

## Behavior-Based Script Language for Anthropomorphic Avatar Animation in Virtual Environments

Dorina C. Petriu <sup>(1)</sup>, Xiao Li Yang <sup>(1)</sup>, and Thom E. Whalen<sup>(2)</sup>

<sup>(1)</sup> Carleton University, Ottawa, ON, Canada

<sup>(2)</sup> Communications Research Centre Canada, Ottawa, ON, Canada

**Abstract** – This paper reports development aspects of a behavior-based script language for the animation of anthropomorphic avatars in virtual reality environments.

**Keywords** – anthropomorphic avatar, animation, script language, virtual reality, behavior-based control

### I. INTRODUCTION

Virtual environment actors are represented by icons, called *avatars*, which can move through and manipulate objects in the virtual environment. Currently, avatars vary from completely abstract geometric figures to rigid models that look like a hybrid between a Lego toy and a Barbie doll. Nevertheless, there is an interest for realistic anthropomorphic model for a multitude of applications such as: multimedia communications, computer graphics in entertainment, experiments in natural language interactions, interpersonal collaboration, and interfaces for avatar control.

*Script languages* are currently developed to allow complex avatar story development. The existent script languages are application dependent. One class of languages is intended to describe movements with little provision for interactions and sensory input, [1] and [2]. Other class is intended for the description of interactions but provide little provision for story development, [3] and [4]. Since in our case the script will have to describe complex stories we will have to combine the features of these two classes. The "interval script language" proposed in [5] could provide the temporal dimension for the script.

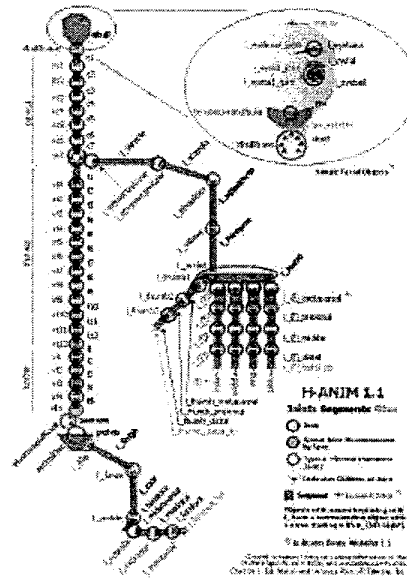
This paper discusses definition and implementation aspects of a behavior-based script language for the autonomous or interactive animation of a standard avatar.

### II. AVATAR STRUCTURE

We are using an "off-the web" *H-Anim* humanoid body representation, Fig. 1, working in any VRML97 compliant browser, [6]. This standard humanoid was created for the express purpose of portability, i.e. to allow using authoring tools from one vendor to be animated using tools from another. No assumptions were made about the types of applica-

tions that will use humanoids. These humanoids can be animated using keyframing, inverse kinematics, performance animation systems and other techniques.

THIS IMAGE IS HERE FOR HISTORICAL PURPOSES ONLY...  
FOR A CORRECT SKELETAL DESCRIPTION FOR H-ANIM 1.1  
SEE: <http://H-Anim.org/Specifications/H-Anim1.1/>



The **Left Hand joints** are: l\_thumbbase, l\_thumbmid, l\_thumbext, l\_indexbase, l\_indexmid, l\_indexext, l\_middlebase, l\_middlemid, l\_middleext, l\_ringbase, l\_ringmid, l\_ringext, l\_pinkybase, l\_pinkymid, l\_pinkyext.

The **Right Hand joints** are: "r\_thumbbase" "r\_thumbmid" "r\_thumbext" "r\_indexbase" "r\_indexmid" "r\_indexext" "r\_middlebase" "r\_middlemid" "r\_middleext" "r\_ringbase" "r\_ringmid" "r\_ringext" "r\_pinkybase" "r\_pinkymid" "r\_pinkyext"

The **Body segments** are: "pelvis" "l\_thigh" "l\_calf" "l\_hindfoot" "r\_thigh" "r\_calf" "r\_hindfoot" "c7" "l\_upperarm" "l\_forearm" "l\_hand" "r\_upperarm" "r\_forearm" "r\_hand" "c4"



Fig. 2. The avatar Nancy, from [7].

### III. CONTROLLING THE AVATAR'S ANIMATION

The basic problem is the control of avatar's individual joints. The standard definition of avatar joints helps the standardization of this *joint-level control*.

The *second level* is the control of basic behaviors/skills, [8], of the humanoid avatar. These basic behaviors can be implemented using analytic inverse kinematic and dynamic transforms, [9] and [10], and/or *Neural Networks* (NN).

The *third level* of control is the script language. It allows the user to input high level English-like instructions commanding the avatar to carry out a series of tasks.

VRML 97, the *Virtual Reality Modeling Language*, [11], is an open, extensible, industry-standard scene description language for 3D scenes, or worlds, on the Internet, [12]. Fig. 3. shows a 3D location created in VRML for Nancy's animation scripts.

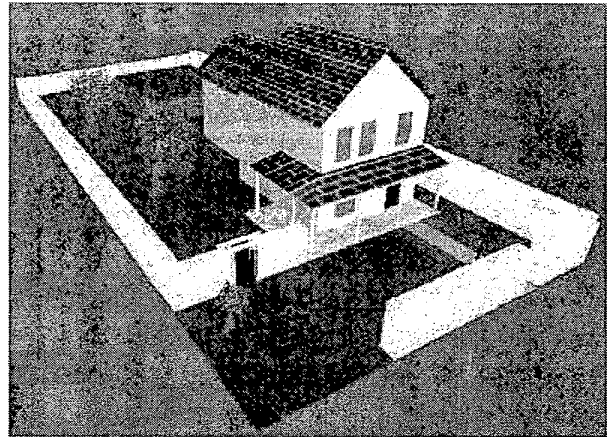


Fig. 3. VRML location for Nancy script playing.

#### 3.1. Joint level control

VRML is not a programming language. It cannot be used to program the joint level movements of an avatar kinematic structure. JAVA, [13], offers a more powerful solution by using an External Authoring Interface (EAI), [14], which defines the way to communicate with a VRML world.

Through the Java interface, the controller in Java connects with the joints and segments of the avatar in VRML. The user can control any joint to translate or rotate through the User Interface. Each joint of the *H-Anim* model may have six degrees of freedom, three rotations and three translations, defined in the 3D Cartesian frame (*X*, *Y* and *Z*) attached to that joint. However, anatomical considerations impose joint specific constraints on the degrees of freedom. For instance, the "sacroiliac" joint could rotate around its *Y* axis from  $-90$  to  $90$  degrees.

#### 3.2. Skill-level control

The skills appear as a preprogrammed purposeful sequence of joint movements. By now, we have implemented 15 skills. These skills are divided in different groups according to the common joints they are using. Skills in the same group are

mutually exclusive, whereas skills from different groups can be executed in parallel. The groups are as follows:

- Group1** : stand, walk, run, jump, turn\_left, turn\_right, go to the door, go to the yard door, open the door
- Group2** : wave\_left\_hand
- Group3** : wave\_right\_hand
- Group4** : kick\_left\_leg, kick\_right\_leg
- Group5** : turn\_head\_left, turn\_head\_right .

#### Basic skills

There are two kinds of basic skills: ones describing independent behaviors of the avatar, and the others describing the interaction of the avatar with the objects in the virtual environment.

For the *independent behavior skills*, the controller defines different joints movements at given times (some movements may be simultaneous, others in sequence). The program controlling the sequence of joint movements can be saved in a data structure or computed on the spot.

Here is an example of how such a basic skill “walk” was implemented by using Java threads. The user can set two walk parameters: “walkSteps” defines the number of steps, and “walkCycle” sets the duration for one step. After the “walk” command and its parameters are given to the *Skill-level Controller* through the *User Interface*, a “walk action” thread is created, which is passed the walk parameters. The *Walk Action Thread* creates a “stage1” thread, which creates in turn three control threads for the left arm, head and right leg. For moving the left arm, the corresponding thread controls three other threads for the left shoulder, elbow and wrist. All the threads for the left arm, head and right leg will send control events to the *JointControl* class, which will forward them to the corresponding *VRML objects*. These objects will change the positions or the orientation of the avatar’s left arm, head and right leg, respectively. After creating the stage1 thread, *Walk Action Thread* will sleep for (42 \*walkCycle) ms before starting the next walk stage. Similarly to the first stage, a “stage2” thread created will create in turn another two threads for the right arm and left leg. For example, the left leg thread controls other three threads for the left hip, left knee and left ankle. These interact with the *JointControl*, and so on. After creating the stage2 thread, *Walk Action Thread* will sleep for (83 \*walkCycle) ms before decrementing the cycle counter and starting the next cycle (i.e., step). The walk will stop if the count for steps becomes zero.

Java threads are used to achieve the concurrent movement of all the joints needed in one stage. The “sleep” times are used to control the time sequence for the whole walk cycle.

The user can set the following parameters for the independent behaviors implemented so far:

- “walk” – cycle (duration for one step), steps (how many steps the avatar can walk);
- “run” – cycle (duration for one step), steps (how many steps the avatar can run);
- “jump” – cycle (duration for one step), steps (how many steps the avatar can jump);
- “turn\_left” – angle (for the whole body to turn);
- “turn\_right” – angle (for the whole body to turn);
- “turn\_head\_left” – cycle (duration for implementing once), times (how many times avatar does this behavior) , angle (the angle to turn);
- “turn\_head\_right” – cycle (duration for implementing once), times (how many times avatar does this behavior) , angle (the angle to turn).

For the *interaction skills*, the behavior of the avatar depends on both the avatar position and the object in the Virtual Environment. An example of such interactive behavior is “open the door”. The user inputs the command “open the door” through the user interface. *Skill-level Controller* will send the events to *JointControl* to get the positions for the doorknob and the avatar from the *VRML objects*. The inverse kinematics is used by *Skill-level Controller* to calculate the angles of joints on the right arm according to the doorknob position and the end point (hand) position. After getting the angles for the whole right arm, *Skill-level Controller* will create a “touch doorknob thread”. This creates other four threads for moving the right shoulder, right elbow, right wrist and right hand at the same time according to the computed angles. After the controller creates the “touch doorknob thread”, it will sleep for 20 milliseconds and wait the rotations and translations in the virtual environment to be done. The controller sends events to *JointControl* for changing the virtual environment to another virtual environment or background.

#### Combining skills

The skills can be combined in sequence and in parallel (concurrently).

When *combined in sequence*, one behavior can begin only after the one finishes.

Basic skills can be *combined concurrently* if they belong to different groups. For example, when the avatar is walking, it can also wave a hand, but cannot walk and run at the same time. Fig. 4 gives a UML sequence diagram that shows how “walk” and “wave left hand” are done in parallel.

The user inputs “walk” for 5 steps first. The *Skill-level Controller* will create a *Walk Action Thread*, which creates other threads as explained before (not shown in the figure) which will send eventually walk events to the *JointControl* class. Assume that the user inputs “wave left hand” when the avatar is still walking. The *Skill-level Controller* will tell the *Walk Action Thread* to stop moving the left hand while continuing

the walk. The *Skill-level Controller* will also create *Wave Left Hand Thread*, which will control the hand waving through the *JointControl* class. After the waving the hand three times, the respective thread will announce the *Skill-level Controller* that is done and will terminate itself. *Skill-level Controller* will then inform the *Walk Action Thread* to resume the left arm's movement when walking, until the five steps are done.

Fig. 5 shows the VRML interface we have developed for prototyping different control levels for the animation of Nancy.

#### IV. SCRIPT LANGUAGE DEVELOPMENT

We are currently developing a higher-level script language to allow the user to create stories (scripts) consisting of more complex behaviors, [15].

An example of such a script is:

*Nancy walks to the door. She looks around after 1 step, sees her neighbor, and waves her hand for 3 times while walking.*

This script will then be parsed into a sequence of basic or combined skills:

##### **Nancy**

*Goes to the door.*

*Turn head begins after one step (this will stop head movement controlled by walking).*

*Turns her head to a specific angle while walking.*

*Wave left or right hand while walking.*

*Continue normal walking to the door.*

If the user doesn't specify explicitly the parameters for every basic skill, some default parameters will be used.

We are currently studying the use of *Neural Networks* (NN) techniques for the development of more life-like personalized skills. These behavioral NN's are trained off-line by sensory-driven examples, [16] and [17].

The script language will support the inherent parallelism in the plan of actions of a multitude of avatars and other agents evolving in a virtual environment. We are using a resource management system with a decentralized architecture, [18], consisting of a script-instruction planner, free space manager, and local controllers for each avatar and mobile agent in the virtual environment. The *script-instruction planner* is responsible for determining which script-instruction can be executed and the generation of the skill level routines for executing that script-instruction. The *free space manager*

allocates free pathways to different avatars and other mobile agents, which have to share the same space in the virtual environment.

#### ACKNOWLEDGMENTS

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada, the National Capital Institute of Telecommunications, and the Communications Research Centre Canada.

#### REFERENCES

- [1] N.M. Thalmann and D. Thalmann, *Synthetic Actors in Computer Generated 3D Films*, Springer-Verlag, Berlin, Germany, 1990.
- [2] K. Perlin, "Real Time Responsive Animation with Personality," *IEEE Tr. Visualization and Computer Graphics*, Vol. 1, No. 1, pp. 5-15, 1995.
- [3] M.C. Buchanan and P.T. Zellweger, "Automatic Temporal Layout Mechanisms," *Proc. ACM Multimedia'93*, pp. 341-350, 1993.
- [4] R. Hamakawa and J. Rekimoto, "Object Composition and Playback Models for Handling Multimedia Data," *Proc. ACM Multimedia'93*, pp. 273-281, 1993.
- [5] C.S. Pinhanez, K. Mase, and A.F. Bobick, "Interval Scripts: A Design Paradigm for Story-Based Interactive Systems," *Proc. CHI'97*, pp. 287-294, 1997.
- [7] \*\*\*, *H-Anim Examples*, <http://www.ballreich.net/vrml/h-anim/h-anim-examples.html>
- [6] \*\*\*, *The Humanoid Animation Working Group*, <http://h-anim.org/>
- [8] C. Archibald and E.M. Petriu, "Skills-Oriented Robot Programming," *Proc. Int. Conf. Intell. Autonomous Syst. IAS-3*, pp. 104-113, Pittsburgh, PA, 1993.
- [9] J. Zhao, and N. Badler, "Inverse kinematics positioning using nonlinear programming for highly articulated figures," *ACM Transactions on Graphics*, 13(4), 313-336.
- [10] H. Ko and N.I. Badler, "Animating Human Locomotion with Inverse Dynamics," *IEEE Computer Graphics and Applications*, pp. 50-59, March 1996.
- [11] \*\*\*, *VRML 97 reference page*, <http://www.web3d.org/Specifications/VRML97>
- [12] \*\*\*, *VRML*, <http://www.vrml.org>
- [13] \*\*\*, *JAVA reference page*, <http://www.sun.com/java/>
- [14] \*\*\*, *EAI reference page*, <http://www.vrml.org/Specifications/VRML97/part1/java.html>
- [15] K. Perlin and A. Goldberg, "Improv: A System for Scripting Interactive Actors in Virtual Worlds," *Computer Graphics*, Vol. 29 No. 3., 1996
- [16] P. Andry, P. Gaussier, S. Moga, J.P. Banquet, and J. Nadel, "Learning and Communication via Imitation: An Autonomous Perspective," *IEEE Trans. Syst. Man Cybern. -Part C: Applications and Reviews*, Vol. 31, No.5, pp. 431-442, Sept. 2001
- [17] C. Luciano, P. Banerjee, and S. Mehrota, "3D Animation of Telecollaborative Anthropomorphic Avatars," *Comm. ACM*, Vol. 44, No. 12, pp. 64-67, Dec. 2001
- [18] D.C. Petriu, T.R. Jones, and E.M. Petriu, "Communicating State-Machine Model of Resource Management in a Multi-Robot Assembly Cell," *ISCA J. Computers Applications*, Vol. 1, No. 1, pp. 43-48, 1994.

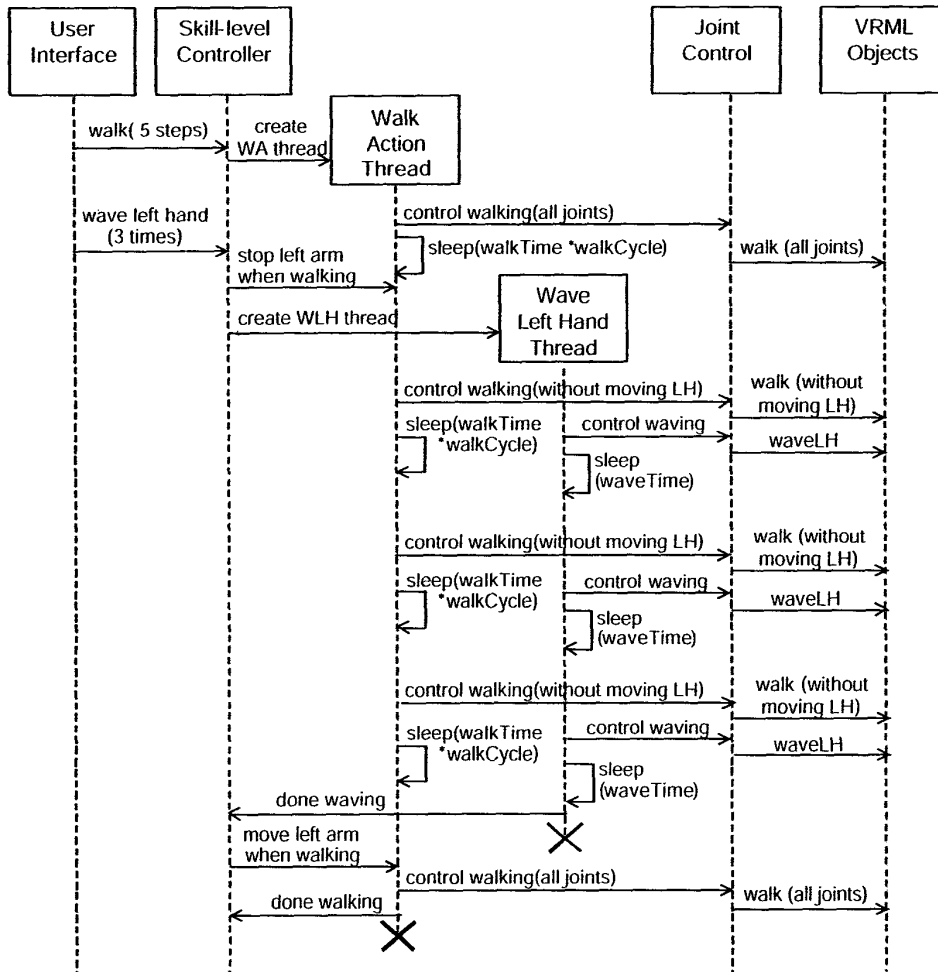


Fig. 4. UML sequence diagram illustrating the "walk" and "wave left hand" done in parallel.

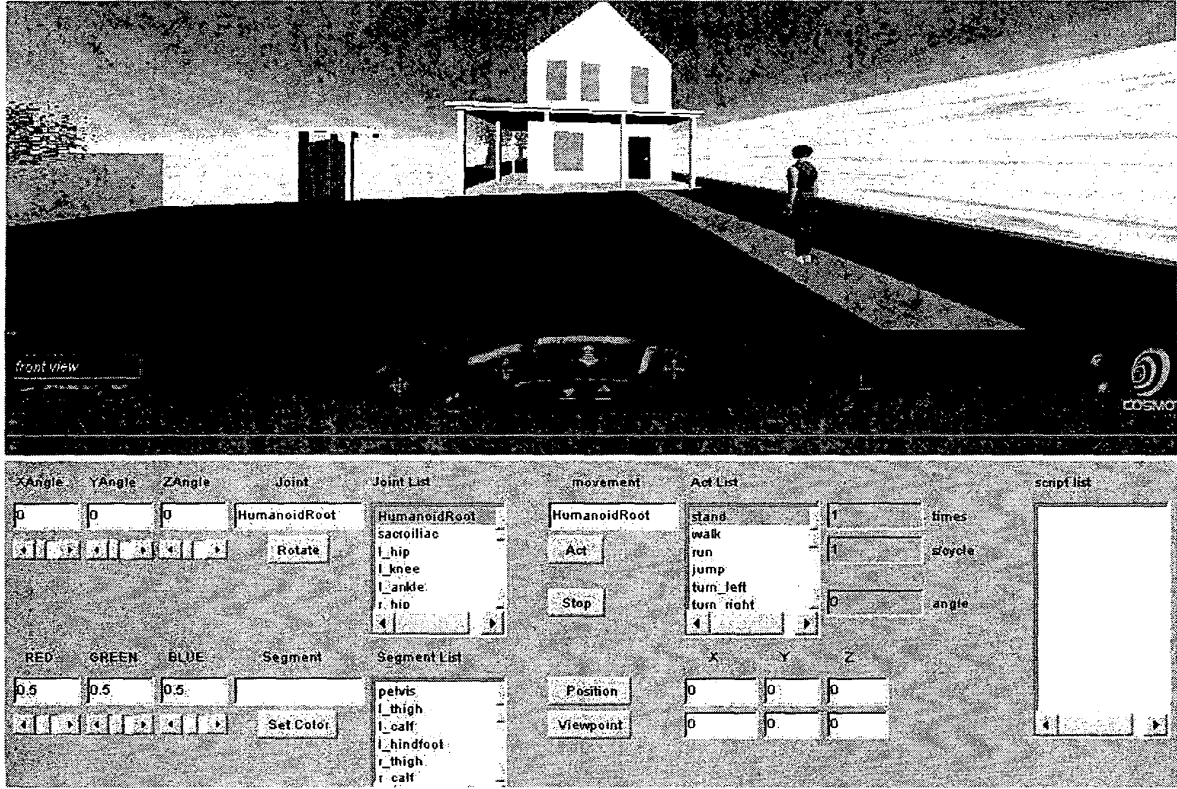


Fig. 5. VRML interface for different level of control for Nancy's animation.