LEA: Software System for Multimedia and Virtual-Reality Web-Based Education and Training

Article · January 2006

DOI: 10.1115/DETC2006-99292

Tamer Wasfy
Indiana University-Purdue University Indianapolis
96 PUBLICATIONS 1,129 CITATIONS

Proceedings of IDETC/CIE 2006
ASME 2006 International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
September 10-13, 2006, Philadelphia, Pennsylvania, USA

DETC2006-99292

LEA: SOFTWARE SYSTEM FOR MULTIMEDIA AND VIRTUAL-REALITY WEB-BASED EDUCATION AND TRAINING

Tamer M. Wasfy
Advanced Science and Automation Corp., Indianapolis, IN

ABSTRACT

LEA (Learning Environments Agent) is a web-based software system for advanced multimedia and virtual-reality education and training. LEA consists of three fully integrated components: (1) unstructured knowledge-base engine for lecture delivery; (2) structured hierarchical process knowledge-base engine for step-by-step process training; and (3) hierarchical rule-based expert system for natural-language understanding. In addition, LEA interfaces with components which provide the following capabilities: 3D near photo-realistic interactive virtual environments; 2D animated multimedia; near-natural synthesized text-to-speech, speech recognition, near-photorealistic animated virtual humans to act as instructors and assistants; and socket-based network communication. LEA provides the following education and training functions: multimedia lecture delivery; virtual-reality based step-by-step process training; and testing capability. LEA can deliver compelling multimedia lectures and content in science fields (such as engineering, physics, math, and chemistry) that include synchronized: animated 2D and 3D graphics, speech, and written/highlighted text. In addition, it can be used to deliver step-by-step process training in a compelling near-photorealistic 3D virtual environment. In this paper the LEA system is presented along with typical educational and training applications.

1. INTRODUCTION

The internet/web has the potential to dramatically increase the accessibility and convenience and decrease the cost of education and training. In order for that potential to be realized, internet/web education and training must provide information transfer and pedagogical benefit which are near (or in some cases better) physical lab/classroom education and training. The main difference between internet/web and classroom education is that the later is delivered using a real live human instructor. Humans are adapted to learn from other humans. A live human instructor in a classroom provides the following capabilities:

- Instruction is presented using natural-language speech including speech variation for example to stress important points.
- Instruction is presented in conjunction with hand and body gestures and lip-synching. For example, some recent studies suggest that gestures are important in human cognition, communication and learning across cultures [1]. Also, many studies suggest that lip-reading is important for speech understanding and cognition.
- The instructor synchronizes the presentation of the material with his speech, pointing, writing, drawing, gestures, and lip movements.
- The student can interrupt the instructor at any time to ask him/her to:
    - Explain a concept that they did not understand.
    - Provide more detailed explanation.
    - Repeat what they said.
    - Skip the explanation if the student already knows the material.
    - Slow down or speed up the presentation.
- The instructor can ask the student questions during the lecture to stimulate his/her thinking and to keep the student engaged in learning.
- The instructor can use real-life objects that he/she are explaining. The instructor can use, move, rotate, and disassemble the objects. For example, the instructor can explain the operating process of a machine by guiding the student step-by-step through the process steps.
- The instructor can guide the student step-by-step through solving any problem related to the course material.

A web-based system which includes the above combined capabilities has the potential to substantially reduce reliance and time requirements of human instructors. In addition, the web-based system will provide additional advantages and capabilities that real live instructors do not provide. Those include:

- The student is in complete control the pace of instruction delivery. The student can pause, rewind, repeat, skip, slow down, and speed-up the delivery of instruction. In addition, the student can be in control of the time and place of instruction delivery. A human instructor will have more restrictions on student control, especially if there is more than one student in the classroom.

- The instruction can be delivered in a much more easy-to-understand and compelling way. For example instead of hand-drawn illustrations drawn on a blackboard, the illustrations can be 3D animated photorealistic models. Also, virtual computer models can be easily and quickly disassembled, rotated, animated, or made semi-transparent to show detailed that would otherwise be hidden in a real machine.
- The instruction can be mass delivered over the internet with significantly less cost than a human instructor.

LEA is a web-based education and training system that comes close to providing the aforementioned combined capabilities. LEA's ultimate objective is to provide education and training capabilities that are near those provided by a one-on-one human instructor. The various components of LEA were presented to various degrees of detail in previous publications [2-6]. In this paper those components are integrated into a unified architecture. This paper is organized as follows. In Section 2 the architecture of LEA is presented. In Sections 3-5, the three components of LEA are described. Applications of LEA are presented in Section 6. Finally some concluding remarks are offered.

2. ARCHITECTURE OF LEA

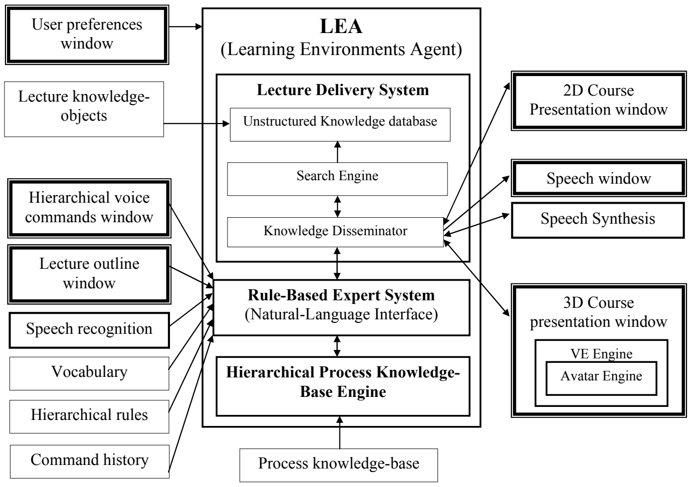


Figure 1 Architecture of LEA.

A schematic diagram of the architecture of the LEA system is shown in Figure 1. LEA consists of three main components:

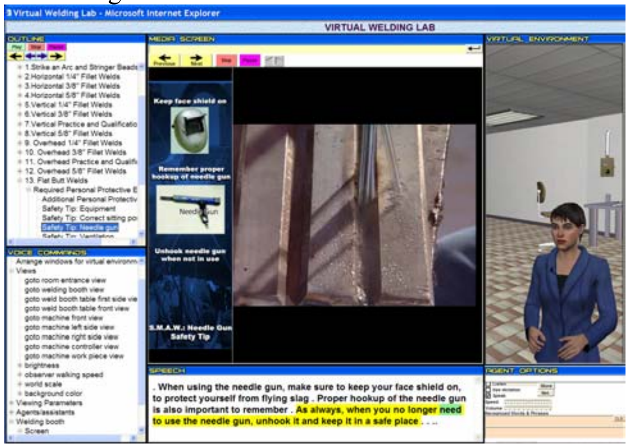
- *Lecture delivery system*. This includes an unstructured knowledge data-base engine, a search engine, and a knowledge disseminator. The knowledge data-base engine handles access to the knowledge-base, which consists of modular knowledge objects in the form of unstructured HTML, XML or text segments. Those segments can have hyperlinks referring to the locations of the multimedia pictures, movies, and 2D/3D animated illustrations files. The search engine is used to search the knowledge database for words or labels. The knowledge disseminator is the traffic controller that connects the user requests to the knowledge-base to produce the output of LEA in the form of instruction. The user requests are issued to the knowledge disseminator from the natural-language interface. The knowledge disseminator then sends the request to the search engine which returns back the appropriate knowledge object(s). The knowledge disseminator then sends these knowledge objects to the 2D and 3D course presentation windows, the speech window to be displayed, as well as to the speech synthesis engine to be spoken. Some of the graphical illustrations in the course windows can be clickable, in which case the knowledge-base request associated with the click event is sent to the knowledge disseminator which fetches the corresponding knowledge object(s).
- *Rule-based expert system (Natural-language interface)* (see Section 4). The rule-based expert system provides natural-language understanding. It processes sentences that are spoken or typed by the user or that are generated by clicking on an outline item (in the outline or hierarchical voice-commands windows), interprets them, and sends what it understood to the knowledge disseminator. For example, if it understands the user sentence as a question then it sends a command to search the knowledge database (see Section 4.2) to the knowledge disseminator along with a list of keywords to search for. If it understands the sentence as a command to make an object semi-transparent, then it sends a script to the knowledge disseminator to be sent via a network socket connection to the 3D or 2D display engines.
- *Hierarchical process knowledge-base*. This component provides step-by-step process training. It stores a hierarchical structured representation of a process as a list of sub-processes, steps, and pre- and post-process constraints. The step in-turn can also have pre- and post-step constraints. When the user issues a natural-language command to LEA to train him/her on a certain process, then the process knowledge-base engine walks the user step-by-step through the hierarchical process while checking for constraint violation.

In addition, LEA includes interfaces to the following components:

- *Speech recognition engine*. LEA has a built-in interface to Microsoft SAPI 5.1. Any SAPI 5.1 compliant speech recognition engine can be used for speech recognition. The user's speech is acquired using a good quality microphone. There are two speech recognition modes in SAPI, namely, continuous dictation mode (with a 30,000+ words vocabulary) and single word/short phrase mode (or command & control). The recognition rate of continuous dictation is typically 75-85% for most users, 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 (~1000 words/phrases) is used and the requirement that the user separates his/her words/phrases by a short 0.2-0.4 sec pause. LEA can use continuous dictation, single word/phrase recognition, or a combination of both for speech recognition. When the two approaches are combined, LEA 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 LEA 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 the regular conversation words (such as "show," "hide," "set," "is," "are," "in," "at,", etc.) all the key words of the educational or training application. LEA determines if a command has ended when the user says special execution words (such as "do it," "execute", or "answer").

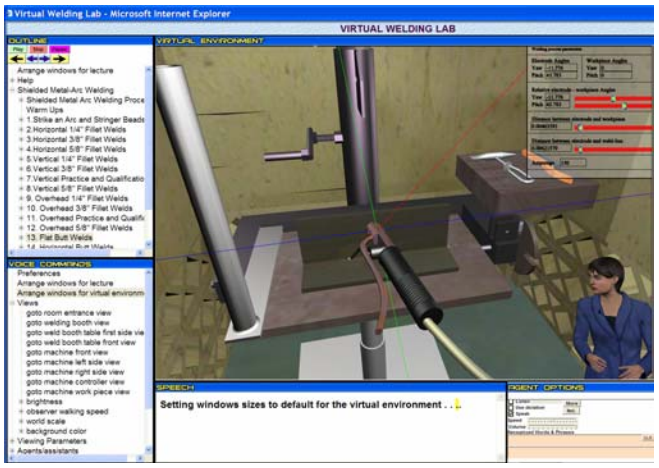
- *Speech synthesis engine*. Any Microsoft SAPI 5.1 compliant speech synthesis (text-to-speech) engine can be used for generating the agent's speech. LEA sends SAPI the text string to be spoken. SAPI uses the speech synthesis engine to generate the speech along with the following events:
    - *Start 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 knowledge disseminator to highlight the sentence that is currently being spoken as well as to run any scripts that is contained within the sentence. Those include scripts to synchronize 2D and 3D animated illustrations with the speech as well as the agent body and hand gestures.
    - *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 knowledge disseminator to highlight the word that is currently being spoken.
    - *Viseme events*. These events are generated in order to do the lip synchronization. They are passed by LEA to the 3D VE engine, which in turn passes them to the agent avatar display module to place the lips of the agent avatar in the proper position.
- *2D multimedia display engine*. Macromedia Flash 8 [7] is used to display animated 2D graphics, which include vector drawings, pictures and movies. LEA interfaces with Flash using JAVA-script through the web-browser.
- *Media player*. Movies can be displayed using any Media player web-browser plug-in such as Microsoft Media Player or Apple Quick Time Player. JAVA-script is used to control the movies (set the play frame/time, play, pause, and resume).
- *3D VE engine*. The IVRESS [8] object-oriented scene-graph display engine is used to display animated 3D virtual environments. Each object in IVRESS 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. IVRESS includes JAVA-script and VB-script to allow writing custom event handling routines. 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 devices (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 [9], pictures such as Bitmaps, PNG, JPEG, and GIF; and movies such as MPEG, AVI, and MNG. Four classes of objects are used to construct a VE in IVRESS:
    - 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.
- *Avatar display engine*. The animated human avatar is driven by the Haptek API [10] and is displayed using IVRESS [8] virtual-reality display engine. A wrapper IVRESS object encapsulates the Haptek API and allows displaying full body textured highly detailed male and female characters in a 3D virtual environment window. The 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.) which are lip and face positions for lip-synching. Also, the API allows setting the character's joints rotations and positions to any desired value. In addition, the IVRESS wrapper object allows animation of the character's hand motions by linear interpolation of the joint positions or angles. In order to effectively deliver the lecture, the agent avatar includes the following capabilities:
    - *Lip synching*. The lips of the virtual instructor avatar are synched with the speech. The text-to-speech engine "visemes" events are used to set the lip and mouth positions during speech and thus provide "lip synching". The visemes are fed from LEA (using SAPI) to the avatar (via IVRESS) using a TCP/IP network connection.
    - *Natural random and cyclic motions*. These include breathing, eye blinking, and natural random hand and body motions. Those motions are generated automatically by the Haptek engine.

- *Gestures*. These include hand gestures and pointing. These are embedded in the knowledge object HTML using a tag to send script to the avatar. An example tag is "<!IVRscript Agent.setSw = "**count L3" IVRend>". The script that is sent by LEA to the avatar (via IVRESS) is "Agent.setSw = "**count L3"", which makes the agent use the left hand to display a count of three.
- *Facial expressions*. These are embedded in the knowledge object HTML using the same script tag as the gestures.


Virtual Welding Lab - Microsoft Internet Explorer
VIRTUAL WELDING LAB
OUTLINE
MEDIA SCREEN
VIRTUAL ENVIRONMENT
Keep face shield on
Remember proper hookup of needle gun
Unhook needle gun when not in use
S.M.A.W.: Needle Gun Safety Tip
VOICE COMMANDS
SPEECH
AGENT OPTIONS

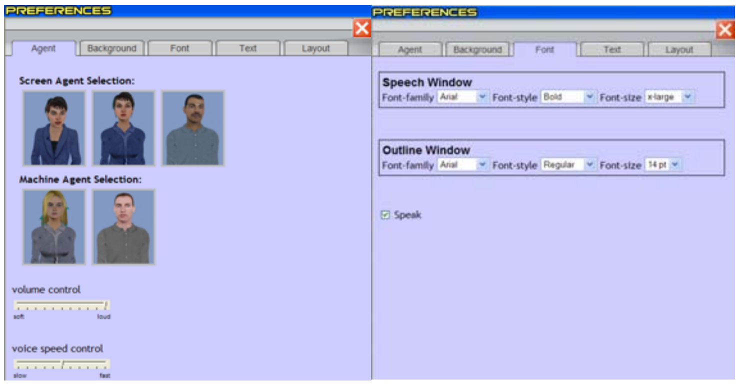
. When using the needle gun, make sure to keep your face shield on, to protect yourself from flying slag . Proper hookup of the needle gun is also important to remember . As always, when you no longer need to use the needle gun, unhook it and keep it in a safe place . . . .

Figure 2 Typical LEA screen for a web-based lecture delivery application on "Shielded Metal Arc Welding."



Setting windows sizes to default for the virtual environment . . .

Figure 3 Typical LEA screen for a web-based training application on "Shielded Metal Arc Welding."



Speech Window
Font-family Arial Font-style Bold Font-size x-large

Outline Window
Font-family Arial Font-style Regular Font-size 14 pt

☑ Speak

Figure 4 Typical tabs from the user's preferences window including Agent tab and font tab.

LEA runs inside a web-browser page. Figures 2 and 3 show typical LEA user interface screens for web-based education and training applications. The interface consists of the following windows:

- *Course presentation window* (Media Screen): shows the course material that includes text and animated 2D and 3D graphics synchronized with the speech. This window is a web-browser that can display: HTML with embedded pictures, movies, and Flash animations. Parts of the illustrations can be clickable such that when the user clicks, LEA jumps to a corresponding outline item.
- *Outline window*: shows a hierarchical-tree of the lecture outline. Branches can be expanded or contracted by clicking on the "+" or "–" buttons that are on the left side of a given branch heading. The student can either let the entire lecture play continuously until its conclusion or he/she can click on any item of the outline to directly go to that item. Navigation buttons at the top of the outline allow the user to: Go forward/backward one outline item; Go forward/backward one sentence; Play/Stop the lecture; Pause or resume playing the lecture.
- *Voice commands window*: shows a hierarchical-tree of voice commands. The user can click on the command from the list, or type/speak the command in natural-language.
- *Speech window*: displays what the agent is saying. The current sentence is highlighted. Also, the current word is highlighted with a different color to help the student follow the agent. The learner can read ahead or look at any text he/she missed in the speech window.
- *3D virtual environment window:* shows a fully interactive "video-game" quality 3D virtual environment that can be used for hands-on training (Figure 3). In addition, a photo-realistic animated virtual human avatar of the instructor can also be displayed within the virtual environment. The lips of the avatar are synched with the speech. The avatar can gesture, point, and display facial expressions (including anger, sadness, happiness, etc.). The user can translate, rotate, or scale the avatar.
- *Preferences window*: allows the user to change the instructor avatar, the voice type, the volume and speed of the speech, the color scheme of the windows, the font sizes, and other display options for the course (Figure 4).

3. UNSTRUCTURED KNOWLEDGE-BASE ENGINE

Unstructured knowledge is natural-language knowledge that is in sources such as scientific papers, documents, web pages, etc. Unstructured knowledge in LEA consists of "knowledge objects" that can be arranged in any order to form a lecture. A given knowledge object can be reused in multiple courses. A knowledge object is a short HTML/XML segment with embedded multimedia content and LEA-specific tags (it can also be a simple ASCII text segment). Self-contained knowledge objects are also used in the SCORM standard [11] which defines a SCO (Sharable content object) as the smallest stand-alone and meaningful component of a course that is reusable, interoperable and modular. SCORM is an emerging standard that is being adopted in online course development tools. Like LEA, the SCORM standard is geared towards single-learner, self-paced and self-directed training. The LMS (Learning Management System) in SCORM is responsible for controlling the delivery and organization of the SCOs. However, the SCORM standard still has a long way before realizing the stated goal of this paper "replacing the human instructor." Unlike LEA, SCORM does not include essential features such as 3D virtual environments, intelligent photo-realistic animated human instructors, process training capability, and a natural-language understanding capability.

LEA knowledge objects are stored in knowledge files. A knowledge-base file is simply a collection of knowledge objects separated by new-lines. Multiple knowledge-base files can to be loaded at the same time. A file is loaded using it's URL. Thus the knowledge files can be located anywhere on the web. LEA includes a text search capability to enable answering the user's questions from the unstructured knowledge-base.

Table 1 Most used LEA tags.

Tag	Description
<**!IVRrun** … *script*… **IVR**>	This runs a JAVA-script from the knowledge object on the course web-browser window.
<**!IVRrunBackground** … *script*… **IVR**>	This runs JAVA-script on the background web browser window**.**
<**!IVRrun0** *...script…* **IVR**> <**!IVRrun1** *...script…* **IVR**> <**!IVRrun*n*** *...script…* **IVR**>	This runs JAVA-script in the web browser utility windows 0, 1, 2, ...
<**!IVAcom** *…command...* **IVA**>	Runs a LEA specific command. These include commands to control the fonts, the window sizes and positions, etc.
<**!IVRscript** …script… **IVR**>	Sends a script to the agent avatar.
<**!IVRalias** *aliasname*>	This gives a knowledge objects an "aliasname" that can be used to call it from the outline. More than one knowledge object can have the same alias. In this case a request for this alias returns all the knowledge objects containing this alias. More than one alias can be associated with one knowledge object.
<**!IVRnoSearch**>	This is a switch that instructs LEA not to include this knowledge object in text searches which look for answers to the user's questions. This is useful to tag, for example, outline knowledge objects which, although may contain the search keywords, cannot be answers to questions.

The human instructor (subject-matter expert) constructs the knowledge-base by identifying the knowledge sources: papers, presentations, web content, etc. (these can also include structured knowledge bases such as an ontology or a concept map). These sources can be used directly as knowledge-base files. Alternatively, the instructor can do some editing to add multimedia content and to add LEA specific tags. For example, the instructor can create a knowledge object by cutting and pasting a text segment from a technical source into the knowledge-base. The text segment by itself is a knowledge object. Optionally, the instructor can embed multimedia content by adding HTML tags for the multimedia content using an HTML editor. Also, optionally the instructor can then add the LEA specific tags for controlling the agent avatar and synchronize the multimedia content with the speech. A list of the most used LEA specific tags that can be embedded in the knowledge object along with a brief description of the function of each tag is given in Table 1.

Figure 5 shows a typical multimedia HTML knowledge object. Figure 6 shows a snapshot of the course corresponding to this knowledge object. The knowledge disseminator sends this knowledge object as is to the course presentation window, which is essentially a web browser. The LEA-specific tags are ignored by the browser because they are in the form of comment tags. The knowledge disseminator also strips all the tags from the knowledge object (thus all what is left is the text of the sentences) and then sends this text to the speech window and to the text-to-speech engine to be spoken. The text-to-speech engine provides the sentence events that are used to synchronize the animations of the graphical illustrations (in the course presentation window), the agent gestures, and the sentence highlighting. The text-to-speech engine also provides the word events that are used to highlight the words and the viseme events that are used for lip-synching of the virtual instructor.

The graphical illustration in the knowledge object in Figure 5 is a FLASH movie. The movie is accessed using its URL and thus can be located anywhere on the web. Each sentence in the knowledge object is separated using a period. The first sentence is "In this brief introduction to machining centers, I will present the following." At the beginning of this sentence a script is sent to the Flash movie in the course presentation window to advance the Flash movie to the next synchronization point "intro". The script is: "*document.intro.TGotoLabel('/','intro'); document.intro.Play();*". The script is sent using the LEA tag "*<!IVRrun ...script… IVRend>*". This tag runs a JAVA-script segment on the course presentation window (the course presentation window is web a browser). Also, a script is sent to IVRESS to instruct the instructor's avatar to display a gesture indicated using the tag "*<!IVRscript ...script… IVR>*". The script that is sent to the avatar is: "*Agent.setSw = "lookeleft b"*" which instructs the avatar to look left. The next sentence is "1, general characteristics of modern CNC machine tools." At the beginning of this sentence the agent avatar is instructed to do a talking left hand gesture using the script "*IVRESSagent.setSw = "talkGestL2 start"*". The rest of knowledge object in Figure 5 consists of sentences with embedded JAVA-script to play the corresponding FLASH movie segment and script to control the agent gestures. At the end of the knowledge object the tag "IVRalias" defines an "alias" for this knowledge object. This alias is used in the outline to request this particular knowledge object using the "what" natural-language rule presented in Section 3.2. Note that more than one knowledge object can have the same alias. In this case a request for this alias returns all the knowledge objects containing this alias. Also, more than one alias can be associated with one knowledge object.

Another capability built into the LEA system is enabling the user to click on "hotspots" in the multimedia illustrations and jump to the corresponding lecture segment. This is enabled using the extra JAVA-script shown in cyan in Figure 5. This script uses the external script support capability in Flash through "FScommand" to instruct LEA to go to a specific knowledge object when the user clicks on the hotspot.

3.1 Lecture Outline

The outline window is shown on the left-hand-side in Figures 2 and 3. A typical outline file (as typed by the course instructor) is shown in Figure 7. The outline consists of the subject headings with child headings indented using tabs. Each heading is "piped" into a question that is sent to the knowledge disseminator (see Figure 1).

The instructor creates the course by writing the course outline shown in Figure 7. The instructor writes the subject headings and the corresponding question, which will be sent to the knowledge disseminator. The question can be in the form of a natural-language question in which case the search engine will search the knowledge base for the “best” answer to that question (see Section 4.2). Alternatively, LEA can search for a specific label in the knowledge-base and display the knowledge objects, which have this label. This is done using the special tag “<!IVRalias label_name>”. The search engine only returns the knowledge object(s) which has the label “label_name”. Thus, the lecture consists of the answers to the outline questions.

```
<div align="center">
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="100%"
height="100%" id="intro">
  <PARAM NAME=movie VALUE="Flash\MCintro.swf">
  <PARAM NAME=play VALUE=false>
  <PARAM NAME=loop VALUE=false>
  <PARAM NAME=menu VALUE=true>
  <PARAM NAME=quality VALUE=best>
</object><br>
<comment>
In this brief introduction to machining centers, I will present the following
<!IVRrun document.intro.TGotoLabel('/', 'intro');document.intro.Play(); IVRend>
<!IVRscript IVRESSagent.setSw = "lookleft b" IVR>.
1, general characteristics of modern CNC machine tools
<!IVRrun document.intro.TGotoLabel('/', '1');document.intro.Play(); IVRend>
<!IVRscript IVRESSagent.setSw = "talkGestL2 start" IVR>.
2, different types of Machining Centers
<!IVRrun document.intro.TGotoLabel('/', '2');document.intro.Play(); IVRend>
<!IVRscript IVRESSagent.setSw = "talkGestR3 start" IVR>.
3, Pallet Changers
<!IVRrun document.intro.TGotoLabel('/', '3');document.intro.Play(); IVRend>
<!IVRscript IVRESSagent.setSw = "talkGestL1 start" IVR>.
4, various Cutting Tools
<!IVRrun document.intro.TGotoLabel('/', '4');document.intro.Play(); IVRend>
<!IVRscript IVRESSagent.setSw = "lookleft a" IVR>.
5, Fixtures
<!IVRrun document.intro.TGotoLabel('/', '5');document.intro.Play(); IVRend>
<!IVRscript IVRESSagent.setSw = "talkGestL3 start" IVR>.
6, programming of Machining Centers
<!IVRrun document.intro.TGotoLabel('/', '6');document.intro.Play(); IVRend>
<!IVRscript IVRESSagent.setSw = "**Lpoint F_up" IVR>.
</comment>
</div>
<!IVRalias intro><!IVRnosearch>
<SCRIPT LANGUAGE="JavaScript" For="intro"
Event="FSCommand(command,args)"><!--
// command is 'call'
switch(args) {
   case 's1': label='general characteristics'     ; alias='modern_chrac'; break;
   case 's2': label='types of Machining Centers'; alias='MC_types';   break;
   case 's3': label='Pallet Changers'            ; alias='MC_pallet';  break;
   case 's4': label='Cutting Tools'              ; alias='MC_tools';   break;
   case 's5': label='Fixtures'                   ; alias='MC_fix';     break;
   case 's6': label='programming'                ; alias='MC_prog';    break;
   default: return;
}
document.cookie = label + ' | what is ' + '<!IVR' + 'alias ' + alias +'>';
//--></SCRIPT>
```

Figure 5 Typical knowledge object. Flash multimedia content tags are shown in blue. HTML tags are shown in green. LEA specific tags are shown in red. The lecture text is shown in black. Tags for the extra navigation script is shown in cyan.

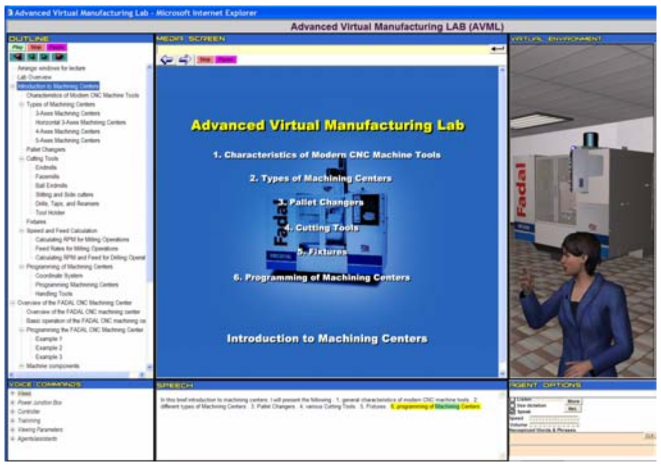

Advanced Virtual Manufacturing Lab - Microsoft Internet Explorer
Advanced Virtual Manufacturing Lab (AVML)
OUTLINE
MEDIA SCREEN
VIRTUAL ENVIRONMENT
Advanced Virtual Manufacturing Lab
1. Characteristics of Modern CNC Machine Tools
2. Types of Machining Centers
3. Pallet Changers
4. Cutting Tools
5. Fixtures
6. Programming of Machining Centers
Introduction to Machining Centers
VOICE COMMANDS
SPEECH
AGENT OPTIONS

Figure 6 Snapshot of the course corresponding to the knowledge object in Figure 5.

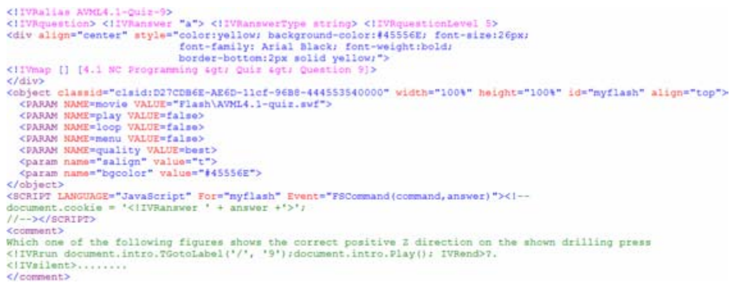
Lab overview | what is <!IVRalias overview>
Introduction to Machining Centers | what is <!IVRalias intro>
  Characteristics of Modern CNC Machine Tools | what is <!IVRalias modern_chrac>
  Types of Machining Centers | what is <!IVRalias MC_types>
    3-Axes Machining Centers | what is <!IVRalias 3axes>
    Horizontal 3-Axes Machining Centers | what is <!IVRalias H3axes>
    4-Axes Machining Centers | what is <!IVRalias 4axes>
    5-Axes Machining Centers | what is <!IVRalias 5axes>
  Pallet Changers | what is <!IVRalias MC_pallet>
  Cutting Tools | what is <!IVRalias MC_tools>
    Endmills | what is <!IVRalias endmills>
    Facemills | what is <!IVRalias facemills>
    Ball Endmills | what is <!IVRalias ballmills>
    Slitting and Side cutters | what is <!IVRalias sidecut>
    Drills, Taps, and Reamers | what is <!IVRalias drills>
    Tool Holder | what is <!IVRalias toolholder>
  Fixtures | what is <!IVRalias MC_fix>
  Speed and Feed Calculation | what is <!IVRalias MC_Speedfeed>

Figure 7 Outline of the course.

3.2 Testing Capability

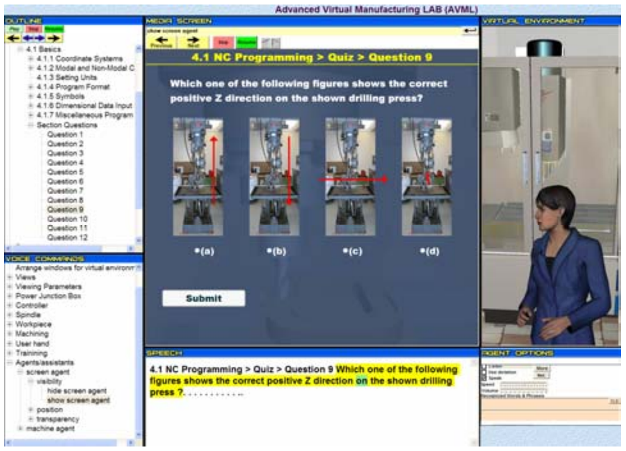
Figure 8 shows a source HTML/XML code of a LEA question knowledge object. Figure 9 shows a screen-shot of this questions displayed using LEA. The following LEA specific tags are used to define a question-type knowledge object:

- *<!IVRquestion>*: Sets knowledge object type to question.
- *<!IVRquestionScore 1>*: Defines the score given to the student for correctly answering this question.
- *<!IVRanswerType [string/real/integer]>*: Defines type of answer for the question. For multiple choice questions the answer type is *string* (‘A’, ‘B’ …).
- *<!IVRanswer "A">*: Defines the correct answer.
- *<!IVRanswerTol 0.1>*: Defines the absolute +/- tolerance for the correct answer for real number answers.
- *<!IVRquestionLevel 5>*: A number between 1 and 10 which specifies the difficulty level of the question.
- *<!IVRanswerTime 30>*: Defines the nominal answer time.



```
<!IVRalias AVML4.1-Quiz-9>
<!IVRquestion> <!IVRanswer "a"> <!IVRanswerType string> <!IVRquestionLevel 5>
<div align="center" style="color:yellow; background-color:#45556E; font-size:26px;
                           font-family: Arial Black; font-weight:bold;
                           border-bottom:2px solid yellow;">
<!IVmap [] [4.1 NC Programming &gt; Quiz &gt; Question 9]>
</div>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="100%" height="100%" id="myflash" align="top">
  <PARAM NAME=movie VALUE="Flash\AVML4.1-quiz.swf">
  <PARAM NAME=play VALUE=false>
  <PARAM NAME=loop VALUE=false>
  <PARAM NAME=menu VALUE=false>
  <PARAM NAME=quality VALUE=best>
  <param name="salign" value="t">
  <param name="bgcolor" value="#45556E">
</object>
<SCRIPT LANGUAGE="JavaScript" For="myflash" Event="FSCommand(command,answer)"><!--
document.cookie = '<!IVRanswer ' + answer +'>';
//--></SCRIPT>
<comment>
Which one of the following figures shows the correct positive Z direction on the shown drilling press
<!IVRrun document.intro.TGotoLabel('/', '9');document.intro.Play(); IVRend>?.
<!IVsilent>........
</comment>
```

Figure 8 LEA question knowledge object.



Which one of the following figures shows the correct positive Z direction on the shown drilling press?

4.1 NC Programming > Quiz > Question 9 Which one of the following figures shows the correct positive Z direction on the shown drilling press ?. . . . .

Figure 9 Snapshots from the AVML showing typical quiz questions implemented using the LEA system.

The student can enter the answer to the question using option boxes (for multiple choice questions), check boxes, or text boxes. LEA supports three types of testing modes:

- *Question by question practice*: In this mode when the student hits the “submit” button, the answer is checked against the correct answer and the agent says whether the answer is right or wrong.
- *Practice test*: In this mode a complete test is offered to the student. The instructor specifies the database of question knowledge objects, the difficulty level of the total test, and the minimum/maximum allowed question difficulty level. LEA then assembles a random test from the database which satisfies the instructor’s requirements. For each question, when the student hits the “submit” button, the answer is checked with the correct answer and the question score is added to the total score. At the end of the practice test the agent speaks the total score and shows a list of the student answers and the correct answer for each question.
- *Official test*: same as practice test except for the following:
  - The instructor specifies the total allowed test time. LEA keeps track of the time during the test.
  - The test score and list of student’s answers are electronically sent to the instructor.

4. RULE-BASED EXPERT SYSTEM FOR NATURAL-LANGUAGE UNDERSTANDING

The rule-based expert system converts the user’s natural-language speech or written text to commands and script that are sent to the knowledge disseminator (Figure 1). Details of the rule-based expert system for natural-language understanding were presented in [2, 4]. The expert system rule hierarchy consists of two types of objects: **Rule** and **Group**.

4.1 Group

The Group object allows grouping a set of rules, including other groups, in order to provide the ability to construct hierarchies of rules. Each group has a name and includes a list of rule names or group names, which are contained within the group (Figure 10).

4.2 Rules

A rule is an object that consists of a name and a list of properties and property values. The properties determine when a rule is triggered and the actions performed by the rule when it is triggered. A rule has the following types of properties:

- *Word properties* used to calculate 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 (Figure 11 and Table 2). The total score for a rule is equal to the summation of the *plusScore* for the *required* that are found, the *minusScore* for the *required* that are not found, and the *scoreOther* for the other words that are neither *required* words nor *ignored* words. If the *plusScore* for the *required* words is negative, this means that if those words are found then the score is reduced. Ignored words do not add or subtract from the score. Any rule with a score above a certain threshold (say 75) is triggered. For example the rule in Figure 11 is triggered by saying: “hide the controller” “turn off control box” or “switch off the controller”, etc.

```
DEF GroupName Group {
children [
        DEF rule1 RuleType1 { … }
        DEF rule2 RuleType1 { … }
        DEF rule3 RuleType1 { … }
        DEF group1 Group { … }
        USE rule4
        USE group2
        …
  ]
}
```

Figure 10 Rules group

```
# hide wind tunnel
DEF tunnel_hide RuleType1 {
   require     50 -100  ["hide" "turn off" "switch off"]
   require     50 -100  ["controller" "control box"]
   ignore         ["the" "a" "from" "me" "my" "view"]
   scoreOther  -10
   state1         "controller"
   reply          "Hiding controller"
   speak          "Hiding controller"
   script         [Cont_Controller.visible = 0]
}
```

Figure 11 Typical rule.

Table 2 Word properties

Attribute	Variables	Description
require	*plusScore* *minusScore* [“word1” “word2”…]	Looks for any of the words listed within square brackets. Adds *plusScore* when a spoken command word is matched, or adds minusScore (generally negative) if none of the words are matched.
ignore	[“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.
scoreOther	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.”

- *Script property* contains the script that is to be sent to the VE upon triggering the rule (Figure 11).
- *Output properties*. The *speak* and *reply properties* output spoken messages and on-screen messages, respectively (Figure 11).
- *Variable manipulation properties* are used to create and set the values of LEA variables. 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 properties* allow the rule to connect to other rules or other rule groups. This allows the formation of the rules’ hierarchy.
- *Feedback properties.* LEA uses the history of an expert user to intelligently provide useful feedback in the form of suggestions to novice users.
- *Process control properties* 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 5).
- *Knowledge-base search properties* allow the rule to initiate a knowledge base search. Figure 12a shows the rule that handles the user’s questions. This rule “requires” the user to say “what”, “tell”, “explain”, etc. to be triggered. Using this rule the system can understand questions such as: “what are the major characteristics of machining centers?” The rule-based expert system recognizes the question by the words “what”. Then the ignored words are stripped out of the question to yield the search string of keywords “major; characteristics; machining; centers.” LEA then retrieves the knowledge object(s) that closely answers this question. The command “searchKnowledge” (Figure 12a) instructs the search engine to search the knowledge base using the search string. The knowledge object(s) with the highest score is

offered as the answer to the user's questions. Note that if the user asks the same question twice the same answer is returned. In Figure 12b the rule for showing the user more information is shown. This rule is triggered when the user says, for example, "tell me more." The required words "tell" and "more" trigger the rule. The command "searchKnowledgeMore" instructs the search engine to find more documents that contain the last search string keywords ranked by relevance. Thus, the first time the user says "tell me more" the knowledge objects that give the highest search score are returned. The second time, the knowledge objects with the next highest score are returned and so on. Depending on the search score the agent will say before speaking a knowledge item an appropriate remark such as "Here is a possible answer" or "I am not sure about this answer." Also, when any knowledge object is presented to the student, it is time stamped. This time stamp is used to insure that the "searchKnowledgeMore" does not keep returning the same answer. Any knowledge item presented within a certain time (say 5 min.) is excluded from the search. If no more information is found, then a web search is initiated using the search keywords. LEA dynamically creates a web page of the top 4 links, displays it, and the virtual instructor speaks the content to the user.

(a)
```
DEF what_rule RuleType1 {
 require  100 -100 [ "what" "tell" "amplify" "explain" "elaborate"
                     "describe" "can" "how" "why" "when" "give detail"
                     "give details" ]
 require -100  0 [ "value" "color" "more" ]
 ignore ["i" "would" "to" "two" "like" "know" "a" "the" "me" "do"
         "does" "about" "on" "regarding" "concerning" "by" "you" ]
 searchKnowledge concept_Phrases concept_Objs 2
}
```
(b)
```
DEF moreinfo_rule RuleType1 {
 require  50 -100 ["tell" "amplify" "explain" "elaborate" "describe" "can"
                   "how" "why" "when" "give detail" "give details"]
 require  50 -100 [ "more" "detail" ]
 require -100  0 [ "value" "color" ]
 ignore ["i" "would" "to" "two" "like" "know" "a" "the" "me" "do"
         "does" "about" "on" "regarding" "concerning" "by"  "you"
         "information" ]
 searchKnowledgeMore concept_Phrases concept_Objs 3
}
```
Figure 12 (a) Typical question rule; and (b) typical "more information" rule.

- *State properties* define the state in which LEA is to be left after execution of a command. State attributes allow LEA 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." LEA 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."

The hierarchical rules approach takes advantage of the object-oriented hierarchical data structure of the 2D and 3D display engines. Typically the rules are organized 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. It then '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 rules containing actions that can be performed on the property.
- *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?"

5. PROCESS KNOWLEDGE-BASE ENGINE

The hierarchical process knowledge base enables LEA to train a student step-by-step how to perform a certain task. A process consists of a set of steps as well as other 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. Not that the process, step and constraint are objects. Figure 13 shows an example of a process, process steps and associated constraints. The process has the following types of properties:

- *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.
- *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 object properties 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 knowledge disseminator to be spoken and converted to a script through the expert system. The script is sent to the 3D or 2D display engines. The passive action is spoken by the agent and is not sent to the expert system (e.g. "verify that breaker 31 15 is closed by viewing that it is illuminated red" in Figure 13). A step also has a property "*runScript*" that specifies a script that is to be sent to the display engine. This allows sending script to the avatar to point at a specific object or displaying a pointing arrow. Similar to processes, steps also have pre-constraints and post-constraints.

The Constraint object properties include a property that sends a script requesting a state variable(s) from the display engine. 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.

```
DEF Proc_ground_maindrive Process {
  objective "ground the tunnel main drive"  ——————  Process objective
  message "Grounding or securing the main drive consists of 21 steps"
  tutorQuestion "Would you me to show you how to ground the main drive?"
  guideQuestion "Would you like me to guide you through grounding the main drive?"
  superQuestion "Would you like to ground the main drive?"
  infoQuestion  "Would you like me to tell you the steps for grounding the main drive?"
  continueRule USE continue_rule
  skipRule  USE skip_rule
  backRule  USE back_rule
  yesRule   USE yes_rule
  noRule    USE no_rule
  abortRule USE abort_rule
  pauseRule USE pause_rule
  doneRule  USE done_rule
  speakTime 3          tutorTime 10          guideTime 60——Process time
  steps [Step {state1 "e e room"                           constants
              action "go to the e e room"
             noAction "My assistant can guide you through the process."
             runScript "EEroomAgent.goto = EEroom_Loc;"
             }
         Step {action "go to the blue marshaling box"
             runScript "EEroomAgent.goto = EEroom_MarshalingBox_Loc;"
             }
          Step {noAction "verify that breaker 31 15 are closed by viewing that it is illuminated red."
             runScript "ree_marshalbox_ind3115.showArrow; "
             postConstraints [DEF cst_ree_marshalbox_ind3115 Constraint
                              {condition "IvServer.sendString = ree_marshalbox_ind3115.value;"
                               trueValue "1"
                               caseFalse "An error was detected." "Breaker 31 15 is open."
                                         "Please perform procedure x to close breaker 31 15."
                              }
                          ]
             }
          Step {action "turn marshaling box key to local"
              runScript "ree_marshalbox_key.showArrow;"
              postConstraints [DEF cst_ree_marshalbox_key_local Constraint
                              {condition "IvServer.sendString = ree_marshalbox_key.value;"
                               trueValue "1"
                              caseFalse "You made a mistake
                                   <!IVRscript tutor.setSw = 'emot eyesmad3 headshake'>."
                                  "You left the marshaling box key on remote
                                  <!IVRscript tutor.setSw = 'emot off' IVR>."
                                  "Marshaling box key should be set to local."
                              }
                           ]
             }
             ...........
             ]
      postConstraints [ USE cst_ree_pcc_indIsolatorGround_on
                        USE cst_ree_pcc_selswitch_remote           Process
                        USE cst_ree_cubicledoor_close              Post-constraints
                        USE cst_ree_cubicle_keyKG_cam_in
                       ]
}
```

Figure 13 Example of a hierarchical process object.

LEA process training supports the following training modes:

- *Process Tutor.* The virtual tutor performs the process steps while the user is watching. The user can pause/resume, repeat (go back) a step, or skip a step.
- *Process Guide*. The tutor guides the user step-by-step through the process. The tutor will not go to the next step until the user says a command such as "go on", "continue", or "proceed." The user has to perform each process step. The agent checks the process constraints to determine if the user performed the step correctly. If a constraint is violated, then the tutor instructs the user to repeat the step. If the user does not perform the step correctly three times in a row, then the tutor performs the 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 Certification*. The tutor instructs the user to perform the process. LEA keeps track of the user's mistakes and lists them at the end of the process. If no mistakes are detected, then the tutor certifies the user in the process.
- *Intelligent Virtual Assistant.* The virtual assistant asks the avatar to perform a process. The assistant performs the process while the user can either watch the agent or do something else.

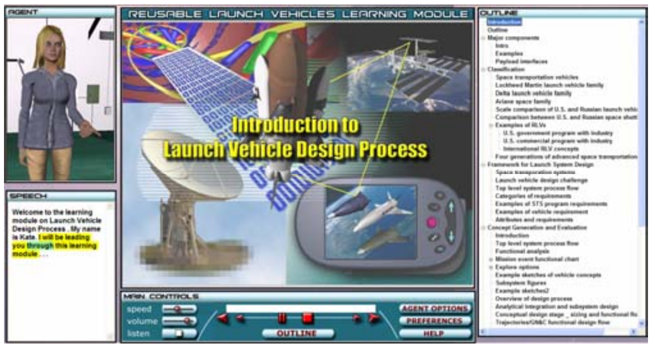
6. APPLICATIONS

Figure 2 shows a screenshot of a lecture driven using LEA on "Shielded Metal Arc Welding." The Figure shows the virtual instructor speaking the material. The virtual instructor lips and gestures are synched with the speech. The speech is displayed and highlighted on the bottom of the screen. The lecture outline is shown on the left-hand-side along with the current outline item that is being delivered. In the center of the screen a multimedia presentation that is synchronized with the instructor's speech and that includes text, pictures and a movie is shown. If the student pauses the lecture, the instructor's speech along with the multimedia illustrations and movie are paused.

Figure 6 shows a screenshot of a lecture on "CNC machining centers." Figure 9 shows a screenshot of a quiz question from that lecture. Figure 14 shows a screenshot of a lecture on "Launch vehicle design process." Note that the location of the instructor, tutor, and outline windows are different than in the previous figures. Also, note that for this lecture a custom navigation screen is used. LEA allows the user interface to be customizable. In addition, LEA windows can be moved and resized by the user.

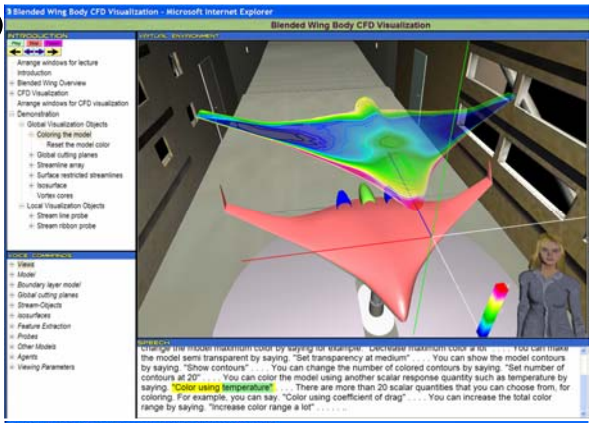
Figure 15 shows a lecture on "CFD visualization of a blended-wing body aircraft." In this lecture LEA is controlling IVRESS (3D VE engine) to display a 3D large-scale CFD dataset [3-5]. The virtual instructor gives the user a lecture on how to use natural-language to control the CFD visualization such as coloring/contouring the airplane using a scalar response quantity such as pressure (Figure 15a); displaying streamlines with animated particles (Figure 15b); displaying an iso-surface of a scalar response quantity; or flow feature extraction such as vortex cores and surface flow characteristics.

Figure 16 shows the instructor showing the student one of the steps a CNC milling machine startup procedure: LEA controls IVRESS to show the arrow pointing at the machine power lever and controls the avatar display engine so that the instructor can walk to face the machine power junction cabinet.

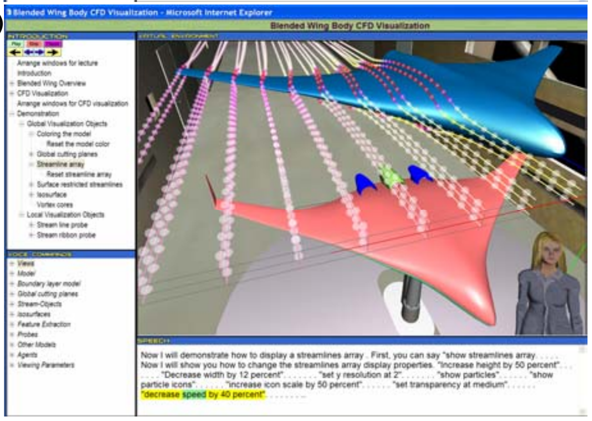

AGENT
REUSABLE LAUNCH VEHICLES LEARNING MODULE
OUTLINE
Introduction to Launch Vehicle Design Process
SPEECH
MAIN CONTROLS
speed
volume
listen
OUTLINE
AGENT OPTIONS
PREFERENCES
HELP

Welcome to the learning module on Launch Vehicle Design Process . My name is Kate. I will be leading you through this learning module . . . .

Figure 14 LEA screen of a lecture on launch vehicle design consisting of: course, agent, speech, outline & navigation windows

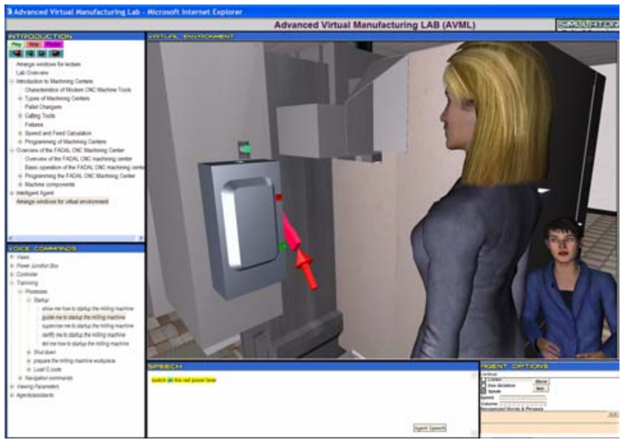
(a)

Blended Wing Body CFD Visualization - Microsoft Internet Explorer
Blended Wing Body CFD Visualization
INTRODUCTION
VIRTUAL ENVIRONMENT
VOICE COMMANDS
SPEECH

change the model maximum color by saying for example. "Decrease maximum color a lot" . . . . You can make the model semi transparent by saying. "Set transparency at medium" . . . . You can show the model contours by saying. "Show contours" . . . . You can change the number of colored contours by saying. "Set number of contours at 20" . . . . You can color the model using another scalar response quantity such as temperature by saying. "Color using temperature" . . . . There are more than 20 scalar quantities that you can choose from, for coloring. For example, you can say. "Color using coefficient of drag" . . . . You can increase the total color range by saying. "Increase color range a lot" . . . . . .

(b)

Blended Wing Body CFD Visualization - Microsoft Internet Explorer
Blended Wing Body CFD Visualization
INTRODUCTION
VIRTUAL ENVIRONMENT
VOICE COMMANDS
SPEECH

Now I will demonstrate how to display a streamlines array . First, you can say "show streamlines array. . . . . Now I will show you how to change the streamlines array display properties. "Increase height by 50 percent". . . . . . . "Decrease width by 12 percent". . . . . . . "set y resolution at 2". . . . . . . "show particles". . . . . . "show particle icons". . . . . . . "increase icon scale by 50 percent". . . . . . "set transparency at medium". . . . . . "decrease speed by 40 percent". . . . .

Figure 15 Typical screens form a LEA lecture on CFD visualization of a blended wing body aircraft.


Advanced Virtual Manufacturing Lab - Microsoft Internet Explorer
Advanced Virtual Manufacturing LAB (AVML)
INTRODUCTION
VIRTUAL ENVIRONMENT
VOICE COMMANDS
SPEECH
AGENT OPTIONS

switch on the red power lever

Figure 16 A snapshot taken during training a step of a CNC milling machine start-up procedure.

7. CONCLUDING REMARKS

LEA, a web-based software system for advanced multimedia and virtual-reality education and training, was presented. LEA provides the following education and training functions:

- Multimedia lecture delivery including synchronized: animated 2D and 3D graphics, speech, and written/highlighted text
- Virtual-reality based step-by-step process training in a compelling near-photorealistic 3D virtual environment.
- Testing capability.

LEA includes many of the capabilities that are currently only available through a human instructor such as: instruction delivery using natural speech and gestures; capability to answer the student's natural-language questions; and ability of the student to pace the instruction delivery.

ACKNOWLEDGEMENTS

Applications of LEA were supported by several federal and state research contracts and grants, namely: NSF STTR grants number 0339024 and 0521710; Indiana 21st Century Research & Technology Fund grant number 0043; Virginia Institutes of Excellence contracts; and NASA sub-contracts. The author would also like to acknowledge the collaboration with various education and research institutions including Paul D. Camp Community College, Franklin, VA and IUPUI, Indianapolis, IN; and companies including Northrop Grumman, Philip Morris and DuPont in developing various LEA applications. The author would also like to thank Haptek Inc. for providing the human avatars' display engine.

REFERENCES

1. Roth, W-M., "Gestures: their role in teaching and learning," *Review of Educational Research*, Vol. 71(3), pp. 365-392, 2001.
2. Wasfy, T.M. and Noor, A.K., "Rule-based natural-language interface for virtual environments," *Advances in Engineering Software*, Vol. 33(3), pp. 155-168, 2002.
3. Wasfy, T.M. and Wasfy, A.M., "Strategy for effective visualization of CFD datasets in virtual environments," ASME Paper No. DETC2003-48294, Proceeding of the DETC: *23rd Computers and Information in Engineering (CIE) Conference*, ASME DETC, Chicago, IL, 2003.
4. Wasfy, A.M., Wasfy, T.M. and Noor, A.K., "Intelligent virtual environment for process training," *Advances in Engineering Software*, Vol. 35(6), pp. 337-355, 2004.
5. Wasfy, H.M, Wasfy, T.M. and Noor, A.K., "An interrogative visualization environment for large-scale engineering simulations," *Advances in Engineering Software,* Vol. 35(12), pp. 805-813, 2004.
6. Wasfy, A.M., Wasfy, T.M., El-Mounayri, H., and Aw, D., "Web-based multimedia lecture delivery system with text-to-speech and virtual instructors," DETC2005-84692, *25th Computers and Information in Engineering Conference,* Long Beach, CA, 2005.
7. http://www.macromedia.com/software/, Macromedia Flash 8 Player.
8. IVRESS (Integrated Virtual Reality Environment for Synthesis and Simulation), http://www.ascience.com/ScProducts.htm, *Advanced Science and Automation Corp.*, 2006.
9. ISO/IEC 14772-1: 1997 Virtual Reality Modeling Language (VRML97), *The VRML Consortium Incorporated*, 1997.
10. www.haptek.com
11. Sharable Content Object Reference Model (SCORM), 2nd Edition, www.adlnet.org, 2004.