TVML (TV program Making Language)
- Automatic TV Program Generation from Text-based Script -

[1)]Masaki HAYASHI, [2)]Hirotada UEDA, [3)]Tsuneya KURIHARA, [4)]Michiaki YASUMURA

[1)] NHK (Japan Brodcasting Corporation) Science and Technical Research Laboratories
1-10-11, Kinuta, Setagaya-ku, Tokyo, 157, Japan. Email: hayashi@strl.nhk.or.jp

[2)]Hitachi-Denshi, Ltd., [3)] Hitachi, Ltd., Central Research Laboratory, [4)]Keio University

Abstract

This paper describes TVML (TV program Making Language) for automatically generating television programs from text-based script. This language describes the contents of a television program using expressions with a high level of abstraction like "title #1" and "zoom-in." The software used to read a script written in TVML and to automatically generate the program video and audio is called the TVML Player. The paper begins by describing TVML language specifications and the TVML Player. It then describes the "external control mode" of the TVML Player that can be used for applying TVML to interactive applications. Finally, it describes the TVML Editor, a user interface that we developed which enables users having no specialized knowledge of computer languages to make TVML scripts. In addition to its role as a television-program production tool, TVML is expected to have a wide range of applications in the network and multimedia fields.

1. Introduction

Broadcasting has a long history as a medium for conveying information. The techniques used in this field, and especially in the production of television programs, have progressed over the years to become highly refined means of presenting information. At present, however, only broadcasting stations are able to use these techniques satisfactorily. In other words, television program production is still the domain of specialists.

In contrast, tools are now being provided that allow individuals to prepare various types of media like documents, pictures, and magazines, as well as Internet-based hypertext, entirely in the privacy of one's own room. One reason for this is that the once limited use of the Internet by scientific and technical professionals has expanded to widespread use by individuals.

Individuals cannot easily produce television programs with conventional techniques because program production requires various video and audio equipment as well as studios, announcers, and so forth. However, if we limit the target programs to the one for information presentation, it becomes quite possible to create studios, announcers, and the like with current technology by computer through the use of computer graphics (CG), voice synthesizers and multimedia computing. This paper describes a mechanism for generating television programs in real time on only a desktop computer through the use of these technologies.

We have developed a new language that can represent an entire television program by text, called TVML [1]-[4](TV program Making Language). TVML is not limited to describing the parts played by actors. Instead, it can be used to describe all program-production work, including pre- and post-production work such as that involving sets, props, cameras, VCR in playback, superimposing, titles, background music, and narration. This paper describes TVML as well as a TVML Player, which converts the television-program script written in TVML to program video and audio in real time. It also describes a TVML Editor, which is a user interface that we have developed for the creation of TVML scripts by users having no specialized knowledge of computer languages.

2. Mechanism of Real-time Generation of Television Programs

The generation of television programs by computer requires some kind of intermediate expressions that can be given to the computer to indicate tasks like directing the performance of an actor and configuring a program. TVML is a text-based language that uses such expressions. The expressions can be easily understood when perused by people and instruct the computer what to do. TVML can describe an entire television program through highly abstract text-based expressions like "title #1" and "zoom-in". The software that we developed for interpreting scripts written in TVML and automatically generating program video and audio in real time is called the TVML Player. This is illustrated in Fig. 1.

The television programs targeted here are of the types of programs that present information such as news, weather forecasts, event guides, and documentaries.

Program production	Method used in TVML Player
- Studio shot	
Studio set	Real-time CG set
Actor	Real-time CG character with speech by voice synthesizer
Lighting	Lighting setup in real-time CG
Camerawork	Camera setup in real-time CG
- Motion picture	
VCR	Movie file playing (QuickTime)
- Title	
Text information	Text layout section of HTML
Static image	Image data file (TIFF)
- Superimposing	
Text information	Text layout section of HTML
- Sound	
Music	Audio file playing (AIFF)
Narration	Synthesized voice
- Video effect	Cut change only
- Sound effect	Audio mixer

Table 1. Elements in actual program production and methods used in TVML Player

For the most parts, television programs like dramas and live variety shows that are essentially one-time creative works are excluded. The production elements of such information-oriented television programs are relatively simple. These elements are shown in Table 1. together with the techniques used in the TVML Player to generate each element by computer. For example, a studio shot is generated entirely by CG in which computer-generated actors in a computer-gen-

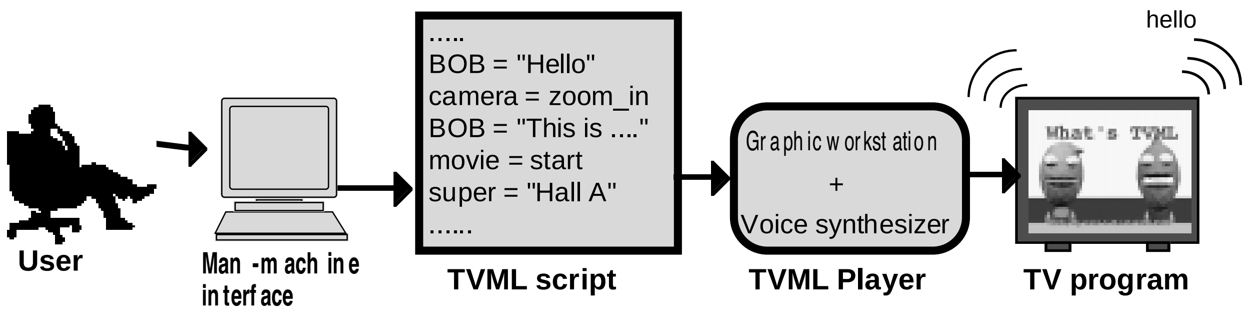

```
.....
BOB = "Hello"
camera = zoom_in
BOB = "This is ...."
movie = start
super = "Hall A"
......
```

Fig. 1. General configuration of system for generating TV programs by TVML

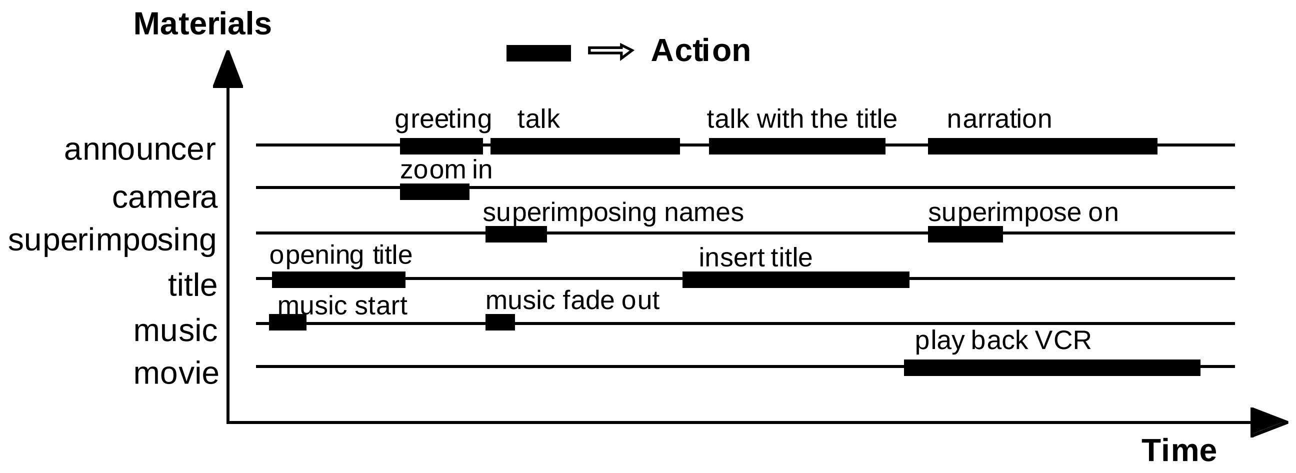


Fig. 2. Example of time, materials and action in TV program

erated set are given speech through a voice synthesizer, and the entire scene is "shot" by a camera within the CG world. A motion picture is achieved by playing back a movie file, titles and captions are generated by using the layout-description part of HTML to display text information, audio is produced by playing back audio files, and narration is generated by using a voice synthesizer. The generation of these elements requires two types of data: script data written in TVML and various forms of reference data.

The next section discusses how a television program should be described on a text basis to establish TVML language specifications.

3. TVML Language Specifications

3.1 The need for TVML

As explained above, we have designed TVML as a text-based language that people can read and understand easily. In this section, we examine the reasons for developing the TVML intermediate language.

The contents of a television program can be divided into the elements given in Table 1, as described in Section 2. These elements are performed at various points in time, sometimes simultaneously. The pattern for the initial segment of a typical news program, for example, is shown in Fig. 2. The model in the figure consists of three main components: time, materials, and action. Text-based scripts may not be conducive to describing actions of multiple elements that are performed simultaneously according to the model. A graphical user interface (GUI), however, can be used for such descriptions. It should be possible to transfer the two-dimensional model in Fig. 2 directly to a GUI. For example, GUIs have become a basic component of non-linear editing because of the dramatic jump in efficiency that they provide. Macromind's "Director," a well-known multimedia production tool, combines a GUI similar to the two-dimensional structure of Fig. 2 with a text-based language for describing actions.

Although GUI appears to be appropriate for describing a television programs, we have designed TVML as a text-based intermediate language. This is because we are considering the generation of TVML scripts by computer in addition to their preparation by people. For example, current research is examining the possibility of assigning indexes to video clips in television programs so that "digest versions" of programs or customized programs created for specific audiences can be prepared. Such research requires that the description of a television program be understood by the computer. It can therefore be said that a GUI is a good method from the viewpoint of human use when

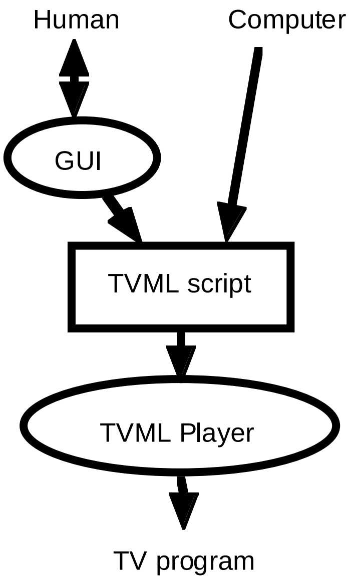


Fig. 3. TVML is a common language between human and computer

describing programs, and that a text-based language is clearly better when a program description must be handled by computer. Therefore, we adopted the mechanism shown in Fig. 3. Here, TVML is positioned as a language that can be used by both people and computers. When people produce programs, some type of GUI is used for describing the programs then outputting TVML scripts, and when computer produce programs, the computer make TVML scripts directly.

3.2 Guidelines for determining specifications

The specifications for the TVML language have been determined with reference to the structure of program scripts used in actual television program production. In addition, to enable TVML to be handled as an intermediate language by the computer as mentioned above, GUI-oriented elements were avoided and specifications were set according to the following guidelines:
(1) Descriptions are to be of the event-driven type without establishing a time axis.
(2) Actions are to be expressed by meaningful words.
(3) The language is to be completely interpretive.

According to guideline (1) above, TVML has been designed so that an event unit corresponds to one line of TVML. Advancing from one line to the next means moving through time. Moreover, a structure is employed that has no conditional branches, loops, and the like as found in standard computer languages like C and FORTRAN. This enables television programs to be conceived and expressed in TVML in much the same way that a person might write a paper or a scriptwriter might write a script. With regard to guideline (2), the problem arises as to what level of abstraction should be used when describing actions. Eventually, of course, the computer must be able to read such descriptions and execute the given actions. The level of abstraction for these descriptions should therefore be set so that the computer is just able to perform the action automatically. This means that, as shown in Fig. 4, the work below where TVML is set is automatically performed by the computer and work above that level is creative work to be performed by people. In other words, TVML is placed at the point where automatic work and creative work meet. The classes of events in TVML are listed in Table 1.

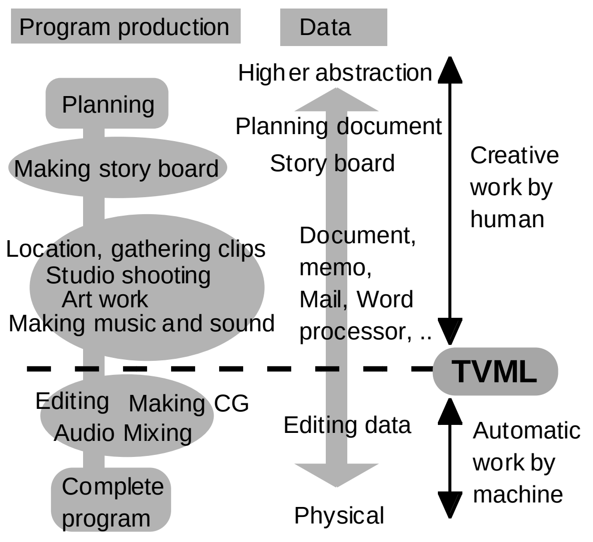


Fig. 4. Level of abstraction where TVML should be setup

Function	Event type	Examples of command (a part)
CG character	character:	talk (make a speech), walk, look (look at something), sit, stand, bow, ...
CG camera	camera:	closeup (focus on something), twoshot (two shot), ...
CG studio set	set:	openmodel (open modeling data), change (change set), ...
CG prop	prop:	position (position prop), openimageplate (make a plate with texture image), ...
CG lighting	light:	model (setup lighting), switch (turn on and off the light), ...
Motion picture	movie:	play (movie file playing), ...
Title	title:	display (display title), ...
Superimposing	super:	on (turn on caption), ...
Sound	sound:	play (audio file playing), mixer (control mixer), ...
Narration	narration:	talk (make a speech), ...

Table 2. Events and associated commands provided in TVML (see Appendix for a complete list)

Finally, guideline (3) is adopted so that the script will be executed in a step-by-step time sequence with a description format that does not generate ambiguities and does not require that the entire script be scanned beforehand. This guideline is also adopted so that TVML can be applied to live programs being shown in real time.

3.3 Basic specifications of the TVML language

The following describes basic specifications of TVML with an emphasis on event description and time description.

(1) Event description

The event format consists of an event type from the list of classes in Table 1, a command provided for the given event type, and a list of parameters required by the command for execution. For example, to make a character named BOB say "Hello, I'm BOB," one would write:

character: talk (name=BOB, text="Hello, I'm BOB.")

Here, "character" is the event type, "talk" is the command, and the items inside the parentheses are parameters. The general format of an event is as follows:

Event type: Command name (arg1=data1, arg2=data2, arg3=data3, ...)

Table 2 lists types of events and associated commands provided in TVML. Here, parameters represented by "arg" in parentheses need not follow a particular order and default values are adopted for any parameters that are omitted. In other words, parameters that are not explicitly specified by the user will be set by the computer. Conversely, the user can specify as many parameters as needed for indicating detailed instructions. For example, if the user desires to indicate a certain speaking speed and an exaggerated character gesture, the event description could be written as follows.

character: talk (name=BOB, text="Hello, I'm Bob, rate=5.0, emotion=excite)

(2) Time description

One event is described on one line in TVML. Upon completion of each event, the system advances to the next line to process the next event. Consider, for example, the following two lines.

character: talk (name=MARY, text="Look at this video.")
movie: playfile (filename=test.mov, from=30, to=450)

Here, the computer-generated character named MARY is made to say "Look at this video," and after this, the motion-picture file named "test.mov" is played back from frame 30 to frame 450. There are actually two types of events: an action event in which time is taken to perform some action (like "sit") lasting from the beginning to the end of the event, and a state event that simple specifies a state change (like superimposing ON) with no elapsed time. Consequently, if a user wants to execute two action events simultaneously, "wait=no" must be added as a parameter to the first action event. This expression will be treated as a state command. For example, to make the characters BOB and MARY bow at the same time and to superimpose text after the bowing is completed, the following event descriptions would be used.

```
character: bow (name=BOB, wait=no)
character: bow (name=MARY)
super: on (type=text, text="Mary & Bob")
```

Here, if "wait=no" is not included in the first line, the actions would be such that MARY starts bowing only after BOB completes his bow.

When "wait=no" is added to an action event and treated as a state event, it becomes necessary to check whether a certain action event has been completed. This is why a wait command is provided for all action events. The format of the wait command is as follows.

```
Event type: wait_Command Name (arg1=data1,
                       arg2=data2, arg3=data3, .....)
```

This wait command blocks further execution of the script until the action specified by the command is completed. In short, the following line:

```
character: bow (name=MARY),
```

is equivalent to the following two lines:

```
character: bow (name=MARY, wait=no)
character: wait_bow (name=MARY).
```

The above structure can therefore be used to describe simultaneously occurring events. For example, if the user wants to superimpose text 1.5 seconds after starting playback of a motion-picture file from frame 100 to frame 200, the following description can be used.

```
movie: playfile (filename=test.mov, from=100,
                                    to=200, wait=no)
wait (time=1.5)
super: on (type=text, text="This is a test movie.")
movie: wait_playfile()
```

In addition to events and wait commands, TVML also provides a several commands that are called directly without being treated as events. These are called "direct commands."

4. TVML Player

4.1 TVML Player operation

The TVML Player is software that reads a program script written in TVML and converts the script to program video and audio in real time. The hardware platform of the TVML Player consists of a Silicon Graphics O2 graphics workstation and a voice synthesizer, as shown in Fig. 5. Text data to be read aloud can be input to the voice synthesizer via a serial port. The audio output of the synthesizer is passed to the A/D input of the O2 workstation, and then the A/D converted data are subjected to a low-pass filter to produce voice-level output. The mouth of the computer-generated character is opened in direct proportion to the magnitude of this output to achieve lip-synching. The TVML Player supports QuickTime and SGI movies as well as motion JPEG as motion-picture files, AIFF and AIFC files as audio files, and TIFF as still pictures. It also supports OpenInventor and VRML 1.0 for the modeling data format in computer-generated characters, sets, and props. The TVML Player features a straightforward user interface with buttons

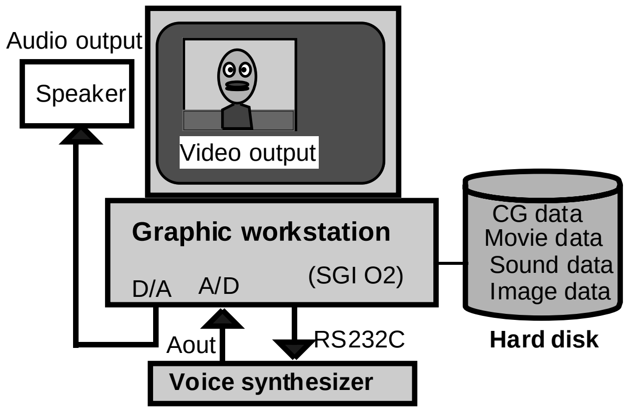


Fig. 5. Hardware platform for TVML Player

for playback, stop, and other functions, enabling selection of TVML-script files and immediate playback. Figure 6 shows an example of a script written in TVML and a sample of a program video generated from this script by the TVML Player.

4.2 TVML Player external control mode

As described above, the basic operation of the TVML Player is to play back a TVML program script from start to finish. However, let us consider for a moment an interactive application that gives a user the choice of changing the story of a program that he or she is currently watching. In this case, there must be some kind of interface to accept a user's selection and some method for dynamically changing the story development based on the selection made[5]. This can be achieved through TVML because a mechanism has been provided to control the TVML Player from an external interactive application program. We point out again that the TVML Player functions completely as an interpreter in which one line of TVML, corresponding to one TVML event, is read in, syntactically parsed, and executed before moving on to the next line. A real-time interactive application can therefore be constructed by having an external application send a script to the TVML Player in an asynchronous manner. The mode which enables a script to be sent to the TVML Player from an external application is referred to as the "external control mode" of the TVML Player.

As shown in Fig. 7, booting up the TVML Player in external control mode allows an external program to write commands to a "control file" provided in the current directory and thereby control the TVML Player. Consequently, if the TVML Player is always left on, an external program can execute any TVML script (of at least one line) at any time as well as suspend external operation at any time. Moreover, TVML Player outputs the script playback status to a "status monitoring file" also provided in the current directory. In external control mode, therefore, with the TVML Player up and running, an interactive application can be constructed by having an external application write commands and send scripts to the control file while checking the status monitoring file.

5. TVML Editor

For non-technical people to use TVML as a program-production tool, a user interface is necessary that enables a user with no specialized knowledge of computer languages to create a television program by intuitive operations. We have also been developing a user interface called TVML Editor[6][7] that incorporates a GUI for such users. The TVML Editor allows users to create a television program by computer

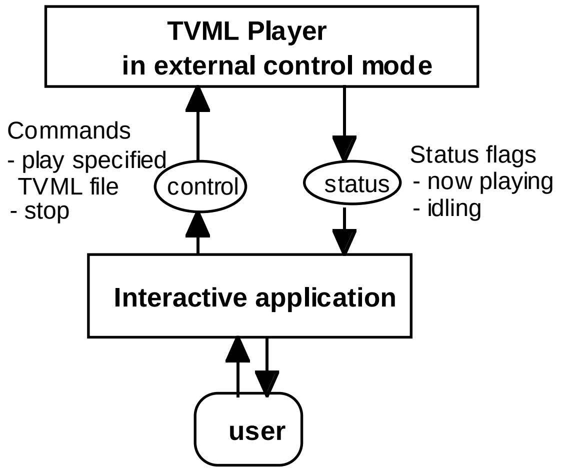


Fig. 7. TVML Player in external control mode

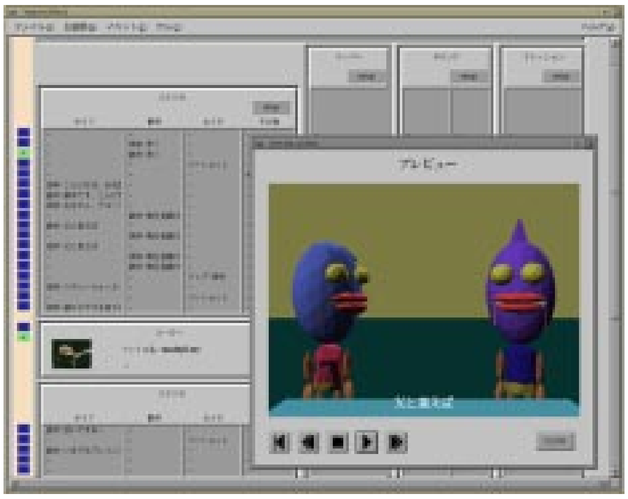


Photo 1. A sample screen of a TVML Editor

mouse and window-based operations while interactively checking intermediate results. In the TVML Editor sample screen shown in Photo 1, a television program is prepared by lining up a studio-shot, a motion picture, and title blocks in a time sequence and making clips where necessary. Each of these blocks is divided into cells, and each cell corresponds to a TVML command. Double-clicking on a cell opens an operation window having buttons, scroll bars, and so forth, that enables parameters such as dialogs to be set up easily. A program created on the TVML Editor can be exported as a TVML script for playback on the TVML Player.

6. Conclusion

This paper described the TVML language, designed for describing television programs by text-based scripts, and the TVML Player that interprets the scripts and outputs program video and audio. We also described the TVML Editor, which is a user interface that enables individuals without specialized knowledge of computer programs to prepare television programs. These developments provide an environment in which anybody can prepare a personal television program in a relatively easy manner on a desktop computer.

One objective of TVML is to popularize program production by making television-program production techniques available to the general user as described above. This is not the only purpose of TVML, however, as many possibilities can be envisioned for it. The following gives some examples.
- The TVML system can be used as a training tool and simulator for professional program directors before they begin producing actual programs.
- As high-performance computers are expected to be incorporated in television sets in the future, a TVML Player installed in the built-in computer can provide a powerful infrastructure for interactive television.
- TVML can be used to achieve automatic program-generation for programs consisting of regular formats and performances like news programs, because core information given by users can be converted to a TVML script automatically using the regular format of the program.
- If the TVML Player is plugged into a WEB browser, program scripts can be distributed and reused over the Internet.
- TVML-based applications can be created that use television programs as a metaphor to present various forms of data like text, pictures, video, and audio stored in databases.

A variety of applications like those described above are currently being studied in a comprehensive manner. We would like to point out here that the research and development of TVML is being conducted on the basis of open research system, and that joint-research partners are always being looked for.

The TVML Player is distributed without charge as freeware. The procedure for obtaining a copy is described at the WEB site given below. This site includes detailed information not given in this paper and we suggest that you pay the site a visit.

http://www.strl.nhk.or.jp/TVML/index.html

References

[1] M. Hayashi, “Automatic Production of TV Program from Script - A proposal of TVML,” Proceedings of the 1996 ITE Annual Convention, S4-3, pp.589 - 592, (1996)

[2] M. Hayashi, “Machine TV Program Generation from Text-based Script,” Proceedings of the Second Symposium on Intelligent Information Media, pp.137 - 144, (1996)

[3] M. Hayashi, Y. Orihara, S. Shimoda, H. Ueda, T. Yokoyama, K. Yaegashi, T. Kurihara, M. Yasumura, “A language specification of TVML (TV program Making Language),” Proceedings of the 1997 ITE Winter Annual Convention, 4-4, pp. 87, (1997)

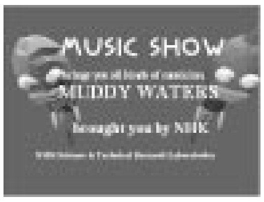
[4] M. Hayashi, Y. Orihara, S. Shimoda, H. Ueda, T. Yokoyama, K. Yaegashi, T. Kurihara, M. Yasumura, “A language specification and a Method for CG Generation of TVML (TV program Making Language),” Proceedings of the Third Symposium on Intelligent Information Media, pp.141 - 148, (1997)

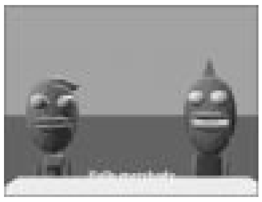
[5] M. Hayashi, “TVML (TV program Making Language) Applied to Interactive Application,” Proceedings of the Information Processing Society in Japan Annual Convention, Demo 4, 3-641, (1998)

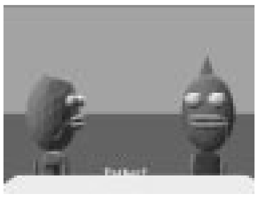
[6] T. Yokoyama, K. Yaegashi, H. Ueda, M. Hayashi, Y. Orihara, S. Shimoda, T. Kurihara, “A TV program generating / interactive aditing system based on TVML (TV program Making Language),” Proceedings of the Third Symposium on Intelligent Information Media, pp.75 -80, (1997)

[7] Ueda, H., Hayashi, M. and Kurihara, T., "DeskTop TV Program Creation - TVML (TV program Making Language) Editor -," ACM Multimedia'98 State of the Art Demos (1998).

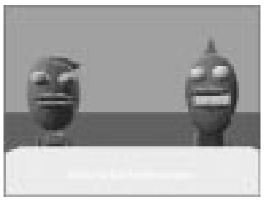
```
// Introducing Muddy Waters by TVML

set: change(setname=default)
character: casting(name=BOB)
character: casting(name=MARY)
character: bindmodel(name=BOB, modelname=BOB)
character: bindmodel(name=MARY, modelname=MARY)
character: position(name=BOB, x=-0.2, y=0, z=0.2, d=180, posture=standing)
character: position(name=MARY, x=0.2, y=0.0, z=0.2, d=180, posture=standing)
character: setvoice(name=BOB, voicetype=e_man)
character: setvoice(name=MARY, voicetype=e_kid)
camera: movement(pan=0, tilt=55.0, x=0, y=0.34, z=-0.39, vangle=50,
transition=immediate)

sound: playfile(filename=muddyhoochie.aiff)
super: on(type=infilehtml, tagname=open.script)
character: walk(name=BOB, x=-0.1, y=0.0, z=-0.3, d=200, wait=no)
character: walk(name=MARY, x=0.1, y=0.0, z=-0.3, d=160)
camera: twoshot(name1=BOB, name2=MARY)
character: sit(name=BOB, wait=no)
character: sit(name=MARY)
super: on(type=text, text = "MARY                BOB")
character: bow(name=BOB, wait=no)
character: bow(name=MARY)

character: talk(name=BOB, text="Hello everybody.")
character: talk(name=MARY, text="Thank you for tuning in.")
character: talk(name=BOB, text="Do you know who's the father of the Blues?")
            character: look(name=MARY, what=BOB)
character: talk(name=MARY, text="Father?")
            character: look(name=BOB, what=MARY)
character: talk(name=BOB, text="Yes.")
            character: look(name=BOB, what=camera)
            character: look(name=MARY, what=camera)
camera: closeup(what=BOB, transition=continuous)
character: talk(name=BOB, text="It's Muddy Waters.")
camera: twoshot(name1=BOB, name2=MARY, transition=immediate)
character: talk(name=BOB, text="Let's watch him anyway.")

super: off()
sound: stop()
movie: playfile(filename=Waltz4.mov, from=130, to=483, wait=no)
            wait(time=0.5)
            super: on(type=text, text="Muddy Waters sings Mannish Boy.")
movie: wait_playfile()
super: off()

character: talk(name=BOB, text="Here is his brief career.")

super: off()
title: display(type=infilehtml, filename=MUDDYCAREER.script, wait=no)
            wait(time=0.5)
            narration: talk(who=BOB, text="He was born in 1915.")
            wait(time=0.5)

character: talk(name=BOB, text="Bye with his great performance.")
character: talk(name=MARY, text="Bye bye.")

super: on(type=infilehtml, filename=endingsuper.script)
movie: playfile(filename=Waltz4.mov, from=613, to=860)
            super: off()
            sound: stop()

title: display(type=pattern, pattern=black, displaytime=0.5)

$ open.script
<BODY TEXT="#ffffaa"><FONT FACE=rock
SIZE=25><BR><BR>
<C>MUSIC SHOW<BR>
<H5>brings you all kinds of musician.<BR><BR>
<H2>MUDDY WATERS<BR><BR>
<H3>brought you by NHK
$

$ MUDDYCAREER.script
<BODY BGCOLOR="#bbbbbb"
TEXT="#222222"><BR><BR><BR>
<C><H2>Muddy Waters <BR>
<L><H4>
(McKinley Morganfield)<BR><BR><H5>
 Birth: Apr 4 , 1915 - Rolling Fork, MS<BR><BR>
 Death: Apr 30, 1983 - Westmont, IL<BR><BR>
  Style: R&B, Electric Chicago Blues<BR><BR>
 Instruments: Guitar, Harmonica, Vocals
$
```







Let's watch him anyway.



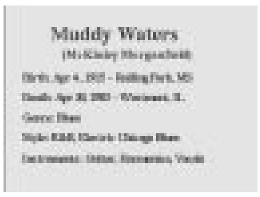

Muddy Waters
(McKinley Morganfield)
Birth: Apr 4, 1915 - Rolling Fork, MS
Death: Apr 30, 1983 - Westmont, IL
Genre: Blues
Style: R&B, Electric Chicago Blues
Instruments: Guitar, Harmonica, Vocals

Fig. 6. An example of a TVML script and a program video generated from the script by a TVML Player

Appendix: TVML command overview

1. CG character command

1.1 **character: casting(name)** - Name CG character
1.2 **character: openmodel(modelname,filename)** - Open the CG character modeling data
1.3 **character: closemodel (modelname)** - Close the CG character modeling data
1.4 **character: bindmodel(name,modelname)** - Bind the CG character modeling data to the character
1.5 **character: setvoice(name,voicetype)** - Set up the CG character's talking voice
1.6 **character: visible(name,switch)** - Show and Hide the CG character
1.7 **character: position(name,x,y,z,d,posture)** - Position the CG character (standpoint)
1.8 **character: talk(name,text,emotion,pausehead,pausetail,rate,pitch, intonation,volume,wait)** - CG character's dialogue
1.9 **character: talkfile(name,filename,emotion,pausehead,pausetail,wait)** - CG character speaks prerecorded dialogue
1.10 **character:walk(name,x,y,z,d,stopmode,pich,compass,stop,wait)** - CG character walks
1.11 **character: stop(name, wait)** - Walking CG character stops
1.12 **character: sit(name, speed, hiplevel, wait)** - CG character sits
1.13 **character: stand(name, speed, wait)** - CG character stands up
1.14 **character: turn(name, speed, style, wait)** - CG character faces different direction
1.15 **character: bow(name, style, speed, level, wait)** - CG character bows
1.16 **character: look(name, what, track, speed, wait)** - CG character looks at something
1.17 **character: gaze(name,pitch, yaw, roll, speed, wait)** - CG character's head turns in specified direction
1.18 **character: shake(name,state, level)** - CG character shakes
1.19 **character: openmouth(name,state, level)** - CG character's mouth opens
1.20 **character: openkeyframe(keyframename, filename)** - Open keyframe data file
1.21 **character: closekeyframe(keyframename)** - Close keyframe data file
1.22 **character: keyframe(name, keyframename, speedratio, repeat, wait)** - Operate CG character by keyframe data

2. Camera command

2.1 **camera: assign(cameraname)** - Name camera
2.2 **camera: switch(cameraname)** - Switch cameras
2.3 **camera: movement(cameraname, pan, tilt, roll, x, y, z, vangle, transition, style, speed, wait)** - Move camera to a specified position
2.4 **camera: twoshot(cameraname, name1, name2, transition, dolly, style, speed, adjustpan, adjusttilt, adjustroll, adjustx,adjusty, adjustz, adjustvangle, wait)** - Twoshot
2.5 **camera: closeup(cameraname, what, transition, dolly, style, speed, adjustpan, adjusttilt, adjustroll, adjustx, adjusty,adjustz, adjustvangle, wait)** - Closeup
2.6 **camera: catch(cameraname, what, track, speed, adjustpan, adjusttilt, adjustroll, adjustx, adjusty, adjustz, adjustvangle, wait)**- Track character only by panning and tilting
2.7 **camera: openkeyframe(keyframename, filename)** - Open keyframe data file
2.8 **camera: closekeyframe(keyframename)** - Close keyframe data file
2.9 **camera: keyframe(cameraname, keyframename, speedratio, wait)** - Operate camera by keyframe data

3. CG studio set command

3.1 **set: assign(setname)** - Name set
3.2 **set: openmodel(setname, filename)** - Open set modeling data
3.3 **set: closemodel(setname)** - Close set modeling data
3.4 **set: change(setname)** - Change set

4. CG prop command

4.1 **prop: assign(propname)** - Name prop (Ex: desk, chair, vase,...)
4.2 **prop: openmodel(propname, filename)** - Open prop modeling data

4.3 **prop: closemodel(propname, filename)** - Close prop modeling data
4.4 **prop: position(propname, x, y, z, pitch, yaw, roll, scale)** - Place the prop
4.5 **prop: visible(propname, switch)** - Show and Hide the prop
4.6 **prop: openimageplate(propname, filename, platesizeh, platesizev)** - Create an image plate prop

5. CG lighting command

5.1 **light: assign(lightname)** - Name lighting
5.2 **light: model(lightname, type, arg0, arg1, arg2, ...)** - Setup lighting type = ambient, flat, point, spot
5.3 **light: switch(lightname, switch)** - Turn lighting On and Off

6. Movie command

6.1 **movie: open(moviename, filename)** - Pre-open movie file
6.2 **movie: close(moviename)** - Stop and close running movie
6.3 **movie: play(moviename, from, to, speedratio, wait)** - Run movie file (In case of pre-open)
6.4 **movie: playfile(filename, from, to, speedratio, wait)** - Run movie file (in case of direct-open)
6.5 **movie: control(type)** - Control running movie type = slow, fast, backward, pause, play

7. Title command

7.1 **title: display(type, arg0, arg1, arg2, ..., wait)** - Display title type = imagefile, infilehtml, htmlfile, pattern

8. Superimpose command

8.1 **super: on(type, arg0, arg1, arg2, ...)** - Superimpose type = text, infilehtml, htmlfile, imagefile
8.2 **super: off( )** - Turn off superimpose

9. Sound command

9.1 **sound: open(soundname, filename)** - Pre-open audio file and name the sound
9.2 **sound: close(soundname)** - Close audio file
9.3 **sound: play(soundname, repeat, wait)** - Play the sound (in case of pre-open)
9.4 **sound: playfile(filename, repeat, wait)** - Play the sound (in case of direct-open)
9.5 **sound: stop(soundname)** - Close the sound
9.6 **sound: mixer(source, action, level, fadeintime, fadeouttime, bgmlevel, wait)** - Operate audio mixer

10. Narration command

10.1 **narration: narratorvoice(voicetype)** - Designate narrator's speaking voice
10.2 **narration: talk(who, text, pausehead, pausetail, rate, pitch, intonation, volume, wait)** - Insert narration

11. Video effect command

11.1 **video: switcher(source, action, transitiontime, effecttype, wait)** - Operate video switcher

12. Direct command

12.1 **wait(time)** - Wait designated time period then return
12.2 **setcaption(switch, displaytime, color, border, borderwidth, bordercolor)** - Closed captions set up
12.3 **displaycaption(state)** - Show and Hide closed captions
12.4 **skipscript(switch)** - S kip TVML script
12.5 **end()** - Finish running TVML script
12.6 **reset()** - Reset TVML Player