MPML3D: a reactive framework for the multimodal presentation markup language

Michael Nischt, Helmut Prendinger, Elisabeth André, M. Ishizuka

Angaben zur Veröffentlichung / Publication details:

Nischt, Michael, Helmut Prendinger, Elisabeth André, and M. Ishizuka. 2006. "MPML3D: a reactive framework for the multimodal presentation markup language." In *Intelligent virtual agents: 6th International Conference, IVA 2006, Marina Del Rey, CA, USA, August 21-23, 2006*, edited by Jonathan Gratch, Michael Young, Ruth Aylett, Daniel Ballin, and Patrick Olivier, 218–29. Berlin [u.a.]: Springer. https://doi.org/10.1007/11821830_18.

MPML3D: A Reactive Framework for the Multimodal Presentation Markup Language

Michael Nischt[1], Helmut Prendinger[2],
Elisabeth André[1], and Mitsuru Ishizuka[3]

[1] Institute of Computer Science, University of Augsburg
Eichleitnerstr. 30, D-86135 Augsburg, Germany
Michael.Rudolf.Anton.Nischt@Student.Uni-Augsburg.De,
Andre@Informatik.Uni-Augsburg.De
[2] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
helmut@nii.ac.jp
[3] Graduate School of Information Science and Technology, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
ishizuka@i.u-tokyo.ac.jp

Abstract. MPML3D is our first candidate of the next generation of authoring languages aimed at supporting digital content creators in providing highly appealing and highly interactive content with little effort. The language is based on our previously developed family of Multimodal Presentation Markup Languages (MPML) that broadly followed the "sequential" and "parallel" tagging structure scheme for generating presynchronized presentations featuring life-like characters and interactions with the user. The new markup language MPML3D deviates from this design framework and proposes a reactive model instead, which is apt to handle interaction-rich scenarios with highly realistic 3D characters. Interaction in previous versions of MPML could be handled only at the cost of considerable scripting effort due to branching. By contrast, MPML3D advocates a reactive model that allows perceptions of other characters or the user interfere with the presentation flow at any time, and thus facilitates natural and unrestricted interaction. MPML3D is designed as a powerful and flexible language that is easy-to-use by non-experts, but it is also extensible as it allows content creators to add functionality such as a narrative model by using popular scripting languages.

1 Introduction

Animated characters were quickly accepted as attractive and engaging mediators for effective human-computer interaction, initially for their mere novelty and entertainment value, but recently more and more for their demonstrated benefit as virtual sales agents, tutors, and social interaction partners, among others [14]. However, when integrated into users' daily life as virtual assistants, users will assume high realism and expressivity of the characters, and more importantly, a high level of interactivity and awareness of their need for high-quality information as well as a natural and enjoyable interaction experience.

The bottleneck for digital content providers typically is the lack of appropriate authoring tools to meet those expectations. For this purpose, we previously designed MPML as a support for non-professionals to create affective multimodal content with life-like characters easily, i.e. without assuming knowledge in a scripting language such as JavaScript or a programming language like Java. Depending on the character system used, including 2D and 3D cartoon-style agents and the Honda humanoid robot ASIMO, different versions of MPML emerged [8]. While the MPML family of markup languages shares a common vision (simple authoring) and some core XML tagging structures, most notably for sequential and parallel character behavior execution, versions with specialized functionality were necessitated by the nature of each character system used and its operating environment (web-based, mobile, real world). In addition, the implementation had to be adapted for each character system, by providing a dedicated parser.

Support for authoring scenarios with frequent interaction among characters and users were not an emphasis of previous versions of MPML, while still possible via (heavily) nested "sequential" and "parallel" tags. In response to the high demand in providing interaction-rich scenarios, we will therefore advocate the next generation of MPML-style markup languages, which is based on a reactive rather than a pre-synchronized behavior model. MPML3D allows content authors to define appropriate verbal and non-verbal responses, which are active during the entire presentation and can therefore suspend the scripted part of the presentation at any time. This paper gives a detailed description of and justification for the re-design of MPML with the overall goal of providing an easy-to-use markup language for highly interactive scenarios.

The rest of the paper is organized as follows. The next section motivates the new language by way of our implemented scenario. Section 3 reports on related work. Section 4 describes the system architecture of our reactive model and its new features. Section 5 is dedicated to an overview of the used tagging structures. Section 6 concludes the paper.

2 Motivating Example

As a show-case for the affordances of a highly interactive scenario, we implemented a virtual sales scenario where a team of two 3D animated agents presents MP3 players to a human user. A professional Japanese character designer for "digital idols" created two highly realistic and expressive 3D agents (female and male), based on the appearance of two famous Japanese actors. Each character can perform body and facial gestures (emotional expressions), speak with proper lip-synchronization and direct its gaze at any specified scene entity as well as the user seated in front of the computer display screen (see Fig. 1).

The story line is that two characters present two MP3 players developed by their enterprise. At the beginning, one character is in favor of a slim, easy to use version and the other one prefers the high-end product with an unparalleled

Fig. 1. Presentation of MP3 players by two Japanese-style 3D characters

feature set. During the presentation they realize, that the gadget presented by the other character would fulfill their particular needs better than what they have presented themselves. Both are startled by that, and try to figure out which player would actually be most attractive to the user. Note that the first part of our presentation is non-interactive and does not put high demands on a markup language. It can in principle be easily dealt with by using available synchronization languages like SMIL [18] and MPML [13,8].

However, in order to increase the level of interactivity and engagement both between characters and between the characters and the user, each character should have the capability to perceive the actions of its counterpart character and the user. This capability is fundamental to lively and engaging human-human communication where participants continuously adapt to the interlocutor's state of attention and emotion, and react accordingly. The new markup language thus provides means for continuously (i) informing each character about the current state of action execution of the other character, and (ii) informing the characters about the state of the user. We handle inter-agent communication and feedback by message-passing, and introduce eye tracking as an input modality to recognize the user's state of attention and interest. Analyzing gaze behavior (or "eye gestures") is a powerful method to detect a user's interest and preference among alternatives [4]. We also allow for simple speech input from the user, following a keyword spotting approach. The second part of the MP3 player presentation thus aims at a high level of interactivity among the participants of the scenario (two characters and one user) and their mutual awareness. This kind of interaction cannot be easily handled by currently available markup languages since it assumes that characters may respond to the other agents (human and virtual) at any time.

3 Related Work

Within the past five years, a range of markup languages and associated technologies were developed in order to direct the verbal and non-verbal behavior of animated agents. The Character Markup Language (CML) puts an emphasis on gesture behavior and its modulation by the agent's emotional state and personality [3]. A similar focus is present in the Parameter Action Representation (PAR), which allows one to specify various action parameters such as purpose, duration, and manner that can be modulated by affect related parameters [1]. While providing effective means to express affect and associated character movement, those languages were designed with professional animators rather than non-expert content authors in mind. The Affective Presentation Markup Language (APML) targets the representation of communicative functions between an agent and a user that may contain the speaker's belief state (certainty of utterance) and intention (request, inform) as parameters [5]. Communicative behavior is also the underlying motivation for the Behavior Expression Animation Toolkit (BEAT) that proposes an elaborate mechanism for the accurate synchronization between speech and conversational gestures of a character [6]. The fine-grained control of parallel and sequential components of gestures was also the motivation behind the development of MURML [9].

All of the above mentioned systems are restricted to synchronizing the behavior of a single character. An interesting approach to authoring multi-character interactions has been suggested in the Inhabited Market Place (IMP) system that creates presentation dialogues automatically by employing a central planner [2]. Character-specific dialogue contributions (e.g. elementary presentation acts) constitute leaf nodes in the decomposed hierarchical plan tree. The IMP system assumes that appropriate STRIPS-style plan operators have been defined, and hence might be an infeasible approach for non-scientists. The system has also been extended to include the user as part of the conversation (and de-centralized planning as a further option), as in the MIAU system [16]. Nevertheless, creating reactive behaviors within MIAU requires basic knowledge of planning formalisms. The Rich Representation Language (RRL) has been developed to specify the interaction between two or more virtual agents [12]. Its use requires less training effort than, for example, the planning mechanism employed in IMP; however, RRL does not deal with anytime user interactions.

There are only a few approaches that explicitly deal with reactive agent behaviors. BEAT accounts for time-line based as well as reactive gesture generation. Nevertheless, reactivity rather refers to events triggered by the speech synthesizer than to user interruptions. Similarly, STEP [17] includes interaction operators to deal with the environment in which the movements and actions take place. ABL (A Behavior Language) is a highly sophisticated language to coordinate multiple characters while being reactive to user input, with the core goal of creating compelling dramatic experiences [10]. Despite its potential for creating highly interactive presentations, generating behaviors with ABL is close to programming in Java, which likely exceeds the skill level of the average content creator. The system most closely related to MPML3D is SceneMaker, a

toolkit for composing interactive performances between life-like characters that are adaptive to user actions [7]. The scene flow is realized by a finite state machine, whereby both nodes and transitions can have playable scenes associated with them. A scene is a pre-scripted dialogue (of variable size) between two characters. Like the MPML3D system, SceneMaker targets non-expert content providers. A major distinction between the two systems is that content representation in SceneMaker is scene-based, whereas in MPML3D, each character has its own representation of possible actions (and their conditions), which is a prerequisite for the desired reactive behavior.

4 Multimodal Presentation Markup Language Based on a Reactive Model

In this section, we propose a major re-design of the MPML language based on the following core observation. Multi-character applications with frequent "barge-in interruption" from the user by speech, gesture, or even gaze and physiological activity will become the rule rather than the exception. MPML as currently defined is either not able to handle such situations (e.g. continuous speech) or cumbersome, as in the case of input-dependent branching. In the following sections we will first describe the new system architecture and we will then explain the new features of MPMP3D.

4.1 MPML3D System Architecture

An overview of the MPML3D system architecture is shown in Fig. 2. Its main components are the *user* and *developer layers*, and the *animation engine*, for which accessibility from the developer layer has to be created.

The main modules can be described as follows. The *user layer* is dedicated to creating the content for interactive presentations. It defines the Schema for the

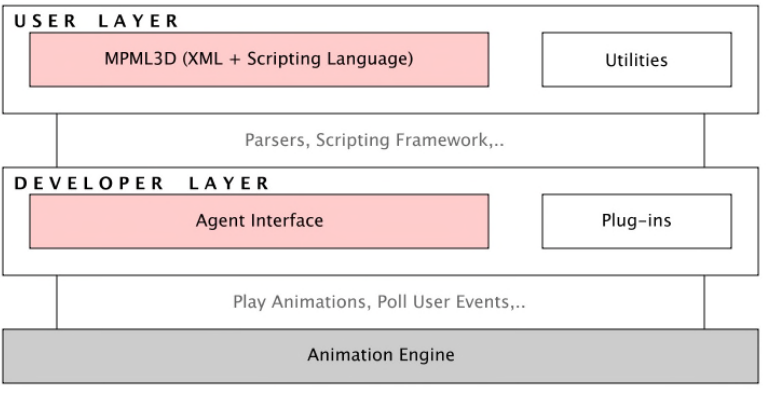

Fig. 2. System architecture

XML-based MPML3D format, to which each instance must conform. In order to maintain a high level of flexibility, statements of a scripting language can be embedded into the markup language. Currently, we only use Javascript (as used in previous MPML versions), but we also envision to support other popular languages like Ruby, Python or Groovy in the near future. Those can be either utilized to access the functionality provided by the developer layer, or to use self-developed Java objects (utilities), e.g. ones that encapsulate emotion or social behavior models.

The *developer layer* can be seen as an intermediate layer, uncoupling the user layer from particular implementations. It is a simple Java API, providing base classes for an agent's actions and its perception capabilities. Although we already provide a basic set of classes, new ones can be easily integrated through the plug-in architecture. By employing annotation and reflection features as offered by languages such as Java or C#, new actions and perceptions can be used in MPML3D without changing the parser. Moreover, this approach allows editors to recognize the plug-ins automatically no matter whether these editors are text based, just provide an auto-completion facility, or are completely graphical editors like the MPML3.0 Visual Editor described in [13].

The *animation engine* itself is not part of the MPML3D system. It is responsible for manipulating a character's internal state and rendering it, and also has to process the user inputs such as gaze behavior or speech. Popular animation engines include game engines, X3D and MPEG4 players, and APIs for mobile phones or physical robots. Currently, we use a self-created system that renders the animated scene using OpenGL and OpenAL. Here, both the environment and the characters are directly transferred from digital content creation tools (Autodesk's 3DS Max or Maya) to the application (see [11] for details).

4.2 New Features of MPML3D

Today's character-based applications demonstrate an increased demand for interactivity and awareness of user input modalities. Although previous versions of MPML were capable of dealing with user input by using conditional statements, the resulting scripts were deeply nested and became cumbersome to extend and read. Specifically, MPML was not designed for anytime interruption of a character's behavior when the user started speaking (or provides other input, e.g. a particular gaze behavior) or when agents interrupt each other.

The issue here is to solve the problem of permanent attendance to react to all possible inputs from the user or another agent. Since previous versions of MPML represent interactive presentations as a branched structure, one had to test all possible user inputs after every action tag and create new character responses for each of those conditional nodes, resulting in an enormous out-degree (if frequent interaction was desirable). In order to overcome this limitation in MPML, the new MPML3D language changes the organization of the presentation to a *reactive model*. In particular, an agent's behavior is determined by its perception that may trigger and interrupt actions, clustered into tasks (i.e. sequences

of actions). This not only simplifies the authoring of a scenario, but also allows one to reuse attentive behaviors across different presentations and scenarios.

The design of a reactive model deviates from the design of MPML that was derived from markup languages that govern the behavior of multiple processes essentially by sequential and parallel execution, e.g. the popular Synchronized Multimedia Integration Language (SMIL) [18], and thus deserves extra justification. As argued in [15], the similarity of markup languages for characters to "easy to understand" languages such as HTML might not be of primary importance since appropriate easy-to-use editing tools will eventually be available for them, and more importantly, those languages might fail to handle the complexity of interaction among agents (including human agents) as observed in human-human face-to-face communication. While the "parallel" tag used in current markup languages assumes independence of behavior, natural communication demonstrates mutual adaptation between speaker and listener (sometimes called "alignment"). For instance, when noticing undesired effects of their utterance in the listener's facial expression, such as irritation, speakers often take counteractions *while* speaking, by adapting e.g. the politeness level of the utterance. These observations suggest an approach like MPML3D that is based on actions *and* (continuous) perception.

The transition from a pre-synchronized model to a reactive model obviously assumes a revision of other aspects of the language as well, specifically the representation of agent actions. In contrast to former versions, where the script for all characters was defined in a single file, the behavior of each agent is now contained in one dedicated MPML3D file, which is seen as the behavior space of that agent. The advantage of such a distributed architecture combined with a reactive model for executing the agents' tasks regards the greater flexibility in adding additional agents to the presentation. Note that individual actions of the same character, e.g. starting a gesture when a specific word is spoken, are synchronized in its own MPML3D file. This approach is not restrictive since a task, consisting of such an arrangement, can be started, interrupted and resumed upon the perception of actions performed by other agents, as well as user input.

The design principles put forth for the MPML family [13,8] also apply to the new MPML3D language, i.e. *Ease of Use* (*Intelligibility*), *Extensibility* (*Accessibility*), i.e. provision for embedded scripting statements for accessing e.g. Java classes from code libraries, and *Easy Distribution.*

5 MPML3D Tagging Structures

In this section we will briefly introduce the tags being used in MPML3D. Like in previous versions, the document is divided into a `Head` and `Body` part after the root element noted as `MPML3D`. The `Head` element specifies general information through HTML-like Meta tags that can be used to define the units for the documents. For instance, the preferred distance measure may be either meters or feet and angles could be defined in radians or degrees, depending on what the author is most familiar with. Furthermore, the character's action and perception

capabilities must be defined here, no matter whether those are included in the standard set or are new ones available through the plug-in mechanism of the developer layer. This is done using the `Extension` tag, which also defines the action class, which can be referred in the `Body` part. Here are two examples:

```
<Extension name="Speak" type="action" class="mpml.Speak"/>
<Extension name="Listen" type="perception" class="mpml.Listen"/>
```

Finally, the `Head` tag can contain a single `Script` node, defining the scripting language. It either references an external file or lists the source code directly. Note that in contrast to JavaScript embedded in HTML, the code is never written as a comment. Although MPML3D authors can create presentations entirely without using an embedded scripting language, the potential benefit is significant. The defined functions can be used as event listeners in order to check conditions for execution, but also every defined `Task`, `Action` and `Perception` given an identifier is exposed to the script. This allows the author to access every public property and method defined in the Java objects of the developer layer. An example excerpt for an event-listener deactivating a task with `id` 'task1' is:

```
<Script language="js">
   function anEventListener() { task1.active = false; }
   .. // other variable, function and object declarations
</Script>
```

The `Body` part defines the presentation flow. In MPML3D, this is done through a list of `Tasks`, which are to be performed but can be interrupted by certain `Perceptions`. A `Task`, on the other hand, corresponds to a list of `Actions` along with instructions when and how to execute them. Consequently, these three elements can be seen as the main components of MPML3D. It is important to notice that all of them are temporal constructs, i.e. they have a beginning and ending state. Therefore, the user-code defined in the `Script` tag can be associated with the optional node attributes `onBegin` and `onEnd`.

We now turn to describing properties of these three tags. We begin with describing the `Task` tag, since it contains the others. Besides the events `onInterrupt` and `onResume`, which are fired when the task is interrupted or resumed afterwards, there are a few other attributes as shown below – all being optional.

```
<Task id="task1" priority="100" active="true" once="true" token="token1">
..
</Task>
```

As mentioned before, an `id` can be assigned to a task to expose it to the script engine. If none is specified, the corresponding Java object is not accessible. The next attribute, called `priority`, defines how urgent a task is. It is used in the default selection mechanism for the task that could be performed. Only a task of higher priority can interrupt another. The default value is zero. The next two attributes (`active`, `once`) are closely related; their specified (default) values

cause a task to be active at the beginning and deactivated after a successful execution. Finally, a task can expose a public `token` while performing. Since this token can be perceived by other characters, it is a convenient way to synchronize the behavior of multiple characters in non-interactive parts of the presentation, which was handled by the "par" tag in previous versions of MPML. By default, a task is not visible to other characters.

Inside the `Task` element we provide (among others) the two command tags, called `Perform` and `Interrupt`, as children. Both can contain perceptions upon which the command is triggered, but before describing those, we will explain their attributes.

```
<Perform interrupt="false" condition="testPerform">
..
</Perform>
<Interrupt resume="false" condition="testInterrupt">
..
</Interrupt>
```

If the `interrupt` attribute is set to `true`, the priority of the task is compared to the one of the currently executed task (if there is any) to decide whether it can be interrupted. Without the attribute, or the value set to `false`, the task will compete for execution with other non-performing tasks depending on their priority only. The `resume` attribute of the `Interrupt` node determines whether the task should be resumed after it is interrupted. Finally, the `condition` attribute can occur within either tag. If present, its value is evaluated along with the perception of the child element by calling script functions. If it evaluates to `false`, the command is not executed.

As stated above, the commands `Perform` and `Interrupt` contain perceptions that may trigger them. A perception belongs to specified `class`. This is not only used to classify the perception, but also defines a possible list of properties which can be declared to narrow down the potential set of user input. Since the character's action and perception capabilities strongly depend on the underlying animation engine, the MPML3D format does not require any specific classes. However, we hope that there will be a standard set established in the future that allows to reuse of presentation parts across individual applications.

Meanwhile, we have defined a few of them that match the requirements of our application scenario described in Sect. 2.

```
<Perception class="Listen" target="NaomiWatanabe" event="end">
 <Property name="text">*music collection*</Property>
</Perception>
```

In this example, the perception is processed as soon as an utterance containing the phrase "music collection" is spoken, but the enclosed command is triggered only when the sentence is finished ("$*\ldots*$" refers to a 'wildcard' or 'regex' construct). In order to change this, one simply has to change the `event` attribute to `begin`. Furthermore, the speaker must be named 'NaomiWatanabe', which is specified by the `target` attribute. Note that by not defining this attribute, any speaker would trigger the command.

In case that more than one observation is required to trigger a command – e.g. in order to have some other character direct its gaze to a specified object, a character might point to it and also refer to it verbally – multiple `Perception` tags can be placed in a node named `All`. By setting its single attribute `order` to `true`, the observation must be in the specified sequence to trigger the command. Finally, the related `Any` tag allows to execute the command if at least one child element is perceived.

Finally, the MPML3D content author has to define the sequence of actions to be executed while performing the task. Due to their generic structure `Action` nodes and their children have a similar syntax to the `Perception` tag. As shown in the example below, all actions are enclosed by the well known `Sequential` and `Parallel` tags. Observe that when considering the behavior of a single character (rather than the behavior of multiple characters), the meaning of those tags corresponds exactly to the meaning in previous versions of MPML.

```
<Sequential>
 <Action class="gesture">
     <Property name="type">BowVeryPolite</Property>
 </Action>
   <Parallel>
       <Action class="speak">
       <Property name="text">Hi, my name is Naomi Watanabe.</Property>
       </Action>
       <Action class="focus">
       <Property name="target">User</Property>
         <Property name="angle">-5.0</Property>
       </Action>
   </Parallel>
</Sequential>
```

According to this example, the character first performs a (very polite) bow gesture and then starts introducing itself by the specified sentence. Concurrently to speaking, the character directs its gaze to the location slightly beside the user (whose exact position can be determined by the eye tracker).

6 Conclusions

This paper describes and justifies a major re-design of the Multimodal Presentation Markup Language (MPML) [8] that was successful in providing non-expert digital content creators an easy-to-use tool for authoring multimodal content with life-like characters. However, MPML was not designed to accept frequent or continuous input from either other characters or the user. Since interaction-rich scenarios are of considerable interest for engaging and natural communication with characters, we propose our new MPML3D language that provides perception (of the behavior of other characters and the user) as a key functionality. The transition from an essentially pre-synchronized model (MPML) to a

reactive model (MPML3D) enables adaptation of character behavior whenever trigger conditions are met throughout the interactive presentation.

The characters in the application scenario described in this paper are able e.g. to attend to the gaze behavior of the user by processing data from a non-contact video based eye tracker. If a relevant gaze pattern is detected, the characters will respond accordingly. For instance, if a user shows (visual) interest in any one of the two MP3 players, the character assigned to promote this product will display happiness about the user's interest, and provide more detailed product information, or even interrupt the other character in its presentation when not currently holding the turn.

MPML3D provides the technology for highly interactive presentations, but currently, resuming a presentation after user interruption is handled in an ad hoc fashion; for instance, incomplete tasks are resumed where they were halted by the interaction. However, it is important to emphasize that typically, interaction (including interruption) between characters is intentionally inserted by the content author to increase the liveliness of the conversation and a resumption strategy is declared. In order to guarantee the overall cohesion and attractiveness of the presentation, a (interactive) narrative model might be added. We also consider adjusting the virtual camera in a gaze-contingent way, e.g. by zooming into the screen area that corresponds to the user's point of interest, or based on some cinematographic principles. MPML3D can be extended by those functionalities in a transparent way while preserving its core purpose as an easy-to-use and powerful authoring language for digital content creators.

Acknowledgments

The first author was supported by an International Internship Grant from NII under a Memorandum of Understanding with the Faculty of Applied Informatics at the Univ. of Augsburg. We would like to thank Dr. Ulrich Apel (NII) for scripting the dialogues and Nikolaus Bee (Univ. of Augsburg, NII) for creating the speech files. The research was supported by the Research Grant (FY1999–FY2003) for the Future Program of the Japan Society for the Promotion of Science (JSPS), by a JSPS Encouragement of Young Scientists Grant (FY2005–FY2007), and an NII Joint Research Grant with the Univ. of Tokyo (FY2005).

References

1. J. Allbeck and N. Badler. Representing and parameterizing agent behaviors. In Prendinger and Ishizuka [14], pages 19–38.
2. E. André, T. Rist, S. van Mulken, M. Klesen, and S. Baldes. 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.
3. Y. Arafa, K. Kamyab, and E. Mamdani. Towards a unified scripting language. Lessons learned from developing CML & AML. In Prendinger and Ishizuka [14], pages 39–63.

4. N. Bee, H. Prendinger, A. Nakasone, E. André, and M. Ishizuka. AutoSelect: What You Want Is What You Get. Real-time processing of visual attention and affect. In *Tutorial and Research Workshop on Perception and Interactive Technologies (PIT-06)*. Springer, 2006. In press.
5. B. D. Carolis, C. Pelauchaud, I. Poggi, and M. Steedman. APML: Mark-up language for communicative character expressions. In Prendinger and Ishizuka [14], pages 65–85.
6. J. Cassell, H. Vilhjálmsson, and T. Bickmore. BEAT: the Behavior Expression Animation Toolkit. In *Proceedings of SIGGRAPH-01*, pages 477–486, 2001.
7. P. Gebhard, M. Kipp, M. Klesen, and T. Rist. Authoring scenes for adaptive, interactive performances. In *Proceedings of 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, pages 725–732, New York, 2003. ACM Press.
8. M. Ishizuka and H. Prendinger. Describing and generating multimodal contents featuring affective lifelike agents with MPML. *New Generation Computing*, 24:97–128, 2006.
9. S. Kopp, B. Jung, N. Lessmann, and I. Wachsmuth. Max – a multimodal assistant in virtual reality construction. *KI Zeitschift (German Magazine of Artificial Intelligence), Special Issue on Embodied Conversational Agents*, 4:11–17, 2003.
10. M. Mateas and A. Stern. A Behavior Language: Joint action and behavioral idioms. In Prendinger and Ishizuka [14], pages 19–38.
11. M. Nischt, H. Prendinger, E. André, and M. Ishizuka. Creating three-dimensional animated characters: An experience report and recommendations of good practice. *Upgrade. The European Journal for the Informatics Professional*, VII(2), 2006, pages 36–41.
12. P. Piwek, B. Krenn, M. Schröder, M. Grice, S. Baumann, and H. Pirker. RRL: a rich representation language for the description of agent behavior in NECA. In *Proceedings AAMAS-02 Workshop on Embodied conversational agents—let's specify and evaluate them!*, 2002.
13. H. Prendinger, S. Descamps, and M. Ishizuka. MPML: A markup language for controlling the behavior of life-like characters. *Journal of Visual Languages and Computing*, 15(2):183–203, 2004.
14. H. Prendinger and M. Ishizuka, editors. *Life-Like Characters. Tools, Affective Functions, and Applications*. Cognitive Technologies. Springer Verlag, Berlin Heidelberg, 2004.
15. T. Rist. Issues in the design of scripting and representation languages for life-like characters. In Prendinger and Ishizuka [14], pages 463–468.
16. T. Rist, E. André, S. Baldes, P. Gebhard, M. Klesen, M. Kipp, P. Rist, and M. Schmitt. A review of the development of embodied presentation agents and their appication fields. In Prendinger and Ishizuka [14], pages 377–404.
17. Z. Ruttkay, Z. Huang, and A. Eliens. Reusable gestures for interactive web agents. In *Proceedings of the 4th International Working Conference on Intelligent Virtual Agents (IVA-03)*, pages 80–87. Springer LNAI 2792, 2003.
18. SMIL. Synchronized Multimedia Integration Language. URL: `http://www.w3.org/AudioVideo`.