ELSEVIER

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Visual Languages and Computing
15 (2004) 183–203

Journal of
Visual Languages
& Computing

www.elsevier.com/locate/jvlc

MPML: a markup language for controlling the behavior of life-like characters

Helmut Prendinger*, Sylvain Descamps, Mitsuru Ishizuka

Department of Information and Communication Engineering, Graduate School of Information Science and Technology, University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

Received 20 January 2003; received in revised form 24 December 2003; accepted 28 January 2004

Abstract

Animated agents have the potential to convey information in a more natural way than other media traditionally used on the world-wide web, such as text, audio, or video clips. They also allow for more natural styles of human–computer interaction than navigation through hypertext documents. In this paper, we will introduce the multi-modal presentation markup language (MPML), which is a powerful and easy-to-use XML-style language enabling content authors to script rich web-based interaction scenarios featuring life-like characters. MPML is a powerful language as it provides controls for the verbal and non-verbal behavior of affective 2D cartoon-style characters, presentation flow, and the integration of external objects, like Java applets. MPML is easy-to-use since we also offer a graphical scripting environment (a visual editor) that facilitates the creation of complex presentation scripts. We will describe the tagging structures of MPML, the MPML3.0 Visual Editor, and illustrate the usefulness of the tagging structures of the language by the actual implementation of a web-based casino where the user may interact with life-like characters.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: Embodied agents; Markup languages; Affective communication; Web-based presentation; Visual editor

1. Introduction

Life-like characters, or animated agents, constitute a promising technology for human–computer interaction. The embodiment of agents provides effective means of

*Corresponding author.
E-mail address: prendinger@acm.org (H. Prendinger).

1045-926X/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jvlc.2004.01.001

imitating human skills such as presenting information or engaging in a conversation [1]. Humans communicate using not only language, but also body gestures and facial expression. The fundamental hypothesis of this research program is that by endowing animated agents with synthetic counterparts of the interaction modalities used by humans, character-based interfaces can be effective tools for enhancing the interaction between humans and computers.

The wide dissemination of animated agents, however, will greatly depend on the availability of appropriate tools that facilitate the scripting (or authoring) of agents with life-like behavior. The tasks for putting animated agents to work are manifold, including the control and synchronization of a character's verbal and non-verbal (embodied) behavior and the coordination of the behavior of multiple characters. Scripting languages are often designed for the representation of one or more levels of character description. Whereas scripting at a 'low level' involves manipulating a character's facial and body motion models, 'high level' scripting is concerned with the synchronization of a character's synthetic speech with appropriate gestures, the expression of emotions, or the synchronization of body motion sequences of a group of characters. Description levels can also be conceived as abstraction levels where higher levels 'abstract from' lower level description concerns. Abstract specifications are then instantiated to low-level body actions by special purpose interpreters [2,3].

While most markup languages developed for life-like characters demonstrate sophisticated control mechanisms, they typically address programming-expert content creators or animators rather than the average user. Here, the 'average user' is seen as someone who wants to add animated agents to a (possibly interactive) web page, without being willing or able to give much thought on the details of character control. Instead, the content author simply wishes to select some behaviors, mostly pointing and conversational gestures, in order to support and enhance the character's verbal behavior. Non-technically oriented authors include content creators for web-based commercial or educational information.

In this paper, we will describe the *multi-modal presentation markup language*, a language specifically designed for non-expert (average) users allowing them to direct the behavior of multiple animated characters when creating web-based (interactive) presentations. Here, we will explain MPML3.0, a particular version of the MPML language family (other members will be briefly introduced in Section 2).[1] First, MPML is a *markup language* compliant with standard XML [4] and hence allows one to script in a style similar to HTML scripting. (XML-style scripting is actually not even necessary since we also provide a visual editor for MPML.) Second, MPML is a language tailored to script character-based *presentations* that can be viewed in a web browser (Internet Explorer). In order to facilitate the generation of different kinds of presentations, MPML provides appropriate tagging structures that enable authors to utilize features that are familiar from presentations given by human presenters in web-based presentation environments, such as dynamic media objects or interaction with the audience. Finally, MPML supports the generation of *multi-modal* presentations, i.e. presentations that utilize multiple mechanisms to encode

[1] Unless indicated otherwise, "MPML" refers to MPML3.0, the language introduced in this paper.

to-be-conveyed information, including 2D graphics and spoken (synthetic) language, music, and video [5]. Here, we will focus on modalities specific to animated agents. Besides synthetic speech, animated characters may communicate information by using multiple modalities, such as facial displays (in order to express emotions), hand gestures (including pointing and propositional gestures), head movements (e.g. nodding), and body posture. Markup languages supporting other types of media on the web are introduced, e.g. in a recent special issue of this journal [6].

The remaining paper is organized as follows. Section 2 reports on other markup languages we have developed and work related to our own. Section 3 is dedicated to motivating requirements for a markup language for web-based presentations. Section 4 introduces the MPML system. We first give an overview of the architecture of the MPML system, and then describe the language specification of MPML. Section 5 describes a visual editor for MPML. In Section 6, the usefulness of the markup language is demonstrated by discussing an actual implementation of a web-based presentation, an interactive casino scenario featuring life-like characters. Sections 7 discusses and 8 concludes the paper.

2. Related work

In recent years, various scripting languages and technologies have been developed to direct the behavior of life-like characters. The virtual human markup language (VHML) provides tagging structures for facial and bodily animation, gesture, speech, emotion, as well as dialogue management [3]. The language is XML-based (i.e. it employs the extensible stylesheet language (XSL) [7]) and covers different abstraction levels. Despite its expressive power, current applications of the language mainly focus on 'Talking Heads' rather than full-body characters. The character markup language (CML) and avatar markup language (AML) focus on sophisticated synchronization of a character's bodily behavior and synthetic speech [2]. Both languages are XML-based and allow to specify character behavior at different levels. Unlike MPML, CML and AML are machine-generated and hence not easy to read or create by non-experts.

The affective presentation markup language (APML) is an XML-based language that allows to represent communicative functions and thus facilitates the scripting of dialogues with embodied characters [8]. By contrast to MPML, the APML language allows to specify the meaning of a communicative act which is then instantiated to a specific communicative signal, depending on the character's personality or culture. The work of Cassell et al. [9] and Kranstedt et al. [10] supports the precise synchronization of synthetic speech and non-verbal behavior.

Character representation languages that do not use an XML-based approach include STEP [11] and PAR [12]. The scripting technology for embodied personal language (STEP) uses distributed logic programming and action composition operators to specify motions of VRML-based characters. The parameterized action representation system (PAR) allows for a fine-grained specification of parameters to modify the execution of character actions, such as duration, purpose, emotion and personality. While those approaches permit more sophisticated control of animated characters, they might counteract our purpose of creating a character markup language that can be used by average users rather than animation experts.

Besides the version of MPML described in this paper (MPML3.0), our efforts in designing markup languages for web-based presentations also lead to the construction of languages that employ XSL to define the form of the MPML content script, rather than a converter. However, none of those languages provides a graphical editor to support the user's scripting effort (although in principle, such an editor could be written).

The following three members of the MPML language family adhere to the technique of using an XSL stylesheet to convert the MPML script to JavaScript. Like MPML3.0, the first two languages use the Microsoft Agent package [13] as an animation engine, whereas the third one uses VRML2.0.

- *MPML2.2a*: This version of MPML supports sequential and parallel behavior (synthetic speech and actions) of multiple characters. It also provides an interface to Macromedia Flash, so that a character can control a Flash movie and a Flash movie may contain triggers for character behavior [14].
- *DWML*: This member of the MPML family is concerned with scripting time-dependent relations between web-based media objects in addition to the 2D animations displaying a synthetic character. The media objects supported by the *dynamic web markup language* (*DWML*) include text, graphics, audio, and video [15]. A time control function allows to define the temporal occurrence of media objects during presentation, which are otherwise (in HTML programming) immediately shown when a web page is loaded.
- *MPML-VR*: Here, a variant of the MPML2.2a language is employed and enriched to control a 3D virtual space and a 3D character. The resulting markup language—*MPML for Virtual Reality*—allows for presentations in three-dimensional space [16].

While the above-mentioned languages primarily focus on alternative ways to integrate life-like characters to web pages, another strand of our research deals with enhancing the flexibility of human–character interaction. The MPML version described in this paper supports two types of user–character dialogue. First, the character may simply respond with a pre-defined utterance dependent on the user's speech input. Second, if MPML is interfaced with the SCREAM system [17], the character's response is decided by its mental makeup (in addition to the user's speech input), but still the responses have to be prepared by hand for each interaction scenario. In order to alleviate the author from the effort to prepare character responses for user input, we recently implemented an interface between MPML3.0 and a popular *chatbot*, the Alicebot [18,19]. The Alicebot provides a large set of responses for a broad variety of topics which are accessible from the web. By using a special type of pattern matching, this chatbot shows response robustness for (almost) any imaginable user input. Moreover, by using artificial intelligence markup language (AIML), authors of interactive presentations may easily define their own response patterns for the character.

3. Requirements for a presentation markup language

The design of a markup language is driven by the requirements of its intended use. From an author perspective, a presentation markup language should provide an easy-to-use and powerful tool for scripting different types of presentations. Authors are typically content experts that seek to distribute information about a certain product which can be a particular creation of a company or a research program pursued by a scientific group. From a user (viewer of presentation) point of view, the scripted content should be easily accessible, and allow for interactivity with the presenter.

In the following, we will propose some requirements for a presentation markup language, and then discuss some features of MPML in the light of those requirements. We start with requirements that are important for presentation authors.

- *Ease of use*: Presentations featuring animated characters should be easy to write and not assume programming skills. Authors should have the option to either directly manipulate the script or use a visual editor with a conventional interface.
- *Intelligibility*: The tags of the markup language should provide names and abbreviations that clearly indicate their meaning (semantics).
- *Extensibility*: The markup language should support authors in specifying new functionality, in order to allow easy adaption to special requirements of content providers.
- *Easy distribution*: The presentation should be easy to install on the user's platform.

Since MPML is an XML-style markup language, scripting is intuitive and accessible even to non-programmers. We do not imply that scripting in XML is a straightforward task—it can become quite involved when the length of the content grows, resulting in long chains of deeply nested tagging structures. However, although scripting in XML requires a certain amount of concentration, it does not require knowledge in JavaScript or VBScript. Following up to the current trend in web content authoring tools for HTML, we have also developed an editor for MPML that depicts the flow of the presentation graphically and hence supports the content author in generating complex presentations.

Tags in MPML are easy to learn and remember, as they follow conventions well known from scripting web pages with HTML, such as the `head` and `body` elements, or from human performances (e.g. the `scene` element). MPML is extensible as it provides tagging structures that interface MPML with the target control language of characters and web pages (JavaScript). Thus, features not supported by MPML can be added at the cost of scripting in the JavaScript programming language. We certainly tried to avoid this situation for character behavior scripting, but we cannot foresee the numerous features which content users would need for web page scripting. Consequently, extensibility of MPML is important in order to guarantee flexibility.

Since a converter transforms MPML to JavaScript, every browser that has JavaScript enabled can run a presentation written in MPML. However, there are restrictions deriving from the agent system used to control the behavior of animated characters, namely the microsoft agent package [13]. Currently, MPML assumes Microsoft Internet Explorer 5.5 (or higher) to run a character-based presentation.

Although MPML is primarily an authoring language, the audience perspective has to be taken into consideration. Besides easy installation (see the 'easy distribution' requirement above), the audience should be able to interact with the animated characters that give the presentation. MPML is designed to allow for *interactive presentations* where the presentation flow halts and waits for the user's speech input to direct the future development of the presentation.

4. The MPML system

In this section, the MPML system will be explained. We will first describe the system architecture, and then introduce the tagging structures that MPML provides to create web-based presentations.

4.1. System architecture

An overview of the MPML system architecture is shown in Fig. 1. There are two main components: The MPML Script text file and the MPML3.0 Visual Editor application program. The MPML Script file contains the tagging structures specifying character and environment control. The MPML3.0 Visual Editor consists of four modules.

- The *Script Loader* module loads the text file containing MPML script and checks the script for syntactical errors.
- The *Graph* module visualizes the script by generating a graphical representation.
- The *Script Saver* module converts the graph to a textual MPML script.

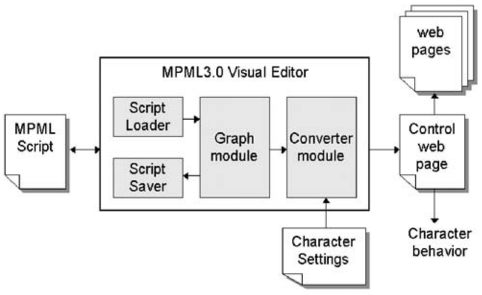


Fig. 1. MPML3.0 system architecture.

- The *Converter* module transforms the MPML script to a language that is executable in a web browser. We use JavaScript in Internet Explorer 5.5 (or higher).

The resulting Control web page is responsible for instructing the characters' behavior and background web pages, i.e. pages depicting the environment that the characters inhabit. The user may also define character specific settings, which are stored in the Character Settings file and processed by the converter. Typically, this file contains speech related parameters for emotion expression.

4.2. MPML specification

A compact way to describe a markup language (a set of tagging structures) is to give its document-type definition (DTD), a set of rules that defines the grammar of an XML document. We acknowledge that DTD technology has been superseded by XML Schemas [20]. Although MPML could be specified by XML Schemas, we opted to present the markup language in a DTD for readability. Alternatively, the tagging structures of the markup language could be written down. However, for space reasons, tagging structures will only be used when showing example scripts.

4.2.1. Introductory tagging structures

Here we will briefly discuss introductory tagging structures. The root element of an MPML script is `mpml` (see Fig. 2). The root element's tagging structure `<mpml>...</mpml>` contains all other tagging structures of the script. The `head` element specifies general information, the title, and the agent(s) that will perform as presenters. Currently, the Microsoft Agent package is used as the default agent system, which includes controls for character animation, a text-to-speech (TTS)

```
<!ELEMENT mpml(head,body)>
<!ELEMENT head(meta*| title?| agent+)>
<!ELEMENT metaEMPTY>
<!ATTLIST metaid          CDATA        #REQUIRED
          description  CDATA        #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT agent EMPTY>
<!--Currently, only Microsoft Agent characters are used-->
<!--TTS Engines other than Lernout & Hauspie TruVoice can be used-->
<!NOTATION lh SYSTEM"B8F2846E-CE36-11D0-AC83-00C04FD97575">
<!ATTLIST agentid         CDATA          #REQUIRED
          system       CDATA"MSAgent" #FIXED
          x            CDATA          #REQUIRED
          y            CDATA          #REQUIRED
          voice        NOTATION lh    #REQUIRED>
<!ELEMENTbody(seq+)>
```

Fig. 2. DTD for introductory elements.

engine, and a voice recognizer [13]. The seq element which may occur one or more times within the <body>...</body> tagging structure refers to the sequence of events comprising the presentation.

4.2.2. Tagging structures for character control

Tagging structures controlling character behavior are summarized in Fig. 3. They allow the script author to make the character speak, think, perform an action, move to a certain location on the screen, and listen to the user's speech command. The speak element may contain a (unordered) sequence of three child elements, emotion, nb, and txt, which provide the means to let the character speak one or more sentences with an affective voice, possibly separated by opening a new balloon after a certain part of the text has been spoken. The character may also speak some text which results from user interaction with the background web page. For instance, if the user previously input his or her name to a form, the character can retrieve the

```
<!ELEMENT speak ((emotion| nb | txt)*)>
<!-- The id of the agent -->
<!ATTLIST speak agent     CDATA #REQUIRED>
<!ELEMENT emotion EMPTY>
<!ENTITY % BASIC-EMOTIONS
      "(fear | anger | sadness| happiness| disgust)">
<!ATTLIST emotion assign %BASIC-EMOTIONS #REQUIRED>
<!ELEMENT think ((nb | txt)*)>
<!-- The id of the agent -->
<!ATTLIST think agent     CDATA #REQUIRED>
<!ELEMENT act EMPTY>
<!-- The id of the agent -->
<!-- An action specified for the agent like Greet, Explain, or Surprised -->
<!ATTLIST act agent       CDATA #REQUIRED
             act          CDATA #REQUIRED>
<!ELEMENT move EMPTY>
<!-- The id of the agent>
<!ATTLIST move agent      CDATA #REQUIRED
              x           CDATA #REQUIRED
              y           CDATA #REQUIRED>
<!ELEMENT listen (heard+)>
<!-- The id of the agent -->
<!ATTLIST listen agent    CDATA #REQUIRED>
<!ELEMENT heard EMPTY>
<!-- A sentence that should be recognized -->
<!ATTLIST heard sentence  CDATA #REQUIRED>
<!ELEMENT nb EMPTY>
<!-- A duration in milliseconds -->
<!ATTLIST nb pause CDATA>
<!ELEMENT txt EMPTY>
<!-- The address of a variable defined in the background web page -->
<!ATTLIST txt target      CDATA #REQUIRED>
```

Fig. 3. DTD for character-related elements.

name, and address the user in a personal way. A balloon serves as a visual support for synthetic speech. By using the `listen` element, the speech recognizer is activated and recognizes sentences which are defined as attribute values of the `heard` element (a child of the `listen` element).

By way of example, assume the script author wants the character to say "Do you want to go out, or do you want to stay at home?", whereby the first part of the sentence should be spoken in a happy voice and the latter part in a sad voice. In MPML script, the following code is used (the character's name is "Shima").

```
<act agent="Shima"act="Thisthatfirst"/>
<speak agent="Shima">
 <emotion assign="happiness"/>
 Do you want to go out,</speak>
<act agent="Shima" act="Thisthatsecond"/>
<speak agent="Shima">
 <emotion assign="sadness"/><nb pause="200"/>
 or do you want to stay at home?</speak>
```

The character starts with performing the gesture denoted by "Thisthatfirst" and then says the first part of the sentence according to the speech parameters for "happiness" (explanations will be given momentarily). Then, the character performs the gesture denoted by "Thisthatsecond", and says the second part of the sentence according to the speech parameters for "sadness" after 200 milliseconds. The second sentence is displayed in a new balloon.

Due to the limitations of the TTS engine, only speech rate and pitch average are currently used to express the vocal effects of five emotions (fear, anger, sadness, happiness and disgust). Following Murray and Arnott [21] we assume that, e.g. if a speaker expresses happiness, then his or her speech is typically faster (or slower) and higher-pitched, whereas a sad speaker's speech is usually slightly slower and lower-pitched.

All gestures are pre-defined 2D animation sequences controlled by the Microsoft Agent package. Depending on the character used, the type and number of available gestures may differ, as well as the names given to animation sequences. Most available characters have about 50 animations at their disposal, including the following:

- *Locomotion*: "Moveup" (character performs a step), "Moveleft", "Show" (character appears), "Hide" (character disappears).
- *Deictic gestures*: "Gestureright" (character points to the right by hand gesture), "Gestureup", etc.
- *Gestures realizing communicative functions* [22] can be implemented as follows:
  - Reacting to new conversation partner. "Alert" (character looks at partner).
  - Take turn. "Lookleft" or "Lookright", then character starts speaking.
  - Give feedback. "Acknowledge" (character nods).

- *Gestures expressing emotions*: "Sad" (character displays sad face and 'hanging' shoulders), "Pleased" (character displays 'big smile'), "Anger" (character displays angry face and stems hands on hips), "Fear" (character lifts arms), etc.

Character animations also contain gestures supporting speech, such as the *contrastive gesture* or *propositional gestures* [22]. The contrastive gesture depicts a 'on the one hand ... on the other hand' relationship if two items are contrasted. This is realized by the sequence of "Thisthatfirst" (left-hand stroke) and "Thisthatsecond" (right-hand stroke). The final view of this animation sequence is shown in Fig. 4. An example of a propositional gesture is the use of both hands to measure the size of an object in symbolic space while saying "there is a big difference". These and many more animations are triggered by using the act tagging structure.

However, for periods of non-activity (i.e. no particular action is to be performed), the character will randomly perform an action from the set of 'idle' behaviors, such as looking left or right, folding arms, etc. Idle behaviors are intended to support the illusion of a character's life-likeness.

4.2.3. *Tagging structures for presentation control*

The flow of a presentation written in MPML is specified by the tagging structures shown in Fig. 5. The scene element determines which characters are visible in the browser window. The seq and par elements are responsible for events that should be executed sequentially or in parallel, respectively, and inspired by the corresponding tags in SMIL [23]. Most importantly, the par element implements a (simple) form of conversational behavior between characters, where a character's speech can be synchronized with another character's behavior, e.g. back channel feedback ("nodding"), as in the following segment.

```
<par>
 <seq><act agent="Shima" act="Congratulate"/>
  <speak agent="Shima">Let us go now!</speak></seq>
 <seq><act agent="Genie" act="Acknowledge"></seq>
</par>
```

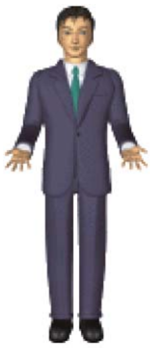


Fig. 4. Contrastive gesture.

```
<!ENTITY % COMMON-CHILDREN
      "(seq | par | page | emotion| execute| wait |
        consult| pause | move | act | speak | think)*">
<!ELEMENT scene %COMMON-CHILDREN;>
<!ATTLIST scene agents    CDATA #REQUIRED>
<!ELEMENT seq %COMMON-CHILDREN;>
<!ENTITY % PAR-CHILDREN
      "((scene| seq | par | emotion| execute| wait | consult|
        pause | move | act | speak | think)*)">
<!ELEMENT par %PAR-CHILDREN>
<!ELEMENT page %COMMON-CHILDREN;>
<!-- The URL of the background page -->
<!ATTLIST page ref        CDATA #REQUIRED >
<!ELEMENT execute EMPTY>
<!-- The address of the function to be executed -->
<!ATTLIST execute target  CDATA #REQUIRED>
<!ELEMENT wait EMPTY>
<!-- The address of the variable used for synchronization -->
<!ATTLIST wait target     CDATA #REQUIRED>
<!-- Halting execution for a duration given in milliseconds -->
<!ELEMENT pause EMPTY>
<!ATTLIST pause pause     CDATA #REQUIRED>
```

Fig. 5. DTD for presentation related elements.

The realization of characters' conversational behavior is restricted by fixed animation sequences (of the Microsoft Agent package) that do no allow for precise timing in synchronization. Another limitation of this package is that multiple characters cannot speak in parallel. However, it is possible to have one character speak while other characters' thoughts are displayed in balloons containing text.

The `execute` element enables the script author to execute a function or method in the background web page, such as a JavaScript function (or Java method, see below). It is typically used to change background web pages. The rationale for the `wait` element is to halt the presentation until the user enters certain information or a media object such as a movie clip stops execution. The `pause` element enables to interrupt script execution for a specified amount of time.

4.2.4. *Tagging structures for interfacing MPML with external objects*

It is sometimes convenient to embed programs written in a different language into a JavaScript based web page, such as a Java applet. The applet becomes an object when loaded into the browser and can hence interact with JavaScript. For instance, André et al. [24] distinguish between *scripted* and *autonomous* character behavior. Whereas a scripted character follows a pre-defined presentation script, an autonomous character decides its behavior depending on its internal mental state. The interface elements in MPML allow for autonomous character control. An extended example of a script that combines scripted and autonomous character behavior will be discussed in Section 6.

```
<!ELEMEMT consult(test+)>
<!-- The address of the variable subject to the test -->
<!ATTLIST consult target  CDATA #REQUIRED>
<!ELEMENT test (seq)>
<!-- The value of the variable -->
<!ATTLIST test value      CDATA #REQUIRED>
```

Fig. 6. DTD for interface elements.

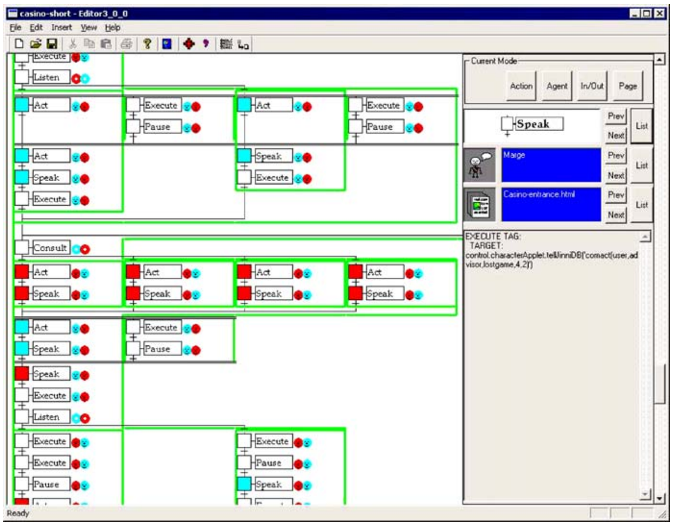
Elements that enable to interface MPML with external objects are summarized in Fig. 6. The `consult` element introduces branching into the otherwise linear presentation script. The value of the `target` attribute is a function of an external object, such as a Java method. Here, the result returned by executing the Java method is tested for string identity with the value of the `value` attribute of the `test` element. When identical, a sequence of events is triggered which corresponds to the chosen branch of the presentation. In Section 6, the operation of those tags will be illustrated by a running example.

5. Working with MPML — the MPML3.0 Visual Editor

Script authors working with MPML may either edit the file containing the MPML tagging structures or manipulate the graphical representation corresponding to the script. Considering the complexity of a script with nested tag structures and the popularity of visual interfaces, working on the graph representation is an attractive alternative to writing tags. However, for convenience, our system supports both types of creating a presentation script that is executable in a web browser. Since scripting complex presentations is an error-prone process, an error handler has been implemented to direct the user to the erroneous parts of the MPML script (see [25]).

We will now introduce a visual editor for presentations marked up with MPML. The MPML3.0 Visual Editor consists of two main windows (see Fig. 7). The window to the left—the (presentation) *Graph* window—shows the graphical presentation of the MPML script, and the window to the right—the *Current Mode* window—displays the current location of user interaction with the graph.

The top field of the Current Mode window provides buttons to insert agent actions to the graph or declare new agents as part of the presentation. The part below of those buttons enables the script author to choose a tag (e.g. `speak`), a character (e.g. "Marge"), and the web page that serves as a background for the characters' performance (e.g. `Casino-entrance.html`). The *Prev* and *Next* buttons allow the user to select the previous or next-listed tagging structure, character, or background pages, while the list buttons show a list of available items. The *List* buttons may also be used to edit the setting of characters, such as their initial position on the screen, or add another background page. The lower part of the Current Mode window depicts the current attribute–value pair of the element whose associated box (configuration)



```
EXECUTE TAG:
  TARGET:
control.characterApplet.tellJinniDB('comact(user,ad
visor,lostgame,4,2)')
```

Fig. 7. The MPML3.0 Visual Editor.

in the Graph window shares the physical location with the current mouse position (i.e. the mouse is 'over' the box). Authors may edit tag elements in the Current Mode window and then drag-and-drop the box associated with the tag at the appropriate position in the graph.

A presentation graph is build up from the following entities. A *node* is displayed as a box configuration with three parts. The rectangle located in the middle refers to a tag element, e.g. `execute` or `speak`. The circle(s) to its right refer to the characters that are visible in the current scene. A circle showing a colored ('Smiley') face denotes the performing character of an action.[2] For convenience of the author, a square box to the left of the rectangle will have the same color (cf. previous footnote) and hence facilitates tracking which character is performing at a node in the graph. The *edges* in the graph can be divided into 'sequential', 'parallel' and 'branching' edges.

- A *sequential edge* is a (down-side) directed arc between two nodes and denotes the next action in the presentation flow.
- A *parallel edge* is a directed arc between a node and a set of nodes each of which initializes a sequence of events, such as actions carried out by multiple agents in parallel (see Fig. 8(a)).
- A *branching edge* is a directed arc between a node and a set of nodes such that the node that satisfies a certain condition initializes a sequence of events. The condition may depend on user interaction or, if autonomous agents are used, the behavior suggested by the reasoning of the character (see Fig. 8(b)).

[2]The non-color view of the editor window shows only different gray shades.

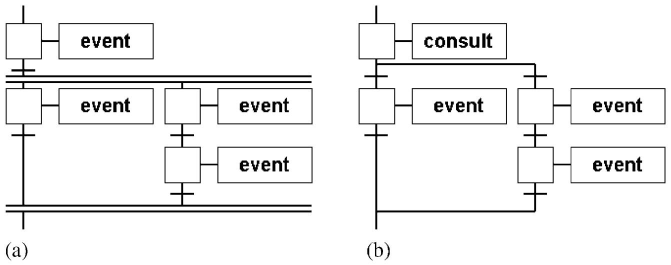


Fig. 8. Examples of a graph with a parallel edge (a) and a branching edge (b). Events can be substituted by characters' actions, speech, loading a web page, etc.

In order to visualize presentation elements connected by different kinds of edges, the presentation graph uses colored frames (cf. previous footnote). The pull-down menus of the MPML3.0 Visual Editor include the standard options of visual interfaces, such as loading and saving a presentation graph or inserting new presentation elements. The toolbar provides buttons to switch between operating on the presentation graph or editing the tagging structures of the corresponding MPML script file, and to launch the presentation, in addition to common tools. In the next section, we will present an extended example of an MPML script. For expository purposes, we will use its tagging structures.

6. Illustration

We will now show how an author may use MPML to script an interactive presentation. As an interaction setting, we will implement a casino scenario where the user and other characters can play the "Black Jack" game. In our game scenario, the table (metaphorically) seats a dealer and two players (see Fig. 9). In the right half of the (Internet Explorer) window, the character "James" is in the role of the dealer. The first seat on the dealer's left (called the 'First Base') is occupied by the user who may interact with the game by uttering one of the sentences displayed in the lower frame window of the (Explorer) window that depicts the current state of the game. To the dealer's right (the 'Third Base'), the character "Al" is in the role of another player. At the bottom-left of the window, the character "Genie" acts as the user's advisor to play the game, and is introduced as "Djinn". In our example, we will focus on scripting a scene where only the dealer and the advisor are present.

The behavior of the dealer and the other player are entirely pre-defined whereas the advisor's behavior is controlled by his mental state. In order to 'script' characters capable of autonomous affective behavior, we have developed the scripting emotion-based agent minds (SCREAM) system, which is extensively discussed in a complementary paper [17]. In short, the SCREAM system is a Java applet that outputs the advisor's answer to the user by accessing a rule-based knowledge base. Most importantly, the SCREAM system decides the emotion of the advisor's

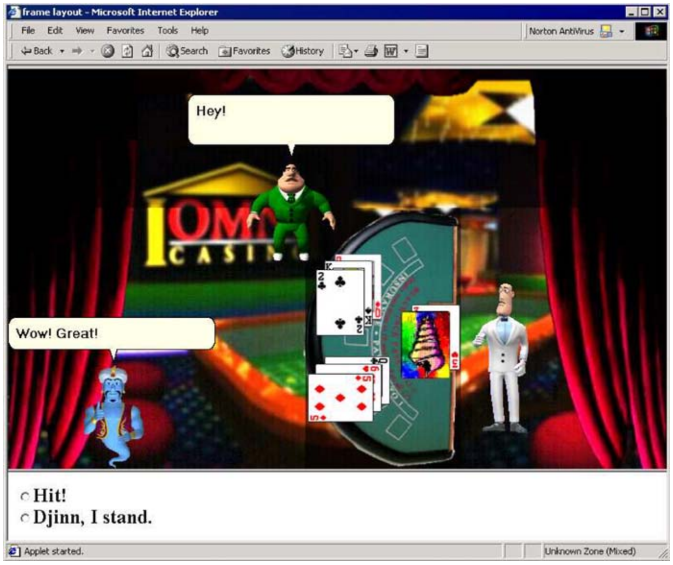

frame layout - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Hey!
Wow! Great!
Hit!
Djinn, I stand.
Applet started.
Unknown Zone (Mixed)

Fig. 9. Casino scenario.

response depending on his mental makeup, the user's choice ("Hit" or "Stand"), and the current game situation. The advisor's mind contains a multitude of parameters, including his beliefs, goals, standards, attitude, personality and peculiarities of the social interaction context. It is important to note that MPML and SCREAM are independent systems. MPML can generate interactive presentations without accessing SCREAM, and the SCREAM system can be integrated to web-based interaction scenarios without using MPML as a character behavior markup language. However, the integration of both systems provides a powerful framework to facilitate both character scripting and context-sensitive affect generation for characters.

Fig. 10 shows a script fragment of the casino scenario that describes the interaction of the user with the advisor and its consequences for the game progress. Emotional markup is omitted for simplicity. In line 1, the character "Genie" is enabled to accept the speech command from the user, followed by a branching edge of multiple alternatives, where part of the first branch is shown (lines 2–35). This branch is chosen when the user utters "Djinn I hit". The user's command has the following consequences. First, in line 5, the bottom frame is 'cleared' by replacing it with a frame window that disables user interaction, denoted by `answer.html`. Next, in line 6, an (embedded) sub-frame window of the main window, `casino-main.html`, is replaced by the new sub-frame window `game3-2.html` that depicts the updated game board state. After Djinn displays the "Uncertain" animation sequence (line 7), he verbally suggests to stand (lines 8–10), and then the bottom frame is being replaced by a sub-frame window (`answer-5.html`) showing a new pair of user choices (line 11). Again, we describe the expansion of only one branch, where

```
1  <listen agent="Genie">
2    <heard value="Djinn I hit">
3    <scene agents="Genie,James">
4      <page ref="casino-main.html">
5        <execute target="control.changebottom('answer.html')"/>
6        <execute target="scene.board.location.href='game3-2.html'"/>
7        <act agent="Genie" act="Uncertain"/>
8        <speak agent="Genie">
9          We have 17 now...That'snot an easy decision,but I would stand.
10       </speak>
11       <execute target="control.changebottom('answer-5.html')"/>
12       <listen agent="Genie">
13         <heard value="Stand">
14           <scene agents="James,Genie">
15             <page ref="Casino-main.html">
16               <par>
17                 <seq><act agent="James" act="GestureRight"/></seq>
18                 <seq>
19                 <execute target="scene.board.location.href='game3-3-win.html'"/>
20                 </seq>
21               </par>
22               <act agent="James" act="Sad"/>
23               <speak agent="James">
24                 The bank gets 24 and looses. Player wins, you're lucky guys.
25               </speak>
26               <execute target="control.characterApplet.tellJinniDB(
27                                       'comact(user,advisor,wongame,4,1)')"/>
28             </page>
29           </scene>
30         </heard>
31         <heard value="Hit">
32           ...
33         </heard>
34       ...
35   </heard>
36   ...
37 </listen>
```

Fig. 10. Example of MPML script.

the user decides to follow Djinn's suggestion. This branch is shown in Fig. 7 after the occurrence of the `listen` tag (lines 13–30). Lines 16–21 code the parallel execution of two actions. The dealer performs the "GestureRight" animation sequence (line 17), thereby demonstrating the new game situation which is simultaneously loaded (line 18–20). Then the dealer verbally and non-verbally expresses his sadness that he lost this round in the game. In lines 26 and 27, the `execute` tag is used to update the advisor's knowledge base telling that the user won the current round, which is internally represented by (round) 4, (choice) 1. Notice that here, MPML interacts with the SCREAM system that will be used to derive Djinn's reaction. The remaining lines 28–37 show some of the required closing tags.

A situation where Djinn receives an instruction on how to respond to the user's choice is depicted in Fig. 11. In line 1, the `consult` tag is used to access the

```
1  <consult target="control.cApplet.askJResComAct('advisor','user','5')">
2    <test value="'rescomact(sorryfor,3,advisor,user,5)'">
3      <scene agents="Genie,James">
4        <page ref="Casino-main.html">
5          <act agent="Genie" act="Uncertain"/>
6          <speak agent="Genie">
7            Oh, you lost.
8            <nb pause="100">I am sorry, this was unlucky.
9          </speak>
10       </page>
11     </scene>
12   </test>
13   <test value="'rescomact(joy,5,advisor,user,5)'">
14     <scene agents="Genie,James">
15       <page ref="Casino-main.html">
16         <act agent="Genie" act="Congratulate_2"/>
17         <speak agent="Genie">
18           And here we are!
19           <nb pause="100"/>We are great today!
20           <nb pause="100"/>And luck is on our side.
21         </speak>
22       </page>
23     </scene>
24   </test>
25   <test>
26   ...
27   </test>
28   ...
29 </consult>
```

Fig. 11. Another example of MPML script.

SCREAM system that processes the user's communicative act and outputs Djinn's current emotional state together with its intensity. For example, the output `rescomact(sorryfor,3,advisor,user,5)` says that the advisor's (i.e., Djinn's) communicative act to the user in situation 5 is 'sorry for' with intensity 3 (see [17] for details). The actual verbal and non-verbal reaction of Djinn is then instantiated within the <`test`>...</`test`> tagging structures. Two of Djinn's responses are given in lines 2–12 and 13–24, respectively. The cardinality of `text` tagging structures following the `consult` tag depends on the granularity of the emotion system underlying the advisor. The graphical representation of the branching edge rooted in the `consult` element can be seen in Fig. 7, where it is located immediately below the occurrence of the `consult` tag approximately in the middle of the Graph window.

7. Discussion

Recent years have seen a growing number of scripting and representation languages for controlling the embodied behavior of synthetic characters. Animating the visual appearance of characters is a difficult task that involves many levels, from changes to each individual degree of freedom in a character's motion model to high level concerns about how to express a character's personality by means of its movements. Beard [26] points out that the main design decision in the construction of a scripting language for synthetic characters is the *level of control* over the character. For instance, a scripting language may cover different (abstraction) levels, such as low-level control over the face, medium-level over the body, and high-level control over (synthetic) voice. Certainly, the desired level of control depends on the application the content author has in mind. While high-level facial control is mostly sufficient for a tutor or presentation character in a technical domain, more detailed (low-level) control might be required for a joke or story telling character.

Given our aim to develop an easy-to-use scripting language for web-based presentations, MPML is not intended as a new character representation language but a convenient tool to control characters that use the Microsoft Agent package as an animation engine. As a scripting language, MPML is located at a high (or abstract) level that allows easy control of pre-defined character motion sequences and their synchronization with synthetic speech. However, the outcome of the synchronization between gestures and speech largely depends on the animation used. Some animations, such as "Acknowledge" ("nod") or "Blink" have very short duration, whereas others ("Announce", "Decline", or "Think") persist during the entire duration of the character's utterance. This simple form of synchronization is sufficient for the application we envision, namely, web-based presentations that rely to a large amount on deictic gestures. A different approach is taken by behavior expression animation toolkit (BEAT) that supports a timeline to accurately synchronize synthetic speech with conversational behavior [9]. More concretely, BEAT uses a pipelined approach where the TTS engine produces a fixed timeline which constrains the subsequently added gestures. However, the sophisticated synchronization of verbal and non-verbal behavior aggravates the real-time rendering of character animation.

Besides genuine character control, another purpose of MPML is to allow for easy integration of life-like characters into web-based presentation scenarios such as interaction with other media objects contained in a web page. This aim is also shared by the PMScript language, which was designed as a language to specify personified media (PM), i.e. avatars that act in online communities [27]. However, since MPML is not a full-fledged environment for web content creation, features not related to character control might be require programming skills.

Systematic experiments with the markup language and the MPML3.0 Visual Editor have yet to be conducted. The visual editor is mostly used by new students whose aim is to work on a topic related to character control. Our (informal) experiences are that both the language and the visual editor are perceived as intuitive and can be used with little instruction. This, however, has to be warranted by principled studies.

8. Conclusion

Life-like characters have recently received a considerable amount of interest as they can be employed in diverse application fields including tutoring, interactive stories, sales and presentation. Despite the popularity of animated agents, tools that would allow content authors to easily integrate this interface technology to web-based presentations are still rare. The purpose of the MPML3.0 Visual Editor for the multi-modal presentation markup language described in this paper is to provide a powerful and easy-to-use character scripting tool for web content experts who are non-programmers.

We have presented the language specification of MPML that contains tagging structures for the control of the verbal and non-verbal behavior of multiple 2D cartoon-style agents, presentation flow control including the possibility of user-agent interaction, and an interface to external web objects, such as Java applets. In order to facilitate the scripting of complex presentation scenarios, we have described the MPML3.0 Visual Editor where content authors can manipulate a presentation graph rather than edit the involved textual script of the presentation. The discussion of an actual web-based presentation implemented with MPML—the interactive casino scenario—was intended to give an idea of how to mark up a presentation that features characters with pre-scripted as well as autonomous behavior. MPML has been set in context by reporting on related markup languages developed in other research institutes and our laboratory. A key advantage of our system in terms of usability is the graphical interface, but this has yet to be proved empirically.

Given the challenging task of building life-like characters that can convincingly assume the role of virtual presenters with interaction capabilities, future avenues for research are manifold. We aim to improve the expressivity and flexibility of web-based interactive presentation characters and keep the specification languages that control them simple enough to be beneficial for the non-technically oriented web content expert. Eventually, we want to model our own characters and implement a visual editor that allows authors to control characters at multiple levels of abstraction.

Acknowledgements

This research was supported by the Research Grant (1999–2003) for the Future Program (“Mirai Kaitaku”) from the Japan Society for the Promotion of Science (JSPS).

References

[1] J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.), Embodied Conversational Agents, The MIT Press, Cambridge, MA, 2000.

[2] Y. Arafa, K. Kamyab, S. Kshirsagar, N. Magnenat-Thalmann, A. Guye-Vuillème, D. Thalmann, Two approaches to scripting character animation, in: Proceedings AAMAS-02 Workshop on Embodied Conversational Agents—Let's Specify and Evaluate Them! 2002.

[3] A. Marriott, J. Stallo, VHML—Uncertainties and problems, A discussion, in: Proceedings AAMAS-02 Workshop on Embodied Conversational Agents—Let's Specify and Evaluate Them! 2002.

[4] XML, Extensible markup language, URL: http://www.w3.org/XML.

[5] M. Bordegoni, G. Faconti, S. Feiner, M.T. Maybury, T. Rist, S. Ruggieri, P. Trahanias, M. Wilson, A standard reference model for intelligent multimedia presentation systems, Computer Standards & Interfaces 18 (6–7) (1997) 477–496.

[6] A. Guercio, T. Arndt, Multimedia computing on the world wide web, Journal of Visual Languages and Computing 13 (2002) 1–2.

[7] N. Bradley, The XSL Companion, Addison-Wesley, London, 2000.

[8] B. De Carolis, V. Carofiglio, M. Bilvi, C. Pelachaud, APML, a mark-up language for believable behavior generation, in: Proceedings AAMAS-02 Workshop on Embodied Conversational Agents—Let's Specify and Evaluate Them! 2002.

[9] J. Cassell, H. Vilhjálmsson, T. Bickmore, BEAT: the behavior expression animation Toolkit, in: Proceedings of SIGGRAPH-01, 2001, pp. 477–486.

[10] A. Kranstedt, S. Kopp, I. Wachsmuth MURML: a multimodal utterance representation markup language of conversational agents, Technical Report 2002/05, SFB 360 Situated Artificial Communicators, University of Bielefeld, 2002.

[11] Z. Huang, A. Eliëns, C. Visser, 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.

[12] N.I. Badler, R. Bindiganavale, J. Allbeck, W. Schuler, L. Zhao, M. Palmer, 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.

[13] Microsoft, Developing for Microsoft Agent, Microsoft Press, Redmond, WA, 1998.

[14] S. Saeyor, Multimodal Presentation Markup Language Ver. 2.2a (MPML2.2a), 2003, URL:http://www.miv.t.u-tokyo.ac.jp/~santi/research/mpml2a.

[15] P. Du, M. Ishizuka, 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.

[16] N. Okazaki, S. Aya, S. Saeyor, M. Ishizuka, A multimodal presentation markup language MPML-VR for a 3D virtual space, in: Proceedings (CD-ROM) of Workshop on Virtual Conversational Characters: Applications, Methods, and Research Challenges (in conj. with HF2002 and OZCHI2002), 2002.

[17] H. Prendinger, S. Descamps, M. Ishizuka, Scripting affective communication with life-like characters in web-based interaction systems, Applied Artificial Intelligence 16 (7–8) (2002) 519–553.

[18] T. Ringate, AIML Reference Manual, A.L.I.C.E AI Foundation, 2001, URL: alicebot.org.

[19] K. Mori, A. Jatowt, M. Ishizuka, Enhancing conversational flexibility in multimodal interactions with embodied lifelike agents, in: Proceedings of Poster Session at International Conference on Intelligent User Interfaces (IUI-03), 2003, pp. 270–272.

[20] XML schema Part 1: structures, URL: http://www.w3.org/TR/xmlschema-1.

[21] I.R. Murray, J.L. Arnott, Implementation and testing of a system for producing emotion-by-rule in synthetic speech, Speech Communication 16 (1995) 369–390.

[22] J. Cassell, Nudge nudge wink wink: elements of face-to-face conversation for embodied conversational agents, in: J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.), Embodied Conversational Agents, The MIT Press, Cambridge, MA, 2000, pp. 1–27.

[23] SMIL, Synchronized multimedia integration language, URL: http://www.w3.org/AudioVideo.

[24] E. André, T. Rist, S. van Mulken, M. Klesen, S. Baldes, The automated design of believable dialogue for animated presentation teams, in: J. Cassell, J. Sullivan, S. Prevost, E. Churchill (Eds.), Embodied Conversational Agents, The MIT Press, Cambridge, MA, 2000, pp. 220–255.

[25] S. Descamps, MPML3.0: towards building a standard for multimodal presentations using affective agents, Master's Thesis, University of Tokyo, January, 2002.
[26] S. Beard, Design decisions underlying virtual conversational character scripting languages, in: Proceedings HF-02 Workshop on Virtual Conversational Characters: Applications, Methods, and Research Challenges, 2002.
[27] T. Takahashi, H. Takeda, Y. Katagiri, Script language for embodied agents as personal conversational media in online communities, in: Proceedings AAMAS-02 Workshop on Embodied Conversational Agents: Let's Specify and Compare Them! 2002.