3-1-1999

Smart Avatars in JackMOO

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Jianping Shi
University of Pennsylvania

Thomas J. Smith
University of Pennsylvania

John P. Granieri
University of Pennsylvania

Postprint version. Published in *IEEE Proceedings of Virtual Reality 1999*, March 1999, pages 156-163.
Publisher URL: http://dx.doi.org/10.1109/VR.1999.756946

Smart Avatars in JackMOO

Abstract
Creation of compelling 3-dimensional, multi-user virtual worlds for education and training applications requires a high degree of realism in the appearance, interaction, and behavior of avatars within the scene. Our goal is to develop and/or adapt existing 3-dimensional technologies to provide training scenarios across the Internet in a form as close as possible to the appearance and interaction expected of live situations with human participants. We have produced a prototype system, JackMOO, which combines Jack, a virtual human system, and LambdaMOO, a multiuser, network-accessible, programmable, interactive server. Jack provides the visual realization of avatars and other objects. LambdaMOO provides the web-accessible communication, programability, and persistent object database. The combined JackMOO allows us to store the richer semantic information necessitated by the scope and range of human actions that an avatar must portray, and to express those actions in the form of imperative sentences. This paper describes JackMOO, its components, and a prototype application with five virtual human agents.

Comments
Postprint version. Published in *IEEE Proceedings of Virtual Reality 1999*, March 1999, pages 156-163.
Publisher URL: http://dx.doi.org/10.1109/VR.1999.756946

Smart Avatars in JackMOO

Jianping Shi, Thomas J. Smith, John P. Granieri, and Norman I. Badler
Center for Human Modeling and Simulation
University of Pennsylvania
Philadelphia, PA 19104-6389
215-898-5862

Abstract

Creation of compelling 3-dimensional, multi-user virtual worlds for education and training applications requires a high degree of realism in the appearance, interaction, and behavior of avatars within the scene. Our goal is to develop and/or adapt existing 3-dimensional technologies to provide training scenarios across the Internet in a form as close as possible to the appearance and interaction expected of live situations with human participants. We have produced a prototype system, JackMOO, which combines Jack*, a virtual human system, and LambdaMOO, a multi-user, network-accessible, programmable, interactive server.* Jack *provides the visual realization of avatars and other objects. LambdaMOO provides the web-accessible communication, programmability, and persistent object database. The combined JackMOO allows us to store the richer semantic information necessitated by the scope and range of human actions that an avatar must portray, and to express those actions in the form of imperative sentences. This paper describes JackMOO, its components, and a prototype application with five virtual human agents.*

1. Introduction

Emerging education and training applications across the Internet require the delivery of distributed, interactive simulation scenarios with virtual human participants. Simulations have long been used as a learning tool, particularly in situations where there are safety and practicality concerns. Such simulations should be extended to a training environment in which the instructor and students are geographically dispersed across the Internet. Intuitively, building simulations supporting human-scale training scenarios will demand a level of realism and inter-agent communication unavailable in currently available simulation (or virtual reality) environments. In particular, we are interested in providing full body virtual human avatars, capable of sophisticated actions and interaction in a virtual environment [3, 1, 14]. Applications currently envisioned include training for checkpoint operations with several real and autonomous individuals, and complex multi-person aircraft maintenance activities.

We have developed a prototype system, JackMOO, which provides us with an environment in which the interesting questions of multi-user distributed training simulations and language control of avatar animation and interaction can be examined. While it is always possible to reduce the complexity of the human model to gain rendering speed or communication efficiency, we deliberately chose to work with a fully articulated and modeled body to explore interpersonal situations where participant avatars and synthetic agents may be interchanged.

2. Smart Avatars

Animating virtual humans involves controlling their parts at the graphical level via joint transformations (e.g., for limbs) or surface deformations (e.g., for face). Motion capture from live participants or algorithmically synthesized motions may be used to animate the 3D model. In real-time applications, avatars may be driven directly by a person's movements [9]. Directly captured motions are natural but lock the user into sensing equipment that may be cumbersome and limiting. Moreover, directly sensing motion is difficult to modify on-the-fly to achieve contextual sensitivity, and the user may be subject to common symptoms such as "groping" for virtual objects, jerky locomotion, annoying head movements, and the overall potential for "simulator sickness." Control without encumbrance leads to vision-based sensing or, as we are interested in exploring, language-based instructions.

We call an avatar controlled via instructions a *smart avatar*. Its actions may be portrayed through synthesized or captured motions and replayed within the current context. Proper movements require that the actions be parameterized in space, manner, and intention and, in turn, that the param-

eters are appropriately specified. Low level, smooth motion transitions are clearly important [21], but a higher level understanding (at the parameter level and beyond) will be essential to sequence and blend motions in human-like ways. Some parameters may come from the instruction itself, others from the local object context, and yet others from the avatar's available capabilities and resources. As the avatar becomes "smarter" it must make more contextual decisions about actions it must perform in order to achieve the requested goals.

We have explored the contextual control of virtual humans and increasingly smarter avatars in a number of experiments including: two person animated conversational "Gesture Jack" [10], Hide-and-seek game players [26], a real-time animated "Jack Presenter" [19], emergency medical training [11] and multi-user "JackMOO" virtual worlds [23]. In this last system we began to explore an architecture for interacting with virtual humans that was solely language-based in order to explicitly approach a level of interaction between virtual humans comparable to that between real people. We focused on instructions for physical action to bound the problem, to empower interesting applications, and to refine a representation bridging language and embodied action.

While simple interactions could be just menu-based, our thesis is that the full power of natural language interfaces for presenting instructions to smart avatars will provide benefits beyond GUIs. This component is just starting to be explored and is our focus here. Instructions are rich in directional references, terminating conditions, manner, purposes, and goals. Instructions are interpreted in a spatial context which establishes (at execution time and without further user input) appropriate access or locomotion paths, body and hand orientation, and action timing. What we learn from a natural language instruction interface would likely be combined with graphical or gestural interface components in a complete simulation training system.

3. JackMOO Architecture

JackMOO is a multi-user distributed virtual environment (DVE) aimed at supporting educational simulation and training scenarios. As shown in Figure 1, it adopts a server/client model. There are two kinds of system components on each client host:

- *Jack* – A 3-dimensional, articulated human figure creation and motion authoring system from Transom Technologies ([3, 1]), and
- JackMOO Client – A Java applet running within Netscape that mediates the flow of control between *Jack* and the LambdaMOO server; and provides the chief user interface to the system.

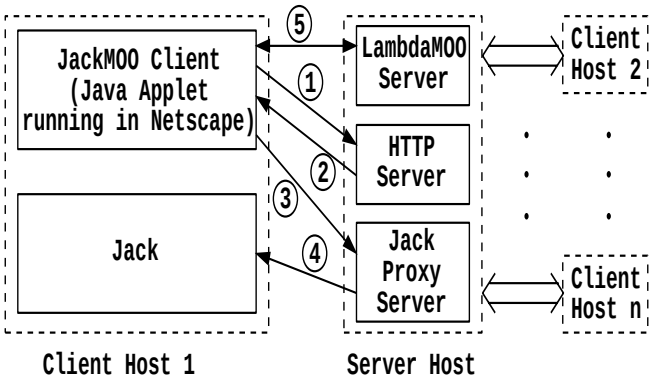

Figure 1. The architecture of JackMOO.

Three kinds of servers are running on the server host:

- LambdaMOO – A multi-user, network-accessible, programmable, interactive system developed by Pavel Curtis at Xerox Parc [12].
- HTTP Server – This server is used by the user to download the JackMOO client written as a Java applet. Currently one of the Java applet security restrictions is that an applet cannot make network connections except to the host that it came from [25]. So to be able to connect to the LambdaMOO server through the JackMOO client, we have to set up the HTTP server exactly at the same host where the LambdaMOO server is running.
- *Jack* Proxy Server – This server works around the strict network connection restriction of Java applet. The *Jack* proxy server resides at the same host as the HTTP server, and is responsible for relaying messages between the JackMOO client and the *Jack* system.

Provided that an end-user is using a workstation running a Java-enabled Netscape and *Jack* system, and all of the servers are running at a known host and listening to certain known ports, the procedure of initialization and logging into the virtual JackMOO world is as follows (see Figure 1):

1. The user asks for the JackMOO client by providing the URL to the HTTP server through Netscape.
2. The HTTP server sends back to Netscape the webpage containing the JackMOO client applet that the user asked for. Netscape then runs the applet.
3. The JackMOO client opens a network connection to the *Jack* proxy server.
4. Upon receiving the request from the JackMOO client, the proxy server opens a network connection to the *Jack* system, which is waiting for an external command

port connection. At this point, the JackMOO client and the *Jack* system are connected together through the proxy server that simply relays every message between them.

5. The JackMOO client opens another network connection to log in to the LambdaMOO server. After the connection is successfully established, the LambdaMOO server sends commands to the JackMOO client to print out welcome messages in the text output of the client, and to initialize the *Jack* system.

After the session of initialization and logging-in, the user can control his/her avatar in the JackMOO virtual world by typing instructions in natural language imperatives through the client. These instructions will then be sent to the LambdaMOO server and, if recognized, trigger corresponding LambdaMOO programs (called verbs) to execute. Verb execution dispatches a sequence of commands to the relevant JackMOO clients, which will do two things: execute non-*Jack* commands themselves, and send the *Jack* commands to their associated *Jack* systems through the proxy server. The latter run in the *Jack* systems to drive the animation.

As additional players connect with the system, their avatars appear on the *Jack* screens of all JackMOO-enabled players. Actions of an avatar are broadcast to each of the connected *Jack* systems, reflecting activity in the JackMOO world.

4. Database Model of DVE

The database of a distributed virtual environment contains the state information of the virtual world, which is the union of all the states of any avatars and physical objects it contains. For any DVE system, where to put the database is a very important issue, because it will affect the bandwidth requirements, end-to-end latency, and scalability of the system.

Figure 2 shows two typical database models for existing DVE systems [17]: the replicated homogeneous database and the shared centralized database.

SIMNET (SIMulator NETworking, the predecessor of DIS, or Distributed Interactive Simulation) [18] uses the replicated homogeneous database model, where in the beginning of a simulation exercise, all participating clients are provided a replica of the database containing the initial state of the virtual world. Then each client is responsible for sending state update information to other clients, and maintaining its own replica of the database by receiving state update information from other clients. The advantage of this model is that the bandwidth requirement is relatively low, because the amount of information flowing among clients is relatively small. Additionally since each client accesses only a local database, the end-to-end latency of a query to

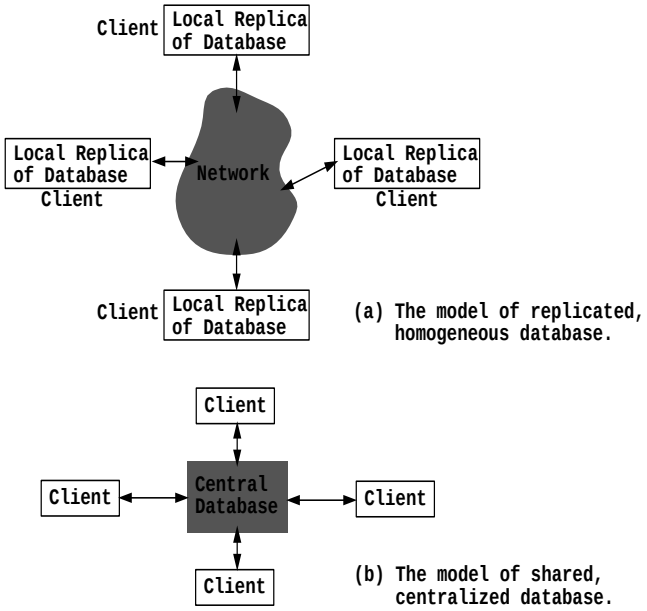

Figure 2. Two typical models for DVE database.

the database is very small. However, this structure cannot be easily extended to accommodate more participants, and as more and more users join in a simulation exercise, the database replica that each client maintains will get bigger and bigger. Furthermore, the consistency of all database replicas is not guaranteed by the model.

On the other hand, the shared centralized database model has only a single database. To retrieve or modify the state of an object, a client must send requests to the central database manager, and wait for a response from it. The advantage of this model is that it is easy to maintain, and is scalable within certain limits with respect to the capacity of the central server. The disadvantage is that the central server can easily become the speed bottleneck as the number of clients increases, and is the single point of failure. Heavy network traffic is another drawback of this model. LambdaMOO [12] is a distributed system that uses this model as its underlying database structure.

Table 1 gives a comparison between these two database model extremes. Based on the comparison, we employ a 'dual-database' model for our JackMOO system (Figure 3). The model divides the whole database into two parts: the object-oriented database, which contains the *semantic state* information of all the objects in the virtual world; and the graphical database, which contains the *graphical state* information of all of the objects:

- Semantic State – the state of an object that is propositional; for example, avatar A_i is in room R_j and in

Database Model	Replicated Homogeneous	Shared Centralized
Bandwidth Requirement	low	high
Speed Bottleneck	no	yes
Single Point of Failure	no	yes
Scalability	hard	easy
End-to-end Latency	low	high
Maintainability	hard	easy
Consistency	weak	strong

Table 1. Comparison of two database models.

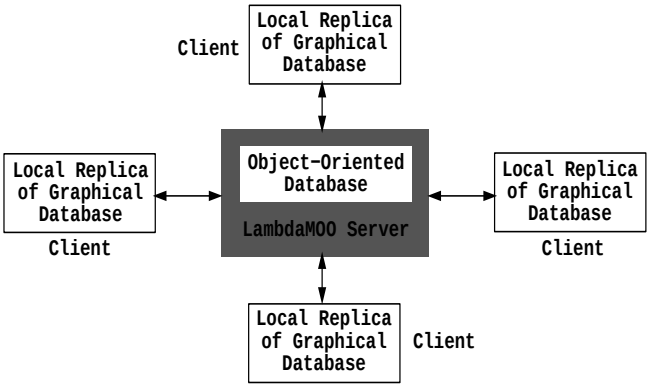

Figure 3. The dual-database model for JackMOO.

posture 'standing.'

- Graphical State – the state of an object that defines its appearance, location, or orientation; for example, the virtual world coordinates of avatar A_i, the joint angle of his left elbow, etc.

In JackMOO, we use the built-in OO database of LambdaMOO (see Section 6) as the OO database containing semantic states of our model, since the messages associated with semantic states are usually small. Meanwhile, each client maintains a local graphical database replica in order to minimize the latency of exchanging graphical state information (usually substantial chunks of bytes) between the client and the database.

The following sections describe the JackMOO main components in detail.

5. *Jack*

Our virtual human model *Jack* includes a programming language interface called Parallel Transition Networks (PaT-Nets) [3]. Intuitively, PaT-Nets are state transition diagrams in which nodes represent executable processes and edges contain conditions which when true cause transitions to another node (process). Combined with message passing and global memory, PaT-Nets provide coordination and synchronization across multiple parallel processes. PaT-Nets provide the mechanism necessary for the realization of complex actions and behaviors on virtual humans.

Of central importance to JackMOO is the association of human action verbs with (possibly several) PaT-Nets. Fundamental movements such as step-forward, walk, reach, turn-around, and look-at are specified in the *Jack* environment in the form of executable programs which are themselves invoked through PaT-Nets. PaT-Nets thus function as a high-level API accessing underlying *Jack* behavior and functionality.

6. LambdaMOO

LambdaMOO is a network-accessible, multi-user, programmable, interactive system well suited for the construction of text-based conferencing systems, educational/training systems and other collaborative software [12]. LambdaMOO's roots are in multi-user, collaborative game environments in which users are represented by a character possessing certain distinguishing attributes. Characters are placed within a virtual reality where they may interact with other users and objects that may be encountered ([7], [8], [13], [20], and [24]), and general virtual society applications ([6], [15], and [22]).

The interface to the LambdaMOO world is text-based. Users communicate with the world by typing one line commands resembling imperative English sentences, for example, "Take the ball," "Strike dragon on the head," etc. Such commands are parsed and executed by the system and result in changes in the virtual reality, such as location of the user's character, the appearance of some object, interactions with other users, etc. Such changes are communicated to the user by means of text: "You pick up the ball," "The dragon appears to be angry and is coming toward you menacingly!" etc. In addition, each of the other players in the user's immediate neighborhood receives text notification of the user's actions: "John has picked up the ball," "The dragon is angrily advancing on Norm," etc. Users may interact with each other on at least two levels: they may "talk" to one or all of the other players; and they may "emote," i. e., reveal emotions to one or more players: "John laughs at Norm's predicament."

LambdaMOO consists of two major components: the server and the database. The server is a program written in a standard programming language that manages multiple network connections, maintains queues of commands to be executed, controls all access to the database, and executes programs written in the LambdaMOO programming language.

The database contains objects representing all components of the virtual reality. Each object consists of properties containing data intrinsic to the object and verbs: programs written in the LambdaMOO programming language that provide the objects with their particular behaviors. A LambdaMOO virtual world is organized as a series of room objects, connected by entrance and exit objects. Rooms provide the containers for other objects in the world. Players, themselves objects, move about the LambdaMOO world, room by room, interacting with objects and other players they may encounter.

Objects are organized into single-inheritance hierarchies. When objects are created, they inherit the properties and verbs of their ancestors. Additional properties and verbs as well as specializations of inherited components may be defined to give the new object its unique behavior and appearance. In particular, the JackMOO world contains a specialization of the LambdaMOO generic room object called a *Jack* Room. This object contains properties and verbs that implement the interface between the LambdaMOO server and the remainder of the JackMOO system. Each LambdaMOO room that has a JackMOO realization is a specialization (child) of the *Jack* Room object containing, in particular, visualization properties with scene information to be loaded into *Jack* for the room and its contents. Specializations (children) of the LambdaMOO generic player object, the *Jack* Player, provide the properties and verbs necessary to define and control the individual avatar representation for a player in the JackMOO system. Default avatar properties are provided in the parent *Jack* Player object for those JackMOO users having no unique avatar representation.

The interface to the JackMOO system is activated when a *Jack* Player enters a *Jack* Room. Verbs and properties defined on the instance of the *Jack* Player issue commands to the user's *Jack* process. The new arrival's avatar appears on the *Jack* displays of all of the room occupants who are JackMOO users and each JackMOO occupant's avatar is displayed on the new arrival's *Jack* screen. Avatar position and orientation information saved as property values on the *Jack* Player object is used to locate the avatars "where they belong" in the scene. Executing appropriate verbs on the JackMOO objects through the JackMOO client moves and controls the avatar.

Each of the JackMOO avatar behavior verbs has the following responsibilities:

- Update the persistent information maintained for the avatar, e. g., change its stored position when a "move" command is received.
- Perform any actions with other objects in the room that may be affected by the verb. If my avatar picks up the red ball, the ball moves from its original location to my avatar's hand.
- Issue commands to the *Jack* component of the system to display avatar change to the end user.
- Broadcast changes to all of the connected JackMOO clients. This provides the synchronization of each of the JackMOO displays as the avatar moves around and interacts in the JackMOO world.
- Broadcast a textual description of the action and its effect on the environment to all players in the room. JackMOO can thus be accessed as a "standard" text-only LambdaMOO world.

These features permit Internet simulation at low (text) bandwidth rates. Text also yields robust communications and minimal transmission latency. The "intelligence" for executing actions is embedded in each client and is therefore independent of network bandwidth once a command is received. Tasks are synchronized by sending text strings informing others that an action is in progress ("Norm is walking across the room"), an action has finished ("Norm stops walking"), or possibly interrupted ("Norm bumped into the obstacle"). More experimentation is planned in this area.

7. JackMOO Client

The JackMOO client is a Java applet, loaded into the user's Netscape session from the HTTP server after the user provides the correct URL to the server. The client has the following functions:

- Establishes a socket connection from the client machine to the LambdaMOO server. This allows the end-user to send commands affecting his avatar's actions and behavior as well as the "standard" MOO interactions.
- Provides the text-oriented user interface to the LambdaMOO server. This interface consists of a text area for commands to be sent to the MOO, and a scrolled text area in which textual responses from the MOO are displayed.
- Mediates the communication between the client machine and the LambdaMOO server. Text messages from the MOO may be in the form of "standard" textual responses displayed in the client's text area; or commands to *Jack* and/or the client which are text strings distinguished by a special prefix.

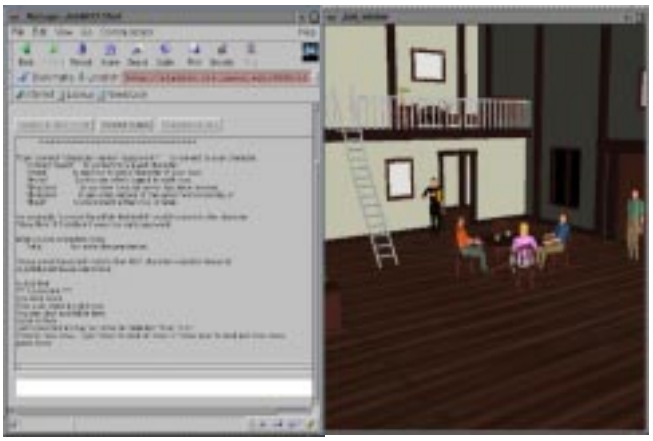

Figure 4. User interface of JackMOO.

8. Example: Jack's MOOse Lodge

In this section, we will discuss an example called "Jack's MOOse Lodge", implemented in the JackMOO environment. The scene is inside a wooden mountain lodge. There is a big room with a dining area with a table and four chairs. To one side of the dining area is a loft, with a ladder leading to it. The loft is surrounded by railings and has a bed on it. Below the loft is an open doorway leading to a kitchen. The main entrance of the lodge has a door with a knob. The test scenario includes five virtual humans: four are user-instructed avatars and one is a semi-autonomous waiter "agent". The four avatars are named 'Bob', 'Norm', 'Sarah', and 'David'. They are all 'smart avatars', and are controlled by different users using computers at different geographical sites. In the following we will use the quoted name ('Bob') to refer to the avatar and the unquoted name (Bob) to refer to the live user who is issuing commands to his/her avatar.

Each user controls his/her avatar via an interface running on the client computer. As shown in Figure 4, there is a Netscape window running the JackMOO client applet on the left side, and a *Jack* window used to display the virtual world on the right side.

The actions that a smart avatar can perform in the lodge include: walking, sitting down (on a chair or on the bed), standing up, talking to others, climbing ladder, opening door, shaking hands, bowing, and drinking. The waiter agent carries a pitcher with some kind of liquid, and acts according to the following rules:

- if an avatar is sitting at the table, and the glass in front of him/her is empty, the waiter will approach the glass, and pour the liquid into it from the pitcher; or
- if the pitcher is empty, the waiter will go into the kitchen through the open doorway, refill the pitcher, and come back out; or
- if nothing needs to be done, the waiter will just stand by the doorway and stay idle.

8.1. Contextual Behavior

The scene begins with 'Bob' entering the lodge while 'Norm', 'Sarah', and 'David' are seated at the table, drinking and talking to each other. To be polite, Bob should issue commands such as "greet ⟨the name of an avatar⟩" to greet other people. Since 'Bob' is a smart avatar, upon receiving the command, he will take different actions suited to different situations:

- *"greet Norm"*. 'Bob' approaches 'Norm', puts forward his right arm, and waits for the response from 'Norm'. Assume that Norm sees the initiating action of 'Bob', and so Norm issues a command "greet Bob". Then 'Norm' will stand up from the chair, turn to 'Bob', grab his right hand, and shake hands with him.
- *"greet Sarah"*. 'Bob' approaches 'Sarah', turns to face her, and bows to her. Upon receiving the command "greet Bob" from Sarah, 'Sarah' will stand up, face 'Bob', and bow back to him.

From these cases we can see that not only 'Bob' shows his contextual behavior by executing different actions to different targets on the same command "greet somebody", but also 'Norm' and 'Sarah' show their contextual behaviors by greeting back in corresponding ways to avatar 'Bob' who has initiated the greetings.

8.2. Preparatory Actions

We have already remarked that a smart avatar may have to execute some preparatory actions before performing the main (requested) action. Now, let us take a look at a more complex case (see Figure 5): suppose that David issues a command "go to bed". The command actually consists of two sub-actions: "walk to the bed", and "sit down on the bed". To be able to walk, 'David' should be in the "standing" posture, so he stands up from the chair first. Then he realizes the bed is on the second floor, and the only way leading to the second floor is via the ladder. So his subsequent preparatory actions should be "walk to the ladder", and "climb the ladder". Thereafter, he can walk to the bed and sit down on it.

In general, preparatory actions may involve the full power of motion planning [4]. The commands, after all, are essentially goal requests and the smart avatar must then figure out how (if at all) it can achieve them. Presently we just use PaT-Nets with hand coded conditionals to test for

Figure 5. Jack's MOOse Lodge: a scene from the example.

likely (but generalized) situations and execute appropriate intermediate actions [26].

8.3. Leader-Follower Relationship

As the example proceeds, 'Norm' invites 'Sarah' to go out for a walk. Sarah accepts the invitation by instructing 'Sarah' to "follow Norm". Then as 'Norm' walks to the door, opens the door, and exits the room, 'Sarah' trails along behind. In this case, a leader-follower relation is established between the avatars. A pursuit locomotion condition is established between the avatars which causes 'Sarah' to follow 'Norm' temporarily.

Note that this leader-follower relationship is totally different from that of pilot/drone, defined in [16]. A pilot is the graphical version of the avatar controlled on the user's own client; the drones are the avatar copies executing on other clients. In the leader-follower model one temporarily yields some aspects of the control of one's avatar to another's lead. Sarah could still instruct her avatar to wave good-bye even as she follows 'Norm' out the door.

9. Future Work

Virtual environments appear natural for training, but a number of complex problems remain to be addressed. A project for team training in checkpoint operations is being undertaken under ONR sponsorship through the University of Houston, the University of Pennsylvania, LinCom Corporation, and Transom Technologies. JackMOO will evolve into a new system to support the additional requirements for multiple human participants.

A distributed virtual environment used for training purpose is likely to have three different types of virtual humans: live participants, semi-autonomous virtual human agents, and smart avatars. The live participants will be one or more trainees. The semi-autonomous agents will be agents that follow a few simple rules. For example, there might be an agent that supervises the trainee; her only responsibility could be to stay in a position which allows her to view both the trainee and the person the trainee is communicating with. The smart avatars will be able to display a more varied repertoire of actions, perhaps initiated by verbal instructions or situation circumstances. These agents may include a subordinate helper or a person who tries to distract the trainee.

Semi-autonomous virtual human agents will need rules to guide their behavior. A good example of a rule-based training agent is the Steve system [14]. Smart avatars will also have a number of context-dependent actions associated with them, but they will be given a sense of role, action-planning, situation, and culture. For example, the waiter in the MOOse lodge might eventually be programmed through natural language instructions that generate the underlying PaT-Net code. Smart avatars from different cultures will act differently in the same situation, as was simply illustrated in the greeting scenarios in the MOOse Lodge. Furthermore, a user interface is necessary to allow users to enter goals and instructions, or to change properties (e.g., culture) for the smart avatars. A natural language interface may be a fruitful medium for specifying these behavioral parameters.

A Parameterized Action Representation (PAR) is the key structure that mediates language-level concepts and smart avatar actions. A PAR includes slots to indicate the agent, the involved objects, applicability conditions, termination conditions, purpose, spatiotemporal terms, and agent manner [5, 2]. The database of PAR prototypes is a dictionary of action definitions: an *Actionary*. The new PAR execution architecture will replace LambdaMOO with a full natural language parser, use the Actionary for interpreting instructions, and make PaT-Nets the virtual parallel machine underlying the simulation process.

The PAR architecture will also be extended for a serverless distributed scenario, in which multiple participants at different geographic sites can participate in a common task and interact with each other. The issues of this distributed virtual environment architecture include coordination, synchronization, real-time interaction, and consistency. We are exploring the use of semantic information packets and PARs to accomplish greater scalability and multi-site coordination while minimizing network traffic.

10. Acknowledgments

This research is partially supported by Office of Naval Research (through Univ. of Houston) K-5-55043/3916-1552793, DURIP N0001497-1-0396, and AASERTs N00014-97-1-0603 and N0014-97-1-0605; Army Research Lab HRED DAAL01-97-M-0198; DARPA SB-MDA-97-

2951001; NSF IRI95-04372; NASA NRA NAG 5-3990; and JustSystem Japan.

References

[1] N. Badler. Virtual humans for animation, ergonomics and simulation. *Proc. Pacific Graphics '97*, 1997.

[2] N. Badler, R. Bindiganavale, J. Bourne, M. Palmer, J. Shi, and W. Schuler. A parameterized action representation for virtual human agents. In *Workshop on Embodied Conversational Characters*, Lake Tahoe, CA, Oct. 1998.

[3] N. Badler, C. Phillips, and B. Webber. *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, New York, NY, 1993.

[4] N. Badler, B. Webber, W. Becket, C. Geib, M. Moore, C. Pelachaud, B. Reich, and M. Stone. Planning for animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation*. Prentice-Hall, 1996.

[5] N. Badler, B. Webber, M. Palmer, T. Noma, M. Stone, J. Rosenzweig, S. Chopra, K. Stanley, J. Bourne, and B. D. Eugenio. Final report to Air Force HRGA regarding feasibility of natural language text generation from task networks for use in automatic generation of Technical Orders from DEPTH simulations. Technical report, CIS, University of Pennsylvania, 1997.

[6] Baymoo. A social Moo with the San Francisco Bay Area as a backdrop, telnet baymoo.sfsu.edu 8888.

[7] Biomoo. BioMOO is a professional community of Biology researchers. It is a place to come meet colleagues in Biology studies and related fields and brainstorm, to hold colloquia and conferences, telnet bioinformatics.weizmann.ac.il 8888.

[8] Cafemoolano, a multi-language educational moo. used and developed by a variety of humanities courses from UC Berkeley and beyond, telnet moolano.berkeley.edu 8888.

[9] T. Capin, H. Noser, D. Thalmann, I. Pandzic, , and N. Magnenat Thalmann. Virtual human representation and communication in vlnet networked virtual environments. *IEEE Computer Graphics and Applications*, 17(2):42–53, 1997.

[10] J. Cassell, C. Pelachaud, N. Badler, M. Steedman, B. Achorn, W. Becket, B. Douville, S. Prevost, and M. Stone. Animated conversation: Rule-based generation of facial expression, gesture and spoken intonation for multiple conversational agents. In *Computer Graphics, Annual Conf. Series*, pages 413–420. ACM, 1994.

[11] D. Chi, B. Webber, J. Clarke, and N. Badler. Casualty modeling for real-time medical training. *Presence*, 5(4):359–366, 1995.

[12] P. Curtis. LambdaMOO, 1997. Xerox PARC Ftp site: parcftp.xerox.com/pub/MOO.

[13] Diversity university, the first moo to be designed specifically for classroom use. telnet moo.du.org 8888.

[14] W. L. Johnson and J. Rickel. Steve: An animated pedagogical agent for procedural training in virtual environments. *SIGART Bulletin*, 8(1-4):16–21, 1997.

[15] Lambdamoo, the original social moo. telnet lambda.moo.mud.org 8888.

[16] Living Worlds, 1997. http://www.livingworlds.com/draft_1/index.htm.

[17] M. R. Macedonia and M. J. Zyda. A taxonomy for networked virtual environments. *the Second IEEE Workshop on Networked Realities*, October 1995.

[18] D. Miller and J. Thorpe. SIMNET: The advent of simulator networking. *Proceedings of the IEEE*, 83(8), Aug. 1995.

[19] T. Noma and N. Badler. A virtual human presenter. In *IJCAI '97 Workshop on Animated Interface Agents*, Nagoya, Japan, 1997.

[20] Pennmoo. telnet ccat.sas.upenn.edu 7777.

[21] C. Rose, B. Guenter, B. Bodenheimer, and M. Cohen. Efficient generation of motion transitions using spacetime constraints. In *ACM Computer Graphics, Annual Conf. Series*, pages 147–154, 1996.

[22] Sensemedia moo. A MOO universe based on Neal Stephenson's novel Snow Crash, telnet sapporo.sensemedia.net 9030.

[23] T. Smith, J. Shi, and N. Badler. Jackmoo, a prototype system for natural language avatar control. In *WebSim*, San Diego, CA, 1998.

[24] Tefcamoo. a virtual space for Educational Technology, Education, Research and Life at TECFA, School of Psychology and Education, University of Geneva,Switzerland, telnet tecfamoo.unige.ch 7777.

[25] The java tutorial. http://java.sun.com/.

[26] T. Trias, S. Chopra, B. Reich, M. Moore, N. Badler, B. Webber, and C. Geib. Decision networks for integrating the behaviors of virtual agents and avatars. In *Proceedings of Virtual Reality International Symposium*, 1996.