



ELSEVIER

Advances in Engineering Software 35 (2004) 337–355

ADVANCES IN
ENGINEERING
SOFTWARE

www.elsevier.com/locate/advengsoft

Intelligent virtual environment for process training

Ayman Wasfy*, Tamer Wasfy, Ahmed Noor

Center for Advanced Engineering Environments, Old Dominion University, Hampton, VA, USA

Received 2 March 2004; revised 1 April 2004; accepted 7 April 2004

Abstract

An intelligent virtual environment is described for training users in the operation of complex engineering systems. The environment combines an intelligent agent facility, for tutoring, guiding and/or supervising the training; an object-oriented virtual environment engine, for displaying the engineering system; and a simulator, for simulating the system controls. The intelligent agent facility includes: (a) a hierarchical process knowledge base, (b) a rule-based expert system for natural language understanding, and (c) a human-like virtual characters engine. Three types of objects are used for representing the process knowledge, namely, processes, steps, and constraints.

An application of the environment to the interactive training for operating a NASA wind tunnel is described. Two agents in the environment can perform several functions, including conducting an interactive virtual tour of the facility; guiding and supervising the training, as well as certifying the trainee.

© 2004 Elsevier Ltd. All rights reserved.

Keywords: Virtual reality; Intelligent agents; Natural language processing; Virtual training environments

1. Introduction

The significant increase in processing power available for rendering in recent years has led to the development of many virtual-reality based simulators for training users in a 'safe' 'near-natural' synthetic environment. Applications of the virtual-reality based training simulators include training for operation of industrial machines [1,2], power-plants [3], vehicle driving, piloting, traffic-control, maintenance simulators [4,5], medical procedures training [6–8], and military operations training.

In order to make the simulators natural, interesting, and engaging, there has been an increasing interest, in developing 'intelligent' photo-realistic synthetic human characters/agents, that can speak and understand natural-language, in order to act as virtual tutors, guides, and/or learning companions. In the last decade, intelligent software agents have demonstrated their potential in supporting many types of practical applications [9], including process control, manufacturing, air-traffic control, information management, E-commerce, computer games, and medical applications. Some of these applications involve problem solving, where the agent proactively performs interactive tasks to support a human user [10]. Some of the characteristics that make

a software agent intelligent are interactivity, reasoning ability, and learning capacity. While there are many intelligent agents applications, the present study focuses on training related applications.

Rich et al. [11] developed a tutor/assistant agent that uses hierarchical process representation or 'plan trees.' The plan trees decompose a process into subtasks and constraints. The learner can select from a set of simple preset utterances to go through the process training. Thus, the approach relies on memorizing or looking-up commands. For a complex process, this could potentially distract from the main purpose of the training, which is gaining proficiency in the task at hand. Also, in the actual task setting, the trainee would interact with a human tutor through natural language. Another agent called Steve that also uses a hierarchical process plan was developed by Rickel et al. [12]. Steve is a half-body torso animated graphical agent for training in VR environments. Steve was used to teach students the operation of complex equipment such as the High Pressure Air Compressor (HPAC), aboard US Navy surface ships. In the HPAC application, Steve could understand a small set of preprogrammed commands. The Steve avatar moved by floating around rather than walking and was able to do simple pointing gestures. Steve's face displayed a few expressions, however his lips were not synchronized to the speech.

* Corresponding author.

Recently, a new version of Steve that was applied to a conversational role-playing application incorporated some natural-language processing capabilities as well as human-like emotional and perceptual capabilities [13]. Commercial software products (developed by Haptik Inc. and Boston Dynamics Inc.) were leveraged to give the new version of Steve a more natural motions, face, and facial expressions. Three Steve agents were used in a military application called the Mission Rehearsal Exercise (MRE), which aims to prepare army officers to face cultural interaction scenarios under stressful conditions. The MRE is a conversational role-playing application where the user has to stick closely to the dialogue in the script and, unlike the HPAC application, it does not involve user hands-on interaction with complex equipment in the environment to perform a process.

In Guinn et al. [14] a spoken-dialogue assistant and trainer called AMAT (Advanced Maintenance Assistant and Trainer) for the maintenance of army tanks was developed. AMAT is based on the AvaTalk [15] agent software. AvaTalk has been used for other training applications such as customer service, interviewing, negotiations, and patient assessment and history-taking. The AvaTalk Language Processor accepts spoken natural-language input and maps them to semantic categories. Then it functions as a speech generator by working in reverse, mapping semantic categories to speech output, facial expressions, gestures, and actions in the environment. AMAT provides verbal cues to soldiers on how to find appropriate diagnostic information and procedures within technical manuals but it doesn't instruct or guide the users through the complex maintenance process itself.

In this paper, an intelligent virtual environment for process training is presented. The environment encompasses the following three components:

- An intelligent agent facility that includes:
 - A natural-language interface (NLI) that incorporates a hierarchical rule-based expert system. The NLI is used for natural-language communication with the user, including both understanding and synthesis [16]. The use of expert systems for natural-language interaction distinguishes the present framework from previous work presented in [11–13].
 - An hierarchical process knowledge engine for accessing the process information, including sub-processes, steps, and constraints, to the intelligent agents. The hierarchical process knowledge-base is similar to the plan-trees developed in Refs. [11,12].
 - Humanoid model engine for controlling and displaying animated human-like avatars that can display a wide range of emotions for realistic representation of the intelligent agents.
- A three-dimensional real-time interactive object-oriented virtual environment, which generates a high-fidelity virtual model of the engineering system (including buttons, knobs, user-interfaces, moving parts, etc.) and interfaces with multimodal input/output devices. The trainees practice the training tasks in the simulated virtual environment.
- A system simulator that incorporates the control logic.

The aforementioned components are integrated in a seamless manner and deployed on the Web. The integration of virtual-reality technology with photo-realistic human character technology, natural-language understanding and synthesis, multi-modal interfaces, and hierarchical process-knowledge enables real-time interaction with humanoids in a visually compelling virtual environment. The environment described herein effectively captures process-knowledge and provides interactive training using natural-language communication. It incorporates natural-language understanding and process workflow knowledge, as well as their consequences in the dynamic simulation environment. The agent guides the user through the steps of complex procedures and explains the implications of action (or inaction) in the environment and the resulting performance outcomes.

The proposed environment can be particularly useful for training the operators of safety-critical engineering systems such as nuclear power plants, space station, wind tunnels, air traffic control, etc. In this paper, a sample application of the environment is presented where a user interacts with two human-like avatars (humanoids) to learn the operational procedure of a wind tunnel. The two humanoids collaborate in tutoring, guiding, and supervising the user in the operation of the wind tunnel.

2. Intelligent training environment architecture

Fig. 1 shows the architecture of the intelligent training environment. The environment consists of three major components:

- The intelligent agent facility (IAF), which includes the following components:
 - Natural-language interface (NLI). The NLI relies on a hierarchical rule-based expert system engine to perform the following functions: interpret the natural-language commands of the user; interpret user pointing and navigation in the VE; and output intelligent humanoid agent(s) speech, gestures, emotions, and lip-synching.
 - Hierarchical process knowledge-base. Using knowledge about task structures and constraints and situational context, the IAF can proactively provide the user with the right information at the right time and place.
 - Humanoid model engine.
- The virtual environment engine, which displays and simulates the engineering facility. The VE engine also manages multimodal input from a variety of sources, including the NLI, hand-held computers, head-tracking, tracked wand, mouse, and keyboard.

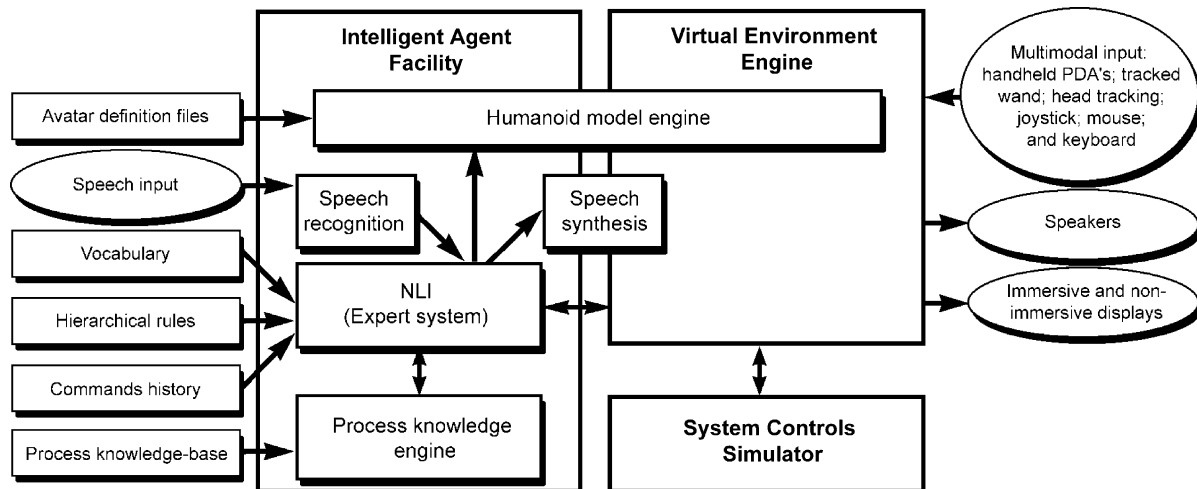


Fig. 1. Architecture of the intelligent training environment.

- A system controls simulator that replicates the system control logic and simulates the physical system response, including the effects of user's actions.

The intelligent training environment integrates a suite of commercial software systems. These include:

- LEA [17] intelligent agent engine.
- IVRESS (Integrated Virtual Reality Environment for Synthesis and Simulation) object-oriented virtual-reality toolkit [18] for generating the virtual environment.
- Microsoft SAPI 5.1 [19] for speech recognition and synthesis.
- Neospeech [20] male and female voices, which are SAPI 5.1 compliant.
- HAPTEK engine [21] for displaying animated emoting near-photorealistic humanoid avatars.

The LEA engine incorporates the hierarchical-rule based expert system and the hierarchical process knowledge base. The hierarchical rule-based expert system is used to understand the user's natural-language commands. The process knowledge-base incorporates the process steps and constraints. LEA also stores the history of the users' interactions with the agent.

The intelligent training environment is web-based. It runs in Microsoft Internet Explorer 6.0 (or higher versions). This is enabled because all the components of the system are packaged as activeX controls that can run on a web page. The system has three main activeX controls: the IVRESS VE engine; the LEA engine; and a hierarchical menu control that can be run in a web page or on a hand-held PDA. The activeX controls download the engineering system model data from a web server. The VE display of the engineering system can either be on the user's desktop computer in a web page, or in an immersive stereoscopic virtual-reality facility such as the CAVE™.

The details of the components shown in Fig. 1 and their role in the environment are described subsequently.

3. Intelligent agent facility

The intelligent agent facility (IAF) includes the rule-based expert system Natural Language Interface (NLI), the hierarchical process knowledge base, and the human-like avatars.

3.1. Natural Language Interface

The function of the NLI is to provide two-way communication with the user in natural-language speech (and/or written text). The NLI accomplishes natural-language understanding by converting the user's natural-language commands (as well as pointing gestures) to script that can be sent to the VE engine. This script can query or change the properties of objects in the VE. The avatar is an object in the VE and hence it can be controlled in the same way as other VE objects. Thus the script can be used to animate the avatar, including arm motions, walking, facial expressions, and lip-synching. The NLI communicates with the user by sending output speech to the speech synthesis engine, as well as by changing the visual state of objects in the VE. The NLI includes the following facilities:

- Speech recognition.
- Text-to-speech (speech-synthesis).
- Hierarchical rule-based expert system engine for speech understanding.
- VE interface, including:
 - Receiving and setting VE object property values.
 - Handling non-verbal input from the VE. The NLI can handle non-verbal communication such as pointing and gazing.
 - Handling the state of the agent's avatar in the VE.

3.1.1. Speech recognition

The user's speech is acquired using a good quality microphone. Any Microsoft SAPI 5.1 compliant speech recognition engine can be used for speech recognition. In the present study the Microsoft speech recognition engine, which is built into SAPI, was used. There are two speech recognition modes in SAPI, namely, single word/short phrase mode (or command and control) and continuous dictation mode (with a 30,000 + words vocabulary). Examples of command and control systems are: Command-Talk [22] and Institute of Simulation and Training's Voice Federate project [23]. The vocabulary of these systems consists of the set of control commands, thus restricting the user to say only programmed commands. The recognition rate of continuous dictation is 75–85% at best, which is too low for the present application. The single word/short phrase recognition rate is above 98% (with 2–3 short training sessions). The high recognition rate of the single word recognition mode is due to the fact that a smaller vocabulary (about 1000 words/phrases) is used and the requirement that the user separates his/her words/phrases by a short 0.2–0.4 s pause. The NLI can use continuous dictation, single word/phrase recognition, or a combination of both for speech recognition. When the two approaches are combined, the NLI first tries to resolve the user's utterance using single word recognition mode. If it cannot, then it tries the continuous dictation mode. This allows the NLI to achieve over 98% accuracy rate while still being able to recognize, with 75–85% accuracy, utterances where the user forgot or chose not to clearly separate words. The vocabulary file for the single word consists of a list of all the possible words/short phrases that can be used in any combination to issue natural-language commands for the specific training application. Typically, the number of words/phrases in the list is about 1000 and includes in addition to regular conversation words (such as 'show,' 'hide,' 'set,' 'is,' 'are,' 'in,' 'at,' etc.) all the key words of the training application.

The speech recognition in the NLI is not as restrictive as strict command and control mode. The main restriction of the mode used herein is that the user has to separate the words clearly during the speech by pausing for about 0.2 s between successive words.

The IAF determines if a command has ended when the user says special execution words (such as 'do it,' or 'execute'). Thus the user does not have to speak continuously and can pause while saying a command. This method is superior to both listening for speech until a sufficiently long pause is detected, and to the push-and-hold method. The listening for pause method does not provide a fast response and tends to cut off users who pause in the middle of a command to figure out what to say next [24]. In the case of push-and-hold method, the user has to push a button to let the system know when to listen, and release the button to let the system know when to stop listening. This does not provide hands-free operation, which is important

for natural interaction in the VE. The special execution word strategy will ignore temporary command interruptions. Some interruptions however maybe permanent, i.e. the user wants the agent to ignore what was said before. Permanent command interruptions are identified based on the length of time of the interruption.

3.1.2. Speech synthesis

Any Microsoft SAPI 5.1 compliant speech synthesis (text-to-speech) engine can be used for generating the agent's speech. In the present study the speech engine of NEOSPEECH [20] was used for male and female voices. The LEA engine sends SAPI the text string that is to be spoken. SAPI along with the NEOSPEECH engine generate the speech along with the following events:

- *Start of sentence event.* This event returns the starting and ending character positions of the sentence that is currently being spoken. This event is used by the NLI to highlight the sentence that is currently being spoken as well as to run any scripts that is contained within the sentence.
- *End of word event.* This event returns the starting and ending character positions of the word that is currently being spoken. This event is used by the NLI to highlight the word that is currently being spoken.
- *Viseme events.* These events are generated in order to do the lip synchronization. Those events are passed by the NLI to the VE engine, which in turns passes them to the agent avatar display module to place the lips of the agent avatar in the proper position.

3.1.3. Hierarchical rule-based expert system

The expert system rule hierarchy consists of rule groups and rules that are loaded from one or more input files.

Rule group

The rules Group object allows grouping a set of rules, including other rule groups, in order to provide the ability to construct hierarchies of rules (Fig. 2). Each group has a name and includes a list of rule names or group names, which are contained within the group. Fig. 3 shows how

```

DEF GroupName Group {
  children [
    DEF rule1 Rule Type1 { ... }
    DEF rule2 Rule Type1 { ... }
    DEF rule3 Rule Type1 { ... }
    DEF group1 Group { ... }
    USE rule 4
    USE group 2
    ...
  ]
}

```

Fig. 2. Rules group.

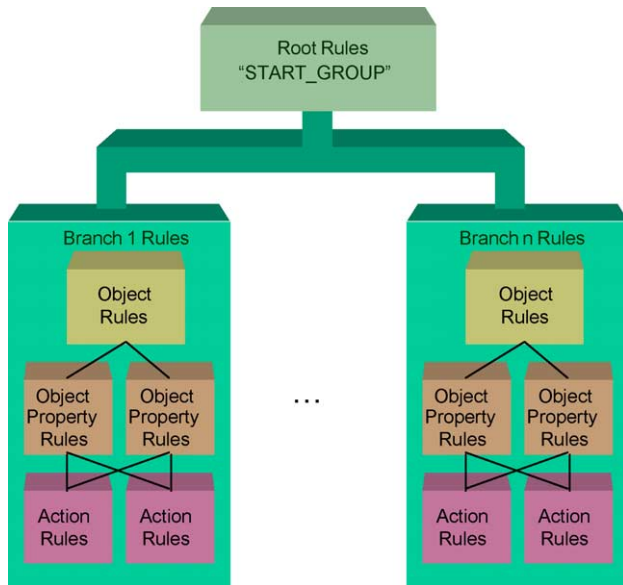


Fig. 3. Rules hierarchy.

the rule groups are used to construct a rule tree. The root rules are placed in a special group called 'START_GROUP'.

The hierarchical rules approach takes advantage of the object-oriented hierarchical data structure of the VE by organizing the rules into three main types, namely, object, property, and action rules:

- *An object rule* is triggered when the object name/alias is found in the user's command. An object rule reflects the structure of the actual VE object. It 'connects' to a rules group containing a set of rules that correspond to the properties of the object.
- *A property rule* is triggered when the property name/alias is found in the user's command. It connects to a group of actions rules that can be performed on the property.
- *The actions rules* contain a set of actions that can be performed on properties. These include:
 - Setting the property to a desired numerical or linguistic value (very high, high, medium, low, very low, etc.). An example of a command is: 'Set the fan speed to high.' In this case, 'fan' is the object, 'speed' is the property, 'set' is the action, and 'high' is the value.
 - Increasing or decreasing the property by a desired numerical value or linguistic value ('increase value a little', 'decrease fan speed a lot', 'reduce pressure by a moderate amount', etc.).
 - Increasing or decreasing the property by a desired percentage.
 - Inquiring about the value of an object's property. For example, 'what is the pressure valve position?'

Rules

A rule consists of a name and a list of attributes (properties) and attribute values. When a rule is triggered

the attributes that it contains are executed. The rule attributes determine the actions performed by the rule. The descriptions of the rule attributes are given in Appendix A. A rule has seven main types of attributes:

- *Word attributes* (see Table A1, Appendix A). These are used to determine whether or not to trigger the rule. This is performed by calculating a satisfaction score for the rule. If that score is greater than a certain threshold, then the rule is triggered. A command consists of a number of words. Each command word is checked against a set of 'required' and 'ignored' words. The total score for a rule is equal to the summation of a *found_score* for the required words that are found, a *not_found_score* for the required words that are not found, and a *scoreOther* for the other words that are neither required words nor ignored words. Note that if the *found_score* for the required words is negative, this means that if those words are found then the score is reduced. This architecture allows issuing the commands in a natural way. For example, for the command 'hide jet', the required words are 'hide' and 'jet'. 'hide' has a number of synonyms such as 'conceal', 'make invisible', 'turn off' and 'switch off'. Also, 'jet' has the following alternative words 'airplane' and 'aircraft'. The ignored words are 'the'. So the user can say 'hide the airplane', 'switch the jet off', 'make the aircraft invisible', 'conceal the jet', and the IAF will recognize all those commands as 'hide jet'. This allows the user to use grammatical (or non-grammatical) constructions that he/she is comfortable with. The NLI also accounts for non-verbal communication such as pointing and looking. For example, the user can simply point at the jet and say 'hide this'. This will be further elaborated in Section 3.1.5.
- *Script attribute*. Contains the script that is to be sent to the VE upon triggering the rule.
- *Output attributes*. The *Speak* and *Reply* attributes output spoken messages and on-screen messages, respectively.
- *Variable manipulation attributes* (see Table A2, Appendix A). These attributes are used to create and set the values of NLI variables; send them to the VE; and receive them from the VE. The values of these variables are stored during the hierarchical evaluation of a command so that they can be accessed by subsequent rules. Any script or output text can contain the names of these variables. Before the script is sent to the VE or before the text string is sent to the speech synthesis engine, the names of the variables are substituted by their values.
- *Rule group hierarchy attributes* (see Table A3, Appendix A). Allow the rule to connect to other rules or other rule groups. This allows the formation of the rules hierarchy.
- *Feedback attributes* (see Table A4, Appendix A). The NLI uses the history of an expert user to intelligently provide useful feedback in the form of suggestions to

novice users. The feedback mode can be switched on or off using the attribute ‘feedBackState.’

- *Process control attributes.* These attributes are used to execute a process or in other words train the user through a process. The process can be executed in four main types of modes: tutor, guide, supervisor, or certification (see Section 3.2).
- *State attributes.* Those define the state in which the NLI is to be left after execution of the command. State attributes allow the NLI to remember information about the last command. This information can be used in the current command so that the user does not have to repeat the context of the command. For example, the user can say ‘turn marshaling box control selector switch to local’. This will trigger the rules, which will execute the command and at the same time set the state to ‘marshaling box control selector switch.’ The next command, the user can say ‘turn it to remote’. The NLI tries to execute the command first without using any states. If it cannot, then it appends the first state to the command and tries to execute it. Then it appends the second state and so on until the command can be executed. Thus, the command will be interpreted as ‘turn marshaling box control selector switch to remote.’

3.1.4. Command history analysis

The present architecture allows for the history for all users’ actions to be maintained. The user’s history is used for contextual analysis to help understand the user’s commands and to provide helpful suggestions to novice users based on an expert user’s history. In contextual analysis, the NLI analyzes the user’s short-term history to determine the context of the conversation and help recognize the user’s commands and, if needed, reverse previous commands. The NLI automatically extracts the context from the previous command(s) by using the rules triggered in the last command(s) to understand the current command. If the command cannot be executed, then the next command in the history is tested. The IAF will repeat this process N times, where N is user defined.

In addition, the NLI uses the history of an expert user to provide useful suggestions to novice users. When a novice user requests suggestions or help from the agent on what to do next, the NLI examines an expert user history and determines what rules the expert user triggered at that point in the hierarchy and suggests the corresponding actions to the user. The NLI examines the novice user’s short-term history in order not to repeat suggestions that have already been made and gets information from the VE in order not to suggest a redundant action that has already been performed in the VE. Timing attributes control the amount of user idle time the NLI waits between providing suggestions to the user. Also, suggestions can be multi-leveled. For example, if the NLI suggests to the user to ‘would you like to increase fan RPM?’ and the says ‘yes’, then the IAF will ask the user ‘by how much?’ to quantify the increase.

3.1.5. NLI Interface with the VE

The NLI can interfaces with the VE through a TCP/IP network socket connection. The NLI ‘get’ rule attribute is used to retrieve data from the VE and place it in an NLI variable. This is done by sending to the VE a script command instructing it to send the value of the desired variable through the socket connection. The NLI ‘script’ attribute is used to send script to the VE in order to change the state of the VE. This script can include commands to modify the value of an object’s property or even to create a new object.

In addition, VE script can be embedded in the agent’s speech (in NLI output attributes) by using a special tag that identifies the script. When the rule containing the speech is triggered, the embedded script is sent when the start of sentence event is triggered. This script can be used to synchronize the agent’s speech with scripted modifications to the VE. Typical uses for speech embedded script are:

- Instructing the agent to carry out motions such as hand/body gestures, pointing, clicking a button, walking, etc.
- Instructing the agent to display a facial expression.
- Modifying the properties of objects in the VE.
- Displaying objects in the VE such as blinking arrows.

3.1.6. Handling of non-verbal communication

The NLI-VE interface also includes the capability to handle non-verbal communication such as pointing and gazing. For example, the user can simply point or look at an object and say ‘hide this’. Also, the user can direct his/her command to a specific agent by looking at the agent.

Hand/head tracking is used to obtain user’s pointing or looking direction. The VE can be instructed (using script) to return a sorted list of all the objects linguistic names that are currently being point or looked at (sorted according to the distance from the user’s head or hand). The ‘testNames’ attribute (see Table A4) is used to test the returned object aliases through the rules hierarchy. The branch that has the highest score is executed. If the same score is returned from two or more branches then the branch with the closest object is executed.

3.2. Hierarchical process knowledge

The hierarchical process knowledge base enables the IAF to have knowledge about a sequence of steps that accomplish a specific objective and the consequences of mistakes. Each process consists of a set of steps as well as other sub-processes. Each process and step can have pre and post constraints. Pre-constraints have to be satisfied before the step/process can be started. Post-constraints have to be satisfied before the step/process is completed. The process hierarchy is illustrated in Fig. 4. The process, step and constraint are objects that have attributes (see Appendix B). Fig. 5 shows an example of a process, process steps

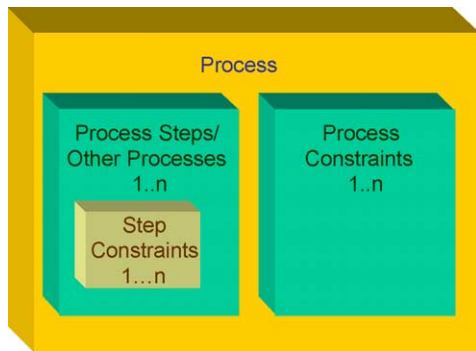


Fig. 4. Process knowledge base.

and associated constraints. The process has the following types of attributes (see Table B1, Appendix B):

- Spoken messages including the process objective and short message which the agent can speak at the beginning of tutoring or guiding the user through this process.
- A set of suggestion questions that the agent can ask the user. If the user answers yes to the question, then the specific process mode (tutor, guide, supervisor, etc.) is executed.
- A set of natural-language navigation rules including the natural-language rules for recognizing: yes, no, go back a step, skip, continue, pause, and abort.
- Pause time constants for the various training modes.
- A list of the process steps including other processes needed to complete the process.
- A list of the process pre-constraints.
- A list of the process post-constraints.

The Step attributes (see Table B2, Appendix B) specify either an action or a passive action (e.g. observing) that is to be performed as part of the step. The action is written as a natural-language command and is sent to the NLI to be spoken by the agent and converted to a script, through the hierarchical-rules, that is sent to the VE (e.g. ‘set the marshaling box control selector switch to local’). The passive action is spoken by the agent (e.g. ‘verify that the indicator 3115 is illuminated red’) and is not sent to the expert system. A step also has an attribute (*runScript*) that specifies a script that is to be sent to the VE concurrent with the step. This allows sending script to the agent to point at a specific object or display a pointing arrow. Similar to processes, steps also have pre-constraints and post-constraints.

The Constraint attributes (see Table B3, Appendix B) include an attribute that sends a script requesting a state variable(s) from the VE. Depending on the value of this variable, the agent can determine if the constraint is violated or not. If the constraint is not violated, then the next step is presented to the trainee. If the constraint is violated, then a message informing the user of the detected constraint violation and proposed corrective measures is spoken to the user. The user is then requested to repeat the step where

the constraint was violated. If the user fails to perform the step three times, then the agent carries out the step.

The IAF can disseminate process knowledge using one of the following training modes. The desired mode is triggered using the rule-based expert system (see Section 3.1.4). The training modes are:

- *Process Tutor*. In this mode, the agent performs the process steps while the user is watching. The user can pause/resume, repeat (go back) a step, or skip a step.
- *Process Info*. This mode is similar to the tutor mode except that the agent will only recite the process steps to the user without demonstrating how they are done.
- *Process Guide*. The agent guides the user step by step through the process. The agent will not go to the next step until the user says a command such as ‘go on’, ‘continue’, and ‘proceed’. The user has to perform each step of the process. The agent checks the process constraints to determine if the user performed the step correctly. If a constraint is violated, then the agent instructs the user to repeat the step. If the user does not perform the step correctly three times in a row, then the agent performs the step.
- *Process Supervisor*. In this mode the agent instructs the user to perform the process. At the end of each sub-process the user lets the agent know that s/he is done. At that point the agent checks the sub-process constraints. If no mistakes were detected, then the agent instructs the user to perform the next sub-process. If mistakes are detected, then the agent lists the mistakes and instructs the user to repeat the sub-process.
- *Process Certification*. This mode is similar to the process supervisor mode except that if mistakes are detected the agent will keep track of the mistakes and only lists them at the end of the entire process. If no mistakes are detected then the agent certifies the user in this process.
- *Intelligent Virtual Assistant*. In this mode, the user asks the agent to perform a process. The agent performs the process while the user can either watch the agent or do something else.

3.3. Interface with the humanoid display engine

The intelligent agent facility visually interacts with the user through full or half-body highly detailed photo-realistic male and female avatars that reside in the virtual environment. The interface between the humanoid avatar display engine and the VE engine is described in Section 4.1. The avatars’ lip movements are synchronized with the speech and their realism is enhanced through a large set of pre-defined gestures and emotional responses. Multiple gestures can be run simultaneously, thus enabling a very large number of combinations of body language. For example, the agent can display a certain emotion by combining a facial expression, a bodily posture, and a gesture. This gives personality to the virtual humans,

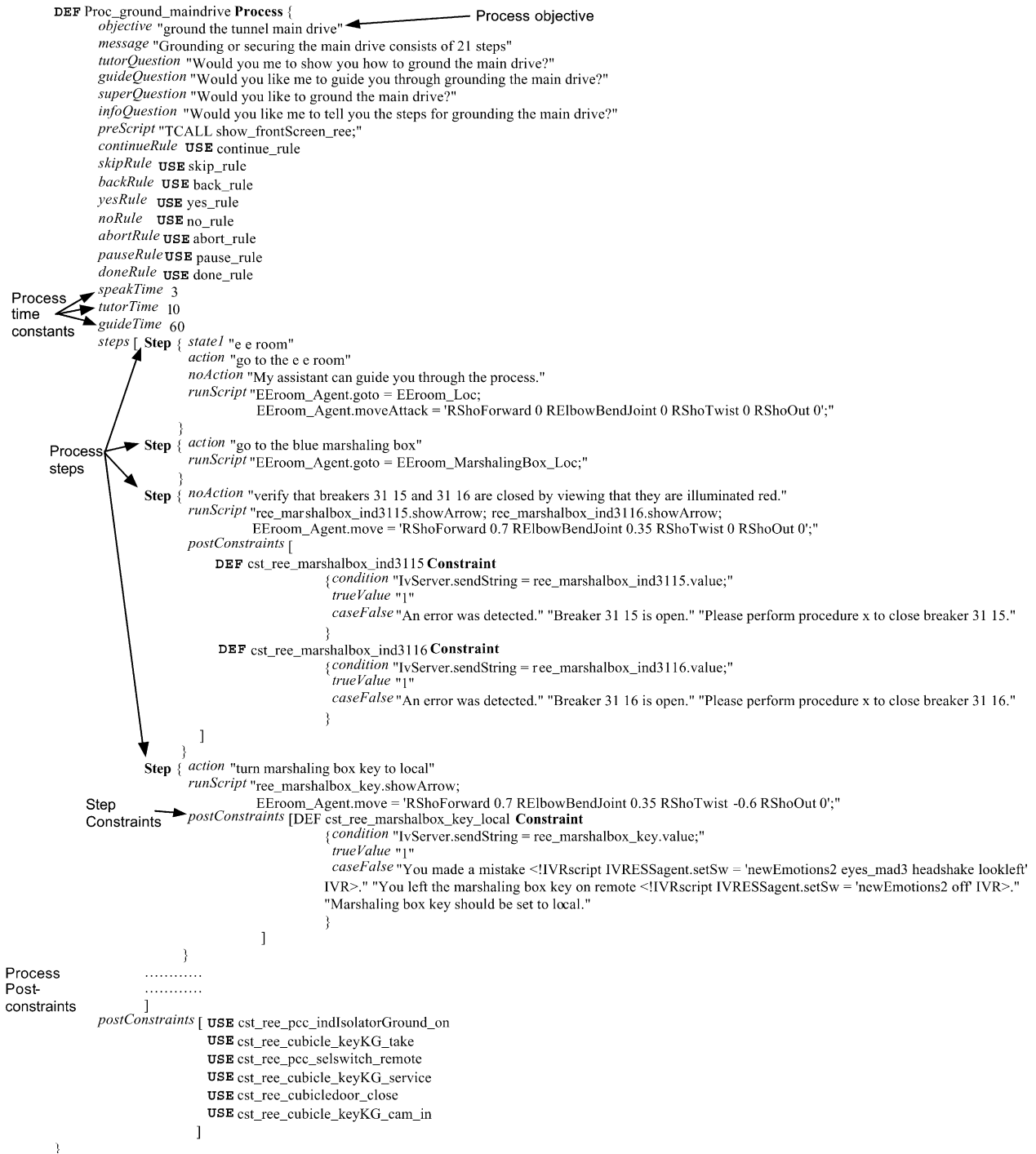


Fig. 5. Example of a hierarchical process object.

thereby making the interaction with the user more interesting.

4. Virtual environment engine

IVRESS object-oriented scene-graph based toolkit is used in the present paper for constructing the VE. Four classes of objects are used to construct the VE:

- Interface objects include many types of user interface widgets (e.g. label, text box, button, check box, slider bar, dial/knob, table, and graph) as well as container objects (including Group, Transform, Billboard, etc). The container allows grouping objects including other containers. This allows a hierarchical tree-type representation of the VE called the 'scene graph'.
- Geometric entities represent the geometry of the various physical components. Typical geometric entities include

unstructured surfaces, boundary-representation solid, box, cone and sphere. Geometric entities can be textured using bit-mapped images and colored using the light sources and the material ambient, diffuse, and specular RGBA colors.

- *Finite elements* represent solid and fluid computational domains.
- *Support objects* contain data that can be referenced by other objects. Typical support objects include material color, position coordinates, and interpolators. For example, a sphere geometric entity can reference a material color support object. Arithmetic operations (such as addition, multiplication and division) and logical operations (such as ‘and’, ‘or’, and ‘not’) can be performed on support objects.

All objects have the same basic structure. Each object has properties that determine its state and behavior, and methods, which are functions that it can perform. In addition, interface objects have events that are triggered when certain conditions, initiated by the user or the passage of time, are met. An event is triggered by calling a script-subroutine associated with that event. The subroutine name consists of the object name concatenated with an underscore and the event name (e.g. *object-name_event-name*). IVRESS-script is an interpreted subset of JAVA-script scripting language that allows setting the properties of the various objects, and writing custom event handling routines. In addition, custom objects can be added to IVRESS by writing C/C++ code for the object and linking that code to IVRESS either dynamically (using a dynamic link library), or statically (by linking with an IVRESS static library file). IVRESS can interface with output devices, including immersive stereoscopic screen(s) and stereo speakers; and a variety of input devices, including body tracking (head and hands), haptic gloves, wand, joystick, mouse, microphone, and keyboard. IVRESS can read and write file formats for geometry data such as VRML 2.0 [25], pictures such as Bitmaps, PNG, JPEG, and GIF; and movies such as MPEG, AVI, and MNG.

4.1. Interface with the humanoid model engine

An IVRESS object that wraps the Haptik humanoid model engine allows loading and displaying full and half body textured highly detailed male and female characters in the virtual environment. A character has a large set of pre-defined gestures. Typical gestures include: looking up, down, right and left; torso bend, twist, and bow; right/left hand; smile; blink; walk; etc. In addition, the gestures also include the visemes (e.g. aa, ih, g, s, eg, uh, etc.) or lip and face positions for lip-synching. Each gesture can take a modifier, which specifies the magnitude/amount of that gesture. Using the IVRESS character wrapper object, the gesture command is sent a specific agent avatar using the script command:

```
Agent_Object_Name.setSw = talkGestL1a
```

where ‘setSw’ is a property of the character wrapper object and ‘talkGestL1’ instructs the Haptik engine to carry out a talking gesture with an amount of ‘a’.

Also, the Haptik engine allows setting the character’s joints’ rotations and positions to any desired value. The wrapper object allows animation of the character hand motions by linear interpolation of the joint positions or angles. For example the commands:

```
Agent_Object_Name.moveAttackTime = 1;
Agent_Object_Name.move = ‘RShoForward 0.7 ×
RElbowBendJoint 0.35 RShoTwist 0’
```

cause the agent to move the right arm in 1 s from the original position to the position specified by the above values of shoulder forward motion, elbow bend, and shoulder twist, where ‘move’ and ‘moveAttackTime’ are properties of the IVRESS character wrapper object.

4.2. Multimodal interfaces

The IVRESS toolkit enables the user to interface with VE input and output devices through output of sensory information and input of commands (see Fig. 1). Output devices include:

- *Immersive stereoscopic display* provided by four $3 \times 3 \text{ m}^2$, 1280×1024 resolution, 24-bit color synchronized stereoscopic back-projected screens arranged as a cubicle room with a front, a floor (front-projected), a left and a right screen (Fig. 1). This configuration provides a field-of-view of at least 180° . Stereoscopic viewing is achieved by displaying the correct perspective view of the model for both eyes of the user using LCD shuttered glasses that are synchronized with the screen refresh rate.
- *Two speakers* are used for the NLI speech output as well as for output of sound effects, and data sonification (Fig. 6).

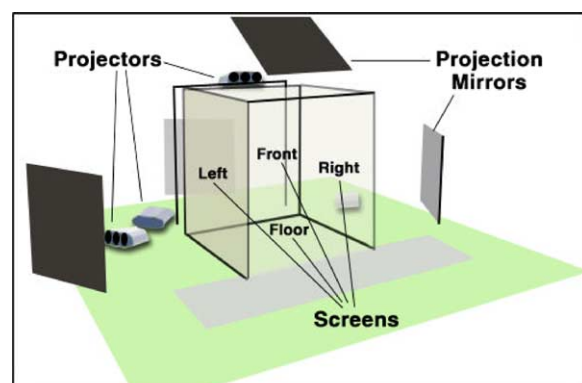


Fig. 6. A four-wall immersive VE facility.

Input devices include:

- *Position and orientation tracking devices* for tracking the position and orientation of the user's body. Tracking receivers are placed on the stereo glasses for head tracking in order to calculate the correct perspective view, as well as a hand-held 'wand' for navigating and pointing in the VE (see Section 2.5).
- *Tracked wand.* The wand has a pressure sensitive 2D joystick and three buttons that can be programmed to perform special functions. The 2D joystick is used to walk or fly in the VE.
- *2D navigation device* such as a mouse, touch pad, or joystick.
- *Microphone* for voice input using to the NLI.
- *Joystick.* If the IAF is used in on the desktop, then a joystick can be used to walk or fly in the VE.
- *Keyboard.* If the IAF is used in on the desktop, then the arrow keys mouse to walk or fly in the VE.
- *Mouse.* If the IAF is used in on the desktop, then the user can use the mouse to click on buttons or drag knobs in the VE.
- *2D hierarchical menu.* The user can control the VE using a hierarchical graphical menu. For example, all the buttons and knobs, and dials can be controlled from the menu. The menu can be displayed as a floating window inside the VE, or can be used on a tablet PC or a hand-held computer connected to the VE computer via wireless LAN. The menu consists of graphical widgets (button; text box; check box; pull-down list box; and slider bar) that are connected to the VE object properties.

5. System controls simulator

The system controls simulator simulates the response of the engineering system in time as a result of user interaction. It consists of script for modeling the physical behavior of the major parts of the engineering system. The simulator includes a high fidelity numerical model of each system component. For example, a wind tunnel drive fan, that is controlled using a proportional controller, is modeled using:

$$I\dot{w} = k(w - w_{\text{Desired}})$$

where I is the moment of inertia of the fan, w is the fan angular velocity, \dot{w} is the fan angular acceleration, w_{Desired} is the desired fan angular velocity, and k is the proportional controller gain. The above equation is integrated in time to give the current fan angular velocity. The user can change the desired angular velocity (w_{Desired}) setting using a knob. Using the above equation, the simulator will slowly accelerate or decelerate the fan to try to match the desired angular velocity. The simulator also includes the facility control logic. This includes simulating the effect of, say, toggling a switch and the current state of the engineering system, on the future state of the engineering system. It also includes propagating events to the various components of the simulator.

6. Application: wind tunnel training simulator

Application of the aforementioned intelligent training environment to NASA Langley's 14×22 ft wind tunnel is described. The tunnel is used to perform experiments on scale models or sections of aerospace vehicles. The tunnel

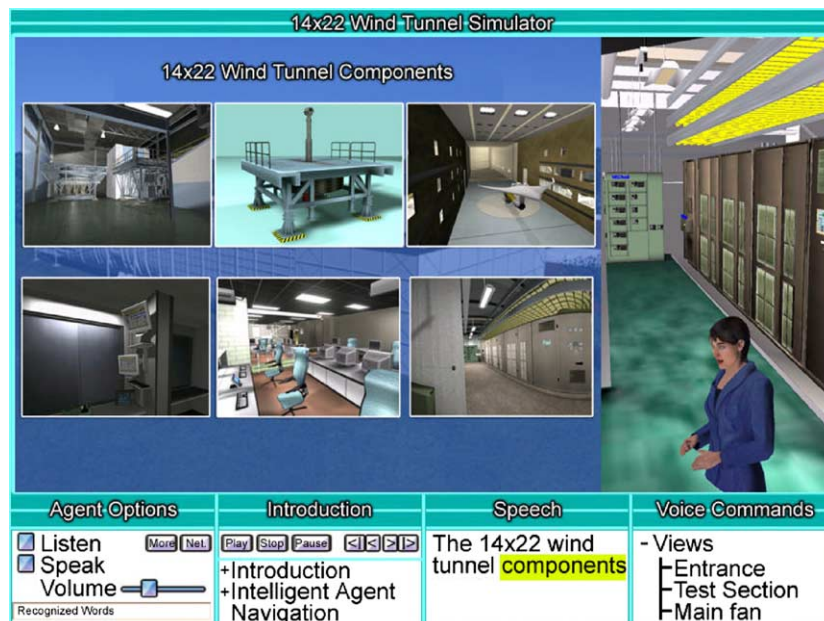


Fig. 7. IAF presenting the components of the wind tunnel.

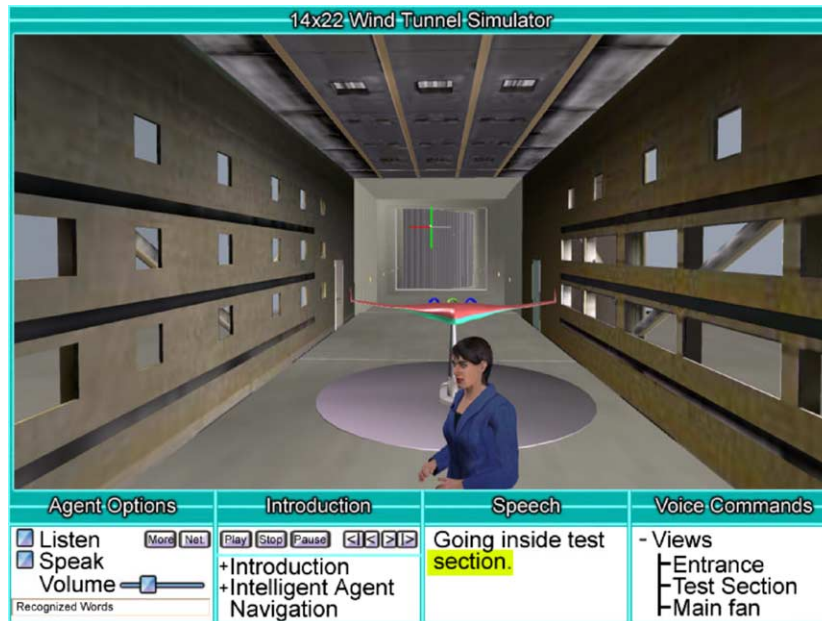


Fig. 8. Test Section and airplane model.

has advanced computer controlled operations. The maximum air speed is 348 ft/s. We will demonstrate how the training simulator is used to provide a safe environment for interactive and realistic training of engineers and technicians.

6.1. Virtual wind tunnel model

A virtual wind tunnel is assembled and displayed using the IVRESS toolkit. The model includes the following major components, which are built using various VE objects:

- VRML photo-realistic geometric model of the wind tunnel. The geometric model is a hierarchical scenegraph consisting, mainly, of textured surfaces (VRML *IndexedFaceSet*). The geometric model was built and textured using Alias' Maya [26] and then exported as a VRML model. Note that hidden objects are not geometrically modeled. The model geometry includes the following components (Fig. 7):
 - Tunnel circuit, including:
 - Test-section (Fig. 8).
 - Experimental aircraft model (Fig. 8).
 - Fan section.

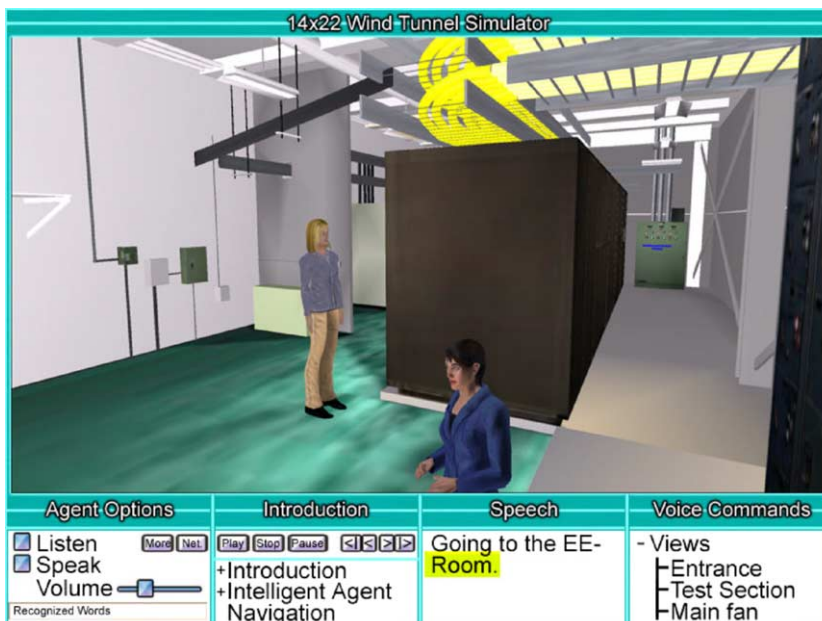


Fig. 9. EE room.



Fig. 10. Control room.

- Tunnel building, including:
 - Electric engineering room (EE room) (see Fig. 9 and left of Fig. 7).
 - Tunnel control room (Fig. 10). In the control room, one of the projectors shows a real-time view of the interior of the test section while another shows a real-time view of the graphical user interface of the tunnel control software.
- Model preparation area.
 - Model setup Carts.
- Tunnel hardware controls. The major tunnel control interfaces are modeled so that the user can interact with them. For example, Fig. 11 shows a virtual model of the tunnel cyclo-converter control panel. It includes the following types of controls: dials, knobs, buttons, switches, and LED indicators. Those are modeled using IVRESS widgets. This allows the user to manipulate the control in the VE by clicking on it or dragging. For example, in Fig. 12, the user can toggle the tunnel control selector key (on the left hand bottom corner) of the EE

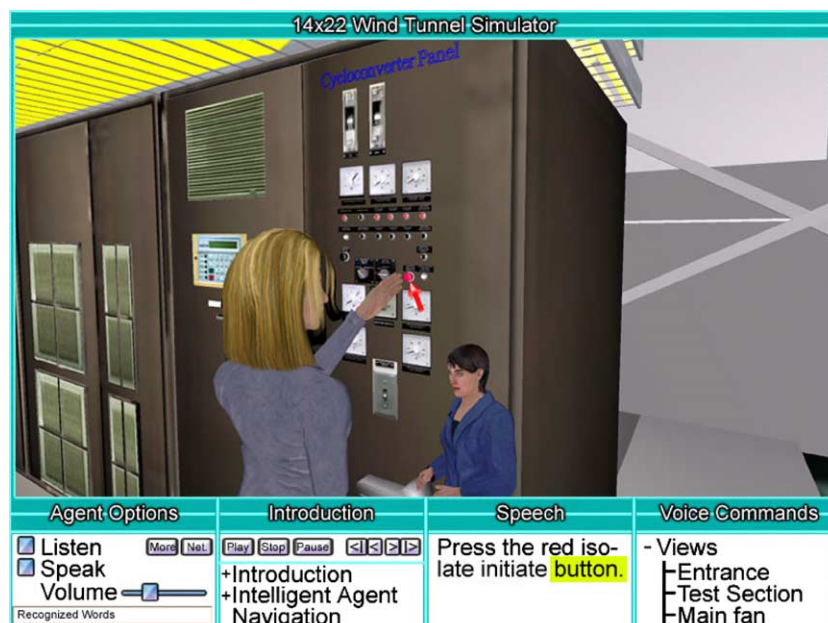


Fig. 11. Cyclo-converter control panel with an ‘always on top’ screen agent interacting with the agent in the environment.



Fig. 12. Blue marshaling box.

room Blue Marshaling Box between ‘Local’ and ‘Remote’ by clicking (using the mouse or wand) on the key. When a control state changes, an event is triggered which runs an associated event-handling subroutine. The script of that subroutine propagates the effect of that change to the tunnel simulator.

- Tunnel control software (Fig. 13). The tunnel control software screens are built using user-interface widgets (labels, text boxes, buttons, check-boxes, sliders, etc.). The software logic and event handling subroutines for all the control software widgets are driven by script.

- Human-like avatars.
- The *Observer* interface object interfaces with the tracked wand and head tracking device. It allows the user to fly-through or walk-through the VE, thus viewing the VE from any perspective. The user can also zoom in and out on desired areas. Multiple observers can be defined and the user can instantaneously ‘tele-port’ between observers.
- A 3D selection object, controlled by the tracked wand, allows selecting, moving and touching objects in the VE. Once the selection bounding box of the selection object

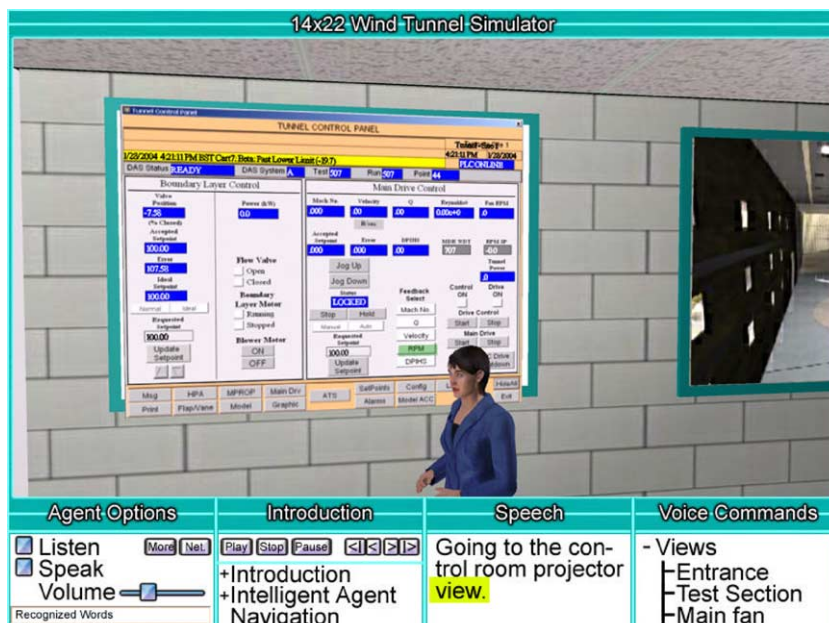


Fig. 13. Tunnel control software projected in real-time on the screen in the control room.

touches an object, a touch event for that object is triggered and the associated sub-routine is executed. Also, a click event is triggered when the selection box is touching the object and the user clicks the first wand function key.

6.2. Tunnel simulator

The tunnel simulator simulates the response of the tunnel as a result of user interaction. It consists of script for modeling the physical behavior of the major parts of the tunnel. A numerical physics-based model of each tunnel component is constructed. These include:

- Main drive.
 - Drive fan
 - Drive lubrication system
- Boundary layer control system.
- Intake flaps.
- Ceiling and wall slots.
- Flow control vanes.

The model interacts with the state of the wind tunnel including, switches, knobs, and buttons, dials, etc. through sending and receiving events.

6.3. Human-like avatars

As shown in Fig. 11, two agents provide assistance to the user in the wind tunnel simulator:

- An ‘always on top of the screen’ agent or screen agent is always visible and provides guidance and answers to the user’s questions.
- An assistant in the environment directly assists the user by showing the correct procedure to perform the required process steps.

Both agents are controlled using the LEA engine. The two agents collaborate to help the user. For example, when the user needs help, he asks the screen agent a question. The screen agent will say the answer and display any visuals, which support the answer. The screen agent will also command the assistant in the environment to actually do a demonstration if necessary. Visual cues such as 3D flashing arrows are used in addition to the agent in the environment pointing to the correct objects in the environment. For example, in Fig. 11, the screen agent is instructing the user to ‘press the red isolate initiate button’ and, in turn, the agent in the environment shows the location of the button.

6.4. Training

The wind tunnel operations consist of three high-level processes: Pre-operations, Operations, and Post-operations. Each high-level process is divided into sub-processes.

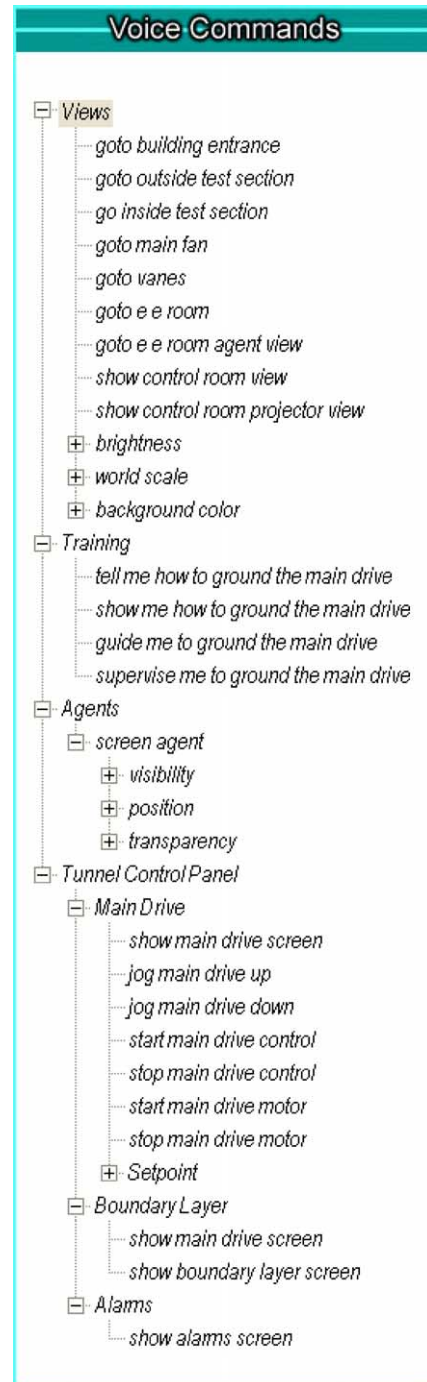


Fig. 14. Voice command hierarchy.

For example, the Pre-operations high-level process is divided into four processes: grounding the main drive, configuring the main drive lubrication, inspecting the wind tunnel, and connecting the main drive. During the training the operators practice their tasks in the simulated wind tunnel.

Fig. 14 shows a hierarchical list of the most frequently used voice commands. Users can click on a command instead of saying it. The Intelligent Agent provides effective training through various modes of assistance to the user, including:

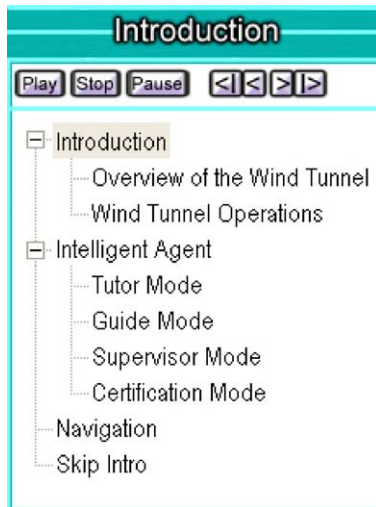


Fig. 15. Introduction to the wind tunnel lecture outline.

- **Introductory lecture.** Before the hands-on training starts, the screen agent can give the trainee a brief introductory overview of the wind tunnel operations. The overview of the wind tunnel is given in the form of a short lecture. The lecture outline is in the window labeled 'Introduction'. This allows the user to skip to specific points in the lecture or return to previous points or even skip the lecture entirely and proceed with the hands-on training right away. Fig. 15 shows an exploded view of the outline. In Fig. 7, the screen agent is describing the components of the wind tunnel to the user. In Fig. 16, the screen agent is giving the user an overview of the Pre-operations processes.
- **Process info.** In this mode the screen agent lists the steps of a process. For example, the user can ask the screen

agent: 'How do I operate the wind tunnel?' and the agent will answer: 'to start the tunnel, first prepare the main drive by performing the pre-ops and then start tunnel operations or ops. Finally, shut down the tunnel by performing the post ops.' User: 'What are the pre-ops?' Intelligent Agent: 'The tunnel pre ops consist of four steps:

- Ground the main-drive
 - Configure the main-drive lubrication system
 - Inspect tunnel circuit
 - Reconnect the main-drive to the cyclo-converter.
- **Process tutor.** In this mode, the agent shows the user how to perform a specific process. If the user asks: 'How can I ground the main drive?' the Intelligent Agent will ask: 'Would you like me to show you how or would you like me to guide you through the process?' If the user says: 'show me' then the screen agent responds by saying: 'my assistant will show you how to ground the main drive.' The agent inside the environment will perform the process steps while the screen agent speaks the step descriptions and the user watches.
 - **Process guide.** In the guide mode, the agent guides the user through the process. In the example above, if the user says 'guide me' then the agent goes into the guide mode. The screen agent tells the user step by step what to do and the user performs all the steps necessary to complete the process. The intelligent assistant in the environment will provide hands-on assistance to the user by showing the location and mode of operation of the controls required to perform the process. The agent will not go to the next step until the user says a command such as 'go on', 'continue', and 'proceed'. If the user makes a mistake during the guide mode, then the screen agent will

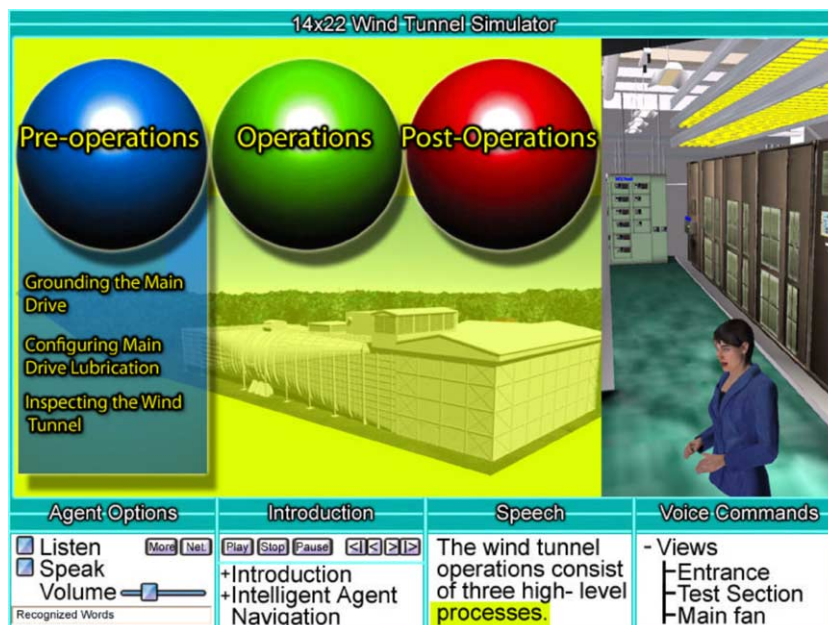


Fig. 16. IAF presenting the pre-operations process.

frown and change the tone of her voice to alert the user to the mistake. The agent's reaction becomes more dramatic if the user repeats the mistake. Finally, if the user repeats the mistake three times, the screen agent instructs the assistant in the environment to demonstrate to the user how to perform the process step correctly.

- *Supervisor mode.* The screen agent supervises the user as he/she performs the process steps. For example, the user can activate this mode by saying: 'watch me ground the main drive.' The IAF would let the user perform the steps necessary to complete the process unassisted and would list the user's mistakes if any at the end of the process.
- *Certification mode.* After the user completes the training, the agent can certify the user in the wind tunnel operations. In this case, the user tells the agent: 'Certify me in the pre-ops.' The agent would let the user complete all the sub-processes of the pre-ops and would tell him/her at the end that he/she is certified if no mistakes are detected. If the user makes any mistakes, then the agent will tell him/her at the end what the mistakes were and will suggest additional training.

7. Concluding remarks

An intelligent simulation environment that can be used for virtual-reality training on simulated engineering systems was described. The three key components of the environment are: an intelligent agent facility comprised of a rule-based expert system natural-language interface, a hierarchical process knowledge-base, and human-like virtual characters; a VE Engine; and a system controls simulator.

The intelligence in the environment is provide through an intelligent agent facility (IAF). Three main features of the IAF are: (1) the agent enables multimodal input from the user such as natural-language spoken or typed questions/commands, pointing and gazing at a certain object, mouse/wand clicking and dragging; (2) synchronized multimodal presentation of the training material, including, synchronized natural-language speech and written text, synchronized motions, gestures, and emotions of a photo-realistic human-like avatar, synchronized illustrations in the VE, including blinking pointing arrow and animations of the physical behavior of the engineering system; (3) integration with a virtual model of the engineering system.

The environment described herein uses the training strategy of 'learning by doing.' During the training the operators practice their tasks in a simulated environment. The agent communicates with the user using natural language, manipulates the simulator of the engineering system, and observes the user's manipulation of the simulator. The agent serve several roles in the environment, including: presenting an introductory lecture; tutoring; guidance; supervision; and certification. In the guide mode the agent gives the user positive or negative feedback after

each step depending on whether the trainee performed the step correctly.

Other related applications, of the proposed environment include process-troubleshooting, remote control of complex engineering systems, and virtual labs and classrooms.

Acknowledgements

The present research was supported by NASA Cooperative Agreement NNL-0-4A-A05A. The authors would like to thank Ajay Kumar, David Dress, Charles Fox, and Patsy Tiamsin of NASA Langley Research Center for providing the data for the 14x22 wind-tunnel virtual model and operational-processes; and Jose Bricio and Derek Wilson of old Dominion University, Center for Advanced Engineering Environments for creating the textured VRML models. The human-like avatars display engine was provided by Haptik Inc. Natural male and female text-to-speech voices were provided by NeoSpeech. IVRESS toolkit and LEA were provided by Advanced Science and Automation Corp.

Appendix A. Rules-attributes descriptions

Tables A1–A4.

Table A1
Word attributes

Attribute	Variables	Description
<i>Require</i>	found_score not_found_score ['word1' 'word2'...]	Looks for any of the words listed within square brackets. Adds found_score (generally a positive value) when a spoken command word is matched, or adds not_found_score (generally negative) if none of the words are matched. Commands are executed when the score value is greater than a threshold value
<i>Ignore</i>	['word1' 'word2'...]	List of words that may be included in the spoken command, but that do not add to the meaning. Those words are ignored and do not contribute to the score
<i>scoreOther</i>	Score	Adds score (generally negative) for other words that are neither required nor ignored words. Thus, if a command contains too many extraneous words then the agent will say 'your command is not clear'

Table A2
Commonly used variable manipulation attributes

Attribute	Variables	Description
<i>Get</i>	[Variable 'script']	Retrieves data from the VE using the specified script. The data is stored in the variable specified as a string
<i>Set</i>	[Variable 'value']	Sets the value of a variable to the specified value
<i>ReadNumber</i>	VariableName	Reads a number from the user's command and places the numerical value in variableName.
<i>readLinguisticNumber</i>	[valueVariable minVariable maxVariable normalVariable]	Converts a linguistic number to the correct range. normalVariable is the normalized linguistic variable (between 0 and 1). 'minVariable' and 'maxVariable' define the range of actual variable. The 'valueVariable' is the output number. This is used to convert an absolute linguistic number to its numerical value such as 'set pressure to high'
<i>readLinguisticIncrement</i>	[valueVariable minVariable maxVariable normalVariable]	Converts a linguistic increment to the correct range. normalVariable is the normalized linguistic variable (between 0 and 1). 'minVariable' and 'maxVariable' define the range of actual variable. The 'valueVariable' is the output increment. This is used to convert a linguistic increment to its numerical value such as 'increase pressure a little bit'
<i>incVarValue decVarValue</i>	'variableName, deltaVariable'	Increase or decrease the value of a variable by a value of deltaVariable
<i>incVarPercent decVarPercent</i>	'variableName, percentVariable'	Increase or decrease the value of a variable by a percentage of percentVariable
<i>incRangeValue decRangeValue</i>	'minVariable, maxVariable, deltaVariable'	Increase or decrease the range between minVariable and maxVariable by a value of deltaVariable
<i>incRangePercent decRangePercent</i>	'minVariable, maxVariable, percentVariable'	Increase or decrease the range between minVariable and maxVariable by a percentage of percentVariable
<i>incMeanValue decMeanValue</i>	'minVariable, maxVariable, deltaVariable'	Increase or decrease the mean value of minVariable and maxVariable by a value of deltaVariable
<i>incMeanPercent decMeanPercent</i>	'minVariable, maxVariable, percentVariable'	Increase or decrease the mean value of minVariable and maxVariable by a percentage of percentVariable

Table A3
Rule group hierarchy attributes

Attribute	Variables	Description
<i>connect</i>	USE GroupName DEF GroupName	Connects this rule to a rules group and then moves control to the rules contained in that group
<i>executeGroup</i>	USE GroupName DEF GroupName	Executes a rules group from within a rule and then returns to the calling rule

Table A4
Feedback attributes

Attribute	Variables	Description
<i>Feedback</i>	USE GroupName DEF GroupName	Uses this rules group for providing feedback to the user. Feedback is activated by setting <i>feedBackState</i> to 1
<i>FeedBackTime</i>	WaitTime	Sets the time between providing feedback responses to the user if <i>feedBackState</i> is 1
<i>FeedBackState</i>	0, 1	0 = turn off feedback, 1 = turn on feedback
<i>FeedBackCommand</i>	'command string'	Defines the feed back command for this rule. This property is not executed. It is checked by the feedback control subroutine to determine whether or not to provide this rule to the user as feedback. The rule is provided if it is not in the recent command history
<i>Question</i>	answerVariable waitTime 'question'	Asks the user a question and waits waitTime sec for the user response. Then puts the response of the user in the variable 'answer'
<i>Command</i>	'command string'	Executes a command as if it was spoken by the user.
<i>CheckRule</i>	yesRule waitTime 'command string'	Performs the following steps Waits 'waitTime' for a response Checks the response against the yesRule If the response does not satisfies the yesRule, then the user has said 'no'. The command is added to the history but it is NOT executed. This insures that the agent does not ask that question again. Then, the rule is execution is ended

(continued on next page)

Table A4 (continued)

Attribute	Variables	Description
<i>CommandOnRule</i>	waitTime yesRule 'command string'	If the response satisfies the yesRule, then the rule execution continues Combines the function of <i>checkRule</i> and <i>command</i> into one function. The following steps are performed Waits 'waitTime' for a response. If the response does not satisfies the yesRule, then the user has said 'no'. The command is added to the history but it is NOT executed. This insures that the agent does not ask that question again. Then, the rule is execution is ended If the response satisfies the yesRule, then the 'command string' is executed as if it was spoken by the user
<i>Wait</i>	WaitTime	Waits a certain amount of time.
<i>ExitRule</i>		Exits the rule
<i>TestNames</i>	[testNumber namesListVariable groupName]	This property can be used to provide feedback from the VE to the agent (such as pointing feedback). The VE returns a list of words in the 'namesListVariable' (which can be obtained using the <i>get</i> property). Each item in the list can then be appended to the command and tested as a valid command starting from the specified 'group'. If the command is valid (score > 70), then it is executed. Only the first 'testNumber' of names are tested. Note that only the command with the highest score is executed. If all the commands have the same score, then the command corresponding to the first name in the list is executed

Appendix B. Hierarchical process information attributes

Tables B1–B3.

Table B1
Process attributes

Attribute	Variables	Description
<i>Objective</i>	'objective string'	Process objective
<i>Message</i>	'message string'	Message spoken at the start of the process execution
<i>continueRule</i>	USE continueRule DEF continueRule ...	Specifies what rule to use in this process to tell the agent to continue to the next step
<i>SkipRule</i>	USE skipRule	Specifies what rule to use in this process to tell the agent to skip the next step
<i>BackRule</i>	DEF skipRule... USE backRule	Specifies what rule to use in this process to tell the agent to go back to the previous step
<i>YesRule</i>	DEF backRule ... USE yesRule	Specifies what rule to use in this process to respond to the agent in the affirmative
<i>noRule</i>	DEF yesRule ... USE noRule	Specifies what rule to use in this process to respond to the agent in the negative
<i>abortRule</i>	DEF noRule ... USE abortRule	Specifies what rule to use in this process to tell the agent to abort this process
<i>pauseRule</i>	DEF abortRule... USE pauseRule	Specifies what rule to use in this process to tell the agent to pause this process
<i>tutorTime</i>	DEF pauseRule...	
<i>guideTime</i>	Time	Pause time in seconds between steps for tutor mode
<i>steps</i>	Time	Pause time in seconds between steps for guide mode
<i>preConstraints</i>	[]	List of the process steps
<i>postConstraints</i>	[]	List of the process preconditions
	[]	List of the process post-conditions

Table B2
Step attributes

Attribute	Variables	Description
<i>action</i>	'command string'	Specify an action command string. The string is first spoken by the IAF, then it is executed as if it was command given by the user
<i>NoAction</i>	'command string'	Specify a passive action string. The string is only spoken by the IAF. This can be used for giving passive commands to the user such as observing or verifying things in the VE (e.g. 'verify that the red LED indicator is illuminated')
<i>preConstraints</i>	[]	Process/step precondition
<i>postConstraints</i>	[]	Process/step post-condition
<i>RunScript</i>	'script'	Sends a script to the VE. This is useful for displaying objects in the VE while the user is listening to or executing the step. For example an arrow can be displayed to point the user to the correct button to press

Table B3
Constraints attributes

Attribute	Variables	Description
<i>Condition</i>	'script'	This script is send to the VE and is supposed to return a string back to IVRESS/Agent. The string is then compared against the 'trueValue' property. If the two strings are the same then the condition is satisfied
<i>trueValue</i>	'trueString'	String to be compared with the string returned from 'condition' to determine if the constraint is satisfied
<i>caseFalse</i>	'message'	Agent spoken message if the condition is not satisfied
<i>caseTrue</i>	'message'	Agent spoken message if the condition is satisfied

References

- [1] Ong SK, Mannan MA. Virtual-reality simulations in a Web-based interactive manufacturing engineering module. *Comput Educ* 2004; in press.
- [2] Lin F, Ye L, Duffy VG, Su C-J. Developing virtual environments for industrial training. *Information Sci* 2002;140:153–70.
- [3] Arjona Lopez MA, Flores CH, Garcia EG. An intelligent tutoring system for turbine startup training of electrical power plant operators. *Expert Syst Appl* 2003;24:95–101.
- [4] Loftin RB, Wang L, Baffes G, Hua G. An intelligent agent training system for space shuttle flight controllers. *Telematics Informatics* 1988;5(3):151–61.
- [5] Vora J, Nair S, Gramopadhye AK, Duchowski AT, Melloy BJ, Kanki B. Using virtual reality technology for aircraft visual inspection training: presence and comparison studies. *Appl Ergonomics* 2002;33: 559–70.
- [6] Mayros J, Kesavadas T, Chugh K, Joshi D, Ellis DG. Utilization of virtual-reality for endotracheal intubation training. *Resuscitation* 2003;59:133–8.
- [7] Nakao M, Oyama H, Komori M, Matsuda T, Sakaguchi G, Komeda M, Takahashi T. Haptic reproduction and interactive visualization of a beating heart for cardiovascular surgery simulation. *Int J Med Informatics* 2002;68:155–63.
- [8] Tsai M-D, Hsieh M-S, Jou S-B. Virtual reality orthopedic surgery simulator. *Comput Biol Med* 2001;31:333–51.
- [9] Jennings NR, Wooldridge MJ. Applications of intelligent agents. In: Jennings NR, Wooldridge MJ, editors. *Agent technology foundations, applications, and markets*. Berlin: Springer; 1998.
- [10] Bui T, Lee J. An agent-based framework for building decision support systems. *Decision Support Syst* 1999;25:225–37.
- [11] Rich C, Lesh N, Rickel J, Garland A. A plug-in architecture for generating collaborative agent responses. *Autonomous Agents and Multi-Agent Systems*, Bologna, Italy; July 2002.
- [12] Rickel J, Johnson W. Animated agents for procedural training in virtual reality: perception, cognition and motor control. *Appl Artif Intell* 1999;13(4–5):343–82.
- [13] Rickel J, Marsella S, Gratch J, Hill R, Traum D, Swartout W. Toward a new generation of virtual humans for interactive experiences. *IEEE Intell Syst* 2002;32–8.
- [14] Guinn CI, Montoya RJ. Natural language processing in virtual reality. *Modern Simulation Training* 1998;6:44–5.
- [15] Hubal R, Guinn C. A mixed-initiative intelligent tutoring agent for interaction training. *Intelligent User Interface Conference*; 2002.
- [16] Wasfy TM, Noor AK. Rule-based natural-language interface for virtual environments. *Adv Eng Software* 2002;33:155–68.
- [17] <http://www.ascience.com/Science/LEA.htm>
- [18] <http://www.ascience.com/Science/IVRESS.htm>
- [19] <http://www.microsoft.com/speech/download/sdk51/>
- [20] <http://neospeech.com/>
- [21] www.haptek.com
- [22] SRI International, SRI CommandTalk System, <http://www.ai.sri.com/natural-language/projects/arpa-sls/commandtalk.html>; 1999.
- [23] Garfield K, Franceschini R, Schricker S, Ulrich R, Alberdeston R, Grosse J, Dumanoir P, Comer B. Modeling voice input and output for individual combatant simulation. 10th Conference on Computer Generated Forces and Behavioral Representation, Orlando FL; May 15–17, 2001.
- [24] Moore R, Dowding J, Bratt H, Gawron M, Gorfu Y, Cheyer A. CommandTalk: a spoken-language interface for battle field simulations. *SRI Int* 1997.
- [25] ISO/IEC 14772-1: 1997 Virtual Reality Modeling Language (VRML97), The VRML Consortium Incorporated; 1997.
- [26] www.alias.com