Behavior Development through Task Oriented Discourse

John R. Lee and Andrew B. Williams
Distributed Intelligent Agent Lab
Virtual Soldier Research Center
Center for Computer Aided Design
Dept. Electrical and Computer Engineering
University of Iowa
Email: {jrlee,abwill}@engineering.uiowa.edu

As digital human animation and simulation systems are coming into widespread use, scenario designers and digital animators must introduce specific tasks, procedures and animation sequences into a digital human's repertoire of capabilities. The parameterized action representation (PAR) architecture allows the animator to introduce dynamic behaviors through natural language instructions. However, this architecture lacks any natural language feedback which would allow the animator to deal with inconsistencies, incompleteness and conflicts within the behavior. In this paper we introduce an extension to a typical PAR architecture that provides that natural language feedback mechanism and define a set of task oriented discourses (TOD) that allow this mechanism to be used in synthesizing and managing behaviors. We demonstrate our behavior development techniques in two applications.

Keywords: behavior synthesis, behavior management, digital humans, task oriented discourse, trainee agents

Introduction

Recent advances in computer graphics and modeling have allowed the digital humans to take form. Although there is still room for advancement, digital humans are breaking ground in motion pictures and television, gaming, simulations, training programs, computer interfaces and more.

A recent article in Computer Graphics Weekly [1] lists the top 10 uses for virtual people (veople) as discussed by the Siggraph-2002 panel entitled "Digital Humans: What Roles Will They Play?". These uses are: **Set Extras**: Crowd scenes, stand-ins, stunt doubles and models for virtual reconstruction of scenes or events. **Leading Actors**: Actors and actresses for motion pictures and television. **Game Players**: Characters for video games, simulation and training programs. **Product Reps**: Personification of products and services. **Spokespersons**: Represent organizations and institutions. **News Commentators**: Virtual newscasters, interviewers and hosts. **Role Players**: A virtual character for engaging in role-playing dialogues. **Biomedical Surrogates**: Anatomic human bodies for testing medical procedures. **Avatars**: Characters that personify and represent you within a virtual environment. **Personal Assistants**: Digital characters that could perform as a personal assistant.

Although these uses are quite diverse, the research presented in this paper specifically benefits the roles in which (1) specific tasks or procedures must be developed or learned or (2) dynamic behavior must be changed or controlled at run time. For instance, a particular set-extra in a video game might be required to play the role of a blacksmith, this extra will have to develop domain knowledge, such as procedures for billowing and forging and will have to demonstrate a basic blacksmith behavior to be convincing.

Although game developers today can write specific scripts tailoring behaviors to agents in video games, our research focuses on using a more modular planning approach that (1) allows agents to reason and develop their own 'scripts'. (2) allows these 'scripts' to be non-homogenous, dependent on personality or other conditions of the agent or environment and (3) allows these scripts to be reused in a variety of applications.

We focus specifically on the problem of allowing a human operator to carry on a *task oriented dialog* with a digital human agent. Through such dialog, it is our hope that not only will the agent be able to learn new tasks, but also develop behaviors and even have an understanding of the intentions of various actions.

We begin by discussing background knowledge in this area, including the PAR architecture we are expanding upon, pedagogical and trainee agents, and task oriented discourse. We then show our system architecture and define a set of task oriented discourse structures that our system implements. We describe two applications that use our techniques and discuss results and future work.

Background

Agents are entities that exist within an environment. An agent perceives its environment through sensors or *perception*, decides how to act, and then acts upon its environment through effectors or *actions*. In our research, a digital human inherits the properties of a simple agent. In particular, we define its interaction in terms of actions which it can animate or perform in a virtual environment and perceptions it has about that environment.

Parameterized Action Representation

One important aspects of a digital human is how its behavior is modeled and how it decides what actions to perform in a given situation. Although, some systems allow scripting languages or even compiled behavior modules, our focus is on allowing the digital human to understand various actions in a given procedure and how they relate to one another. The *parameterized action representation* (PAR) [2, 3] provides such a mechanism.

PAR Objects

A PAR object gives a complete description of a given action. The action may have different participants, target objects, and details on how it is to be carried out, which are called parameters. The parameters of a PAR are detailed in [2]. A subset of these parameters are: **Objects**: A physical object in the virtual environment which can be manipulated or land-marked for use by an agent. **Agent**: Performer of the action. **Applicability Conditions**: Conditions required for this action to be applicable. This may refer to agent capabilities or physical object configurations. These are conditions in which an action or sub-action can make these conditions true. **Start/Result**: The given start and resultant time and state information. **Subactions**: A graph representing the sub-actions that are taken to perform the given action. There may be multiple ways of multiple steps to complete the action, and the sub-actions may include sub-actions themselves. **Core Semantics**: Several components of meaning of the action. These include Preparatory Specifications, Post-conditions, Motion, Force and Path components. **Purpose**: Purpose of the action, including conditions to achieve, actions to generate, and actions to enable. Terminating Conditions: Any conditions which will terminate the action other than the purpose of the action. **Agent Manner**: The constraints on the manner in which the action is to be performed.

There are two distinct forms for a PAR [2]: The *uninstantiated PAR* (UPAR) contains generic information on performing a given action such as the execution steps and pre-conditions and post-conditions. The *instantiated PAR* (IPAR) overrides a particular UPAR with information such as the performer, objects and manner of a specific action. In object oriented programming terms, a UPAR can be conceptualized as a class definition and an IPAR can then be conceptualized as a particular instantiation of that class.

PAR Architecture

The Parameterized Action Representation (PAR) architecture as described [2, 3, 4] is designed to allow a human operator to give natural language instructions to an avatar to carry out in a virtual environment.

This architecture uses a natural language to PAR translator which accepts natural language instructions and converts these instructions into IPARs using a specialized database called the Actionary(TM) which contains instances of all physical objects, UPARs and agents in the environment. These IPARs are fed into a planner, then to an execution engine, and finally an agent process. Eventually motion data is generated and visualized through digital human rendering.

PAR Research Areas

PAR research began with using natural language instructions to dynamically change agent behavior. Initially these instructions were either immediate (*Go to door*), or conditional (*When there is a driver, watch the driver*). [5] later extended this by adding a planning system and allowing these instructions to be negative directives or constraints (*Do not walk on the grass*). Other researchers are generating PARs from gestures and other non-verbal communications rather than through natural language alone.

Not only does PAR allow an agent to act, plan, and reason about actions, it also includes parameters for personality and mood in modeling agents. [6] discusses how PAR can be used in representing agent behaviors modified by personality and emotion.

These methods mainly concentrate on developing executable PARs (IPARs) given a library generic actions (UPARs). However, [4] suggests that the natural language interface may be used to construct new UPARs and [7] discusses the process of creating new UPARs through observing human motion information.

Types of Agents

Smart Avatars

In a video game, a player's avatar would normally be controlled by a joystick or game-pad with pre-defined movements assigned to configurable buttons. In motion pictures, an animated character may be controlled by motion capture devices or animator puppetry tools. However, when controlling multiple avatars, or avatars with a variety of possible actions, some form of enhanced interface is required for greater effectiveness.

A *smart avatar* is defined by [2] as a human representation controlled via instructions from a live participant. In other words, a smart avatar can execute orders given to it at runtime. We extend this by saying then that a *smarter avatar* should be able to provide feedback on given instructions to the live participant who gave those instructions, thus creating a means for task oriented discourse.

Pedagogical agents

A *pedagogical agent* is an agent specifically designed to teach. The most common role of a pedagogical agent is a tutor agent. A *tutor agent*'s objective is to assist students in learning how to perform procedural based physical tasks. Students can either observe tasks being explained by a tutor agent, or they can attempt to perform the tasks themselves, under the supervision of the tutor agent. One such system is illustrated in [8] where STEVE (Soar Training Expert For Virtual Environments) trains students on the operating procedures of naval equipment. STEVE focuses on integrating non-verbal queues and gestures in order to enhance the tutoring experience.

Another tutoring agent in [9], called PACO (Pedagogical Agent for COllagen), focuses on collaboration between agents and students.

Trainee Agents

Trainee agents are specifically designed for learning domain knowledge and procedures. There are three traditional methods for teaching an agent this information. *Programming by demonstration* allows an agent to learn by observing a human user performing the tasks. *Direct specification* allows the user to directly instill domain knowledge in an agent through some language or other knowledge medium. *Autonomous experimentation* allows the agent to experiment within the environment to learn tasks and their consequences.

[10] describes Diligent, a system which incorporates all three of these methods to create a trainee agent which can be taught by a human expert. The motivation for Diligent is to learn procedural task information that it can then teach as a pedagogical agent. Diligent is directly tied to the Steve system discussed in the previous section.

Task Oriented Discourse

Task Oriented Discourse is a task based formalization of the communication between two or more entities. Most research on task oriented dialogues involves a tutor agent providing guidance to a human student. Such research has been going on for over thirty years [11]. [12] discusses the task oriented dialogs of four separate pedagogical agents; TOTS, STEVE, PAT and PACO.

Other systems such as Task Oriented Dialog Understanding System (TODUS) [13] focus entirely on task automation, where the emphasis is placed on command execution rather than developing behaviors.

Task oriented discourse is an essential aspect to multi-agent system collaboration. Agents must coordinate their actions to work toward a common goal (cooperation), or they must adapt to one another for different goals (competition). Although many of the early collaboration

mechanisms dealt with the actions themselves, current approaches such as [14] try to develop the intentions of participants for better collaboration.

Task Oriented Discourse Architecture

The architecture we use for our task-discourse agent is outlined in figure 1. Specific modules of the system are described in detail.

Operator Interface

The operator interface provides a component in which single sentences of task-based dialog can be received by and displayed to a human operator. Although our current system implements this as a text based console window, there is a desire to integrate speech recognition and voice synthesis for a more natural interaction.

Language to PAR Translator

This component translates complex sentences of language into instantiated PARs so that the system can reason about the dialog in a more structured way. Uninstantiated PARs are provided by the task knowledge base and specific keywords are recognized by utilizing both the task knowledge base and environment annotation.

First, keywords like 'then, while, when' are used to detect the structure and break-up complex sentences into simpler sentences by creating PARs and utilizing fields (precondition, sequences) to represent this structure. Then keywords in these simpler sentences are translated into agents, objects, actions, and manners to be used for behavior execution. At the end of the translation process all keywords should have been used or discarded.

PAR to Language Translator

The PAR to language translator is responsible for transforming PAR, given a particular context into a sentenced statement to represent the personified statement of an agent. This is the crucial component of turning a command driven system into a command discourse system.

Task Knowledge Base

The task knowledge base contains both uninstantiated PARs and knowledge about perception, state and action reasoning. Due to space considerations, the knowledge base is only briefly discussed.

Uninstantiated PARs represent generic actions and are organized by the knowledge base as a hierarchy, with specialized actions inheriting from more abstract representations. For example, the action of opening a door has certain properties, such as the requirement that the door is closed, and the consequence that the door will be open. However, an automatic door, a door that swings freely and a door with a handle all have different procedures to open them. All of these procedures can inherit from the core action of opening a door. The core action can be used in higher level procedures such as entering a room where the details of how to open the door can be hidden.

The knowledge base contains information for reasoning about perceptions, states and actions. For example, a firing range can be considered safe if no one is shooting and the all-clear signal has been broadcast. This allows certain procedures to require that the firing range is safe, without having to know the details about that requirement.

Environment Annotation

Environment annotation is an important aspect of designing a virtual simulation in which agents can fully interact. If a scenario designer imports the model of a chair, how should the agent see it as such? Rather than recognize the distinct properties of a chair, such as physical structure and shape, objects are annotated by a designer by giving it embedded attributes that represent knowledge about the object. Such knowledge includes the type of the object, names it might be called by human operators, and common sense information such as how to pick it up, or how to operate it. For example, a mug will have a grasping vector suggesting how an agent will orient its hand to pick up the mud by its handle.

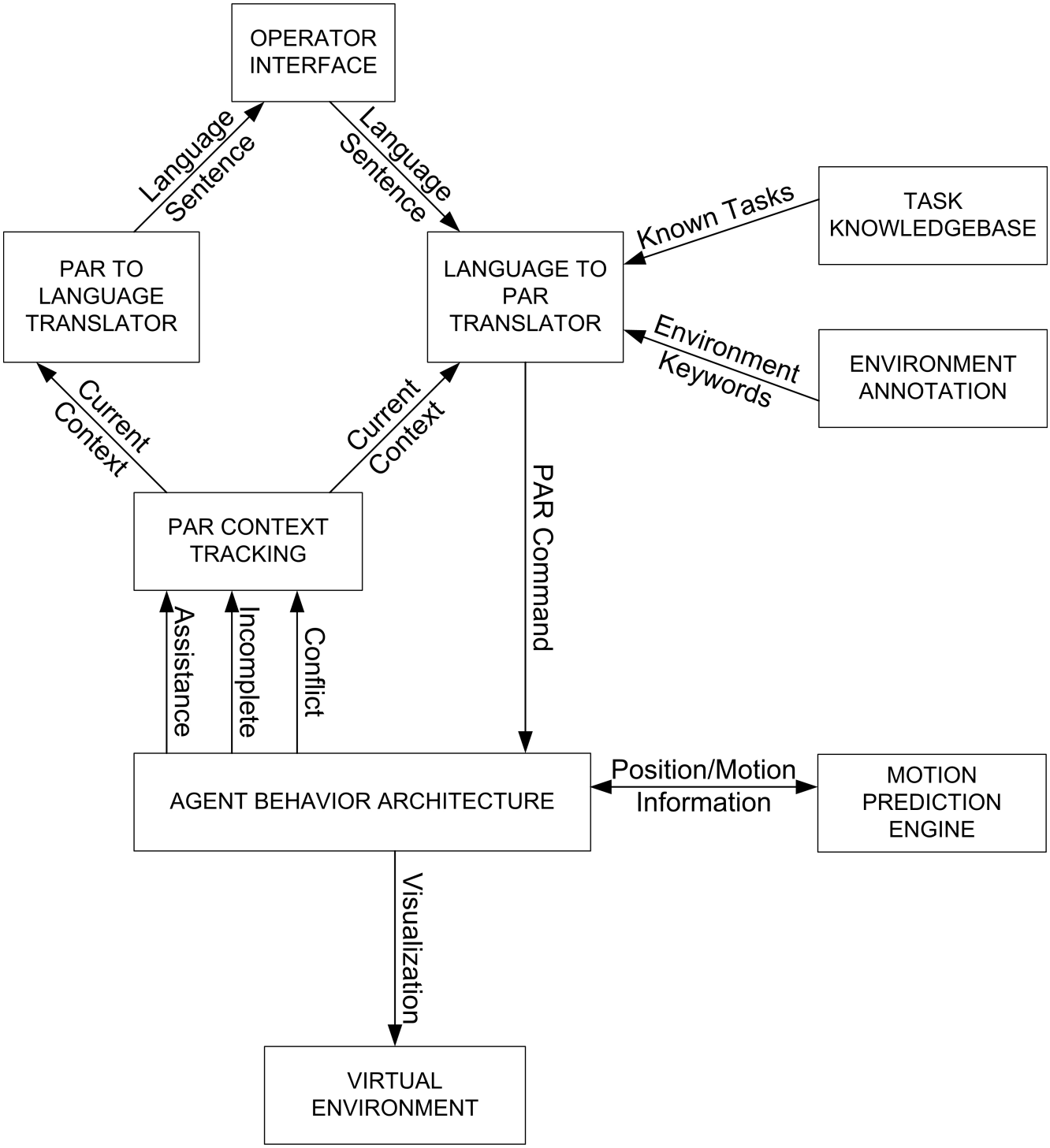

Figure 1: VSRC Task Discourse Architecture

These annotations are important in the language to PAR translator. They expand upon the recognizable keywords and properly translate them into the objects that they represent. Also, animation and modeling information in PARs will use attributes to produce more realistic behavior.

PAR Context Tracking

The context tracker receives PARs or sets of PARs that are incomplete, in direct conflict, or otherwise not executable (perhaps the system does not know how to carry out a particular command). It then creates a context based on these PARs and sends it to the PAR to language translator to inform a human operator of the problem. It also provides the language to PAR translator with the current context so that the translator may attempt to apply incoming dialog to a previously mentioned context.

Agent Behavior Architecture

The agent behavior architecture simply executes PARs by breaking them down into performable actions and executing those actions. If a PAR is incomplete upon receiving it from the translator, if several commands are in conflict, if it does not know how to carry out a particular command, or is having difficulty with preconditions or requirements, then it will send the troubling information to the PAR context to ask for assistance or resolution from the human operator.

If a PAR is executable, it will generate the animations it should carry out by requesting position and motion information from the motion prediction engine. It then produces those movements in its virtual environment.

Motion Prediction Engine

The motion prediction engine assists digital humans in adding realistic behavior by providing realtime motion and position optimization routines. The Virtual Soldier Research Center [15] is one of the groups developing motion prediction routines, which allow the application to anatomically model digital humans and record feasibility, ergonomics and other human factors data. This module is outside the scope of this paper, but is included briefly because they generate animation information for rendering the digital human.

Virtual Environment

The virtual environment contains all motion and position information on all of the agents and objects in the animation. It is able to produce real time visualizations of the agent interactions, thus providing visual feedback to the human operator who can see the orders being carried out.

Implementation

We have used the VirTools[16] development package for our virtual environment. We have implemented PAR in both VirTools Scripting Language (VSL) and C++ based building blocks.

Language Based Control

Our digital human must be able to relate language instructions to performable actions and consequences. We do this by allowing the architecture of our agent to relate with PARs and performing translation to and from PARs, for a human operator. Our digital human takes the role of the smarter avatar, attempting to execute standing orders while keeping various constraints. Our agent is also capable of resolving conflicts, asking information about orders, and reasoning about the intentions of the human controller.

It is important to note that the research of this paper is not focused on the natural language processing aspects but on behavior development. Therefore, rather than dealing directly with natural language we have defined a language for the human operator, the details of which is beyond the scope of this paper.

Categorization

We define our task oriented discourse in terms of the following categorization of utterances (based in part on [17, 13]). A *command* (C) is an explicit action to be taken by the smart avatar. This command may be constraints or standing orders, both immediate and conditional. A *query* (Q) asks about state or environment information. An *informative* (I) provides necessary information about the environment in general or about a particular task. A *clarification* (L) provides information about resolving task conflicts, such as priority information.

Types of Discourse

In this section, we will define several types of discourse that we have enabled in our system. An example will be given to illustrate each type, and a basic rationalization of how each type is carried through will be provided.

All examples provided refer to the laboratory application described later in this paper. All instructions are given to Eva. Pierre is another agent in this environment. The environment layout is illustrated in figure 2.

In the symbol [1:2], the 1 represents who is generating the utterance, U for the human operator, E for Eva, and S is what is visualized; the 2, represents the utterance categorization as listed above. So if the user (U) were to give a command it would be labeled [U:C].

Simple Instructions

A simple instruction is the assertion of a direct command. An example of a simple instruction is given below:

```
[U:E] walk to keyboard.
```

The simple instruction is the lowest form of instruction to be entered, usually it will represent a single action and thus be translated into a single IPAR.

Complex Instructions

A complex instruction is the combination of several simple instructions through the use of connecting keywords. Two examples of a complex instruction is given below:

```
[U:C] walk to keyboard then walk to map.
[S   ] Eva walks to the keyboard.
[S   ] Eva walks to the map...

[U:C] close all door.
[S   ] Eva walks to door 1.
[S   ] Eva closes door 1.
[S   ] Eva walks to door 2
[S   ] Eva closes door 2...
```

The complex instruction is translated into a series of IPARs, interconnected in a structure that is based on the keywords used to interconnect the simple commands.

Information Query

A user may obtain information about the environment or information about the agent's current state. A request for such information is done through an information query. An example of an information query is provided below:

```
[U:Q] How many doors are open?
```

This is a simple case of asking the agent for environment information. Simple statements like "how many" are programmed to search objects that match the given information. In this case it looks through all objects recognizing keyword, 'door', and has the attribute of being 'open'.

Conditional Instructions

Conditional instructions allow actions to be executed or not executed based on conditional information about the agent or the environment. An example of a conditional instruction is provided below:

```
[U:C] if you are at keyboard then type on keyboard.
[S  ] Eva is typing on the keyboard.
```

These can be coded directly into IPARs or interconnected IPARs. The conditional requirements are coded into the structure of the IPAR directly.

Instructions with Memory

Instructions can include memory, for use in sequences of instructions. Often the memory is represented as object labels. An example of such a sequence is provided below:

```
[U:C] read message from screen then walk to pierre then tell pierre message.
[S  ] Eva reads message from screen.
[S  ] Eva walks to pierre.
[S  ] Eva tells Pierre message.
```

A non-physical object ”message” is obtained from the screen. That same object is referenced in a later action. This represents memory. This is coded directly into IPARs and subsequently executed.

Standing Orders

Standing orders are persistent instructions that must be constantly checked, constrained or executed. A few examples of standing orders are provided below:

```
[U:C] always stand by pierre.

[U:C] whenever door opens then walk to keyboard.
```

The standing order is generated into an IPAR and is asserted into the system as a either a constraint (as in the first case) or an order that is continually checked and executed (as in the second case).

Incomplete Instruction Resolution

When instructions are found to be incomplete, the system is capable of querying the user for more information. This is called incomplete instruction resolution and two examples is provided below:

```
[U:C] hide.
[E:Q] hide from what?
[U:I] pierre.
[S  ] Eva hides from Pierre.

[U:C] hide.
[U:Q] hide from what?
[U:C] walk to keyboard.
[S  ] Eva walks to keyboard.
```

In the incomplete instruction resolution the first command translates into an IPAR which gets sent to the behavior architecture. However, in this particular case a necessary requirement of the IPAR was not satisfied (that there is a target object). This IPAR was marked as *incomplete* and sent to the PAR context tracker. The context generated a message asking for the required information, which comes across as a query.

In the first case the user input, 'pierre', is not yet known to be an informative statement. It is translated into a PAR with no action word. This PAR is then compared with the context and found not to conflict. These two PAR's are merged into a single PAR which is subsequently executed.

In the second case, the user input, 'walk to keyboard', was translated into a PAR that conflicted directly with the context, thus it was deemed as a new statement, and was sent to execution engine.

Order Conflict Resolution

When the user inputs more than one standing order and those orders are found to be conflicting, the system is capable of notifying the user that there is a conflict. Currently, no planning is used, only current actions can produce a conflict. The user may then resolve the conflict. An example of order conflict resolution is provided below:

```
[U:C] always stand by pierre.
[U:C] hide from pierre.
[E:I] (Always stand by Pierre) (hide from Pierre) in conflict!
[U:L] hide
[S  ] Eva hides from Pierre.
[S  ] Eva walks to Pierre.
```

Assuming Pierre is not by the keyboard, the first two commands in this example are in direct conflict. The conflict is detected within the behavior architecture. These PARs are packaged and sent as *conflict* to the PAR context tracking, which in turn generates a message to the user.

The user then inputs a subsequent command. If the command is not in conflict with either PAR in the context, such as 'tell pierre message', then it is executed and the context is forgotten. The conflict is resolved by canceling the second command (hide from pierre).

However, if the new command is in conflict with either PAR in the context, than the command is deemed as clarification. The new command overrides any conflicting information in the context and is sent to execution. However, as soon as the command is completed, the standing order 'always stand by pierre' comes back into play and Eva moves back to Pierre. If the second command were (always hide from pierre) then she would stay hiding.

In addition, during conflict resolution there are two clarification commands: *neglect stand by*, in this case places a higher priority on completing the hiding task and a lower priority on completing the always standing by task. *forget stand by* cancels the standing order for standing by Pierre.

Perception Knowledge Generation

If the system does not know how to execute user instructions, it may ask the user for assistance. Any assistance the user provides can be learned. When information about the state of the object is learned then perception knowledge is generated. An example of perception knowledge generation is provided below:

```
[U:C] if door1 is secure then walk to keyboard
[E:Q] when is door1 secure?
[U:L] when door1 is closed and door1 is locked
[S  ] Eva walks to keyboard.
```

In this case new domain knowledge is introduced. The idea that the object door1 could be 'secure'. The system, not knowing how to deal with this state sends information to the context as *assistance*. When the subsequent user command is seen as an incomplete PAR (most likely 'if' or 'when' without any action information) it is deemed as clarification and is applied to the PAR in context. In this case, perception knowledge about merging state information about a door into a new state is learned and added to the Task Knowledge Base.

Action Substitution

If the system is incapable of executing a specific instruction, the user may suggest another instruction. The substitution of another instruction is learned. An example of action substitution is provided below:

```
[U:C] carry box to door1.
[E:I] the box is too heavy!
[U:C] instead push box to door1.
```

Action substitution presents alternate actions that can be performed in varying circumstances. This allows for a great deal of flexibility and adaption in behaviors of this domain.

The first order could not be carried out because the precondition for lifting the object could not be met. It was therefore sent to the context which generated the resulting informative message. The second command was recognized as an alternative not only because of the keyboard instead, but also because the generated PAR shared a great deal of requirements and consequences of the PAR then in context.

Prerequisite Action

If the system is incapable of executing a specific instruction due to environment states. The user may suggest an action or set of actions that can be taken to change the environment. The system learns that these actions can be used in order to allow other actions to be performable. Two examples of this is provided below:

```
[U:C] open door1
[E:I] Action Failed: can not open door1!
[U:C] first unlock door1
[S  ] Eva unlocks door1.
[E:C] now open door1
[S  ] Eva opens door1.

[U:C] open door2
[S  ] Eva walks to door2.
[S  ] Eva unlocks door2.
[S  ] Eva opens door2.
```

The prerequisite action allows for programming the applicability conditions of the UPAR for opening of a door. In this case the agent did not know that you can not open a locked door. However, through this discourse and the agent learns preparation activity for a particular action, which is then demonstrated.

The first command ’open door1’ fails and thus is sent to the PAR context tracking as ’assistance’. The second command, containing the keyword, ’first’, allows this action to execute and merges it with the context PAR. The third command, containing the keyword, ’now’, is a recognized as a repetitive form of the PAR in context. After realizing this repetition and the commands subsequent success, the command of ’unlocking’ the door is registered into the UPAR for ’open’/’door’ as a preparation step.

Consequence Implication

A user may show how certain environment states change by showing the environment states, performing an action or set of actions, and showing the resulting states. The system learns which actions can change the environment states. An example of this is provided below:

```
[U:I] door1 is not locked.
[U:C] lock door1
[S  ] Eva locks door1.
[U:I] door1 is locked
```

Consequence information is learned through performing an action and noting the change of the environment based on this action. This is tracked by making informative statements (observations) about the environment. Informative observations are sent directly to the context tracker because there is nothing to execute. The direct conflict between two observations (not locked - locked) demonstrates the change in the environment and is recorded in the Task Knowledge Base.

New Action Development

New actions may be introduced by the user. First the instruction is asked of the system. If the system does not know how to perform the instruction it will ask the user for assistance. The user may then explain how to perform this action. An example of action development is

provided below:

```
[U:C] secure room
[E:Q] How do I, 'secure room'?
[U:C] secure all doors
[S  ] Eva walks to door1.
[S  ] Eva locks door1...
[U:I] done
```

In this example, a brand new command 'secure room' is introduced. A PAR is created, however the agent architecture does not have the necessary information for executing it. It is sent to the PAR context tracker and marked as *assistance*. The PAR context tracker than produces the inquisitive message on how to perform the action. The user then provides a series of commands that will perform this procedure and ends this list with the 'done' informative statement. The system recognizes that the sequence has been completed and creates a new UPAR for this action and adds it to the Task Knowledge Base.

Example: Laboratory Virtual Environment

The Laboratory Virtual Environment is a simulation environment created with the VirTools development package. It demonstrates basic core behaviors that are programmed and scripted into the agents. Core actions are created using the VirTools AI package and adding several actions such as opening and locking doors. This environment was created to test our task oriented discourse techniques, but will soon be replaced to continue development on our human posture and motion prediction application.

In this environment, Eva and Pierre are two digital humans who work in a LAB. The environment is annotated with keywords and specific object properties.

Example: Human Posture and Motion Prediction

We are currently working on creating a digital human for simulation in industrial and military applications. Our digital human is capable of generating multiple constraint posture and motion prediction optimizations for realistic action behaviors. The system is designed to be used for ergonomics, feasibility and visualization.

Discussion

We have expanded the PAR architecture to create a feedback mechanism so that a digital human can produce messages about commands back to the user. We have developed a set of task oriented discourse situations and demonstrated how they can be used to create and manage dynamic behavior as well as teach domain knowledge to digital humans.

Future Work

Our task oriented discourse (TOD) is only a beginning and we plan to incorporate techniques discussed in Diligent [10] for capturing and learning actions in a virtual environment. This would allow the digital human to learn from observing other agents, as well as humans embodied in haptics. Eventually we would like to transform our trainee agent into an apprentice agent that is fully capable of developing domain knowledge, learning and reasoning about tasks and their intentions. It will also learn through not only observing humans and agents, but collaborating and interacting with them.

Acknowledgements

This research was performed at the Digital Human Laboratory of the University of Iowa, College of Engineering and Center for Computer Aided Design. Support for the laboratory's digital human research is provided by Digital Humans and Virtual Reality for Future Combat Systems under U.S. Army Contract No. DAAE07-03-Q-BAA1.

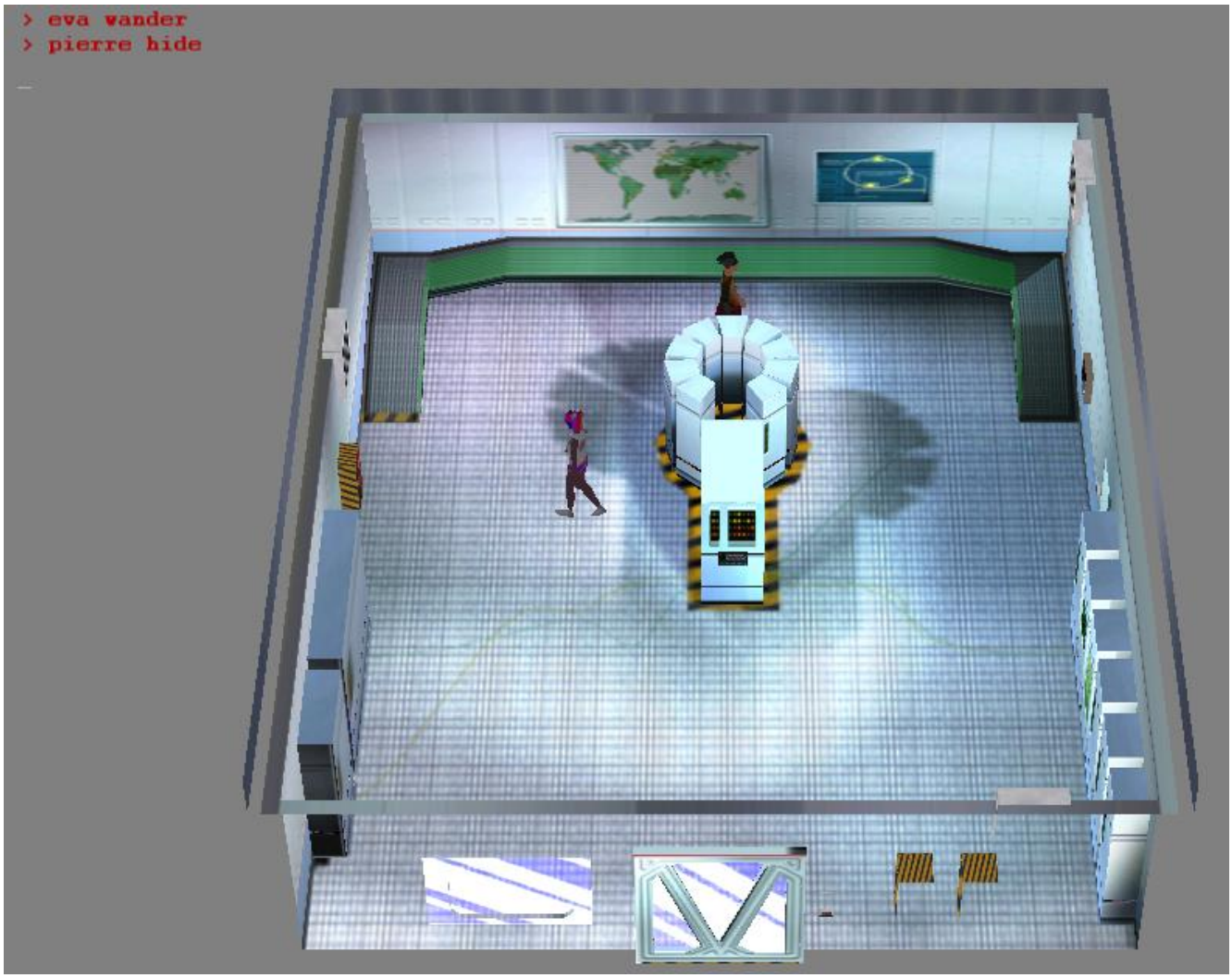

Figure 2: Laboratory Virtual Environment Application

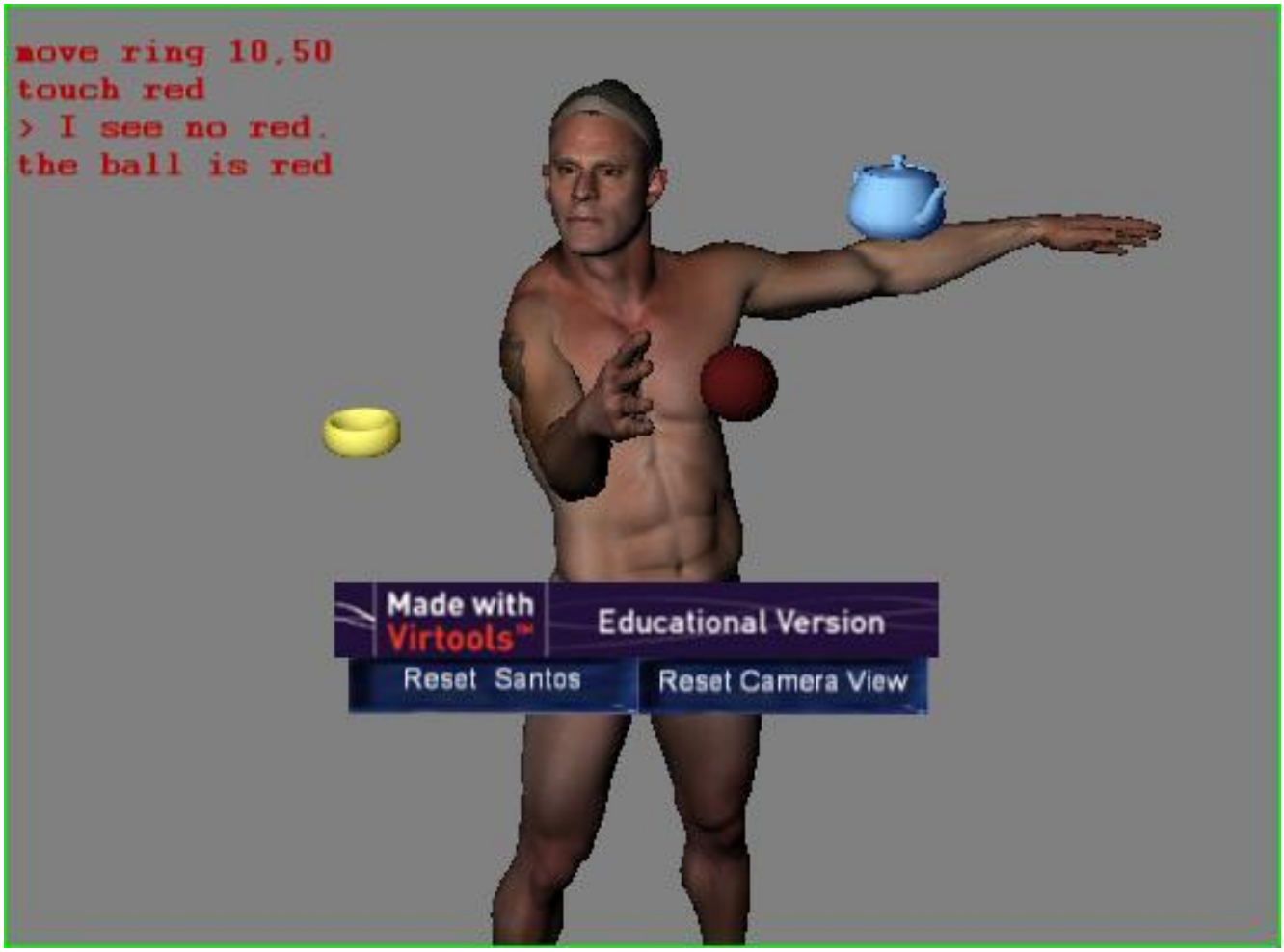

Figure 3: Human Posture and Motion Prediction

References

[1] Phil LoPiccolo. What are 'veople' for? *Computer Graphics World*, November 2002.

[2] Norman I. Badler, Rama Bindiganavale, Jan Allbeck, William Schuler, Liwei Zhao, and Martha Palmer. A parameterized action representation for virtual human agents. *Workshop on Embodied Conversational Characters*, October 1998.

[3] Rama Bindiganavale, William Schuler, Jan M. Allbeck, Norman I. Badler, and Aravind K. Joshi. Dynamically altering agent behaviors using natural language instructions. *Autonomous Agents*, June 2000.

[4] Rama Bindiganavale, William Schuler, Jan M. Allbeck, Norman I. Badler, Aravind K. Joshi, and Martha Palmer. Dynamically altering agent behaviors using natural language instructions. Autonomous Agents, June 2000.

[5] Jan M. Allbeck, Rama Bindiganavale, Karin Kipper, Michael B. Moore, William Schuler, Norman I. Badler, Aravind K. Joshi, and Martha Palmer. Authoring embodied agents' behaviors through natural language and planning.

[6] Jan M. Allbeck and Norman I. Badler. Toward representing agent behaviors modified by personality and emotion. In *Workshop Embodied conversational agents - let's specify and evaluate them!*, Bologna, Italy, 2002. AAMAS.

[7] R. Bindiganavale. *Building Parameterized Action Representations from Observation.* PhD thesis, University of Pennsylvania, 2000. Appears as Technical Report MS-CIS-00-17.

[8] Jeff Rickel and W. Lewis Johnson. *Task-Oriented Collaboration with Embodied Agents in Virtual Worlds.* MIT Press, 2000.

[9] Jeff Rickel, Neal Lesh, Charles Rich, Candace L. Sidner, and Abigail Gertner. Using a model of collaborative dialogue to teach procedural tasks. AIED Workshop on Tutorial Dialogue Systems, May 2001.

[10] Jr. Richard Angros, W. Lewis Johnson, Jeff Rickel, and Andrew Scholer. Learning domain knowledge for teaching procedural skills. Autonomous Agents and Multi Agent Systems, 2002.

[11] B. J. Grosz. The structure of task oriented dialogs. In *In Proceedings of the IEEE Symposium on Speech Recognition*, Pittsburgh, PA, April 1974.

[12] Jeff Rickel, Rajaram Ganeshan, Charles Rich, Candace L. Sidner, and Neal Lesh. Task-oriented tutorial dialogue: Issues and agents. In *Working Notes of AAAI Fall Symposium on Building Dialogue Systems for Tutorial Applications*, November 2000.

[13] Richard D. Amori. A natural language interface for task oriented activities. In *Proceedings of the third international conference on Industrial and Engineering Applications of Artificial Intelligence*, Charleston, South Carolina, 1990.

[14] K. E. Lochbaum. A collaborative planing approach to discourse understanding. Technical report, Harvard University, 1993.

[15] Virtual Soldier Research Center. Vsrc website. http://digital-humans.org.

[16] VirTools Development Team. Virtools dev 3.0. http://www.virtools.com.

[17] Ananlada Chotimongkol. Dialog structure for task-oriented conversations. April 2004.