January 2000

A Parameterized Action Representation for Virtual Human Agents

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

Ramamani Bindiganavale
University of Pennsylvania

Juliet C. Bourne
University of Pennsylvania

Martha Palmer
University of Pennsylvania

Jianping Shi
University of Pennsylvania

See next page for additional authors

Postprint version. Published in *American Association for Artificial Intelligence, Spring Synposium*, 2000.

A Parameterized Action Representation for Virtual Human Agents

Abstract

We describe a Parameterized Action Representation (PAR) designed to bridge the gap between natural language instructions and the virtual agents who are to carry them out. The PAR is therefore constructed based jointly on implemented motion capabilities of virtual human figures and linguistic requirements for instruction interpretation. We will illustrate PAR and a real-time execution architecture controlling 3D animated virtual human avatars.

Comments

Postprint version. Published in *American Association for Artificial Intelligence, Spring Synposium*, 2000.

Author(s)

Norman I. Badler, Ramamani Bindiganavale, Juliet C. Bourne, Martha Palmer, Jianping Shi, and William Schuler

A Parameterized Action Representation for Virtual Human Agents

Norman Badler, Rama Bindiganavale, Juliet Bourne
Martha Palmer, Jianping Shi, William Schuler
Center for Human Modeling and Simulation
Computer and Information Science Department
University of Pennsylvania
Philadelphia, PA 19104-6389

Abstract

We describe a Parameterized Action Representation (PAR) designed to bridge the gap between natural language instructions and the virtual agents who are to carry them out. The PAR is therefore constructed based jointly on implemented motion capabilities of virtual human figures and linguistic requirements for instruction interpretation. We will illustrate PAR and a real-time execution architecture controlling 3D animated virtual human avatars.

Introduction

Only fifty years ago, computers were barely able to compute useful mathematical functions. Twenty-five years ago, enthusiastic computer researchers were predicting that all sorts of human tasks from game-playing to automatic robots that travel and communicate with us would be in our future. Today's truth lies somewhere in-between. We have balanced our expectations of complete machine autonomy with a more rational view that machines should assist people to accomplish meaningful, difficult, and often enormously complex tasks. When those tasks involve human interaction with the physical world, computational representations of the human body can be used to escape the constraints of presence, safety, and even physicality.

Virtual humans are computer models of people that can be used

- as substitutes for "the real thing" in *ergonomic* evaluations of computer-based designs for vehicles, work areas, machine tools, assembly lines, etc., *prior to the actual construction of those spaces*;
- for *embedding real-time representations of ourselves or other live participants* into virtual environments.

Recent improvements in computation speed and control methods have allowed the portrayal of 3D humans suitable for interactive and real-time applications (Badler, Phillips, & Webber 1993). There are many reasons to design specialized human models that individually optimize appearance, character, performance, or intelligence. Here we will focus on the problem of *making virtual humans respond to verbalized commands in a context-sensitive fashion.*

In this discussion we first define the kinds of real-time virtual human characters we wish to control. This leads to a focus on language-based control or instruction interpretation. To support *both* language understanding and animation, we define a *Parameterized Action Representation* (PAR). We then give an outline of an architecture for interpreting (executing) such represented actions, and show how they control embodied agent models. Several examples show how the characters can interact when given context-dependent instructions. Finally, we show how a conversation results in establishing a relation between agents.

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 are used to animate the 3D model. In real-time applications, *avatars* are human representations driven directly by a real person. Physically sensed motions are natural but lock the user into wearing equipment which may be cumbersome and limiting. Moreover, directly sensed motion is difficult to modify on-the-fly to achieve subject or environmental sensitivity (Gleicher 1998; Bindiganavale & Badler 1998), and the user may be subject to common symptoms such as "groping" for virtual objects, jerky locomotion, and annoying head movements. Control without encumbrance leads to vision-based sensing (Metaxas & Essa 1997) or, as we are exploring, language-based instructions.

In general, an embodied (human-like) character that acts from its own motivations is often called an *agent*. In this sense, an avatar is an agent that represents an actual person. Its actions may be portrayed through captured or synthesized motions performed in the current context. This requires parameterization and, in turn, proper specification of parameters. We call an avatar *controlled via instructions* from a live participant a *smart avatar*. Parameters for its actions may come from the instruction itself, others from the local object context, and yet others from the avatar's avail-

able capabilities and resources.

We have explored the contextual control of embodied agents, avatars, and smart avatars (with and without conversation) in a number of experiments including: two person animated conversation ("Gesture Jack") (Cassell *et al.* 1994), medic interventions and patient physiological interactions in MediSim (Chi *et al.* 1995), a real-time animated "Jack Presenter" (Noma & Badler 1997), and multi-user "Jack-MOO" virtual worlds (Shi *et al.* 1999). 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 enable interesting applications (Firby 1994; Johnson & Rickel 1997), and to refine a representation bridging natural language and embodied action.

Language-based Control

In order for smart avatars to respond to instructions expressed in natural language input we have to be able to *coordinate verbal and physical representations of actions.* The basic linguistic representation of an action is the predicate argument structure that indicates the participants and the particular action involved, such as `slide(John, box)`, but certain kinds of actions can sometimes integrate additional information and use it to enrich the action description. Motion verbs often specify details with respect to the path the object in motion will take. These can include the medium of the path (ex. a gravel road, the air, the sea) as well as specific locations for endpoints such as sources and goals (*to the store, across the room, from the beach, home*). Our common sense tells us that path information is relevant to the description of actions involving motion, but more significantly from a methodological point of view, linguistic evidence points to the facility with which path prepositional phrases can modify descriptions of motion events. It is critical for generic representations of actions to adequately specify any and all possible enrichments of the basic predicate-argument structure so that these types of extended meanings can be accommodated. The Parameterized Action Representation (PAR) (Badler *et al.* 1997) includes slots for many types of information that can sometimes occur linguistically as adjuncts to the main verb phrase rather than as part of the basic sub-categorization frame, or even possibly in separate sentences. These slots include spatio-temporal information such as paths (*to the store, across the room*), manner information that is often expressed as adverbs (*quickly, carefully*), and applicability and terminating conditions that can be inherent but can also be specified (*until the door is open*).

This rich representational structure also allows our PARs to capture the physical or performance attributes of movements and actions. For example, it allows differentiating between a simple change of location description such as *go to the door*, and a more idiomatic expression such as *go to bed* which brings with it a wealth of cultural habits about preparing for a night's rest. With the former action, there is just a single move event involving a single participant, and the termination is achieved when that participant arrives at the door. The latter action involves a series of actions that can be as diverse as changing clothes and brushing teeth, and terminates when the participant actually lies down on the bed. This requires the embedding of several simple actions into a single complex action. In this way, we can use a PAR as a common representation to capture verbal *and* physical descriptions of both simple actions, *go to the door*, and complex actions, *go to bed.*

In addition to commands for the immediate performance of actions, we would also like to use natural language input to convey conditional actions that need to persist in the avatar's memory and may be triggered by future circumstances. For example, "If you agree to go for a walk with someone then follow them," is a standing instruction that would be triggered whenever the avatar agreed to go for a walk. This requires a sophisticated natural language processing system with broad syntactic coverage and close integration of syntax and domain-specific lexical semantics that can interact with the underlying graphical world model. We are using as the basis of our parser the XTAG Synchronous Tree Adjoining Grammar system in $C++$ (Palmer, Rosenzweig, & Schuler 1998).

Parameterized Action Representation (PAR)

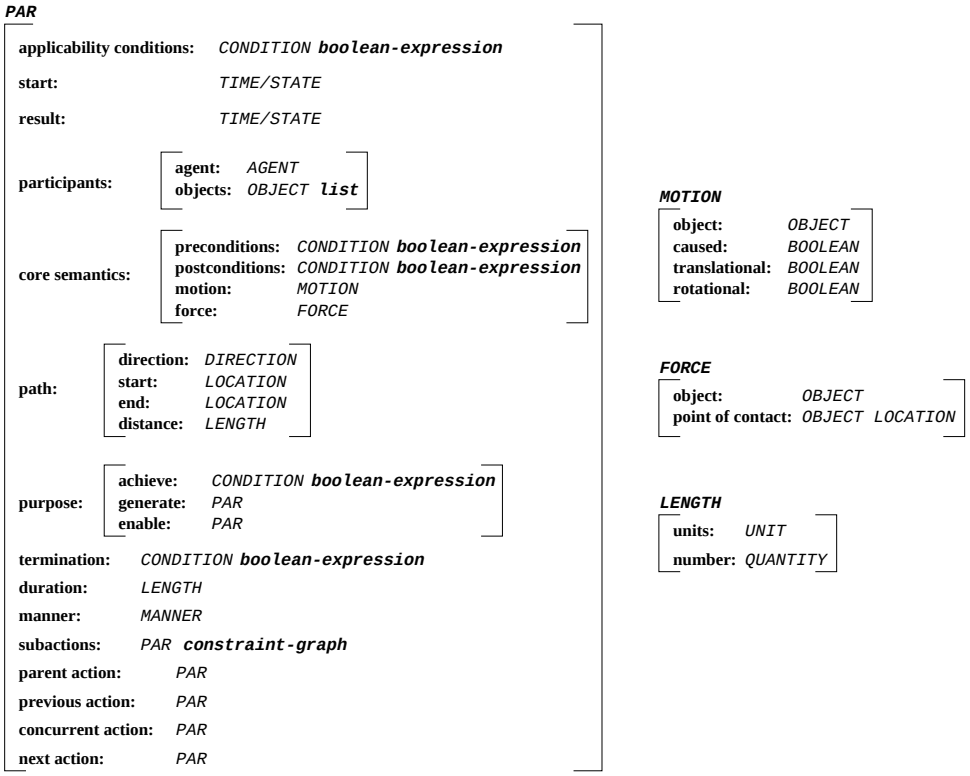
A PAR (Badler *et al.* 1997) gives a complete description of an action. We call it "parameterized" because an action depends on its participants (agents and objects) for the details of how it is accomplished. A PAR includes a number of conditions such as *applicability* and *preparatory* that have to be satisfied before the action is actually executed. The action is terminated when the *terminating* conditions are satisfied. A PAR can be represented syntactically as shown in Fig.1. In this section, we first briefly describe some of the terminology and concepts used to define a PAR and then describe the architecture that we have designed that interprets the PAR.

Terminology

To simplify the explanations, we consider an example of a PAR used to represent the instruction

Walk around the room.

Physical Objects: This is the list of objects referred to within the PAR. We have prepended the name *physical* to distinguish them from the *object* terminology used in object oriented languages. In our example, the *room* and *other objects contained in the room* are the physical objects. Each physical object has a number of properties associated with it and is stored hierarchically in a database.



,

Figure 1: Syntactic Representation of PAR

Agent: This is the agent who will be executing the action. In our example, the *avatar of the user* is the implied agent. An agent is a special type of object and has additional capabilities including a set of actions that the agent is capable of executing.

Applicability Conditions: These conditions specify what needs to be true in the world in order to carry out an action. They may refer to inherent properties of the agent (*e.g.*, agent capabilities) or physical objects (*e.g.* object configurations). Hence, if the applicability conditions are not satisfied (*i.e.* evaluate to FALSE), no sub-actions or subgoals are generated to make the conditions true. In our example, one of the applicability conditions may be *Is the agent capable of walking?*. If the agent is not capable of walking, the applicability conditions are not satisfied and the action of the agent walking around the room is aborted.

In some instances, the applicability conditions may also point to other actions. For example, a specific door may have to be opened by a sliding action. That door then associates sliding with opening. Hence, when the applicability conditions of the door are checked for opening, the sliding action *("slide", agents:("a1"), objects:("door")))* is returned and the current action of opening the door will be replaced by the action of sliding the door. The implementation details of this are discussed later in the paper.

Preparatory Specifications: This is a list of <*Condition, Action*> statements. The conditions are evaluated first and have to be satisfied before the current action can be executed. If the conditions evaluate to FALSE, then the corresponding action is executed: it may be a single action or a very complex combination of actions, but it has the same format as the execution steps (defined below). In our example, one of the conditions can be *standing?(agent)* and the corresponding action can be *("stand", agents:("Jack")))*. If the agent is not standing, *e.g.* if he is sitting or prone, then he changes to the standing posture.

Execution Steps: A PAR can either describe a complex action or a primitive action. The execution steps contain the details of action executing after all the conditions have been satisfied. A complex action can list a number of sub-actions which may need to be executed in sequence, parallel, or a combination of both. An example of a complex action specification is shown in Fig. 2.

Termination Conditions: This is a list of conditions which when satisfied complete the action. Determining a terminating condition from the main verb or attached clauses is discussed in (Bourne 1998).

```
complex =
 ('SEQUENCE',
   ('PARJOIN',
     ("par1",agents:("a1"),objects:("o1", "o2")),
     ("par2",agents:("a1"),objects:("o2"))),
   ("par3",agents:("a1"),objects:("o1")),
   ('PARINDY',
     ('SEQUENCE',
       ("par4",agents:("a1"),objects:("o2","o3")),
       ("par5",agents:("a1"),objects:("o3"))),
     ("par6",agents:("a1"),objects:("o1","o2")))
   ("par7" agents:("a1"),objects:("o1","o3"))
 )
actions = {'COMPLEX': complex}
```

Figure 2: Specification of a Complex Action

Post Assertions: This is a list of statements or assertions that are executed after the termination conditions of the action have been satisfied. These assertions update the database to record the changes in the environment. The changes may be due to direct or side effects of the action.

PAR representations: A PAR appears in two different forms:

UPAR(Uninstantiated PAR): We store all instances of the uninitialized PAR in a database in a hierarchical tree. A UPAR contains default applicability conditions and preconditions for the action, and also points to the executable actions that actually drive the embodied character's movements. The UPAR does not contain information about the actual agent or physical objects involved. In essence, UPARs comprise a dictionary of available actions.

IPAR (Instantiated PAR): An IPAR is a UPAR instantiated with information or pointers to a specific agent, physical object(s), manner, and termination conditions. Any new information in an IPAR overrides the corresponding UPAR default. An IPAR can be created by the parser (one IPAR for each new instruction) or can be created dynamically during execution.

Architecture

Fig. 3 shows the architecture of the PAR system. We briefly describe the important modules.

NL2PAR: This module consists of two parts: parser and translator. The parser takes a NL instruction/command and outputs a tree identifying the different components as noun, verb, adverb, preposition, etc. For each new instruction, the translator uses the output of the parser and information stored in the database to first determine the correct instances of the physical object and agent in the environment. It then generates the instruction as an IPAR.

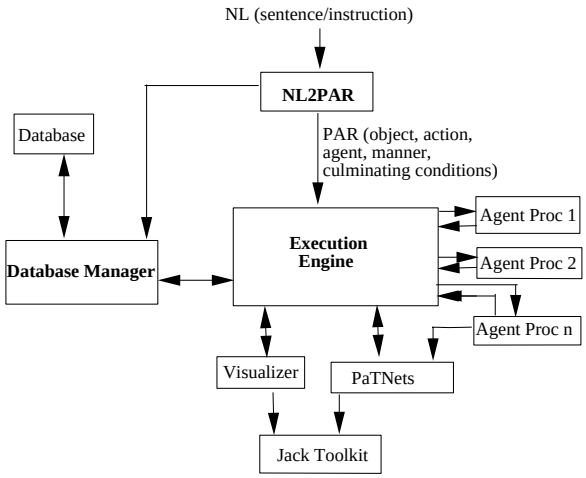


Figure 3: PAR Architecture

Database: All instances of physical objects, UPARs, and agents are stored in a persistent database. The physical objects and UPARs are stored in hierarchies.

Execution Engine: The execution engine is the main controller of the system. It maintains the global timer/controller, sends commands to the visualizer to update the displayed scene, and accepts a PAR from the NL2PAR module and passes it on to the correct agent process.

AgentProcess: Each instance of an agent is controlled by a separate agent process, which maintains a queue of all IPARs to be executed by the agent. For each IPAR, the applicability conditions are first checked. If they are not satisfied, the entire process is aborted after taking care of failure conditions and proper system updates. If the applicability conditions are satisfied, the preparatory conditions are then checked. If any of the corresponding preparatory actions need to be executed, an IPAR is created (using the specified information of the UPAR, agent, and the list of objects) and added to the agent's existing queue of IPARs. It should be noted that the queue of IPARs is a multi-layered structure. Each new IPAR created for a preparatory action is added to a layer below the current one. The current action is continued only after the successful termination of all the preparatory actions. If the current action is very complex, more IPARs are generated and the depth of the queue structure increases.

The queue manager of actions in an agent process is implemented using a PaT-Net (Parallel Transition Network) (Badler, Phillips, & Webber 1993). A PaT-Net is a simultaneously executing finite state automata in which the nodes are associated with actions and connections between the nodes are associated with transition conditions. The PaT-Nets make it very easy to wait on completion of actions before moving onto the next action, to execute actions in

parallel or in sequence, and to dynamically extend the action structure by invoking other PaT-Nets from nodes of the current one. During the execution phase, a PaT-Net is dynamically created for each complex action specified in the execution steps or in preparatory specification. Each sub-action corresponds to a subnet in the PaT-Net. The PaT-Nets are also used ultimately to ground the action in parameterized motor commands to the embodied character.

Toolkit: We use the Transom *Jack*®toolkit and OpenGL to maintain and control the actual geometry, scene graphs, and human behaviors and constraints.

Implementation

We have implemented PAR using C++ and *Python* (Lutz 1996). Python is an interpreted object-oriented language and is quite compatible with C++. This makes it easy for both the action and object hierarchies to be visible from both *Python* and C++. It also provides serialization and persistence which is ideal for database implementation. UPARs and IPARs may be dynamically created. As *Python* can be extended and embedded in C++, objects of different data types can be passed between them. The applicability conditions, preparatory specifications, and executable actions are all written by end-users on the fly as simple Python scripts which can be easily tested. The objects passed from *Python* to C++ are all perceived by C++ to be of a single *Python* object type which could later be typecast to different types. This allows conditions to return the various test results as either a Boolean type or a Python string. The agent process is capable of expanding this string into a new action and adding it to the agent's queue.

The system is built in two layers. The bottom layer contains the implementation of the core system and is in C++ and *Python*. The top layer has a user-friendly interface. The user needs to interact only with the top layer. In the future, this system will be extended to work in a distributed environment with heterogeneous clients on different platforms.

Example: Jack's MOOse Lodge

In this section, we will discuss an example called "Jack's MOOse Lodge", which was initially implemented in the JackMOO environment (Shi *et al.* 1999). We will describe the flow of control in the PAR system within the context of this example.

The scene (Fig. 4) is the inside of a mountain lodge built from wood. There is a big room with a dining area containing 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 "smart" avatars and one

Figure 4: Jack's MOOse Lodge: a scene from the example.

is a semi-autonomous waiter "agent." The four avatars are named 'Bob', 'Norm', 'Sarah', and 'David', respectively. In a distributed environment, each avatar could be controlled by different users at possibly different geographical sites. In a non-distributed environment, the instructions can be specifically directed to an avatar: *e.g.*, "Sarah, walk to the door". The parser is capable of interpreting and generating the correct IPAR for this. 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 responsible for issuing commands to his or her avatar.

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 (a ladder), opening a door, shaking hands, bowing, and drinking. The database stores the definitions (UPARs) for each action, containing the necessary applicability conditions, preparatory specifications, and possibly PaT-Nets.

The waiter agent is autonomous and is controlled by its own agent process. 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.

The waiter agent obtains all its environment state information from the working memory. Based on the environment conditions (state of pitcher or the avatar's glass), the agent process for the waiter adds the corresponding actions to its queue.

Conversation

The avatars can appear to hold conversations with each other. To do this, an instruction is given to the avatar to say something, *e.g.*, *Norm, say Hello to Sarah*. The parser module NL2PAR generates the IPAR where "Norm" is the agent, "say" is the verb, "Hello" is the text to be spoken and "Sarah" is the person (object) to be spoken to. The UPAR for "say" contains the preparatory specification of turning to the person (or object). So, if 'Norm' is initially turned away from 'Sarah', he will first execute the command "turn" towards 'Sarah' and then can say "Hello" to "Sarah". A text-to-speech converter is used to generate the actual speech.

Contextual Behavior

The avatars can interpret the same action in different ways depending on the context and the state of the environment. In our example, 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. Both then release the grasp and return their arms to a neutral pose.
- *"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 'Bob' shows his contextual behavior by executing different actions to different targets given the same command "greet somebody". 'Norm' and 'Sarah' show their contextual behaviors by returning the greeting to 'Bob' in a corresponding fashion.

Go-to Action The *goto* ⟨*location*⟩ is a complex action. In our example, we have two instances of *goto*: *goto bed* and *goto door*. There is a semantic difference between the *goto* in *goto bed* and *goto* ⟨*location*⟩. The *goto bed* has "lying down on the bed" as the termination condition. All the other *goto* actions usually terminate after reaching the target location. Hence we create a special UPAR associated with *goto-bed*. This UPAR is a child of the *goto* UPAR in the action hierarchy. *Goto-bed* is a complex action containing as sub-actions *goto-location*, *sit on bed* and *lie down on bed* which are all to be executed in sequence.

The *goto-location* is itself a complex action. It involves path planning and invocation of suitable sub-actions to reach the target along the planned path. For instance, the bed in our example is in the loft which can be accessed only by climbing a ladder. One of the applicability conditions of *goto-location* is *path-exist?*. To test for the condition, a path planning algorithm is invoked which takes into account the current state of the environment and returns either TRUE, FALSE or a *Python* string describing the action to be taken. The returned string is similar to the action description shown in Fig. 2. From this a new complex IPAR is built and added onto the queue. For the example of reaching the bed, the new action to be executed may be a sequence of *"walk" to the ladder*, *"climb" the ladder*, *"walk" to the bed*. After the agent has reached the bed, the sub-actions of *sit on bed* and *lie down on bed* are executed.

In general, preparatory actions or applicability conditions may involve the full power of motion planning (Badler *et al.* 1996). 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 use PaT-Nets with hand coded conditionals to test for likely (but generalized) situations and execute appropriate intermediate actions (Trias *et al.* 1996).

Leader-Follower Relationship

PAR also supports a leader-follower relationship where one avatar follows another avatar's actions. As our example proceeds, 'Norm' invites 'Sarah' to go out for a walk. This is done by issuing a command, "Norm, say 'Will you go for a walk with me?' to Sarah". When 'Norm' is heard saying this , Sarah can be instructed to accept the invitation: "Sarah, say 'yes'.". (Recall that, were the users on separate clients, identifying the avatar that is to speak would be unnecessary.) This is an example of a conversation modifying the state of the environment, namely, to initiate a leader-follower relationship. Then as 'Norm' starts walking to the door, 'Sarah' is instructed to follow 'Norm' by issuing the command, "Sarah, follow Norm". As 'Norm' walks to the door, opens the door, and exits the room, 'Sarah' trails along behind. In this case, 'Norm' is the leader and 'Sarah' is the follower. A pursuit locomotion condition is established between the avatars which causes 'Sarah' to follow 'Norm' temporarily. 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. Explicit commands would have to be issued to break the relationship.

Discussion

The PAR architecture and its implementation is intended to provide a testbed for real-time conversational agents who work, communicate, and manipulate in a synthetic 3D world. Our goal is to make interaction

with these embodied characters the same as with live individuals. We have focused on language as the medium for communicating instructions, and finite state machines as the controllers for the output movements.

The structure described here is the basis for a new kind of dictionary we call an *Actionary*. A dictionary uses words to define words. Sometimes it grounds concepts in pictures and (in online sources) maybe even sounds and video clips. But these are canned and not *parameterized* – flexible and adaptable to new situations the way that words appear in actual usage. In contrast, the Actionary uses PAR and its consequent animations to ground action terms. It may be viewed as a three-dimensional (spatialized) environment for animating and evaluating situated actions expressed in linguistic terms. The actions are animated to show the meaning in context, that is, relative to a given 3D environment and individual agents.

The Actionary will facilitate:

- The translation of human action instructions into sample action execution for training and education (*e.g.*, foreign language learning).
- The translation of action descriptions (instructions) across languages, especially where the motion verbs types differ significantly (*e.g.* Chinese to English).
- The study of conceptual aspects of verbs that are overlooked in standard human dictionaries.
- Low bandwidth communication of multi-person activities: by transmitting textual instructions (which are very compact), smart avatars at the receiving end can interpret instructions via an Actionary. (This is sensible because we believe that the power of local computation will increase faster than the network bandwidth into consumer homes, for example). This can enable remote job training, video/virtual conferencing, equipment operation and maintenance, 3D virtual communities, online playrooms, and so on.

A language-to-action interface interpreted through the Actionary helps free users from physical sensing devices and offers the potential for less expensive hardware deployment, realistic computational loads, decreased communication bandwidth requirements, and an undeniably natural user interface.

References

Badler, N.; Webber, B.; Becket, W.; Geib, C.; Moore, M.; Pelachaud, C.; Reich, B.; and Stone., M. 1996. Planning for animation. In *Interactive Computer Animation*. PrenticeHall. N. M-Thalmann and D. Thalmann (eds.).

Badler, N.; Webber, B.; Palmer, M.; Noma, T.; Stone, M.; Rosenzweig, J.; Chopra, S.; Stanley, K.; Bourne, J.; and Di Eugenio, B. 1997. 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.

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

Bindiganavale, R., and Badler, N. 1998. Motion abstraction and mapping with spatial constraints. In *Workshop on Modelling and Motion Capture Techniques for Virtual Environments*. Geneva, Switzerland: IFIP. to appear.

Bourne, J. 1998. Generating adequate instructions: Knowing when to stop. In *Proceedings of the AAAI/IAAI Conference,Doctoral Consortium Section*. to appear.

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

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

Firby, R. 1994. Task networks for controlling continuous processes. In *Proc. of the Second International Conference on AI Planning Systems, Chicago IL*.

Gleicher, M. 1998. Retargetting motion to new characters. In *Computer Graphics Proceedings*, 33–42. SIGGRAPH.

Johnson, W., and Rickel, J. 1997. Steve: An animated pedagogical agent for procedural training in virtual environments. *ACM SIGART Bulletin* 8(1):18–21.

Lutz, M. 1996. *Programming Python*. O'Reilly.

Metaxas, D., and Essa, I. E. 1997. *Nornrigid and Articulated Motion Workshop Proceedings*. IEEE Computer Society.

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

Palmer, M.; Rosenzweig, J.; and Schuler, W. 1998. *Predicative Forms in NLP*. Kluwer Press. chapter Capturing Motion Verb Generalizations with Synchronous TAG. to appear in.

Shi, J.; Smith, T. J.; Granieri, J. P.; and Badler, N. I. 1999. Smart avatars in jackmoo. In *IEEE Virtual Reality '99 Conference*. Submitted to.

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