDINATION_BOARD,"pro has a go")
dialog_2({"pro, should I enable you for commanding?"},{"cpo ... enable ... commanding"},{},20.0,false)
say("In work")
talk_on("FMT_COORD")
Send enable command
say("Pro enabled for commanding rack four")
```

Figure 5. Dialogs interventions are conditioned by facts determined at other stages of the dialog. These facts are asserted on blackboards shared among teammates. In the example, the PRE_COORDINATION_BOARD is shared by the POD, OC and CPO who will join the POD's loop to talk to the PRO. The CPO will wait for the POD's activation approval before talking to the PRO to enable him for commanding. The dialog interventions are modeled in terms of high-level dialog pattern behaviors (e.g., dialog_2) which hide the logic that the dialog interaction supposes.

(local memory), (iii) have access to blackboards (shared memory), (iv) are polymorphic – the same behavior is executed differently by each teammate, and (v) can execute arbitrary perceptual or action-oriented code (e.g., query a database, interact with the simulator). We used the Stottler Henke SimBionic tool to create and execute such BTNs [Fu and Houlette, 2002]. SimBionic allows scenario authors to create teammate behaviors by drawing them producing a flow-chart-like graphical representation (Figure 4). Specifically, actions are represented as rectangles, behaviors as boldfaced rectangles, conditions as ovals, and connectors as lines. Scenario authors can attach as many variable assignments, complex expressions, and explanatory comments as they like to any of these elements.

When modeling using BTNs, authors must decide on the basic building blocks (actions, predicates, and key behaviors shared by simulated entities), and use those building blocks to define specific behaviors for the simulated teammates in the scenario. The actual modeling process most often follows a top-down design, where teammate's high-level behaviors are successively refined. Next we present some actions, predicates and behaviors used to model simulated teammates in our NASA application.

Simulated teammates actions

Actions represent the primitive things an agent can do. In general, there are two kinds of actions: domain independent and domain dependent actions. Domain specific actions include:

- listen_on(loops): after an agent executes this action, the simulator will send to the agent all the utterances said on the loops the agent connected to.
- selectUtterance(utterancesPatterns,parameters,chosenUtterance): this action lets an agent select a sentence (*chosenUtterance)* from a parametrized set of possible patterns (*utterancesPatterns*). For example, if *utterancesPatterns* = {"$0,1$, $0,1$ here"} and parameters={"PD", "Advasc PD"} then there are four possible sentences defined by the patterns,

namely, “PD”, “PD here”, “Advasc PD”, and “Advasc PD here”. The *selectUtterance* action will select one of these four sentences providing an alternative to explicitly typing all the sentences and adding some variation to the utterances used by simulated teammates.

Domain independent actions exist that let an agent access and manipulate system data. For example, the action *enqueue(event,simulationTime)* informs the simulator of an event that should happen at the specified time. For example, the execution of

enqueue(createSimulationEventUpdateState("ER4.polling.state", "on"), getCurrentSimulationTime())

states that the polling state on express rack four should be set to *on* right now. For convenience, behaviors are created that hide primitive actions like *enqueue* and allow scenario authors to define teammates' behaviors using domain specific terms. For example, executing the behavior

setPollingState(“ER4”,”on”)

will have the same effect as calling the action above.

Simulated teammates predicates

Predicates are used to check conditions or to access data. As with actions, predicates are domain dependent or independent. For example, the predicate *isExpectedUtterance(context,utterance)* lets an agent decide whether a given utterance is expected in the current context. The domain independent predicate *getSimulationValue(variableName)* returns the current state of the given simulation variable.

Simulated teammates behaviors

On starting a scenario, simulated teammates (agents) are created for each position involved in the scenario. The tutor itself is also represented by a simulated teammate. Each teammate starts running a behavior called “MyJob” which describes what the teammate will do during the scenario. Using the polymorphism facilities of BTNs, the definition of “MyJob” is tailored to each position. For the most part, teammates' behaviors follow NASA's procedures but are augmented with dialog capabilities, as discussed below.

Dialogs are represented by BTNs showing the logic of the different interventions the agent should carry on during the dialog (Figure 5). Each agent uses a *blackboard* where it stores relevant information that may affect the dialog execution. The agent uses its blackboard to determine whether it should engage in an intervention (e.g., to avoid asking for information already provided in the dialog). For example, during the initial coordination, the THOR will store in its blackboard the temperature and flow rate of the payload whenever the student provides this information. Should the student not provide the information, then the THOR will ask the student for the information. As suggested by the example, dialogs between the trainee and the simulated teammates may take different courses depending mainly on the information provided by the trainee.

Many dialogs follow the same pattern of interactions or dialog rules. Authors are provided with a library of dialog macros (implemented by BTNs) representing these common interactions. These macros simplify the authoring of dialogs: SMEs just specify some input data to the macro (e.g., which sentences to say, which sentences to hear) and the macro definition hides the logic of the dialog to the SME (e.g., ask a question, wait for the answer, if the answer is not given in a timely manner then ask again, do not ask more than n times).

CITTP ARCHITECTURE

The high-level architecture of the Computerized Individual Trainer for Team Performance (CITTP) system is shown in Figure 6. The trainee interacts with the ITS through an *instructional interface* which is mainly composed of the *simulator* interface and the *speech recognition engine*. The instructors interact with the ITS through *authoring tools* used to define the datasets (cylinders in the figure) needed by the tutor, simulated teammates and the speech recognition components. The *course manager* allows instructors to organize scenarios into courses (set of related scenarios) and provides facilities to administer these courses. The *runtime engine* provides the communication and coordination facilities that allow the tutor, simulated teammates and the instructional interface to interact with each other. Next we describe the datasets and CITTP's modules responsibilities.

CITTP Main Datasets

The main knowledge sources used by the different agents in the system are the procedure model, the scenario model and the student model. The purpose of this knowledge is described next:

- **Procedure model**: The procedure library codifies the acceptable procedures that a trainee should follow, and the circumstances under which he may follow them. Codification of the protocols for the use of each

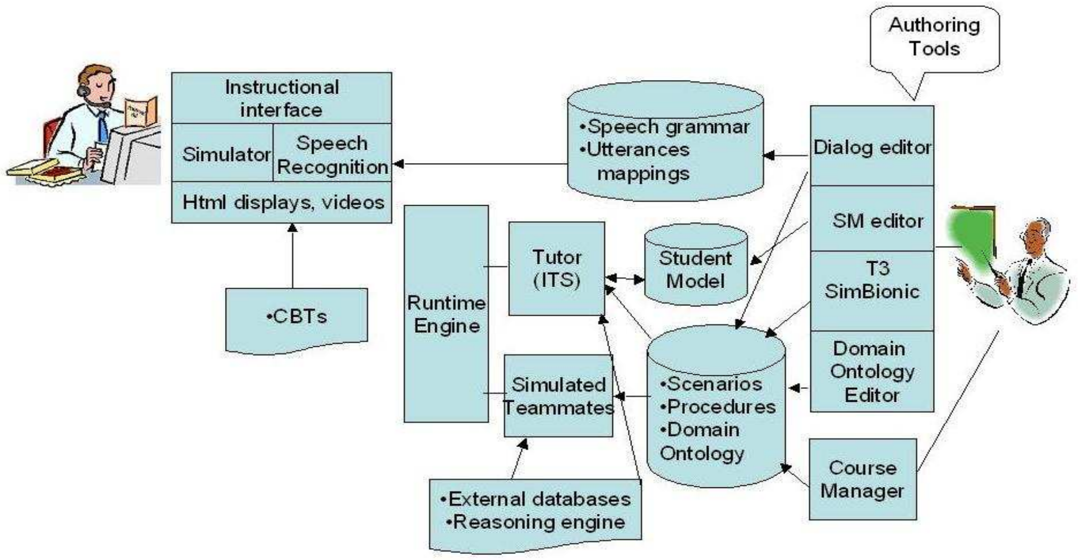


Figure 6. CITTP architecture. Rectangles represent system components and cylinders data sets (see text for details).

procedure enables the tutor to determine what is expected from the student and to monitor his progress during a scenario.

• **Scenario model**: This library contains the information needed to support simulation-based training and intelligent tutoring: representations of the acceptable student responses and/or means to automatically assess the acceptability and efficacy of the student's responses. A scenario also contains a description of the main scenario's events (e.g., an equipment malfunction) and the conditions where such events should happen (e.g., start simulation time 30 minutes before the planned activation and move the simulation time to activation time after the PRO coordinates activation with the THOR). The tutor uses these events to influence the expected behavior of the student and simulated teammates by causing changes in the simulation state. *The behavior of the team is controlled by changing the environment conditions where the team interacts*. This approach circumvents the problem of defining tailored simulated teammates and changing the simulation to support a particular scenario.

• **Student model**: The student model stores the main aspects describing the student performance as a team member. These include the student mastery over principles associated with his task or role in the team, his expertise on the other tasks or roles within the team, and his awareness of other team members' competencies.

• **Domain Ontology**: The domain ontology represents the concepts and relationships among these concepts proper of the application domain. Procedures made implicit reference to this ontology and assumes that the trainee knows what these basic concepts are about. The tutor uses the ontology to make basic common-sense inferences that determine the expected trainee's actions. For example, when the trainee is to activate the ASVASC payload, the tutor knows that this payload is in ExPRESS rack four, and that the commanding console for a rack X is USX-Nominal. From these facts, the tutor infers that the expected user action is "lookAtConsole US4-Nominal".

• **Speech grammar and utterance mappings**: Utterance mappings are used by the speech engine to associate a content representation with a given utterance. This representation is available to the simulated entities and the tutor. Speech grammars define the set of sentences that the application will recognize. Having such a grammar considerably improves the speech recognition rate.

CITTP Architectural Modules

Instructional Interface

The instructional interface is the system component the student interacts with. From the user perspective this usually means a GUI composed by the simulator's GUI, the tutor interface, the speech recognition and text-to-speech (TTS) facilities. From the system perspective, the instructional interface is a façade for all these components that provide the basic user interaction functionality. Other system components communicate with the instructional interface by sending messages to it, without directly accessing the actual components providing the functionality. Dispatching one instructional message may involve using different components: for example, flashing a control in the simulator's GUI while producing an audio instruction.

Run Time Engine
The run time engine acts as a controller providing the communication and coordination facilities that allow the tutor, simulated teammates and the instructional interface to interact with each other. Internally, these components implement a common "agent" interface: they subscribe to the run time engine which (i) distributes available data to interested agents, and (ii) provides facilities for running different agents executing parallel behaviors in the same thread of execution.

Authoring Tools
The different data sets and modules comprising CITTP have corresponding authoring tools. These authoring tools are aligned with our emphasis on having domain experts and instructors to enter the knowledge required by the ITS. Our authoring tool allows instructors to create procedures by demonstrating them, generalizing the procedure demonstration to recognize other valid sequences of actions, and annotating the procedure so that the tutoring system can identify the principles or concepts the student applied correctly or incorrectly [Ong and Noneman, 2000].

Tutor
The tutor does three main functions: selects an appropriate training scenario, monitors the student actions while practicing a scenario, and evaluates the student actions. These three functions are described next.

- **Scenario selection**: the tutor selects a training scenario depending on the student model. Different strategies are used for this purpose. For example, a strategy is to determine the set of procedures that the student masters the least and then choose the most specific scenario where these procedures are illustrated.
- **Monitoring and tutoring**: the tutor monitors the student actions comparing them to those of the procedure being practiced in the scenario. This monitoring is not trivial given the expressiveness of the task definition language: the student has many procedures (or actions) he can perform at the same time, there are different ways to perform a procedure, and the appropriate actions to perform depend on other teammates' actions (and decisions). The state of the monitoring defines the set of expected student actions: information used by the tutor to coach the student. The tutor coaches the student by providing hints, giving explanations on how and why to do an action, and demonstrating the execution of an action.
- **Evaluation and debriefing**: as a result of monitoring the student, the tutor determines which student's actions are right (expected) or wrong (unexpected). After the exercise concludes, the tutor determines which procedures (and so associated principles) the student got right and wrong. It then decides which principles the student has mastered based on his performance history with respect to the principles. The debriefing shows the student which principles were passed or failed and provides links to related material for remediation.

Simulated Teammates
Simulated teammates execute the procedures corresponding to the team role they play. Simulated teammates are specified using Behavior Transition Networks (BTNs). Dialogs are represented by BTNs showing the logic of the different interventions the agent should perform during the dialog. Each intervention is a dialog itself or a basic intervention which can be of three types: a set of utterances to be told, a set of utterances to be heard, or an assertion of what can be derived from the dialog at the state on which the assertion is made. These assertions are stored in a blackboard, and referred to by conditional transitions in the dialog. Information derived from previous states in the dialog is used by conditions on the BTN transitions to decide whether the simulated teammate should engage in an intervention.

The speech recognition engine transforms trainee spoken utterances into a user defined utterance representation. This representation is passed to the simulated teammates, who extract information from this representation to guide its dialog interventions. Which information is expected from a given utterance is determined by the current state on the BTN defining the dialog. This state provides the context needed to interpret a received utterance. This approach to natural language understanding presents a compromise between a deep understanding and a shallow understanding of utterances.

CONCLUSIONS

In this paper we described CITTP (Computerized Individual Trainer for Team Performance), an intelligent tutoring system (ITS) framework for individuals to rehearse their team related tasks using computer based simulations. Next we summarize the main aspects of the system:

- **Tutoring of dynamic complex scenarios**: in addition to monitoring student actions, the tutor monitors the state of the simulation, and dynamically determines the expected behavior of the trainee. Changes in the simulation state are not only caused by trainee actions, but also by simulated teammate's actions or by the physical laws of the system the trainee controls.

- **Procedure representation**: the expected student behavior is encoded in a language that models tasks as partial order of subtasks, and provides constructors to represent conditional branching, looping, and disjunctive alternatives to perform a task. Our task representation language and authoring tools provide a crucial trade-off between system usability and knowledge representation power.
- **Definition of simulated teammates' behaviors**: simulated teammates behaviors are represented using Behavior Transition Networks (BTNs), a finite state machine-like behavior language. A graphical yet powerful programming language is used to define these BTNs. This representation mechanism allows authors the definition of sophisticated teammate's behaviors including dialogs with the trainee or other teammates. Libraries of high-level domain dependent behaviors are provided to facilitate the definition of teammate behaviors.
- **Dialog templates library**: the system provides a library of dialog templates representing the most common high-level dialog patterns used by simulated teammates. The logic of these dialog templates is expressed using BTNs that describe the different interaction between the participants in the dialog.
- **Access to existing systems**: the definition of the tutor and simulated teammates can make use of existing data sources (e.g., a database) and functionalities provided by commercial off-the-shelf products (e.g., speech recognition and text-to-speech engines, rule-based systems).

FUTURE WORK

Our current research involves enhancing CITTP in the following two aspects:

- **Tutoring of error actions**: the tutor will coach the trainee whenever he performs actions that are not expected according to the procedures that apply in the scenario. For example, the tutor will be able to detect that the student is skipping some steps in the procedure, and depending on the requirements of the procedure, the tutor will either suggest the trainee to execute skipped steps or to continue the procedure execution from its current state.
- **Speech recognition, text to speech and dialog definition tools**: dialogs will be defined by speaking the utterances that may occur during the dialog. These utterances are used to automatically update the grammar used by the speech recognition engine, and to update the behavior of simulated teammates and the definition of the expected trainee's dialog interventions. Both the tutor and the simulated teammates will be able to understand different trainee's spoken utterances that convey the same information. When adequate, authors can specify only the content of simulated teammates utterances (e.g., ask for the flow rate) and the system will automatically generate the appropriate utterance (e.g., what is the expected flow rate?, which flow rate should I expect to see?).

ACKNOWLEDGEMENTS

The work reported here was funded by NASA under SBIR contract NNM04AA83C.

REFERENCES

Fu D., and Houlette R. (2002). "Putting AI in Entertainment: An AI Authoring Tool for Simulation and Games". *In IEEE Intelligent Systems, July-August.*

Luperfoy S., Domeshek E., Holman E., Struck D. and Ramachandran S. (2003). "An Architecture for Incorporating Spoken Dialog Interaction with Complex Simulations". *Proceedings of the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC).*

Ong J. and Noneman S. (2000). "Intelligent Tutoring Systems for Procedural Task Training of Remote Payload Operations at NASA". *Proceedings of the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC).*

Scolaro J. and Santarelli, T. (2002). "Cognitive modeling teamwork, taskwork, and instructional behavior in synthetic teammates". *Proceedings of the Eleventh Conference on Computer Generated Forces and Behavioral Representation*. Orlando: Institute for Simulation and Training.

Zachary W., Santarelli T., Ryder J., Stokes J., Scolaro D., Lyons D., Bergondy M., and Johnston J. (2001). "Using a community of intelligent synthetic entities to support operational team training". *Proceedings of the Conference of Computer Generated Forces, Virginia Beach, VA.*, pages 215-224