User-Definable Rule Description Framework for Autonomous Actor Agents

Narichika Hamaguichi[1], Hiroyuki Kaneko[1], Mamoru Doke[2], and Seiki Inoue[1]

[1] Science & Technical Research Laboratories, Japan Broadcasting Corporation (NHK)
1-10-11, Kinuta, Setagaya-ku, Tokyo, 157-8510, Japan
{hamaguchi.n-go, kaneko.h-dk, inoue.s-li}@nhk.or.jp
[2] NHK Engineering Services, Inc.
1-10-11, Kinuta, Setagaya-ku, Tokyo, 157-8540, Japan
douke@nes.or.jp

Abstract. In the area of text-to-video research, our work focuses on creating video content from textual descriptions, or more specifically, the creation of TV program like content from script like descriptions. This paper discusses a description framework that can be used to specify rough action instructions in the form of a script that can be used to produce detailed instructions controlling the behavior and actions of autonomous video actor agents. The paper also describes a prototype text-to-video system and presents examples of instructions for controlling an autonomous actor agent with our designed descriptive scheme.

Keywords: Autonomous Actor Agent, Digital Storytelling, Text-to-Video, TVML, Object-Oriented Language.

1 Introduction

Research into digital storytelling has attracted considerable interest in recent years, and one approach of producing computer graphics (CG) video content from textual descriptions has inspired a number of studies around the world. (We refer to this approach as "text-to-video" [1])

In text-to-video production, figuring out how to make the actor agents (CG characters) in the video act and behave naturally is critically important. In big production animated films, the mannerisms and behavior of actor agents can be manually edited on a frame-by-frame basis which is extremely costly in terms of man-hours, time, and budgets. But for smaller scale or personal video productions such lavish and costly production techniques are impractical, thus creating a demand for an autonomous method of controlling actor agents. A number of studies have addressed this issue of autonomous actor agents [2].

Researchers have also investigated language-based descriptive methods of producing video content and controlling the actions and behavior of actor agents, including a specially designed scripting approach [3] and a method of controlling the behavior of actor agents using natural language instructions [4]. Most of these

J.A. Jacko (Ed.): Human-Computer Interaction, Part III, HCII 2009, LNCS 5612, pp. 257–266, 2009.

studies of autonomous actor agent action and ways of describing such actions are only able to describe such actions using a limited vocabulary based on rules set up in advance under limited conditions. The problem is that if the user wants to add a new autonomous action rule or wants to modify an existing rule, there are very few schemes giving users access to the rules, and the expandability of those that do provide access is quite limited.

This led us to design a system that is functionally separated into two parts—a video content production part and an object-oriented description part— and instead of using a special proprietary language, the object-oriented description part uses a dynamic programming language that can be run as is from the source code without compiling the code beforehand. Users are thus able to add, modify, and reuse actor agent action rules by directly accessing the source code. Moreover, because this approach is based on an existing programming language, it is infinitely expandable according to the whims and desires of the user. In the next section, we will lay out the principle issues that will be addressed in this paper.

2 Requirements

In this section we consider the requirements needed to enable users to represent rules that control the behavior and actions of actor agents.

2.1 Openly Modifiable Object-Oriented Language

The object-oriented approach permits functions to be encapsulated which are highly beneficial in terms of reusability, and today many advanced programming languages have adopted the object-oriented approach. There are essentially two types of object-oriented languages: languages that are executed after the source code is compiled (compiler languages), and languages that are not pre-compiled but are interpreted at run-time (dynamic programming languages or interpretive languages).

In compiler languages, the source code is separated from the executable file, and because the executable file is a *black box*, the user is unable to modify functions or copy and reuse portions of functions even if he is able to use functions created by others.

To achieve our objectives, we need a language that will enable users to add, modify, and reuse rules for controlling the behavior and actions of actor agents. In short, we need an object-oriented dynamic programming language that permits the user to access the source code.

2.2 Versatile Layered Structure for Different Types of Users

With the goal of using descriptive language to control the actions of actor agents, our first concern is to achieve the desired behavior or action using the simplest possible expressions without inputting detailed instructions. The problem with the object-oriented languages described earlier is that expressions inevitably become much more complicated than can be handled by a simple script language the more encapsulation and reuse is involved.

The level of language used also varies depending on the type of user and the intended use. For example, if one wants to produce a simple video clip with a minimum of time and effort, the level of language abstraction and types of data manipulated are very different than if one wants to produce content in which very detailed actions and timing of the actor agents are critically important.

In order to address these issues, we adopted a three-layer structure that can be tailored to different kinds of users and different intended uses. The lower layer (detailed description instructions) is for pros enabling descriptions supporting detailed video content to produce professional-grade video content. The upper layer (simple description instructions) is for beginners or amateurs. It hides the complexity of detailed descriptions, and allows amateur users to produce video content using relatively simple descriptions. The rules that control the actions and behaviors of actor agents are described in the middle layer sandwiched between the upper and lower layers. Essentially, this layer converts the relatively simple instructions received from the upper layer to the more detailed expressions required by the lower layer.

3 Language Design and Prototype System

Based on the requirements outlined in the previous section, we designed a prototype system consisting of two parts as illustrated in Fig.1: an Object-Oriented Description Module and a Presentation Module. The Object-Oriented Description Module is a three-layer structure as described above. It consists of a Simple Scripting Layer, an Automatic Production Layer, and a Detailed Description Layer, and runs using the dynamic programming language Python.

A series of rough sequential instructions similar to the script for a TV program are described in the upper Simple Scripting Layer. The instructions are in a format that even someone with little or no experience with programming languages can understand and edit.

The rules controlling the behavior and actions of the actor agents are described in the middle Automatic Production Layer. The layer receives the rough instructions described in the upper Simple Scripting Layer, acquires the situation using the functions of the lower Detailed Description Layer, then automatically determines the specific actions and behavior of the agents based on rules in the middle layer, which are sent to the lower Detailed Description Layer for execution.

The lower Detailed Description Layer provides a simple wrapped interface with a TVML Player [5] in the Presentation Module, and through this intermediary wrapper, the Detailed Description Layer obtains the video states and delivers the instructions. Instructions are created using a descriptive language called TVML (TV program Making Language), and the states are acquired using an external application program interface called TvIF. TVML is a self-contained language featuring all the capabilities needed to create TV program-like video content, including detailed control over the speech and movement of actor agents, cameras, display of subtitles, and so on. Essentially, complex descriptive instructions are substituted for detailed control.

The TVML Player uses software to interpret a TVML Script, then generates video content using 3D computer graphics, synthesized speech, and other production

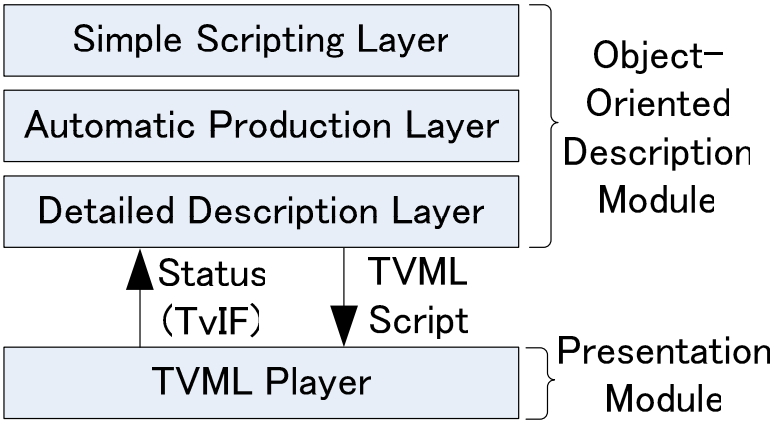

Fig. 1. Layered structure of the object-oriented video content production framework

techniques. The TVML Script only provides the TVML Player with a one-way stream of instructions, and the states are returned by way of the TvIF interface. Moreover, the TVML Player was developed using C++. This relieves the user of dealing with the technically challenging aspects of production such as handling 3D computer graphics which is done by the TVML Player, but the internal operation of the TVML Player itself is unalterable.

By adopting the layered approach outlined above, users can employ whichever layer is best suited to their skills and objectives. And because users have direct access to the source code of each layer, they can add, modify, reuse and inherit classes of rules controlling the actions of actor agents.

4 Application Examples for Each Layer

In this section we will provide description and application examples for each layer.

4.1 Simple Scripting Layer

The Simple Scripting Layer is the upper layer. It is based on a very intuitive format: a sequential string of instructions without any inherent control structure, much like the script of a TV program. It is thus transparent and easily manipulated by anyone, even people with little or no experience with programming languages.

Application Example. Fig.2 shows a typical example of an application in which a description in the Simple Scripting Layer is edited, and the internal descriptions written in Python are the followings. These descriptions in the Simple Scripting Layer consist of a simple line-by-line sequence of unstructured instructions. So using a tool such as illustrated in Fig.2, the user can easily edit the script in much the same way as

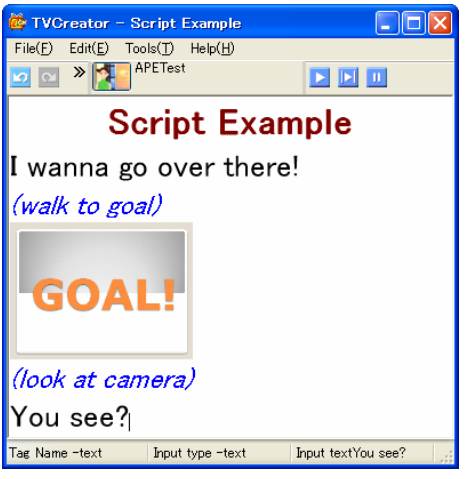

Fig. 2. Application example for editing the descriptions in the Simple Scripting Layer

using a word processor. Any user capable of using a word processor is thus capable of producing video content!

Description Examples

```
import apetest   #Import of Automatic Production module
ape=apetest.APETest()                #Constructor
ape.title("Script Example")
ape.text("I wanna go over there!")  #Speech
ape.action_walk_to_goal()                    #Action
ape.subimage("goal.jpg")                     #Show image
ape.action_look_at_camera()
ape.text("You see?")
ape.end()
```

4.2 Automatic Production Layer

The specific rules that control the actions and the behavior of actor agents on the basis of instructions received from the Simple Scripting Layer are described in the Automatic Production Layer.

Description Examples. Here is an example of descriptions in the Automatic Production Layer. Action rules are represented in classes, and inherit a new class called *APEBase*. Basic action rules are defined in *APEBase*, so in order to create a

new action rule, a user only needs to create or describe a rule that is different from that in the *APEBase*. The module for producing new action rules in this way is called the Automatic Production Engine (APE) [6].

```
goal.x=4
goal.z=0
class APETest(APEBase):
  …
  def setup(self): #Initialization
    self.A=tvml.Character(filename="bob.bm", x=-4)
    self.obst1=tvml.Prop(filename="tsubo.obj", x=1,…
    …
  def text(self, value):     #Speech
    self.A.talk(text=value)
    …
  def subimage(self, value):#How to show image
     self.img=tvml.Prop()
     self.img.openimageplate(filename=value,
                      platesizeh=3.6, platesizev=2.7)
    self.img.position(y=2, pitch=270)
    …
  def action_walk_to_goal(self):  #How to walk to goal
    props=getPropNameList()  #Get all prop names
    for prop in props:
      loc=findPath(obstacle=prop)  #Find a path per prop
      self.A.walk(x=loc.x, z=loc.z)      #Simple walk
    self.A.walk(x=goal.x, goal.z)
```

As one can see, several method subroutines are defined: the *setup* method deals with initialization, the *text* method relates to the speech of actor agents, and the *subimage* method relates to how images are presented. The *action_walk_to_goal* method is a new subroutine created by the user that instructs the actor agent to avoid obstacles as it proceeds to a goal (the actual description has been simplified in this example). Previously, the only walk-related method defined in the lower Detailed Description Layer instructed the action agent to proceed in a straight line from its current position to the goal. The new *action_walk_to_goal* subroutine calculates a path from the position and size of obstacles (3D bounding box), thus

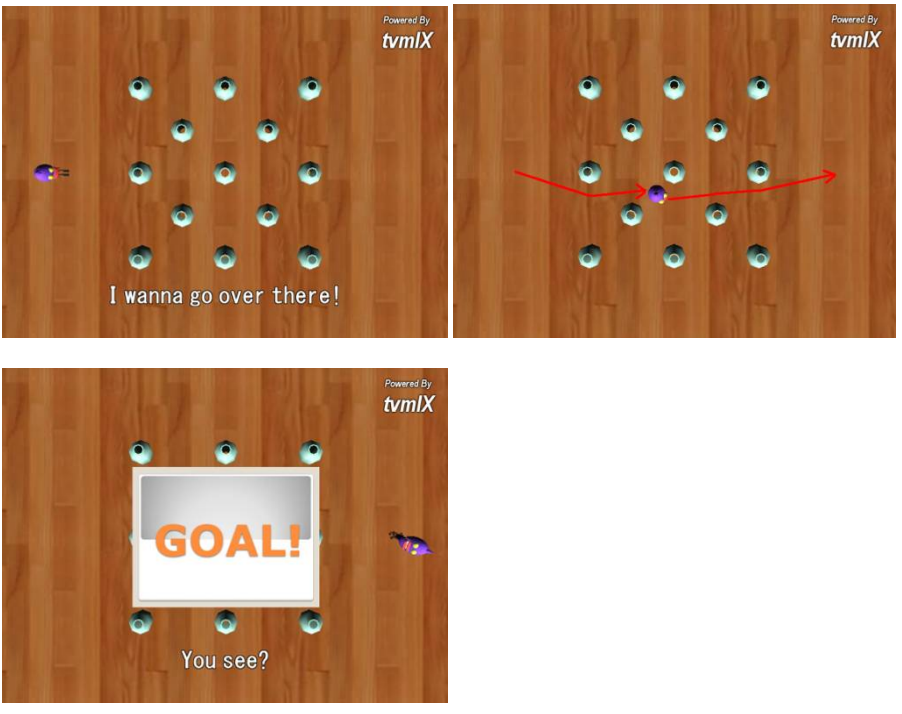

Fig. 3. Operation of the *action_walk_to_goal* method

enabling the user to define an action rule permitting the agent to proceed to a goal without bumping into things.

Output Example. Fig.3 shows an example of how the *action_walk_to_goal* method is run based on descriptions in the Simple Scripting Layer.

Users are thus able to add and modify the rules controlling the actions and behavior of agents in the Automatic Production Layer. Significantly, newly added action rules can be used and manipulated using an easy user-friendly format from the upper Simple Scripting Layer just like any other instruction. Here we have discussed a user-defined action rule enabling agents to avoid obstacles, but all sorts of powerful rules can be created in the same way, such as:

–
– - Actor agent actions playing to a particular camera:
– - Actions of an agent can be controlled to play to a particular camera by acquiring the names, positions, and angles of the cameras.
– - Actor agent actions can be synched to a movie file:
– Actions of an agent can be synchronized to the playback timing of a movie by acquiring the playback timing of the movie file.
– - Actor agent behavior can be synched to speech:

- The expressions and gestures of an agent can be synched to the character strings of the synthesized speech lines spoken by the agent.
- - Evaluation of the output screen layout:
- The layout or composition of the output video screen can be evaluated and the agent's actions adjusted to the layout by acquiring an on-screen 2D bounding box.

These various types of automatic production rules are actually executed by the functional capabilities of the Detailed Description Layer. Let us next take a closer look at the Detailed Description Layer, which must be endowed with powerful capabilities in order to execute these rules.

4.3 Detailed Description Layer

The TvIF/TVML are wrapped by the method subroutines that are incorporated in the Detailed Description Layer. This layer can obtain a comprehensive range of states in the TVML Player including the prop bounding box, camera information, movie playback timing, and a host of other states. Table 1 shows some of the state acquisition methods that are incorporated in the Detailed Description Layer.

Note too that all of the states incorporated in the TVML Player—orientation angle and coordinates of the actor agents, speed, timing, and so on—can be directly controlled by the TVML Script. This allows more experienced users who want direct control over the production and editing of their video content to directly work at this layer.

Description Examples. Here are some typical examples of Detailed Description Layer descriptions. These descriptions enable detailed control over the movement of actor agent joints, camera movements, and a host of other variables.

```
buddy.gaze(pitch=-30, wait=NO)

buddy.turn(d=-120, speed=0.5)

buddy.definepose(pose=GetWhisky, joint= LeftUpperArm,
                            rotx=-105.00, roty=25.00)

buddy.definepose(pose=Getwhiskey, joint=Chest,
                      rotx=5.00, roty=-15.00, rotz=0.00)

buddy.pose(pose=Getwhiskey, speed=0.25, wait=NO)

tvml.wait(time=0.7)

cam1.movement(x=-0.49, y=1.57, z=1.75, pan=400,
                    tilt=-5.00, roll=1.00, vangle=45.00,
                     transition=immediate, style=servo)

whisky.attach(characternamе=buddy, joint=RightHand,
                                                switch=ON)
```

Output Example. Fig.4 illustrates how the agent actually moves based on the Detailed Description Layer descriptions listed above.

Table 1. Typical state acquisition methods incorporated in the Detailed Description Layer

getCharacterLocation	Current position of an actor agent
getCharacterTalkingText	Character string currently spoken by an actor agent
getCameraCurrent	Name of camera currently selected
getCameraLocation	Current location and angle of camera
getPropNameList	List of prop names
getPropBoundingSolid	3D bounding box of a prop
getPropBoundingBox	2D bounding box of an on-screen prop
getMovieCurrentTime	Playback position of a movie file

Fig. 4. Movement based on Detailed Description Layer descriptions

5 Conclusions

In this work we designed an object-oriented scheme for building a descriptive language framework enabling users to add, modify, and replay rules controlling the behavior and actions of autonomous actor agents. By dividing the object-oriented description into three layers—the Simple Scripting Layer, the Automatic Production Layer, and the Detailed Description Layer—we have implemented a structure that can be tailored to different kinds of users and different intended uses. This scheme allows users themselves to describe rules for controlling the behavior and actions of autonomous actor agents by editing the Automatic Production Layer.

Leveraging this Automatic Production Layer based scheme, we plan to design a wide range of autonomous actor agents and develop applications that use the agents.

References

1. Bindiganavale, R., Schuler, W., Allbeck, J., Badler, N., Joshi, A., Palmer, M.: Dynamically Altering Agent Behaviors Using Natural Language Instructions. In: The 4th International Conference on Autonomous Agents, Proceedings, Barcelona, Spain, pp. 293–300 (2000)
2. Funge, J., Tu, X., Terzopoulos, D.: Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters. In: The 26th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1999), Proceedings, Los Angeles, USA, pp. 29–38 (1999)
3. Hamaguchi, N., Doke, M., Hayashi, M., Yagi, N.: Text-based Video Blogging. In: The 15th International World Wide Web Conference (WWW 2006), Proceedings, Edinburgh, Scotland (2006)
4. Hayashi, M., Doke, M., Hamaguchi, N.: Automatic TV Program Production with APEs. In: The 2nd Conference on Creating, Connecting and Collaborating through Computing (C5 2004), Kyoto, Japan, pp. 20–25 (2004)
5. http://www.nhk.or.jp/strl/tvml/
6. Perlin, K., Goldberg, A.: IMPROV: A System for Scripting Interactive Actors in Virtual Worlds. In: The 26th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1996), Proceedings, New Orleans, USA, pp. 205–216 (1996)