

Animating virtual humans involves actions like posture changes and balance adjustments; reaching, pointing, grasping, giving-taking and other arm/hand gestures, locomotion, eye gaze and head gestures, facial expressions and speech. These animations involve controlling their parts at the graphical level via joint transformations (e.g., for limbs) or surface deformations (e.g., for face). Software packages, such as Unigraphics' Jack, provide tools for creating animations at this level. Joint angles can be set and interpolation between joint angles can be used to generate actions. Behavioral systems such as balance control are also provided. Though essential, these systems are just tools. Hand construction of actions for virtual humans using these tools can be tedious and time consuming. Using these tools in programs for procedural animation can create robust, reusable actions. We call these programs *motion generators*, and they are an essential part of our PAR system.

PAR is our *parameterized action representation*. PAR was designed to be an intermediate between high-level control, such as natural language, and animations. It, therefore, eases the creation of new virtual environment scenarios for non-animators. Creating a virtual environment scenario using the Jack toolkit, for example, requires:

- Creating or importing the necessary geometry,
- Placing the figures in correct initial locations in the environment,
- Creating actions for the figures,
- Creating code for the interaction of the human figures with each other, one or more trainees, and the environment.

The first two elements are the same when using PAR for virtual environment scenario creation. However, the work involved with the last two elements is greatly reduced by using PAR. There are a couple of options for the creation of actions using the Jack toolkit. One can import motion capture data or create key frame animation, but both of these options are very dependent on the context in which the action will be performed and therefore inflexible and less than ideal for a dynamically changing virtual environment. One can also use the inverse kinematics that the toolkit provides. This can provide more general actions such as general reaching, but the inverse kinematics in the toolkit is not as robust as one would like. Finally, one can create programs that use the low-level tools provided by the toolkit. These programs can provide reusable, robust actions. Unfortunately, the creation of such programs requires extensive knowledge of graphics and human movement. Even then, the resulting actions may appear unnatural and detract from the virtual training environment. It is also necessary to create natural transitions between actions.

When using PAR, few or no actions need to be created. The PAR Actionary™ is a database of parameterized actions that are linked to motion generators. These motion generators are in turn linked to the Jack toolkit API. The motion generators in PAR include generators for; locomotion, gesturing, facial expressions, gross body movements, and reaching and its family of actions. Many years of research in human modeling and simulation have generated PAR's motion generators. While improvement and refinement of these programs continues, they are dynamic and robust enough for ever changing virtual environments including user input. Additionally, motion capture data can be automatically parameterized and stored as PARs for use in the PAR system (Bindiganavale 2000). Motion capture data creates the most natural looking actions, and parameterizing them into their most essential components increases their robustness. Finally, the way in which actions are given to the characters to be performed provides methods for natural action transitions. Each character in the virtual world has an associated action queue. By examining this queue, one can dynamically

determine the sequence of actions the character is about to perform and determine the necessary action transitions.

The Jack toolkit includes access to the position and orientation of objects and modules for collision detection between objects, but no higher-level interaction facilities. Coordination between characters must be hand scripted before the simulation begins and user interaction is limited. PAR expands these capabilities to a much higher level. The object hierarchy of the Actionary™, in addition to elementary graphics parameters, contains information about the semantics of objects, thus allowing characters a richer interaction with the environment through higher level planning. Each character in the environment has an associated agent process that is responsible to determining what action to perform and calling the motion generators necessary to perform it. During this process it can query the object hierarchy to determine the current state of the environment. This hierarchy is continuously updated during the simulation. Hence, each character can dynamically react to changes in the environment. Such reaction is impossible in pre-scripted scenarios such as those built with lower level tools such as the Jack toolkit. PAR also contains message-passing facilities so that characters can communicate and coordinate with each other.

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. These require parameterizing and, in turn, proper specification of parameters.

Building a virtual human model that admits control from sources other than direct animator manipulations requires an architecture that supports higher level expressions of movement. Although layered architectures for autonomous beings are not new (R. Brooks 1989, D. Zeltzer 1990), we have found that a particular set of architectural levels seems to provide efficient localization of control for both graphics and language requirements. Our multilevel architecture is grounded in typical graphical models and articulation structures, and includes various motor skills for endowing virtual humans with useful abilities. The higher architectural levels organize these skills with parallel automata and use a conceptual representation to describe the actions a virtual human can perform.

A typical virtual human model design consists of a geometric skin and an articulated skeleton. Usually modeled with polygons to optimize graphical display speed, a human body can be crafted manually or shaped more automatically from digitized body. The surface may be rigid or, more realistically, deformable during movement. Deformation demands additional modeling and computational loads. Clothes are desirable, though today, loose garments have to be animated off-line due to computational complexity (P. Volino 1995). The skeletal structure is usually a hierarchy of segments connected together by joint rotation transformations (N. Badler 1993). The body is moved by changing the joint angles and its global position and location. In sophisticated models, joint angle changes induce geometric modifications that keep joint surfaces smooth and mimic human musculature within a character's particular body segment (J. Wilhelms 1997, B.J. Ting 1998, M. Cavazza 1998). A virtual human's actions can be generated either directly by live motion capture or by motion generators, which are procedures for changing joint angles and body position.

Motion generators should be capable of:

- Playing a stored motion sequence that may have been synthesized by a procedure, captured from a live person, or scripted manually
- Blending one movement into another, in sequence or in parallel
- Generate various motor skills like:
  - Changing postures and adjusting balance
  - Reaching and other arm gestures
  - Grasping and other hand gestures
  - Locomoting, such as stepping, walking, running, and climbing
  - Looking and other eye and head gestures
  - Facial expressions, such as lip and eye movements
  - Physical force- and torque-induced movements, such as jumping, falling, and swinging

Numerous methods help create each of these movements. But, we want to allow several of them to be executed simultaneously. A virtual human should be able to walk, talk, and chew gum at the same time. Simultaneous execution also leads to the next level of our architecture's organization: parallel automata.

Our parallel programming model for virtual humans is called Parallel Transition Networks, or PaT-Nets (Badler N. 1993). Other human animation systems, including Motion Factory's Motivate and New York University's Improv (Perlin K. 1996), have adopted similar paradigms with alternative syntactic structures. In general, network nodes represent processes. Arcs connect the nodes and contain predicates, conditions, rules, and other functions that trigger transitions to other process nodes. Synchronization across processes or networks is made possible through message-passing or global variable blackboards to let one process know the state of another process. The benefits of PaT-Nets derive not only from their parallel organization and execution of low-level motion generators, but from their conditional structure. Traditional animation tools use linear timelines on which actions are placed and ordered. A PaT-Net provides a nonlinear animation model, since movements can be triggered, modified, and stopped by transitions to other nodes. Nonlinear animation is a crucial step toward autonomous behavior, since conditional execution is key to a virtual human's inter-activity, reactivity, and decision-making.

Providing a virtual human with humanlike reactions and decision-making skills is more complicated than just controlling its joint motions from captured or synthesized data. Simulated humanlike actions and decisions are how we convince the viewer of the character's skill and intelligence in negotiating its environment, interacting with its spatial situation, and engaging other agents. This level of performance requires significant investment in action models that allow conditional execution.

PaT-Nets are effective but must be hand-coded in C++. No matter what artificial language we invent to describe human actions, it is not likely to represent exactly the way people conceptualize a particular situation. We therefore need a higher-level representation to capture additional information, parameters, and aspects of human action. We create such representations by incorporating natural-language semantics into a parameterized action representation.

A PAR (N. Badler 1997) gives a complete description of an action. The PAR has to specify the agent of the action, as well as any relevant objects and information about paths, locations, manners, and purposes for a particular action. There are linguistic constraints on how this information can be conveyed by the language; agents and objects tend to be verb arguments, paths are often

prepositional phrases, and manners and purposes might be in additional clauses (M. Palmer 1998). A parser maps the components of an instruction into the parameters or variables of the PAR, which is then linked directly to PaT-Nets executing the specified movement generators.

The PAR was designed to bridge the gap between natural language and animations. We use the example "Walk to the door and turn the handle slowly" to illustrate the function of the PAR. Whether or not the PAR system processes this instruction, there is nothing explicit in the linguistic representation about grasping the handle or which direction it will have to be turned, yet this information is necessary to the action's actual visible performance. The PAR has to include information about applicability and preparatory and termination conditions in order to fill in these gaps. It also has to be parameterized, because other details of the action depend on the PAR's participants, including agents, objects, and other attributes. The representation of the "handle" object lists the actions that the object can perform and what state changes they cause. The number of steps it will take to get to the door depends on the agent's size and starting location.

Some of the parameters in a PAR template are defined in the following ways:

- Participants:

- Agent: The agent executes the action. In our example, the walking and turning actions share the same agent. We assume that the agent refers to a human model or a physical force like gravity (in which case the agent is understood to be causal and not volitional) or a process simulator. An agent has a specific personality and a set of actions it is capable of performing. The agent can also be considered to be capable of playing different roles. For each role, the agent performs different actions. So, instead of maintaining one long list of actions, we could group these actions under different roles. For example, the actions involved while driving a car like grasping a steering wheel, sitting with foot on the accelerator pedal, etc., would be grouped under the "car-driver" role. Each of the listed actions is a primitive action. Unlike for the objects, each action is associated with a set of preparatory conditions (test for reachability, etc) which check if the agent can perform the action. If not, another set of primitive actions is generated for that agent which have to be completed before the current action can be performed.

The agent type also has a field for specifying nominal values and the distribution type and range for some of the actions and state space descriptors. For example, the walking rate of the agent could be specified to have a nominal value of 2 steps per second with a normal distribution and standard deviation of 1. This gives a range of values over which the walking rate can be varied.

- Objects: The object type is defined explicitly for a complete representation of a physical object and is stored hierarchically in a database. Each object in the environment is an instance of this type and is associated with a graphical model. The state field of an object describes a set of constraints on the object which leave it in a default state. The object continues in this state until a new set of constraints is imposed on the object by an action that causes a change in state. The other important fields are the reference coordinate frame, a list of grasp sites and directions defined with respect to the object. In our example, the walking action has an implicit floor as an object, while the turn action refers to the handle.

- Start: This is the time at which the action begins.
- Result: This is the time at which the action ends.
- Applicability conditions: The applicability conditions of an action specify what needs to be true in the world in order to carry out an action. These can refer to agent capabilities, object configurations and other unchangeable or uncontrollable aspects of the environment. The conditions in this boolean expression must be true to perform the action. For "walk," one of the applicability conditions may be "Can the agent walk?" If conditions are not satisfied, the action cannot be executed. In some instances, the applicability conditions may also replace an action with a more specific one: opening the door might be specialized to a sliding action if that is what this particular door calls for.
- 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 proceed. If the conditions are not satisfied, 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 described below. In general, actions can involve the full power of motion planning to determine, perhaps, that a handle has to be grasped before it can be turned. The instructions are essentially goal requests, and the smart avatar must then figure out how (if possible) it can achieve them. We use hand-coded conditionals to test for likely (but generalized) situations and execute appropriate intermediate actions. Adding more general action planners is also possible, since the PAR represents goal states and supports a full graphical model of the current world state (T. Trias 1996). In our example, one of the conditions to be checked could be standing?(agent) and the corresponding action could be ("stand",agents: ("Jack")). If the agent is not standing, e.g., if he is sitting or prone, then the action causes him to change to the standing posture.
- Sub-actions: Each action is organized into partially ordered or parallel substeps, called sub-actions. Actions described by PARs are ultimately executed as PaT-Nets.
- Execution Steps: A PAR can describe either a primitive or a complex action. The execution steps contain the details of executing the action after all the conditions have been satisfied. If it is a primitive action, the underlying Pat-Net for the action is directly invoked. A complex action can list a number of sub-actions that may need to be executed in sequence, parallel, or a combination of both. A complex action can be considered done if all of its sub-actions are done or if its explicit termination conditions are satisfied.
- Core semantics: These semantics represent an action's primary components of meaning and include motion, force, path, purpose, termination conditions, post-assertions, duration, and agent manner. For example, "walking" is a form of locomotion that results in a change of location. "Turning" requires a direction and an end point.
- Manner: Manner specifications describe the way in which an agent carries out an action. We define manner as composed of Effort and Shape parameters that are derived from Effort Notation (Bartenieff 1980) expressing the quality of a movement. Each parameter takes on a real value in the range from -1 to 1.

Effort elements are weight, space, time, and flow and can be combined and phrased to vary the dynamics of the movement. Shape elements are respectively advancing/retreating, spreading/enclosing, rising/sinking and opening/closing and can be combined to affect the form of the movement.

- Termination Conditions: This is a list of conditions which when satisfied complete the action. A termination condition can be determined from the main verb or attached clauses (J. Bourne 1998).
- 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.

A PAR can appear as one of two different forms: uninstantiated PAR (UPAR) and instantiated PAR (IPAR): We store all instances of the UPAR, which contains default applicability conditions, preparatory specifications, and execution steps, in a hierarchical database called the Actionary. An IPAR is a UPAR instantiated with specific information on agent, physical object(s), manner, termination conditions, and more. Any new information in an IPAR overrides the corresponding UPAR default.

For more information of PAR and its use with virtual environments for training please see our publications (Badler et al. 2002, Allbeck et al. 2002, Allbeck and Badler 2002, Ashida et al. 2001, Badler and Allbeck 2000, Bindiganavale et al. 2000, Badler et al. 2000).

#### Referencea:

J. Allbeck and N. Badler. Toward Representing Agent Behaviors Modified by Personality and Emotion, "Workshop Embodied conversational agents - let's specify and evaluate them!" at AAMAS 2002, Bologna, Italy.

J. Allbeck, K. Kipper, C. Adams, W. Schuler, E. Zoubanova, N. Badler, M. Palmer, and A. Joshi. ACUMEN: Amplifying Control and Understanding of Multiple ENTities, Proc. of Autonomous Agents and Multi-Agent Systems, ACM Press, Vol. 1. pages 191-198. July 2002, Bologna, Italy.

K. Ashida, Seung-Joo Lee, Jan Allbeck, Harold Sun, Norman Badler, and Dimitris Metaxas. Pedestrians: Creating Agent Behaviors through Statistical Analysis of Observation Data. Computer Animation 2001.

N. I. Badler and Jan M. Allbeck. Towards Behavioral Consistency in Animated Agents. Proceedings of Deformable Avatars 2000.

N. Badler, J. Allbeck, L. Zhao, and M. Byun. Representing and Parameterizing Agent Behaviors, Proc. Computer Animation, IEEE Computer Society June 2002, Geneva, Switzerland.

N.I. Badler, R. Bindiganavale, J. Allbeck, W. Schuler, L. Zhao, and M. Palmer. "Parameterized Action Representation for Virtual Human Agents. In J. Cassell (ed.), "Embodied Conversational Agents." MIT Press, 2000, pages 256-284.

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

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

Bartenieff and D. Lewis. *Body Movement: Coping with the Environment*. Gordon and Breach Science Publishers, New York, NY, 1980.

R. Bindiganavale. Building parameterized action representations from observation. Technical Report, MS-CIS-00-17. PhD Thesis. July 2000. University of Pennsylvania.

R. Bindiganavale and N. Badler. Motion Abstraction and mapping with spatial constraints. In *Modelling and Motion Capture Technologies for Virtual Environments*, International Workshop, CAPTECH'98, Geneva, Switzerland. pp 70-82.

R. Bindiganavale, W. Schuler, J.M. Allbeck, N.I. Badler, A.K. Joshi, and M. Palmer. Dynamically Altering Agent Behaviors Using Natural Language Instructions. *Autonomous Agents 2000*, pages 293-300.

J. Bourne. Generating adequate instructions: Knowing when to stop. In *Proc. of the AAAI/IAAI Conf., Doctoral Consortium Section*, Madison, WI, 1998

R. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. *Neural Computation*, 1(2), 1989.

M. Cavazza, R. Earnshaw, N. Magnenat-Thalmann, and D. Thalmann. Motion control of virtual humans. *IEEE Computer Graphics and Applications*, 18(5), pages 24-31, Sept.-Oct. 1998.

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

K. Perlin and A. Goldberg. Improv: A system for scripting interactive actors in virtual worlds. In *ACM Computer Graphics Annual Conf.*, pages 205-216, 1996.

B.J. Ting. Real time human model design. PhD thesis, CIS, University of Pennsylvania, 1998.

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.

P. Volino, M. Courchesne, N. Magnenat-Thalmann. Versatile and efficient techniques for simulating cloth and other deformable objects. In *Computer Graphics*, SIGGRAPH, pp 137-144, 1995.

J. Wilhelms and A. van Gelder. Anatomically-based modeling. *Proc. ACM SIGGRAPH Annual Conf.*, pages 173-180, July 1997.

D. Zeltzer. Task-level graphical simulation: Abstraction, representation, and control. In N. Badler, B. Barsky, and D. Zeltzer, editors, *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, pages 3-33, Morgan-Kaufmann, San Francisco, 1990.