

Visual Agent Programming (VAP): An Interactive System to Program Animated Agents

Kamran Khowaja¹ and Sumanta Guha²

¹Department of Computer Science, Isra University, Pakistan

²Computer Science & Information Management,
Asian Institute of Technology, Thailand

kamran_khowaja@hotmail.com, guha@ait.ac.th

Abstract. An interactive system in which the user can program animated agents visually is introduced: the Visual Agent Programming (VAP) software provides a GUI to program life-like agents. VAP is superior to currently available systems to program such agents in that it has a richer set of features including automatic compilation, generation, commenting and formatting of code, amongst others. Moreover, a rich error feedback system not only helps the expert programmer, but makes the system particularly accessible to the novice user. The VAP software package is available freely online.

1 Introduction

During the last few years animated agents, known also as life-like characters, have become increasingly popular. They emulate humans in application and perform the same tasks that are performed by a human. The objective is to afford the user an experience of human-computer interaction similar to that of human-human interaction. Animated agents are deployed in a variety of applications ranging from E-commerce to electronic learning environments [1], [2], [3], [5], [7], [9], [11]. Fig. 1 shows programmable agents currently available from Microsoft.

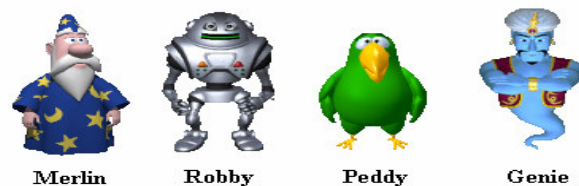


Fig. 1. Agents provided by Microsoft.

There are a few programming environments – both commercial, as well as freeware – currently available for the scripting of animated agents. A very popular one is the

2 Kamran Khowaja1 and Sumanta Guha2

Multimodal Presentation Markup Language (MPML) Visual Editor [6], [10], [12], which allows a content author working with the MPML scripting language to edit a file containing MPML script, as well as to manipulate the graphical representation of the script. However, a significant weakness of this editor is that it provides little by way of feedback or error messages that would enable a user to be able to resolve errors in the MPML script. Neither does it allow access or modification of the finally generated JavaScript code within the editing environment. MPML2.2a [4], DWML [10] and MPML-VR [13] use XSL style sheet to convert MPML script to JavaScript. VHML provides tagging structures for facial and body animation, gesture, emotion, as well as dialogue management [8].

Microsoft Agent Scripting Helper (MASH) provide users an easy to use interface so as they can record and playback their presentations. This is done by dragging & dropping Microsoft Agents on the screen and directing them what to perform next [14]. In addition to the fact that the software is not available for free, the major drawback of MASH is that the generated code is offered for users to edit as well. If user (intentionally or un-intentionally) makes changes in the code there is a possibility that it might generate errors and users are not informed in such situations.

Currently, programming animated agents can be a tedious task, especially for novice-intermediate users who do not have good coding skills. Our goal is to provide an environment that will enable even the inexperienced user to script and deploy agents, thus bringing their utility to a wider audience. This environment, called Visual Agent Programming (VAP), not only provides a graphical user interface (GUI) to program Microsoft Agents, but also enables the user to program these agents manually while simultaneously viewing the underlying script that is generated. An added benefit of VAP's transparency is that a user tends to quickly acquire an understanding of how to use life-like agents to maximum impact in their web application. VAP provides as well a complete and informative debugging environment. VAP statements are automatically compiled when new are added or old are edited, and errors indicated. Microsoft agents currently programmable in VAP are those shown in Fig. 1.

2 VAP Architecture

The architecture of VAP is shown in Fig. 2. There are three main components: the script file, the language settings and the VAP visual editor. The script file contains the script to load characters on a webpage, as well as the actual animation and language-specific script. The language settings contain information related to the languages supported by VAP, their syntax and parameters. The visual editor itself consists of six modules.

- The script loader loads the scripts and passes statements to the program hierarchy generator module.
- The program hierarchy generator creates the tree structure of statements to show parent (function, loop and condition) and child (statements in block) relationship.

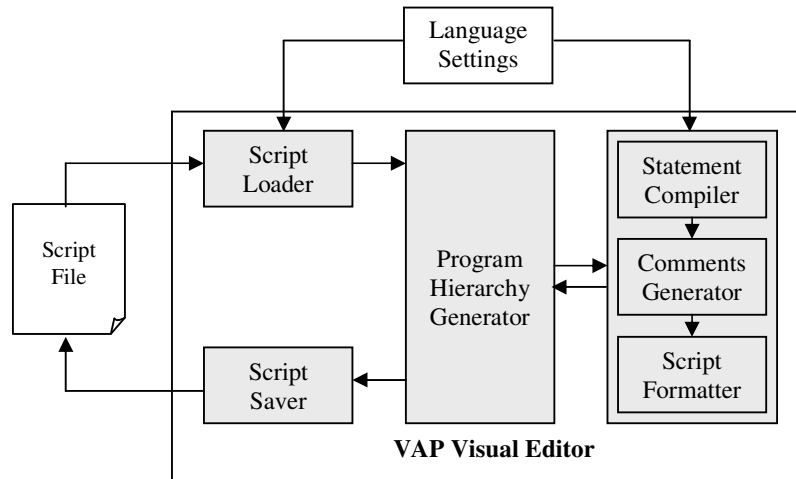


Fig. 2. VAP System Architecture

- The script saver module saves back the structure created by program hierarchy generator to a script file.
- The statement compiler checks the arguments of each statement and informs the user of errors if any.
- The comments generator module generates comments for each statement.
- The script formatter formats the comments and script generated.

VAP supports both VBScript and JavaScript, unlike most of the competing agent programming environments that support only JavaScript.

3 VAP Features

We describe next the distinguishing features that we believe make VAP a superior choice for programming animated agents. Hopefully, the reader already has had an opportunity to run VAP as described in the previous section. This will make the following easier to appreciate.

Stickiness: To make VAP convenient for the novice programmer to use, a feature called stickiness is implemented. Whenever the user either right or left clicks an agent, VAP saves the name of the agent for later use. By default, “Peedy” is the selected agent in VAP. In either case, when the user creates a new command by right clicking on an agent and then selecting command, or right clicking on the program hierarchy and then selecting command, the system will create a new command by automatically adding the name of the agent for whom this command is added. Commands related to a particular agent can be accessed through the agent menu as well.

Auto-compilation: Commands are auto-compiled to show error messages for missing parameters when:

1. A new command is added to the program hierarchy, or
2. Parameters of a command are modified.

Fig. 3 shows the flowchart of automatic compilation. Table 1 shows the commands, their parameters, the types of values expected, and whether the parameter value is optional or not.

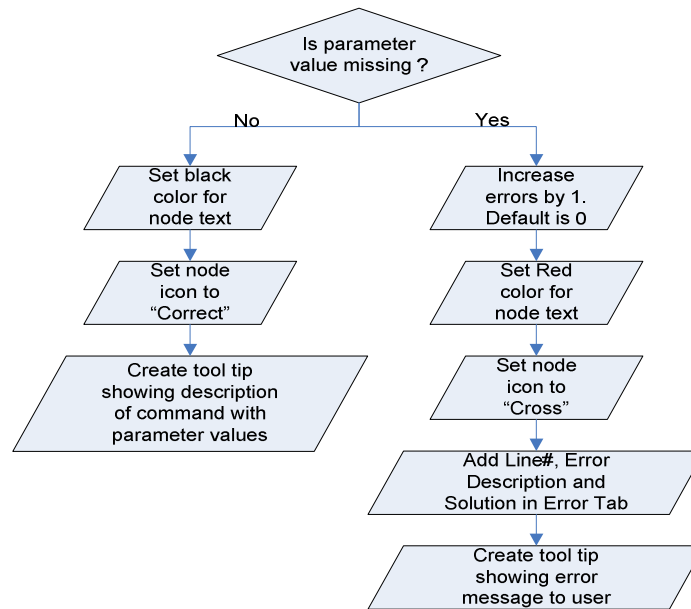


Fig. 3. Flowchart of auto compilation.

Table 1. Command information

Command	Parameter	Data Type	Required/Optional	Default Value
Speak*	Text	String	Required	Click to modify
Move To*	X	Integer	Required	0
Move To*	Y	Integer	Required	0
Play*	Animation	String	Required	-
Alert	Message	String	Required	Click to modify
Print	Message	String	Required	Click to modify
Loop	From	Integer	Required	0
Loop	To	Integer	Required	0
Loop	Inc/Dec	Integer	Required	0

Function	Name	String	Required	-
Function	Parameters	String	Optional	-
Call	Name	String	Required	-
Call	Parameters	String	Optional	-
Math	Expression	String	Required	-
Condition	Type	String	Required	#

Notes:

- indicates that no default value is provided for this command.

indicates that the parameter can accept only one of the selected values (If, Else If, or Else).

* indicates that the command is not agent-specific and can be applied only to agents.

Auto-generation: Program code for a command is automatically generated by the system when the user either creates a new command, or modifies an existing one. Generated script depends very much on the scripting language that is selected. To simplify the interface, script related to loading agents is not shown, as it is not directly relevant to animating the agent. The difference can be seen from fig. 4 and 5, where the former shows only script related to animation, while the latter shows the entire source.

Auto-commenting: The system generates automatic comments for the user and places them before each command and function describing what it will do together with its parameter values. The current version of the system does not allow the user to insert its own comments. Fig. 4 and 5 show example of comments generated by the system.

Auto-formatting: Even if the scripting code in the original file is not properly formatted, the system will re-generate formatted code for the user when the file is saved again. See Fig. 5 and 6, where the latter is unformatted, while the former has been formatted by the system.

Reverse Engineering: This feature of the system allows the user to load scripting code and create program hierarchy. The following steps are performed when the user does reverse engineering.

1. Generate error if the syntax of the command is not correct and set the icon of the node accordingly.
2. Create a tool tip for each command.
3. Create comments for each command.
4. Format comments and commands.

Program code is generated according to the scripting language used. Comments are ignored when reading a file, but generated and inserted when the file is saved back.

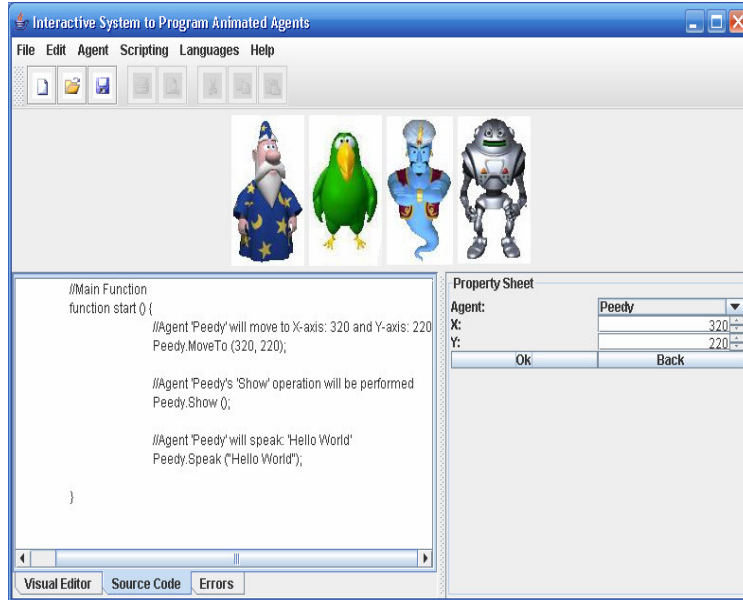


Fig. 4. Partial source code of Hello World.

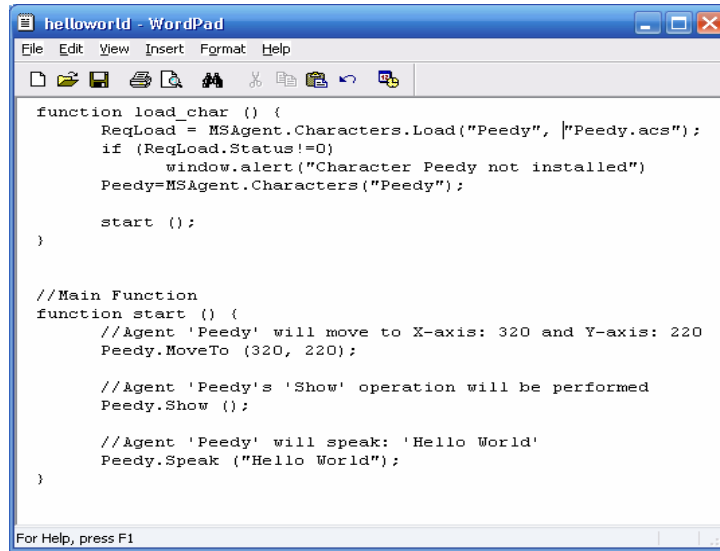


Fig. 5. Complete source code of Hello World.

```

function load_char () {
    ReqLoad = MSAgent.Characters.Load("Peedy", "Peedy.acs");
    if (ReqLoad.Status!=0)
        window.alert("Character Peedy not installed")
    Peedy = MSAgent.Characters("Peedy");

    start ();
}

//Main Function
function start () {
    Peedy.MoveTo (350, 300);
    Peedy.Show ();
    Peedy.Speak ("First Sample");
}
    
```

Fig. 6. Source code of Hello World before using the system.

Error Message: VAP provides a rich error feedback environment. Fig. 7 shows an example in which an error in a statement is highlighted in red and the icon of the statement turned to cross indicated error. Fig. 8 shows the system telling the user what the error is, where and why it has occurred, and how to resolve it. Consequently, VAP requires little programming experience as the user is constantly guided by error messages.

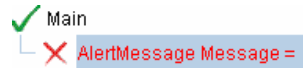


Fig. 7. Error indication in statement.

Error on Line# 2 [Click to Go](#)

Description: Error occured because you either forgot to specify 'message' for 'Alert' command.

Solution: Go to Line# 2 and specify 'message' to remove this error

Fig. 8. Error description and solution.

4 VAP Comparisons

Table 2 shows a comparison between VAP and two currently popular systems. Significant differences can be seen in features such as code conversion, error debugging, automatic compilation, comments generation, etc.

Table 2. Comparison matrix

Option	VAP	MPML	MASH
Underlying Source Code Generation Language	JavaScript and VBScript	JavaScript	JavaScript, VBScript, Visual Basic, VBA for Office etc
Usage Policy	Free	Free	Free: 30 days Single User: \$25 Educational Site: \$250
Type of System	Desktop/Internet	Desktop	Desktop
Underlying System	Microsoft Agents	Microsoft Agents	Microsoft Agents
Code Conversion	JavaScript to VBScript and vice versa	None	JavaScript, VBScript, Visual Basic, VBA for Office etc
Error Description	Detailed Error Description with solution	None	None
Programming Comments	Automatic comments are generated with every single command. Users cannot create their own comments	None	Limited automatic comments are generated by MASH. However, users can create their own as well
Compilation	Automatic compilation of statement when added/edited	Program structure is compiled when saved	None
Character Type	Full Body	Full body	Full body
Audience	Novice – Intermediate	Novice – Intermediate	Novice – Advanced

5 Conclusions and Future Work

We believe that the VAP software package, as well as its underlying ideas, will help popularize the deployment of life-like agents in web applications by providing an easy-to-use GUI for their scripting and animation. Planned extensions to VAP include the following:

1. Development of more sophisticated interaction between user and system.
2. Support for concurrent execution of statements.
3. Tagging of statement to overcome problems of absolute locations.
4. Equipping agents with a human voice.
5. Enable speech recognition by agents.

References

- [1] Andre, E., Rist, T., Mulken, S. van, Klesen, M., and Baldes, S.,. The automated design of believable dialogue for animated presentation teams. In J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, editors, *Embodied Conversational Agents*, pages 220–255. *The MIT Press, Cambridge, MA, 2000*.
- [2] Badler, N.I., Allbeck, J., Bindiganavale, R., Schuler, W., Zhao, L., and Palmer, M. Parameterized action representation for virtual human agents. In J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, editors, *Embodied Conversational Agents*, pages 256–284. *The MIT Press, Cambridge, MA, 2000*.
- [3] Badler, N.I., Bindiganavale, R., Allbeck, J., Schuler, W., Zhao, L. and Palmer, M. Parameterized action representation for virtual human agents, in: J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.), *Embodied Conversational Agents*, *The MIT Press, Cambridge, MA, 2000*, pp. 256–284.
- [4] Du, P., Ishizuka, M., Dynamic web markup language (DWML) for generating animated web pages with character agent and time-control function, in: *Proceedings (CD-ROM) IEEE International Conference on Multimedia and Expo (ICME2001)*, 2001.
- [5] Huang, Z., Eliens, A., Visser, C., STEP: a scripting language for embodied agents, in H. Prendinger (Ed.), *Proceedings PRICAI-02 International Workshop on Lifelike Animated Agents. Tools, Affective Functions, and Applications*, 2002, pp. 46-51.
- [6] Ishizuka, M., Tsutsui, T., Saeyor, S., Dohi, H., Zong, Y. and Prendinger, H.. MPML: A Multimodal Presentation Markup Language with Character Agent Control Functions, 2000.
- [7] Kitamura, Y., Tsujimoto, H., Yamada, T., and Yamamoto, T. Multiple character-agents interface: An information integration platform where multiple agents and human user collaborate. In *Proceedings First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, pages 790–791, New York, 2002. *ACM Press*
- [8] Marriott, A., Stallo, J., VHML – Uncertainties and problems, A discussion, in: *proceedings AAMAD02 Workshop on Embodied Conversational Agents – Let’s Specify and Evaluate Them! 2002*
- [9] Marsella, S.C., Johnson, W.L., and LaBore, C. Interactive pedagogical drama. In *Proceedings 4th International Conference on Autonomous Agents (Agents-2000)*, pages 301–308, New York, 2000. *ACM Press*
- [10] Okazaki, N., Aya, S., Saeyor, S., Ishizuka, M, A multimodal presentation markup language MPMLVR for a 3D virtual space, in: *Proceedings (CD-ROM) of Workshop on*

10 **Kamran Khowaja¹ and Sumanta Guha²**

Virtual Conversational Characters: Applications, Methods, and Research Challenges (in conj. with HF2002 and OZCHI2002), 2002

- [11] Predinger, H., Descamps, S., Ishizuka, M., Scripting Affective Communication with Life-Like Characters in Web-based Interactions Systems. *Applied Artificial Intelligence*, 16 (7-8): 519-553, 2002
- [12] Predinger, H., Descamps, S., Ishizuka, M., MPML: a markup language for controlling the behavior of life-like characters, *Journal of Visual Languages and Computing*, Elsevier, January 2004.
- [13] Saeyor, S., Multimodal Presentation Markup Language Ver. 2.2a (MPML2.2a), 2003, URL: <http://www.miv.t.u-tokyo.ac.jp/~santi/research/mpml2a>.
- [14] MASH: <http://www.bellcraft.com/>.
- [15] VAP: <http://www.cs.ait.ac.th/~b101650>.