

Hierarchical Animation Control of Avatars in 3-D Virtual Environments

Xiaoli Yang, *Member, IEEE*, Dorina C. Petriu, *Senior Member, IEEE*, Thom E. Whalen, and Emil M. Petriu, *Fellow, IEEE*

Abstract—This paper reports the development of a three-layer hierarchical control system for the animation of anthropomorphic avatars in three-dimensional virtual environments. The lower layer controls the movement of the avatar's joints, the second layer defines skills or basic behaviors (such as walk, run, and jump), and the highest layer executes a behavior-based script language that can be used to describe stories to be enacted by the avatars. In order to verify how flexible the proposed hierarchical approach is, the two lower control layers were implemented in two different ways, without affecting the script language layer.

Index Terms—Animation, anthropomorphic avatar, behavior-based control, script language, virtual environments.

I. INTRODUCTION

VIRTUAL environment actors are represented by icons, called *avatars*, which can move around 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 interest for realistic anthropomorphic models 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.

There are a variety of approaches to avatar animation that attempt to simulate real human beings as closely as possible in three-dimensional (3-D) virtual environments. The Virtual Reality Modeling Language (VRML) [6], [20], [30] is frequently used for the definition of the avatars, and more rarely used as the visualization and integration technology. Only some simple and repeatable procedures can be simulated this way, as VRML itself is not a programming language and does not give the user many options to interact with the virtual environment.

Another approach is to represent avatars by real-time agents [4]. For the avatar movements, a time series of joint angles are stored so that specific motions can be replayed under real-time

constraints [8]. The major advantages of prestored motions are safer algorithmic security and fast execution speed (by minimizing the computation). However, the disadvantage is that prestored motions lack generality (since every joint must be controlled explicitly) and anthropometrical extensibility (since changing joint-to-joint distances will change the computed locations of end effectors such as feet, making external constraints and contacts impossible to maintain) [4].

The Improv System developed by Perlin at New York University mainly controls avatars by behavioral scripts [14]–[16]. The behavioral scripts are designed such that they can be translated easily from a given storyboard. In this system, noise techniques are used to change image rendering to character movement. Characters created in this manner can smoothly blend and layer animations. The system is useful for predefined scenarios in the virtual environment. External functions need to be added if the approach is used in a real-time controlled virtual environment and the characters need to make their decisions often.

Agent Common Environment (ACE), developed by the EPFL computer Graphics Laboratory, is a platform for virtual human agent simulation that is able to coherently manage a shared virtual environment [10]. This system can simulate scenarios in the virtual environment. It provides built-in commands for perception and acting. It also defines an external collection of behavioral plug-ins. The user can use these behaviors to create a story for the virtual environment, but must extend some control functions if (s)he wants to use this system for a specific task.

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, [22] and [15]. Other class is intended for the description of interactions, but provide little provision for story development, [5] and [9]. Since, in our case, the script will describe complex stories, we will combine the features of these two classes. The “interval script language” proposed in [19] could provide the temporal dimension for the script.

This paper is an extended version of the work presented by the authors in [18], [23]. It describes a three-layer hierarchical control system for the animation of avatars, which are represented according to the International ISO/IEC Humanoid Standard proposed by the Humanoid Animation Group [25]. The lower layer controls the rotation of the avatar's joints. The second layer defines skills or basic behaviors (such as walk, run, and jump), which can be combined in sequence or in parallel. The last level of abstraction is represented by a script language, which can be used to describe stories that will be enacted by the avatars. In

Manuscript received November 3, 2003; revised September 2, 2004. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada, in part by the National Capital Institute of Telecommunications (NCIT), and in part by the Communications Research Centre (CRC) Canada.

X. Yang is with the Department of Electrical Engineering, Lakehead University, Thunder Bay, ON P7B 5E1, Canada.

D. C. Petriu is with the Systems and Computer Engineering Department, Carleton University, Ottawa, ON K1S 5B6 Canada.

T. E. Whalen is with the Communications Research Center, Ottawa, ON K2H 8S2, Canada.

E. M. Petriu is with the School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, ON K1N 6N5, Canada.

Digital Object Identifier 10.1109/TIM.2005.847134



Fig. 1. Examples of avatars: Nancy and SphereMan.

order to verify how flexible the proposed hierarchical approach is, the two lower control layers were implemented in two different ways, without affecting the script language layer.

II. AVATAR ANIMATION

VRML is a well-known language for describing interactive 3-D objects and worlds. The design of VRML makes it easy to be used on the Internet, intranets, and local client systems [13], [30]. The 3-D virtual environment and avatar in this paper are defined in VRML.

Fig. 1 shows two examples of avatars chosen to test our avatar animation control system: Nancy, created by 3Name3D Company [30], and SphereMan found at [26]. They are defined in VRML based on the standard Humanoid—H-Anim body representation [25], that was created for the express purpose of portability. No assumptions were made about the types of applications that will use such humanoids. These humanoids can be animated using keyframing, inverse kinematics, performance animation systems and other techniques.

According to H-Anim, an avatar is composed of joints and segments. The difference between various avatars that are defined according to the standard consists in the segment shapes and the positions of their joints. We chose to control the joint rotations for animating the avatar, so that the difference among avatars does not influence the control system. Thus, in our case, the control is not affected by the shape of segments or by the distance between joints. Therefore, the control system can be reused without any modification for any avatar that is defined according to the H-Anim standard.

As mentioned before, our animation system is composed of three hierarchical layers: joint control, skill control, and script language. The design goals were to make the control system easy to develop and extend, by keeping a low dependency between layers. In order to prototype and test different animation control levels, we have developed a control panel, which allows us to test individually any joint movement, basic skill or script-language sentence.

The two lower layers, joint control and skill control, were implemented in two different ways, in Java/VRML and Java/Java3D, as described in the following subsections. Both

implementations support the same interface with the script language layer, which can run on either one without any change.

A. Joint Control

The lowest layer controls all the joints of the avatar, like sacroiliac, elbow, wrist, hip, and so on. For each avatar, the designer may choose to implement only a subset of the joints defined in the H-Anim standard; for example, Nancy has 45 joints and the SphereMan 77 joints. The whole avatar body of a H-Anim model is considered as a special joint that may have six degrees of freedom, three rotations and three translations defined in the 3-D Cartesian frame (X , Y , and Z) attached to it. Each individual joint may have three degrees of freedom (i.e., three rotations without any translation). However, anatomical considerations impose joint specific constraints on the degrees of freedom. For instance, the “sacroiliac” joint can rotate around its Y axis from -90° to 90° ; the “knee” joint can rotate around its X axis from 0° to 180° , without any rotation around the Y axis. These constraints are defined in our implementation for each joint and enforced by the control system.

The following subsections describe two implementations of the joint control layer, one in VRML/Java, and the other in Java3D/Java. The user can control any individual joint through the control panel in both implementations, and find out its rotation restrictions. An error message pops up if a joint movement specified by the user does not respect the constraints.

1) *VRML/Java Joint Control*: In our first implementation, the control of the avatars defined in VRML according to the H-Anim standard, was developed in Java. Since VRML is not a programming language, it cannot be used to program complex movements such as those of an avatar. A combination of VRML and Java offers a more powerful solution, as Java is a programming language able to realize a more flexible and detailed control of the VRML world than VRML itself. The communication between the VRML objects and the Java control takes place through the external authoring interface (EAI) [28], [29]. Any node in VRML can have a corresponding Java control object, with which is communicating through events provided by EAI. The control code for the rotation/translation of every joint is defined in a separate Java object running in its own thread of control (by using a Java thread). The reason for using a thread for each joint is to give the impression that the different joints involved in a basic behavior are moving simultaneously.

Fig. 2 shows the classes used for controlling a joint in the VRML/Java version. The user inputs a joint rotation command through the *ControlPanel*. The *Controller* class gets the command and its parameters, verifies it and passes it to another class, *VbRotation*, which calculates the actual rotation data to be used by *JointThread*. In turn, *JointThread* generates an event that triggers the *JointControl* class, which will communicate with the corresponding VRML node through EAI; the desired command will be performed in the VRML world.

2) *Java3D/Java Joint Control*: The second implementation uses Java3D for the modeling of the avatar and its environment. The virtual environment and avatar defined initially in VRML, are converted to Java3D by using the so-called “file loader” CyberVRML97 [27], which reads and writes VRML files, sets, and gets the scene graph information, draws the geometries, and

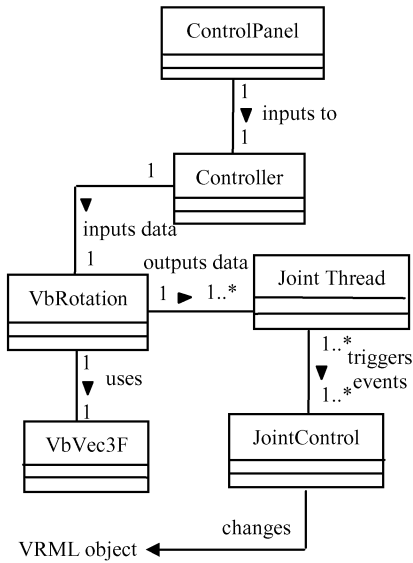


Fig. 2. Class diagram for joint control in VRML/Java.

runs the behaviors. The VRML objects are first loaded with CyberVRML97, then converted automatically into corresponding Java3D objects. Therefore, we do not have to define Java objects for each VRML node by ourselves, like in the first implementation. Java3D is a high-level scene graph-based API [28]. It uses either DirectX or OpenGL low level API to take advantage of the 3-D hardware acceleration. This gives the platform-independence one expects from a Java API, along with a high level of performance needed by today’s sophisticated 3-D applications.

The actual control of the Java3D objects was also implemented in Java, only this time the interface between the Java control code and the Java3D objects is seamless. Since the rendering is entirely the responsibility of Java3D and the slow interface between Java and VRML is no longer used during the animation, the control code is simpler and faster. We did not need to use different threads for different joints, as in the previous implementation. Fig. 3 shows the class diagram for controlling a joint in this case. The only classes required are *Control Panel*, *Controller*, and *JointControl*, which communicates directly with the Java3D objects that perform the command.

From the user’s point of view, the avatar’s movements are smoother and the animation looks more natural in the second implementation than in the first, according to our experiments.

B. Skills

The role of the second control layer is to combine different joint rotations according to a predefined schedule in order to create skills (also known as basic behaviors) [3]. The skills appear as a preprogrammed purposeful sequence of joint movements, and may require the use of analytic inverse kinematics and dynamic transforms, [24], [11], and/or Neural Networks [21]. So far we have implemented 15 skills 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:

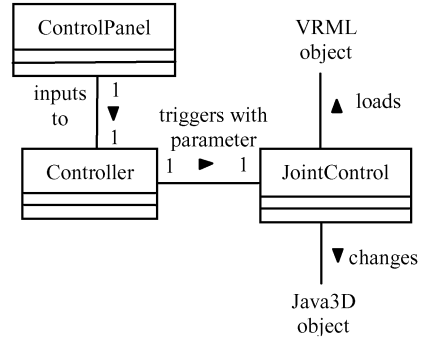


Fig. 3. Class diagram for joint control in Java/Java3D.

- **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.

There are two kinds of basic skills: some are describing independent behaviors of the avatar, and others the interaction of the avatar with objects from the virtual environment. Examples of independent behaviors performed by the avatar alone without interacting directly with the environment, are: walking, running, and waving a hand. The schedules for the joint movements corresponding to three such behaviors (“walk,” “run,” and “jump”) are taken from the VRML Nancy example given in [30]. These schedules are described by fixed parameters, such as the timing of different joint movements, the angles for raising the legs and rotating the arms, etc. Therefore, the skills in our implementation are always executed in the same way. We are studying how to realize more flexible skills, by taking two approaches. The first approach is to add noise at run time to the parameters describing a basic behavior, in order to introduce slight variations in the execution and make it look more natural (for example, not all the steps will look exactly the same). Another approach is to use *neural networks* (NNs) techniques for the development of more life-like personalized skills. These behavioral NNs are trained off-line by sensory-driven examples [2], [12]. This way the avatar can imitate, for instance, the walking or running style of different people.

Interactive behaviors are those where the avatar interacts with objects from the virtual environment. For example, “turn on/off the computer” is an interactive behavior. In this case, the avatar needs to locate some objects in the virtual environment and interact with them. Since the relative position of the avatar with respect to different objects changes dynamically, we have to obtain the control data at run time. For example, when receiving the command “open the door,” the avatar could be in different places, standing at different angles. In order for the avatar to touch the doorknob, the inverse kinematics algorithm [24] is used to compute at run time the parameters for the movements of the corresponding joints.

We separated the control of different behaviors into different classes, in order to make them easier to use from the higher control layer. New classes can be built if we want to add new behaviors without affecting the existing ones.

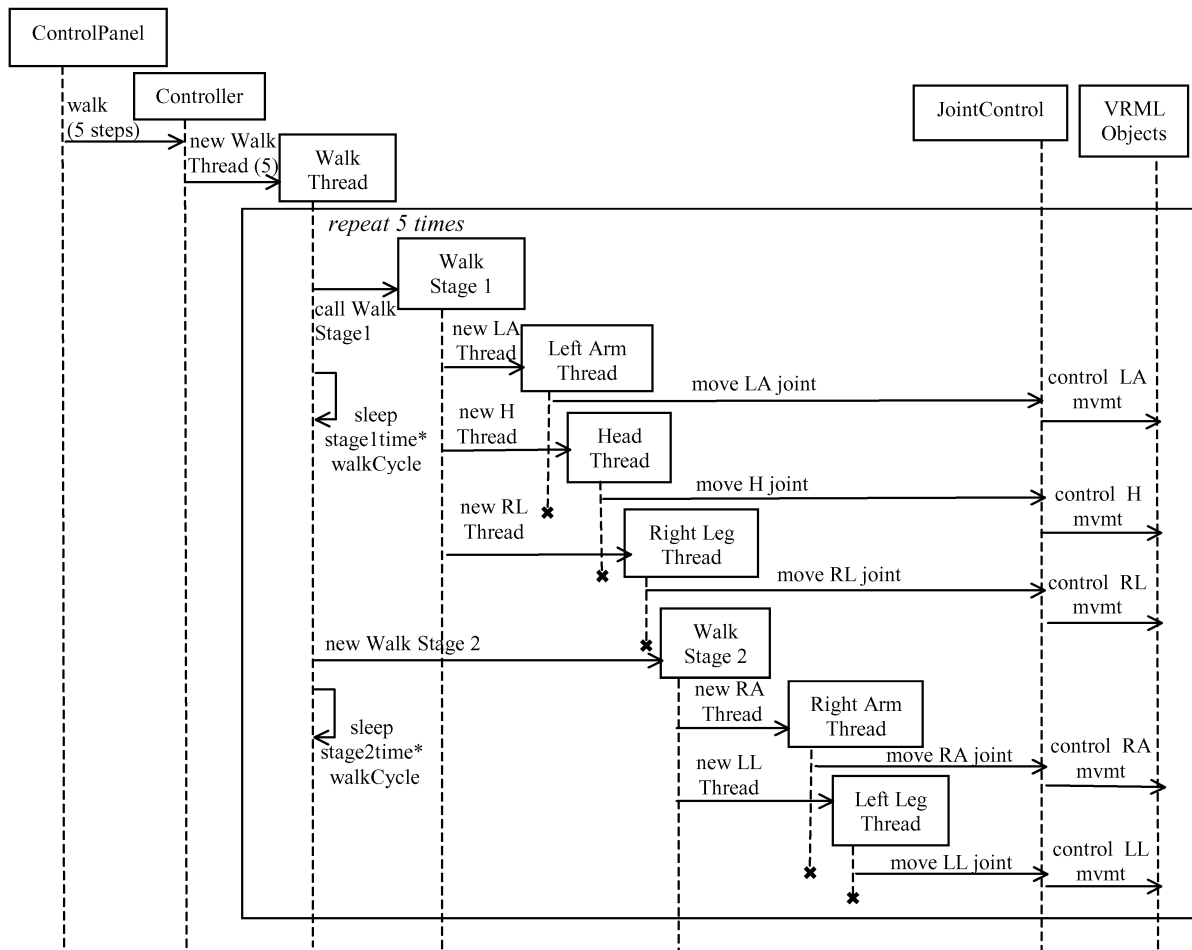


Fig. 4. UML sequence diagram illustrating the “walk” behavior in VRML/Java.

By combining the skills together in sequence or in parallel, the user can get a composed behavior. For example, by combining “go to the door” and “turn the doorknob,” one can obtain the behavior “open the door.”

1) *VRML/Java Skills Control*: Each skill is implemented by concurrent joints movements executed at different times according to a predefined schedule. In the first layer, different joint movements are controlled by different classes, which run in their own thread of control. This produces a smoother avatar animation by giving the appearance that the joints are moving simultaneously.

In order to separate the responsibilities related to a behavior, we used three classes working together for each independent behavior. For example, the walking behavior is realized by the following:

- 1) *WalkData* class, which stores the data (i.e., the schedule) for different joint movements;
- 2) *Walk* class, which defines different joint groups at different times;
- 3) *WalkThread* class, which controls the timing and coordinates the different groups.

In order to add noise or use NNs for the control of a behavior, the “data” class will be changed or connected to algorithm classes for computing the parameters that make the skill look more natural or imitate a certain style.

Fig. 4 shows a simplified UML sequence diagram to illustrate the “walk” behavior. There are 21 walk stages in the definition of “walk.” A stage represents a different combination of joints that are moving simultaneously at a specified time. The Walk thread controls all of the walk stages according to the schedule. The figure only shows two walk stages repeated for five steps. The user inputs the “walk” command through the *ControlPanel*, giving the following parameter values: five steps and a walk cycle of 2 s/step. The *Controller* will create the *Walk Thread* that initiates in turn 21 walk stages for each step. Each walk stage determines different joints to move concurrently, by creating a new thread for every joint. The *Walk Thread* will repeat all the walk stages for the desired number of steps. The repeating part is shown enclosed in a box in the sequence diagram.

Fig. 5 shows the sequence diagram for the parallel composition of two skills: “walk” and “wave left hand.” Assume that the user inputs the command “walk 5 steps” first. The *Controller* creates a *Walk Thread*, which creates in turn *Walk Stages* 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” while the avatar is still walking. The *Controller* will inform the *Walk Thread* to stop moving the left hand while continuing the walking behavior. After “wave left hand” is done, the *Walk Thread* can resume the control of the left arm.

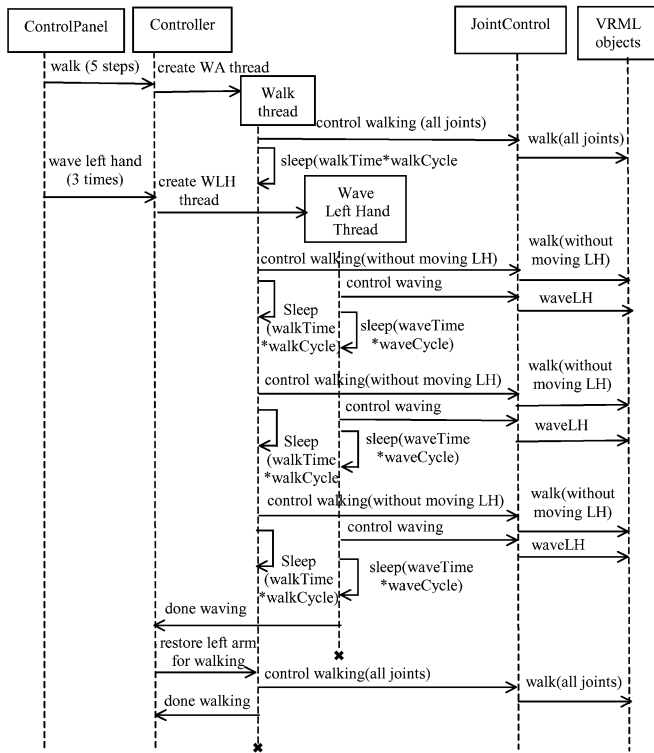


Fig. 5. UML sequence diagram for the parallel composition of “walk” and “wave hand” behaviors in VRML/Java.

For the *interaction skills*, the behavior of the avatar depends on both the avatar position and some objects 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 appropriate 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 ms and wait for the rotations and translations in the virtual environment to be done.

The main disadvantage of the VRML/Java implementation is the large communication overhead between VRML and Java. Another disadvantage is the large number of threads required for the numerous joints of the avatar. On one hand, the number of threads is limited by the underlying Java virtual machine; on the other hand, multithreading context switching comes with its own overhead in terms of execution time.

2) *Java3D Skill Control*: After loading the VRML objects into Java3D, we can use special Java3D functions for the avatar control. The main advantage of using Java and Java3D for animation control is platform independence—the code can run anywhere provided that the Java Virtual Machine is supported. Another advantage is that the execution time is much faster than

in the VRML/Java case, leaving available CPU time for more complex animations.

In Java3D, there are two special classes, which enhance the perceived quality of the animation, named *Alpha* and *Interpolator*. Interpolators are customized behavior objects that use an *Alpha* object to provide smooth animations of the visual objects [29]. The interpolator defines the end points of the animation. *Alpha* controls the animation with respect to timing, and shows how the *Interpolator* will move from one defined point to another at times given by the *Alpha* object. It was obvious just by looking at the animations obtained with the two approaches described here that the one using Java3D had a higher quality.

To separate the responsibilities related to a behavior in the Java3D implementation, we used two classes for each independent behavior. For example, the walking behavior is realized by the following:

- 1) *WalkData* class, which stores the data (i.e., the schedule) for different joint movements;
- 2) *Walk* class, which uses the Java3D *alpha* and *interpolator* objects to control the joint movements at different times.

Fig. 6 shows the sequence diagram for the parallel composition of “walk” and “wave left hand” in Java3D/Java. The user inputs the command “walk 5 steps” through the ControlPanel. The controller will set up the timer to control the duration of the behavior and triggers the *Walk* class to initiate the movement. Assume that the user enters the command “wave left hand” while the avatar is still walking. The Controller will stop the moving of the left arm according to the walking behavior, and triggers the behavior “wave left hand.” Assuming that the waving completes first, the control of the left arm will return under the control of the walking behavior.

By comparing Figs. 2–4, it is easy to see that the control is simpler in the Java3D/Java implementation than in the VRML/Java version.

C. Script Language

The third layer of control is the script language which is a restricted English-like language described by a context-free grammar. The language has its own interpreter to recognize and execute the commands. The interpreter translates the stories entered by users into a set of commands that represents skills, and triggers their execution by the avatars. The script language layer is dependent only on the interface provided by the skill control layer, but it is independent of the specific implementation of the two lower layers.

1) *Grammar*: Our intention was to design an English-like script language that would allow the user to express easily a short story or a set of commands to be enacted by the avatar. However, it is well known that an unconstrained natural language is difficult to parse, so we decided to settle for a script language defined by a context-free grammar. This type of grammar is largely used for the definition of today’s programming languages. Efficient parsers can be built for context-free grammars.

The script language defined here is constrained by:

- 1) the size of its vocabulary (terminal symbols) that depends on the set of skills implemented in the second layer of control;

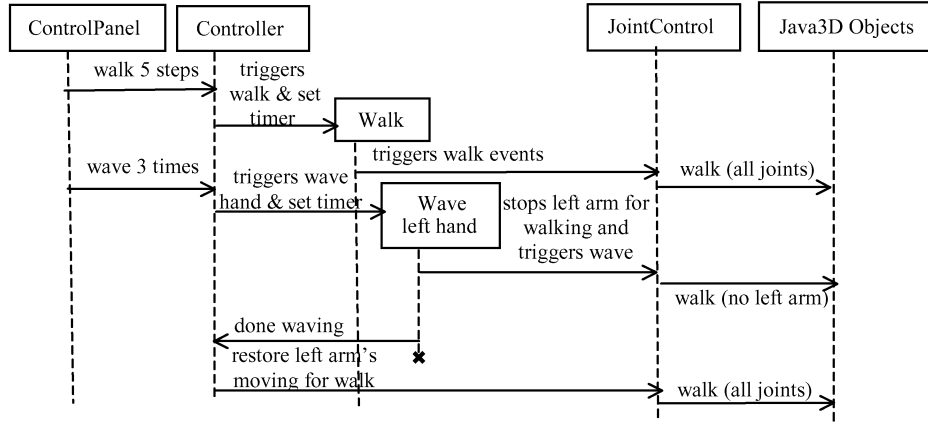


Fig. 6. UML sequence diagram for the parallel composition of “walk” and “wave left arm” in Java3D/Java.

TABLE I
GRAMMAR FOR THE SCRIPT LANGUAGE: THE SET OF RULES

Left-hand side		Right-hand side
<story>	::=	<sentence> <sentence><story>
<sentence>	::=	<simple sentence>.” <composed sentence>.”
<composed sentence>	::=	<simple sentence> <separator> <composed sentence> <simple sentence>
<separator>	::=	“,” <conjunction>
<conjunction>	::=	“and”
<simple sentence>	::=	[<subject>] <predicate>
<predicate>	::=	<verb> [<adverb>] * [<direct object>] [<indirect object>]
<direct object>	::=	[<article>] [<adjective>] <noun>
<noun>	::=	[<article>] <noun>
<indirect object>	::=	[<article>] [<adjective>] * <noun> <preposition> [<article>] [<adjective>] * <noun>
<subject>	::=	“Nancy” “she”
<verb>	::=	“stand” “wait” “run” “jump” “open” “wave” “kick” “walk” “go” “turn”
<adjective>	::=	“left” “right”
<noun>	::=	“door” “arm” “leg” “head” “yardDoor” “table” “window” “tree” “chair” “light” “computer”
<adverb>	::=	“fast” “faster” “slow” “slower”
<article>	::=	“a” “the”
<preposition>	::=	“to” “on”

2) the sentence syntax defined by the grammar.

According to [1], a context-free grammar G is a 4-tuple (V, Σ, R, S) , where:

- V is a finite set of nonterminal symbols;
- Σ is a finite set of terminal symbols;
- R is a finite set of rules that define each nonterminal as a string of symbols (either terminal or nonterminal);
- $S \in V$ is the start symbol.

For the script language grammar defined here, the set of rules R is given in Table I, whereas V, Σ , and S are defined as follows:

$$V = \{ \langle \text{story} \rangle, \langle \text{sentence} \rangle, \langle \text{composed sentence} \rangle, \langle \text{simple sentence} \rangle, \langle \text{subject} \rangle, \langle \text{predicate} \rangle, \langle \text{direct object} \rangle, \langle \text{indirect object} \rangle, \langle \text{separator} \rangle, \langle \text{noun} \rangle, \langle \text{article} \rangle, \langle \text{adjective} \rangle, \langle \text{verb} \rangle, \langle \text{adverb} \rangle, \langle \text{preposition} \rangle, \langle \text{conjunction} \rangle \}$$

$$\Sigma = \{ \langle \text{Nancy} \rangle, \langle \text{she} \rangle, \langle \text{stand} \rangle, \langle \text{wait} \rangle, \langle \text{run} \rangle, \langle \text{jump} \rangle, \langle \text{open} \rangle, \langle \text{wave} \rangle,$$

$$\langle \text{kick} \rangle, \langle \text{walk} \rangle, \langle \text{go} \rangle, \langle \text{turn} \rangle, \langle \text{left} \rangle, \langle \text{right} \rangle, \langle \text{door} \rangle, \langle \text{arm} \rangle, \langle \text{leg} \rangle, \langle \text{head} \rangle, \langle \text{to} \rangle, \langle \text{on} \rangle, \langle \text{yardDoor} \rangle, \langle \text{table} \rangle, \langle \text{window} \rangle, \langle \text{tree} \rangle, \langle \text{chair} \rangle, \langle \text{light} \rangle, \langle \text{computer} \rangle, \langle \text{fast} \rangle, \langle \text{faster} \rangle, \langle \text{slow} \rangle, \langle \text{slower} \rangle \}$$

$$S = \langle \text{story} \rangle.$$

How can a string of terminal symbols be generated from the rules? We begin with the start symbol and choose a rule to be applied (the rule must have the start symbol as its left-hand side). By “applying the rule,” we replace the nonterminal represented by the left-hand side of the rule with its right-hand side. If the resulting string contains nonterminal symbols, we choose rules to apply for each of one of them. We continue making replacements according to the rules until, eventually, the current string contains only terminals [20].

We have adopted the following convention for expressing the composition of skills (each one represented by a simple sentence):

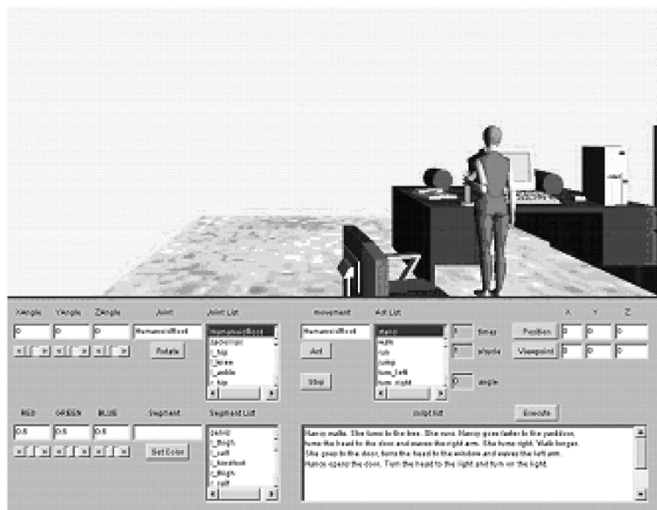


Fig. 9. Nancy turns on the light.

waves the right arm. She turns right. Walks longer. She goes to the door, turns the head to the window and waves the left arm. Nancy opens the door. Turns the head to the light and turns on the light.”

After pressing the “execute” command button, the whole text will be parsed, then the corresponding commands will be executed. In Fig. 8, Nancy is executing the first sentence “Nancy walks;” and, in Fig. 9, the command “turns on the light.”

III. CONCLUSION AND FUTURE WORK

This paper proposes a hierarchical three-layer control system for anthropomorphic avatar animation in a 3-D virtual environment, and describes its design and implementation. The lower layer controls the movements of avatar’s joints; the middle layer controls the skills and their composition, and the higher layer implements and executes a script language. We used two different approaches to implement the first and second layer of control without affecting the highest level, and compared their advantages and disadvantages.

This hierarchical approach, inspired from the field of robotics, has the following advantages:

- 1) user friendliness, as the complexity of the lower layers is hidden from the user. With a concise script sentence, the user can trigger complex actions involving multiple skills and a very large number of joint movements;
- 2) decoupling between different control layers, which means that the internal implementation of a layer can be changed without affecting the others, as long as we keep the same interfaces;
- 3) flexibility and extensibility at every layer.

The hierarchical animation control proposed in the paper targets interactive virtual reality applications, such as on-line training, e-commerce, computer games, etc. In such applications, it is important to respond in real-time to concise user commands that may trigger complex yet repetitive actions. However, a textual script language may not be convenient for movie animation where, traditionally, the storyline is expressed

graphically as a storyboard, which is a sequence of successive snapshots that describe better than text the artistic vision of the creators [32].

In future developments, more functions and behaviors could be added to the avatar. Artificial intelligence techniques can be used to enable the avatar to make decisions in impromptu situations without the interference of human beings. Another big extension is to introduce more avatars into the virtual environment and to develop collaborative interactions among avatars. Speech and text can be also added to enrich the whole activities in the virtual environment. Different behaviors can be integrated into models or plug-ins that can be used in other software applications, such as computer games, educational applications, weather forecast or news report. All these extensions will have an impact on the script language for avatar animation. The language will become more complex, and maybe closer to English than it is now.

REFERENCES

- [1] V. Aho, R. Sethi, and J. D. Ullman, *Compilers, Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley, 1986.
- [2] 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. C, Appl. Rev.*, vol. 31, no. 5, pp. 431–442, Sep. 2001.
- [3] C. Archibald and E. M. Petriu, “Skills-oriented robot programming,” in *Proc. Int. Conf. Intelligent Autonomous Systems IAS-3*, Pittsburgh, PA, 1993, pp. 104–113.
- [4] N. Badler, C. Phillips, and B. Webber, “Simulating humans,” in *Computer Graphics Animation and Control*. New York: Oxford Univ. Press, 1993.
- [5] M. C. Buchanan and P. T. Zellweger, “Automatic temporal layout mechanisms,” in *Proc. ACM Multimedia*, 1993, pp. 341–350.
- [6] S. Chenney, J. Ichnowski, and D. A. Forsyth, “Efficient dynamics modeling for VRML and java,” in *Proc. VRML*, Monterey, CA, Feb. 1998, pp. 15–25.
- [7] K. S. Fu, R. C. Gonzales, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*. New York: McGraw-Hill, 1987.
- [8] J. Granieri, J. Crabbtree, and N. Badler, “Off-line production and real-time playback of human figure motion for 3D virtual environments,” in *Proc. Virtual Reality Annual International Symp.*, Research Triangle Park, NC, Mar. 1995, pp. 127–135.
- [9] R. Hamakawa and J. Rekimoto, “Object composition and playback models for handling multimedia data,” in *Proc. ACM Multimedia*, 1993, pp. 273–281.
- [10] M. Kallmann and D. Thalmann, “A behavioral interface to simulate agent-object interactions in real-time,” *Proc. Computer Animation*, pp. 138–146, 1999.
- [11] H. Ko and N. I. Badler, “Animating human locomotion with inverse dynamics,” *IEEE Comput. Graph. Appl.*, vol. 16, no. 2, pp. 50–59, Mar. 1996.
- [12] C. Luciano, P. Banerjee, and S. Mehrota, “3D animation of telecollaborative anthropomorphic avatars,” in *Comm. ACM*, vol. 44, Dec. 2001, pp. 64–67.
- [13] D. R. Nadeau. (2004, Sep.). Visualization in VRML. [Online]. Available: <http://www.sdsc.edu/~nadeau/Talks/WorldMovers/>
- [14] K. Perlin, “An image synthesizer. Computer graphics,” in *Proc. SIGGRAPH*, vol. 19, 1985, pp. 287–293.
- [15] K. Perlin, “Real time responsive animation with personality,” *IEEE Trans. Vis. Comput. Graph.*, vol. 1, no. 1, pp. 5–15, Mar. 1995.
- [16] K. Perlin and A. Goldberg, “Improv: A system for scripting interactive actors in virtual worlds,” in *Proc. ACM SIGGRAPH*, New York, 1996, pp. 205–216.
- [17] D. C. Petriu, T. R. Jones, and E. M. Petriu, “Communicating state-machine model of resource management in a multi-robot assembly cell,” in *ISCA J. Comput. Appl.*, vol. 1, 1994, pp. 43–48.
- [18] D. C. Petriu, X. L. Yang, and T. E. Whalen, “Behavior-based script language for anthropomorphic avatar animation in virtual environments,” presented at the IEEE Intl. Symp. Virtual and Intelligent Measurement Systems, Anchorage, AK, 2002.

- [19] C. S. Pinhanez, K. Mase, and A. F. Bobick, "Interval scripts: A design paradigm for story-based interactive systems," in *Proc. CHI, 1997*, pp. 287–294.
- [20] S. Ressler, A. Godil, Q. Wang, and G. Seidman, "A VRML integration methodology for manufacturing applications," presented at the 4th Symp. Virtual Reality Modeling Language, Paderborn, Germany, 1999.
- [21] C. Rigotti, P. Cerveri, G. Andreoni, A. Pedotti, and G. Ferrigno, "Modeling and driving a reduced human mannequin through motion captured data: A neural network approach," *IEEE Trans. Syst., Man, Cybern., A: Syst., Humans*, vol. 31, no. 3, pp. 187–193, May 2001.
- [22] N. M. Thalmann and D. Thalmann, *Synthetic Actors in Computer Generated 3D Films*. Berlin, Germany: Springer-Verlag, 1990.
- [23] X. L. Yang, D. C. Petriu, T. E. Whalen, and E. Petriu, "Script language for avatar animation in virtual environments," in *Proc. IEEE Intl. Symp. Virtual Environments, Human-Computer Interfaces and Measurement Systems*, Lugano, Switzerland, Jul. 2003, pp. 101–106.
- [24] J. Zhao and N. Badler, "Inverse kinematics positioning using nonlinear programming for highly articulated figures," in *ACM Trans. Graph.*, vol. 13, pp. 313–336.
- [25] —, (2004, Sep.). The Humanoid Animation Working Group. [Online]. Available: <http://www.h-anim.org>
- [26] —, (2004, Sep.). Virtual Humanoids in VRML. [Online]. Available: <http://ligwww.epfl.ch/~babski/vrml.html>
- [27] —, (2003, Aug.). CyberVRML97 Virtual Reality Modeling Language Development Library. [Online]. Available: <http://www.cybergarage.org/vrml/cv97/cv97java/>
- [28] —, (2004, Sep.). Java3D API. [Online]. Available: <http://java.sun.com/products/java-media/3D/>
- [29] —, (2004, Sep.). Java3D API Tutorial. [Online]. Available: <http://developer.java.sun.com/developer/onlineTraining/java3d/>
- [30] —, (2004, Sep.). H-Anim Examples. [Online]. Available: <http://www.ballreich.net/vrml/h-anim/h-anim-examples.html>
- [31] —, (2004, Sep.). VRML 97 Reference Page. [Online]. Available: <http://www.web3d.org/x3d/specifications/vrml/>
- [32] —, (2004, Sep.). Pixar Homepage. [Online]. Available: <https://renderman.pixar.com/>



Xiaoli Yang (M'03) is currently an Assistant Professor with the Department of Electrical Engineering, Lakehead University, Thunder Bay, ON, Canada.

She was with the DISCOVER Laboratory, School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, as a Postdoctoral Fellow before she joined Lakehead University.



Dorina C. Petriu (M'90–SM'03) received the Dipl.Eng. degree in computer engineering from the Polytechnic University of Timisoara, Timisoara, Romania, and the Ph.D. degree in electrical engineering from Carleton University, Ottawa, ON, Canada.

She is a Professor with the Department of Systems and Computer Engineering, Carleton University. She was a contributor to the UML Profile for Schedulability, Performance, and Time, recently standardized by OMG. Her current research projects include automatic derivation of software performance models from UML design specifications, scalability analysis of virtual private networks, and software development and modeling for virtual environments and robotics. Her research interests are in the areas of performance modeling and software engineering, with emphasis on integrating performance engineering into the software development process.

Dr. D. C. Petriu is a Member of the ACM.



Thom E. Whalen received the Ph.D. degree in experimental psychology from Dalhousie University, Halifax, NS, Canada, in 1979.

He has been conducting research in human-computer interactions at the Communications Research Centre (CRC), Ottawa, ON, Canada, a research institute of the Government of Canada, since 1979. His current research interests include access to information through natural language queries, retrieval of images using visual features, Web-based training, and the control of avatars in shared virtual environments.

He is also an Adjunct Professor with the Psychology Department, Carleton University, Ottawa, and with the Department of Management Science and Finance, St. Mary's University, Halifax.



Emil M. Petriu (M'86–SM'88–F'01) received the Dipl.Eng. and Dr.Eng. degrees from the Polytechnic Institute of Timisoara, Timisoara, Romania, in 1969 and 1978, respectively.

He is a Professor at the School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada, where he has been since 1985. His research interests include test and measurement systems, interactive virtual environments, intelligent sensors, robot sensing and perception, neural networks, and fuzzy control. During his career, he

has published more than 200 technical papers, authored two books, edited two other books, and he currently holds two patents.

Dr. Petriu is a Fellow of the Canadian Academy of Engineering and Fellow of the Engineering Institute of Canada. He is currently serving as a Member of the AdCom, and Chair of TC-15 Virtual Systems and Co-Chair of TC-28 Instrumentation and Measurement for Robotics and Automation of the IEEE Instrumentation and Measurement Society. He is an Associate Editor of the *IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT* and member of the editorial board of the *IEEE Instrumentation and Measurement Magazine*. He is a corecipient of the IEEE's Donald G. Fink Prize Paper Award for 2003 and the 2003 recipient of the IEEE Instrumentation and Measurement Society Award.