MPML3D: Scripting Agents for the 3D Internet

Helmut Prendinger, Member, IEEE, Sebastian Ullrich, Arturo Nakasone, and Mitsuru Ishizuka, Member, IEEE

Abstract—The aim of this paper is two-fold. First, it describes a scripting language for specifying communicative behavior and interaction of computer-controlled agents (“bots”) in the popular three-dimensional (3D) multi-user online world of “Second Life” and the emerging “OpenSimulator” project. While tools for designing avatars and in-world objects in Second Life exist, technology for non-programmer content creators of scenarios involving scripted agents is currently missing. Therefore, we have implemented new client software that controls bots based on the Multimodal Presentation Markup Language 3D (MPML3D), a highly expressive XML-based scripting language for controlling the verbal and non-verbal behavior of interacting animated agents. Second, the paper compares Second Life and OpenSimulator platforms and discusses the merits and limitations of each from the perspective of agent control. Here, we also conducted a small study that compares the network performance of both platforms.

Index Terms—Artificial, augmented, and virtual realities, Graphical user interfaces, Synchronous interaction, Visualization, Markup languages, Scripting languages.

1 INTRODUCTION

METAVERSEs as first envisioned in the scifi-novel “Snow Crash” [1] are manifesting themselves nowadays as online virtual worlds [2] and are becoming increasingly popular [3]. While such environments are mostly used for entertainment and business purposes, they recently also started to attract the attention of the scientific community [5], [6], [7], [8]. (In this paper, the terms “metaverse” and “virtual world” will be used interchangeably.)

Second Life (SL) is a prominent example of such a virtual online world [9]. SL provides a free networked multi-user three-dimensional (3D) environment and is very popular with an increasing amount of registered users (over 16.8 million as of May 2009) and about 40,000 to 60,000 users online at any time. The main features of SL are:

1) Support of social interactions between user avatars (called SL “residents”) via the Instant Messaging interface.
2) Support for user-created content.
3) Economy with a marketplace and own currency, called “Linden dollars” that convert to US dollars.

Users of SL can design their own objects, such as buildings, vehicles, or even entire ecosystems, and upload this content to their privately owned virtual locations – ‘parcels’ of an ‘island’ (a land unit in SL), or entire islands – which are open to the SL community.

OpenSimulator (OpenSim) is an open source project aiming at the creation and deployment of virtual worlds [10]. Since the beginning of 2007, it is being developed under the BSD (Berkeley Software Distribution) license. The goal of the originators is to provide an open and extensible platform, which can be run on virtual worlds creators’ own servers, rather than on servers of Linden Lab (the company running SL). Otherwise, the motivations, goals and challenges of OpenSim are quite similar to those of Second Life.

Second Life and OpenSim have also attracted the attention of companies like Intel and IBM, where those virtual worlds are promoted as first examples of the coming 3D Internet. To emphasize the vision of a network of interconnected 3D virtual worlds, rather than web pages, we reserved “3D Internet” to denote the target domain of our scripting language.

Besides contributions from interested individuals, many institutions – both commercial and academic – have recently opened their presence in Second Life. It was soon noticed, however, that simply building a visually impressive place is not sufficient for an attractive presence in an inherently social space like a metaverse. (For that reason, some companies closed their presence in SL after the initial hype was over.) The key to the success of an island is to provide visitors an interactive and entertaining experience. Some islands demonstrate a high level of interaction for the simple reason that many avatars gather there. Yet, many islands are much less frequented and can give an uncomfortable (lonely) feeling to the visitor.

An obvious solution to attract visitors to islands in SL and OpenSim is to populate these virtual places with “bots”, computer-controlled virtual agents, which may play the roles of guides, receptionists, guards, or other visitors of the island. Note that in the gaming world, bots are rather called NPCs (Non-Player Characters),
and bots should not be confused with avatars, which are controlled by users. In computer games, avatars are called “characters” [2]. It is rather surprising that bots are currently almost completely missing from SL and OpenSim. We can only speculate about the reason: since bots have to be programmed in the C# language, common content creators of metaverses might lack the skill to specify the behavior of bots.

Therefore, in order to support non-computer science professionals, we have developed an XML-based agent authoring language for virtual worlds, based on our previously developed versions of the Multimodal Presentation Markup Language (MPML) family [11], [12], [13], [14], [15]. In this paper, for the first time, we will present the complete scripting language schema of MPML3D for Second Life, which is freely available (since Dec. 2008) as an open source project for content creators [16] and developers [17]. In order to keep our discussion as relevant as possible, we also compare Second Life to the emerging open source OpenSimulator platform.

The paper is structured as follows. In Section 2, we will briefly describe the history of our MPML authoring language and review related work on markup languages for virtual animated agents. Section 3 reports on the technical background, benefits and limitations of SL and OpenSim. Section 4 covers a description of our MPML3D framework. First, we show how the system architecture has been redesigned in order to support the requirements of supporting different target platforms. Then we provide information about the integration into the two virtual world systems of SL and OpenSim. Section 5 is dedicated to describing the tagging structures of MPML3D. We provide the complete XML Schema, and demonstrate the expressiveness of MPML3D by using examples of an online video clip. In Section 6 we will compare implementation details and capabilities of SL and OpenSim. Additionally, network speed has been measured and will be discussed. Finally, in Section 7 we will summarize the results and conclude the paper.

2 BACKGROUND AND RELATED WORK
A variety of markup languages has been proposed for behavior planning of animated agents, or life-like characters (see [18], [19], [11], [20], [13] and other papers in a collection dedicated to scripting life-like characters [21]). We first describe the development of our own markup language (MPML) from a historical perspective, and then report on similar efforts on scripting animated agents.

2.1 A Short History of the Multimodal Presentation Markup Language
We have been developing the Multimodal Presentation Markup Language (MPML) since 1999 [22], initially for the 2D cartoon-style characters of the Microsoft Agent package [23]. A milestone has been the MPML version described in [11], which provided rich control functionality and a visual editor for multimodal content creation with 2D agents. The language has been extended to include affective behaviors and was ported to other platforms, including animated agents in cellular phones and the HONDA Asimo humanoid robot [12]. In 2006, MPML was completely redesigned [13]. Conditional branching based on “sequential” and “parallel” structures was abandoned, and task selection management was introduced to support interaction-rich scenarios with reactive agents that ‘perceive’ their environment. Reflecting the emphasis on 3D agents, the language was henceforth called MPML3D. Based on the highly natural 3D models from a Japanese artist specialized in digital idols (see Fig. 1), we added facial emotion expression and smooth movement [24], and also gesture parametrization, “interruptibility” of behaviors and gesture blending [25]. An application based on our agents won the best application award of GALA (Gathering of Animated Lifelike Agents) in 2006 [26].

However, the MPML3D authoring language could not be distributed effectively because it was strongly connected to our original 3D agents (Fig. 1). Thus, we decided to extend and adapt MPML3D to Second Life, because in this virtual world users can easily create their own avatars and a scripting language like MPML3D could be used by any SL resident. A preliminary and partial language definition of MPML3D for Second Life is described in a short paper [14]. Another short paper compares the SL and OpenSim platforms [15].

2.2 Scripting Virtual Agents
One of the earliest scripting languages is the Virtual Human Markup Language (VHML) [19], an XML-based language aimed at directing a ‘Talking Head’. It addresses the control of various aspects of human–computer interaction, including facial animation, body animation, and dialogue management. The language also provides
controls for speech, emotion, and gesture. For instance, if an utterance should be delivered in a cheerful manner, the smile tag can be used to produce the appropriate prosodic and behavioral effect. Unlike our markup language, which emphasizes multi-agent interaction and SL, VHML is directed towards single facial agents and web-based environments.

The Behavior Markup Language (BML) refers to a broader effort in controlling communicative channels of virtual agents [20], [27]. BML extends and supersedes markup languages that were previously developed by BML project members, including BEAT (Behavior Expression Animation Toolkit) [18], MURML (Multi-modal Utterance Representation Markup Language) [28], APML (Affective Presentation Markup Language) [29], and RRL (Rich Representation Language) [30]. The BML project aims to develop a representation framework for describing both non-verbal and verbal real-time behavior that is independent of the particular graphical realization. BML is an XML-based language that supports the description of behaviors at different levels of detail, e.g., by embedding a more detailed behavior specification into the BML tagging structure. On the other hand, BML itself can be embedded in another XML document. The development of the BML language involves the majority of research groups in embodied conversational agents (ECA). Hence, many projects use BML, including ECA systems, behavior planners and realizers, and tools (see [27] for details). The vision of BML is to develop a general platform that allows to combine behavior planners and behavior realizers easily. In contrast to the BML vision, the goal of MPML3D is more specific. MPML3D is developed only for multi-user virtual worlds like SL in mind, and tries to provide support for agent-based content creation in that environment. Besides this, BML and MPML3D share many features, such as setting synchronization points and trigger conditions.

Significant research has been conducted on the animation of characters and their interaction with objects in a virtual environment (see e.g. [31], [32], [33], [34], [35]). E.g., the scenario language introduced in [35] addresses the control of semi-autonomous agents. It targets the temporal management of behaviors based on James Allen’s temporal logic [36], which defines thirteen possible relations between two intervals A and B, such as A equals B, A before B, A meets B, A overlaps B, A during B, and so on. The scenario language supports the hierarchical description of scenarios and scheduling at simulation time. The scenario language is built on C++ and can embed C++ code. Although the availability of programming constructs and instructions adds significantly to the expressivity of this language, it limits the use for average content authors, who typically have only little programming background. Our MPML3D language can produce most of Allen’s temporal relations, albeit in a different way, which is expected to be more intuitive for common content creators (see Sect. 5.2.1). Notably [37] is one of the earliest works to mention Second Life avatars.

It is important to note that we do not claim to advance the field of virtual worlds or agent animation. On the contrary, the fidelity of agents (bots) in SL or OpenSim is significantly lower than the state-of-the-art in animation research (e.g. [34]). Furthermore, the perception capabilities of SL agents cannot compete those of agents in earlier work (e.g. [33]). Instead, we aim to develop a markup language that can fully exploit the current state-of-the-art in technology for popular online virtual worlds like SL and OpenSim. Unlike online game worlds, programming interfaces are available and an online developer community exists. In our view, it is those features that make the development of a dedicated markup language an important and valuable endeavor.

There are only few scripting languages that are directed for use in recent online virtual worlds or meta-verses. One example is an authoring language that integrates the previously discussed Behavior Markup Language into EVE Online, a massively multiplayer online role-playing game (MMORPG) [27]. This feature has been announced by the game producers in 2006 [38], but appears to be still under development. It will allow players to interact with autonomous agents and to automate coordination of non-verbal social behavior. It is an official addition to the game by the developers themselves. Because of the space setting of the game and the closed source game engine, however, content creators are not able to create their own custom scenarios in this online world.

The work described in [39] focuses on the evaluation of social behavior with a simple agent in SL. The agent is driven by LSL (Linden Scripting Language) and is mainly used for data logging and traversing through SL. A very basic type of navigation has been implemented (walking in random directions until either an obstacle or an avatar is found). Furthermore, the agent is able greet avatars by their names and perform gestures. A more sophisticated approach to crowd simulation and collision avoidance using SL bots is introduced in [40]. The authors use libsecondlife (an unofficial API for SL, which has been superseded by libOpenMetaverse [41]) to implement a motion controller and direct bots in the virtual world.

In summary, we have not found any authoring language that would support content creators to easily script agents (bots) for a widely used multi-user 3D environment. Until now, there seems to be only one, very specialized approach to scripting agents in SL [39].

3 Second Life and OpenSimulator

In this section we will give an overview of the technical details of Second Life and the OpenSimulator project. We will describe the infrastructures, the programming interfaces, and supported entities and resources. These details are essential for obtaining a better understanding of the capabilities and limitations of either platform.
3.1 System Setup

SL is a client/server application for multiple networked users. To use SL, one needs to register an account for an avatar (graphical self-representation of the user). Similar to HTML, with a web server and browser application, the 3D content is hosted on cluster network servers and streamed to the client application. However, the crucial difference is that content in SL is protected by a digital rights management system. This encourages users to have their own virtual property, to create new 3D models and to participate in the economic system of SL.

OpenSim is a server system for virtual worlds implemented in C#. The system is almost completely compatible to the SL protocol. For that reason, the official client of SL can be used to connect to an OpenSim server.

3.2 Scripting Languages

The official scripting language in SL is called “Linden Scripting Language” (LSL). This programming language allows one to assign scripts to in-world objects. An example of a simple object is a box that displays a welcoming text message and note card to users in close proximity. An example of a complex object is a vehicle that can be used for fast transportation, simulation or entertainment, and employs the integrated HAVOK physics engine in SL. With over 300 library functions and different data and message types, scripts can control the behavior of virtual objects and communicate with other objects and avatars. Limitations of the scripting language include time delays for movement of objects (0.2 sec) and memory constraints for scripts (16 KB). In OpenSim there are several languages for in-world scripting: LSL, OpenSim Scripting Language (OSSL), and C# scripts.

3.3 Programming Interfaces

libOpenMetaverse (previously libsecondlife) is an unofficial component that provides APIs for SL, which can also be used for OpenSim, and is being developed by an open source community [41]. It connects to SL as an alternative client and has access to some of the data that is provided to the client. Among many functions for controlling the avatar, it contains methods for communication, navigation and object manipulation. Thus, compared to the LSL, its main advantages include (1) responsiveness (i.e., no implicit time delay), (2) no memory constraints, and (3) the possibility to use more complex data structures and algorithms.

OpenSim as an open source project allows one to extend algorithms or replace them. A modular architecture ensures extensibility by implementing new plugins.

3.4 Avatars and Resources

The appearance of avatars in SL is defined by the standard male or female body shape. This shape can be changed in-world by anthropometrical parameters like height, length of limbs, and diameters to create individual body types. Virtual cloths, skin textures, hair styles and accessories can be either modeled by the user or purchased in the virtual economy of SL. Animation files, which can be applied to the avatars, must be uploaded in the BVH format [42]. To create new animation files or to re-target existing motion from other systems, reference models for male and female SL avatars exist [43]. These models contain the hierarchical setup of the skeleton, which is comparable to the reference models of the H-Anim standard [44].

SL supports spatial 3D sound (i.e., sound panning and attenuation) and also allows for streaming audio. Audio sample files can be uploaded by the users and are hosted on the server. Each file is restricted to a maximum length of 10 secs and must have a sample rate of 44.1Hz.

The avatar appearance modeling in OpenSim is similar to SL. However, in combination with an alternative viewer from [45], OpenSim supports the usage of arbitrary 3D meshes that are rendered by the open source 3D engine OGRE. This approach leads to higher flexibility as direct joint manipulation enables new algorithms for skeletal and facial animations. For example, it allows to parameterize animations, to implement inverse kinematics and accurate lip synchronization, which can be blended with different facial displays expressing emotions. This option is not naturally supported by Second Life, but external solutions are currently being implemented with interesting results [46].

3.5 Comparison Between Second Life and OpenSimulator

Summarizing the previous sections, both SL and OpenSim can be seen as promising platforms for the 3D Internet. Despite their common features, both have advantages and shortcomings (see Table 1). On the one side, SL has a large user community, a significant amount of content, and it is relatively stable. On the other side, OpenSim is highly extensible but still in alpha stage and sometimes unstable. Very recently, IBM and Linden Labs announced interoperability between SL and OpenSim by allowing avatars to teleport from one world to the other by proposing a new open grid protocol [47]. Because of these reasons, we decided to support both platforms.

4 MPML3D FOR VIRTUAL WORLDS

Our new system integrates the Multimodal Presentation Markup Language 3D (MPML3D) into virtual worlds and is based on the MPML3D framework [13], [14]. In this section, we will first give an overview of our system architecture in Sect. 4.1 and then describe its main components. The tagging structures (and the XML Schema) of the adapted MPML3D language will be described in the next section.
4.2 MPML3D Backend

The MPML3D backend implements the scenario (scene setup and scene plot/content) as defined in the MPML3D script, and acts as a host for one or more frontends. It integrates the parser for the XML-based script source with the dynamic runtime representation of all interdependencies between and the hierarchy of agents’ conversational activities and perceptions, the so-called ‘activity network’. In order to keep this part of the MPML3D system flexible, the backend handles scene entities and activities in an abstract way, leaving the actual implementation to the frontends.

Each frontend realizes a user interface – comprising multimodal output as well as user feedback channels – for interactive content scripted via MPML3D. A frontend provides a concrete implementation for supported entity types along with their actions and entity states. After connecting to the backend, the frontend prepares all required output resources (e.g., generating speech files with a Text-to-Speech system or logging with special user accounts into the chosen virtual world) according to the received scene setup information. After starting the plot (of the script), it performs activities as requested by the backend. It also processes user feedback resulting in state changes and forwards these changes to the backend. Further, the frontend performs state matches as required by the backend and thus helps to determine if perceptions are triggered.

During runtime, the MPML3D backend accepts incoming network connections from frontends (see Fig. 3). In our example, the MPML3D script file for the presentation is parsed, the information for the bot entities ‘yuki’ and ‘ken’ are loaded, and so is the activity network. For initialization, frontends (one or multiple instances are possible) have to register themselves at the backend. They receive all information from the backend about the scene setup (entities and additional scene specifications), and in addition, they receive information about the actions to be performed within the script. After initialization, the backend waits for the beginning of the scene plot, which is triggered by the frontend. Then the backend issues ‘start’ and ‘stop’ commands, respectively, for those actions, as defined by the scene plot of the MPML3D script. The frontend in turn performs the output for each action according to its specification and notifies the backend when an action is completed.

4.3 SL and OpenSim Frontends

The SL frontend is responsible for the presentation of the MPML3D content in SL. It is implemented in C# and makes use of the libOpenMetaverse API (as described in Sect. 3.3) to connect to SL. For OpenSim we have ported the SL Frontend. Because of the compatibility of OpenSim to the SL protocol (see Sect. 3.1), the implementation could be adapted without major changes. In future work, we want to achieve a tighter integration into OpenSim by creating a plugin.
After registering at the MPML3D backend, the scene data is received. Text-based sentences that are specified in the MPML3D script must be synthesized with the help of a Text-to-Speech (TTS) system and are then uploaded to SL. On the subsequent execution of the script, this step becomes obsolete. The backend sends the scene setup for each humanoid entity (bot), and logs the used bots, according to user accounts. Note that bots also require user accounts, which have to be registered/provided by the content creator beforehand.

To illustrate the specific procedures for SL, we indicate how the initialization phase and presentation phase work (shown in Fig. 3). After the initialization is finished, the SL frontend monitors the virtual environment in SL for incoming events. In the example depicted in Fig. 4, two bots are waiting in idle mode. As soon as a visitor writes “Hello” into the chat (instant messaging) channel, a perception is triggered which creates a plot activation event in the SL frontend. This event in turn sends a ‘start plot command’ to the backend, which then begins the presentation. Controlled by MPML3D, the bots show verbal and non-verbal behavior using timed speech output, gesture playback and basic lip synchronization, which have been implemented as functions in the frontend.

Fig. 4. Example of a visitor avatar (middle) attending to a presentation given by two MPML3D scripted agents in SL. The presentation has been started by an in-world perception triggered by the visitor avatar (user) writing “Hello” into the chat channel.

5 SCRIPTING MULTIMODAL PRESENTATIONS WITH MPML3D

In this section, we will describe the elements of the Multimodal Presentation Markup Language 3D (MPML3D) language. [48] provides a detailed introduction to MPML3D including installation instructions, a large number of examples, and a reference section of MPML3D elements (release date: December 19, 2008).

The root element of an MPML3D script is <MPML3D>. The tagging structure <MPML3D> ... </MPML3D> contains all other tagging structures, in particular ‘tasks’ – the building blocks of the presentation – and ‘perceptions’ – a specification of what the scripted bots perceive about each other, or user-controlled avatars. The complete MPML3D schema representation is shown in Fig. 5.

5.1 Initializing SL Bots

The Head element contains the information required to initialize the bots (see Fig. 6). The Entity element is used to define the SL avatars that will take part in the presentation. Since two or more avatars are used in a typical scenario, all Entity elements have to be placed inside a tag structure called Entities. This element has no attributes and its only purpose is to serve as a entity container. Although the Entity element defines three types of attributes, only two of them are used for SL: (a) type, which is almost always human, and (b) name, by which the avatar will be identified in the script. The resourcePath attribute was used in our previous version of the system [13], in which custom avatars were utilized. There it was used to indicate the path from which the avatar 3D appearance was stored.
Fig. 5. MPML3D XML Schema specification.

In order to define the properties of an avatar, we defined an element called Property. Using this tag inside the Entity structure, a script author can define the information needed by MPML3D to log in the avatars into SL. Currently, there are only three properties supported: (a) agent_name, which specifies the full name of an avatar as “Firstname Lastname”, (b) agent_pwd, which specifies the password for the avatar, and (c) landmark_pos, which specifies a location into which the avatar will log in.

In the example segment depicted in Fig. 6, the landmark position (line 6) is represented by its LLUUID (Linden Lab Universally Unique Identifier), which can be retrieved from the SL interface. Utterances of bots have to be prepared and uploaded beforehand. One convenient way is to create an object in SL, e.g. an invisible cube, which the bot can ‘wear’ so that it can serve as a container for the audio files.

5.2 Performing a MPML3D Script

We now turn to the contents of the Body element of MPML3D. In MPML3D, so-called ‘tasks’ contain information about the activities of bots. The Action element defines a command for activities that is executed by an entity. MPML3D currently defines five commands:

- wait: The avatar waits for a specific number of milliseconds.
- write: The avatar uses the SL chat channel to convey a message.
1 <Parallel>
2 <Action name="yukiSpeak2">yuki.speak("To start the presentation
3 provided on the website, you have to create two agents in Second Life.
4 One male, one female.")</Action>
5 <Action startOn="yukiSpeak2[2].begin">yuki.gesture("turnleft")</Action>
6 <Action startOn="yukiSpeak2[8].begin">ken.gesture("turnright")</Action>
7 <Action startOn="yukiSpeak2[18].begin">yuki.gesture("COMPARE")</Action>
8 </Parallel>

Fig. 7. Sub-action synchronization.

- **gesture**: The avatar performs a particular gesture.
- **speak**: The avatar plays a speech file that has previously been stored in its speech cube.
- **walk**: The avatar walks or runs.

The statements about bots’ perceptions and the bots’ activities form the ‘activity network’ (which was introduced in Sect. 4.2).

### 5.2.1 Temporal Management

Activities can be performed sequentially (default) or in parallel. With the **Sequential** element one can specify that activities follow immediately one after the other (temporal interval \( A \) meets interval \( B \)).

The **Parallel** element refers to the parallel execution of two or more activities. The activities are carried out either by a single agent, or by multiple agents. For synchronizing the behavior of a single agent, this relation should be interpreted as Allen’s [36] ‘\( A \) starts \( B \)’, i.e. both \( A \) and \( B \) start at the same time, but do not necessarily terminate at the same time, rather than ‘\( A \) equals \( B \)’ (\( A \) and \( B \) start and end at the same time). Gesture animations in SL have a fixed duration and verbal utterances have variable length, depending on the sentence to be spoken. Hence gesture and utterance will typically not finish at the same time.

The **Parallel** element can also be used to implement ‘sub-action synchronization’ (\( A \) during \( B \)), which is a powerful construct to encode natural synchronization of verbal and non-verbal behavior of an agent. Sub-action synchronization means that some actions, e.g. a gesture, are fully contained within the duration of the performance of another action, typically an utterance.

Fig. 7 shows an example of sub-action synchronization from our MPML3D promotional video, which can be watched on our “Global Lab” channel on YouTube [49]. The timing of the gestures is specified by the numbers in square brackets, which refer to the count of the words in the sentence. E.g., \( \text{startOn=}`yukiSpeak2[2].\text{begin}` \) specifies that a ‘turnleft’ (posture) will be performed upon the second word in Yuki’s utterance (“To start the presentation...”). Similarly, a ‘COMPARE’ gesture is triggered upon the utterance of the 18th word (“... Second Life. One male, one female.”)

Note that at the beginning of the eighth word (“... on the website...”) it is the bot Ken turning to the right (line 6), which is an example of parallelism of behavior among multiple agents. For different bots, we can also specify Allen’s ‘\( A \) overlaps \( B \)’ relation, where interval \( B \) starts, whereas \( A \) is not completed. This is implemented through the use of perceptions (see Sect. 5.3).

### 5.2.2 Gestures and Locomotion

In the previous section, the gestures written in lowercase belong to the gesture set of SL (about 130), whereas the gestures written in uppercase refer to our original 72 gestures (see [48] for details). The available gestures can be broadly categorized into the following types:

- **Deictic gestures**: POINT_LEFT, POINT_RIGHT, etc.
- **Communicative gestures**: AGREE, GREET, GOOD-BYE, bow, clap, turnleft, turnright, etc.
- **Iconic and metaphoric gestures** (parameters in brackets): 1 ‘showing height’ (high, medium, low), ‘show size’ (small, medium, large), etc
- **Emotion-related gestures**: express_afraid_emote, express_angr_emote, express_laugh_emote, express_surprise_emote, etc

The selection of appropriate gestures for speech and their correct timing are challenging and time-consuming tasks for content creators. Hence we have developed a system that automates these tasks [50]. Content creators also have to add (appropriate) emotional expression to achieve the impression of natural agent behavior. For that purpose, we have designed a system, which analyzes a sentence and ‘recognizes’ the emotion expressed in the sentence [51]. Both of those systems can generate MPML3D scripts as output.

An MPML3D script may also include commands for agent locomotion.

1 <Action>yuki.walk(x,y,z)</Action>
2 <Action>yuki.walk(run,x,y,z)</Action>
3 <Action>yuki.walk(S,3)</Action>

The first two statements specify that the bot “Yuki” walks (line 1) or runs (line 2) to the position \( x, y, z \). The third statement specifies that the bot walks southward for 3 seconds. The agent may also walk to the north, west, east, or combinations thereof (northwest, etc).

At this stage of development, we have not implemented any advanced method for collision free locomotion of multiple agents. An approach to navigating bots in SL can be found in [40].

1. Since animations in SL cannot be parameterized, the ‘parameters’ simply refer to different animation sequences. For the use of MPML3D with parameterized gestures, see [25].
5.3 Perceptions

Perceptions are a powerful device to implement non-linear scripts. A perception is characterized by a precondition that is tested during runtime. When the precondition eventually holds (and possibly further optional conditions hold), the perception is triggered and consequently, some task is launched.

When defining a perception, the user can specify:
(1) a name, which will identify the perception in the script, (b) the maximum number the perception can be activated, using the maxActivations attribute, (c) if the perception is active or not, using the activated attribute, and (d) which task is to be executed when the perception is triggered, using the onOccur attribute.

In the case that we need to launch tasks given a specific combination of triggered perceptions, MPML3D defines two perception container elements called All and Any. The Any element allows authors to define a perception container that is triggered when one of the perceptions defined inside the container structure is triggered, whereas the All element allows authors to define a container that is triggered if all perceptions are triggered. Since the use of these elements is recursive and non-exclusive (e.g. an Any tag can contain not only perceptions, but also other Any elements, and even All elements inside), authors can flexibly use these containers to create any perception combination they might need. The attributes for these elements are the same as the attributes for the Perception element.

We have implemented two types of perceptions: (1) internal perceptions relate to events which are defined within an MPML3D script, e.g. an utterance spoken or gesture performed by another scripted agent; (2) external perceptions are defined for events outside the MPML3D script, in particular, instant messages sent by (user) avatars.

5.3.1 Internal Perceptions

Internal perceptions are used to specify reactions of scripted agents to other scripted agents. These perceptions are specified through the use of the onEvent and onChange statements, which are included as content of the Perception element. The four internal perceptions are:

- **onEvent “saysWord”**: This statement triggers a perception when an avatar utters a specific word.
- **onEvent “saysPhrase”**: This statement triggers a perception when an avatar utters a specific phrase.
- **onEvent “performsGesture”**: This statement triggers a perception when an avatar performs a specific gesture.
- **onChange “StateParameter”**: This statement triggers a perception when a state parameter changes its content to a specific value.

Let us have a look at an example from our MPML3D promotional video.

In Fig. 8, the perception defined in lines 1–3 is triggered when Ken says the phrase “how can I do it”, i.e. the corresponding audio file is played, as specified in the task named “kenSpeaking”. Initially, the perception named “kenSaysHowCanI” is not activated (activated="false"). It is activated through the attribute–value pair activate="kenSaysHowCanI" in the task “kenSpeaking”. The perception can be activated only once (maxActivations="1").

The attribute minor specifies whether a task or action will be restarted after it is interrupted. The default value is false: once interrupted, the task or action will not be resumed afterwards. The attribute priority determines whether a task can interrupt another task. In our example, the task “yukiInterrupts” has priority 10 (line 14), and hence can interrupt the lower priority task “kenSpeaking”, which has priority 0 (line 4).

Perceptions may also have an effect on so-called “state parameters”, which can be defined in the Entity tagging structure within the Head environment of the MPML3D script. For instance, a state parameter can set the mood of a bot to a certain value.

Certain (scripted) events may have an effect on the bot’s mood. Such a situation can be specified in the form of a perception.

The effect of the perception is stated as an action.

As a consequence of Ken’s bad mood, a task aimed at ‘repairing’ the mood of this bot can be launched, using the startOn="ChangeKensMood" attribute–value pair.

5.3.2 External Perceptions

External perceptions were implemented in order to support reactions of scripted agents (bots) to user-controlled agents (avatars). The only external perception defined in MPML3D for now is onEvent “hearsPhrase”, which is triggered when a (non-scripted) SL avatar utters a specific phrase, which is heard by a pre-specified listener avatar in the script.

The script shown in Fig. 9 encodes a scenario where the female bot called “yuki” (SL name “buebchen Magic”) reacts to the user’s (SL name “Lia Rikugun”) chat message “Hello” (see Fig. 4).

The perception is ‘globally’ activated (line 11, Fig. 9). Whenever an avatar writes “Hello” as specified in the perception (lines 11–12), the task named “yukiAnswers” (lines 13–16) will be triggered. In order to be able to receive messages from a user, the entity type user has to be declared for a specific bot (line 7), and the bot needs the ‘hearing capability’ for the named user (e.g. “yukiEar”), as a listener property (line 4).

http://mc.manuscriptcentral.com/tvcg-cs
5.4 Task Attributes and Activity Structures

As seen from the previous examples, Task elements define sequences of actions, which are executed sequentially. A Task structure in MPML3D has a thorough set of attributes that allows authors to define when and how a task will be executed in a flexible way. The set of task attributes that are currently supported by MPML3D is:

- **name**: This attribute specifies the name by which the task will be identified in the script.
- **priority**: This attribute is used to determine the priority of the task compared to other tasks, so that when two or more tasks need to be executed, tasks with higher priority are executed first.
- **repeat**: If this attribute is set to “true”, the task will be repeated indefinitely, except if the minor attribute was also set to “true”. In this case, if the task is interrupted, it does not resume its execution.
- **minor**: This attribute is used to prevent a task from being interrupted when it is set to “true”.
- **resume**: This attribute specifies if the task execution will be resumed if interrupted.
- **startOn**: This attribute specifies the perception that will trigger the execution of the task.
- **onBegin**: This attribute lets the author define an onEvent statement that will be executed when a task begins its execution.
- **onEnd**: This attribute lets the author define an onEvent statement that will be executed when a task finishes its execution.
- **onInterrupt**: This attribute lets the author define an onEvent statement that will be executed when a task is interrupted.
6.1.1 Gesture Playback

To animate the virtual characters we have implemented basic support to start and stop previously uploaded or officially provided gestures in SL. Due to technical restrictions in SL, it is not possible to change the playback speed or to blend animations. Moreover, one cannot perform low-level animation, because for now it is not possible to manipulate the joints of an SL character directly. Projects like Avatar Puppeteering [46] are being developed to overcome this problem, but they are not included as part of SL standard software package yet. Once this technology is available, we will consider its inclusion into our MPML3D tag structures.

In OpenSim, however, it is possible to change the playback speed and to blend animations. Additionally, the OpenSim developer team is currently working on support for low-level animation. This will allow for parameterized gestures and individual pointing gestures.

6.1.2 Speech Output

We have identified three different approaches for speech output in SL: (1) playback of previously uploaded audio clips, (2) streaming audio that is generated on the server-side in nearly real-time, and (3) text (input) that is intercepted on the client-side and synthesized by locally installed TTS software. For the most accurate timing, the first approach is preferred. To organize speech clips in a meaningful way, we have created a 'speech cube' in SL. This cube is a small transparent object that is attached to the agent and serves as a 'container' for the speech clips. It has an associated LSL script that manages these speech-clips, and can start, pause, resume, and stop these files by commands from the SL frontend.

In OpenSim, speech output can either be realized by uploading audio clips and sending playback commands (as in SL), or, alternatively, by streaming of audio files.

6.1.3 Pseudo-lipsync

Lip synchronization is currently not supported in SL and no third-party solutions exist. The two programming interfaces of SL provide no access to the internal data structures, which would enable the implementation of a new facial animation system. As a tentative method, we implemented pseudo lip synchronization, using randomized “mouth open” animations.

To achieve a good lip-synchronization, work on a client-side solution is required. [52] is an alternative to the official client of Second Life and could be used for such an approach in future work. Alternatively, the rendering (visual) quality should also improve greatly by the contributions of [45] which will be integrated as modules to OpenSim.

6.2 Challenge: Timing Issues

Due to the distributed server/client architecture of the system, it is difficult to achieve accurate timing. The most promising solution is to synchronize the events on the server-side. As an example we demonstrate the synchronization of speech and gestures (see Fig. 10). Because the speech data must be transferred to the client, we first issue a command to pre-buffer the audio file. After a specific waiting time, the frontend sends the playback commands for speech and gesture, which achieves synchronized presentation of speech and gesture. The waiting time compensates for the network delay/latency between server and client. In the next section, we will have a closer look at the network.

6.3 Network Metrology

We have conducted a network metrology to identify bottlenecks and to estimate required resources. In order to characterize the traffic and to identify the possible evolution of it, several passive and active measurement tools have been used. Because Transmission Control Protocol (TCP) behavior depends on bandwidth, latency,
6.3.2 Bandwidth

### a) inter-server

<table>
<thead>
<tr>
<th>Phase direction</th>
<th>init sent</th>
<th>presented sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPML3D</td>
<td>33.6KB</td>
<td>540.1KB</td>
</tr>
<tr>
<td>SL login</td>
<td>80.8KB</td>
<td>4.8KB</td>
</tr>
<tr>
<td>SL region</td>
<td>459.1 KB</td>
<td>28.1KB</td>
</tr>
<tr>
<td>SL client</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
</tbody>
</table>

### b) server-client

<table>
<thead>
<tr>
<th>Phase direction</th>
<th>init sent</th>
<th>presented sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL login</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>SL region</td>
<td>n.a.</td>
<td>4.3MB</td>
</tr>
<tr>
<td>SL client</td>
<td>n.a.</td>
<td>6.1MB</td>
</tr>
</tbody>
</table>

#### TABLE 3

Measured total throughput for all server components and the client. Please note that the client is not affected by the init phase before the presentation and thus we have made no measurements for that part.

#### TABLE 4

Measured average bandwidth/throughput for all server components and the client. Please note that the client is not affected by the init phase before the presentation and thus we have made no measurements for that part.

In order to measure the bandwidth consumption we have used the passive network probing tool *ntop*. This open source tool allows to monitor the network traffic according to different protocols and to create statistics. We have analyzed the same two routes as before. Because the protocols of SL and OpenSim are very similar, we just present the results of the communication with the virtual worlds server of SL. Still, we have divided the communication into two parts (compare Fig. 3): the initialization phase and the presentation phase. For the benchmarking we have used a MPML3D script with two bots that utilizes many different aspects of the scripting language. The initialization phase takes about 14 seconds and the presentation is approximately 324 seconds long. First we give an overview of the accumulated values in Table 3 and then show the average throughput in Table 4.

By comparing these values, we noticed that the MPML3D server receives almost double the amount of data compared to the SL region. This can be explained by the fact that the bots connect as virtual clients and hence the data has to be sent twice from the SL region.

#### TABLE 2

Measured round trip times in milli-seconds for communication between the different modules of the system comparing SL and OpenSim.

<table>
<thead>
<tr>
<th>Route system</th>
<th>a) inter-server</th>
<th>b) server-client</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>47.350</td>
<td>34.041</td>
</tr>
<tr>
<td>avg</td>
<td>47.843</td>
<td>0.096</td>
</tr>
<tr>
<td>max</td>
<td>48.121</td>
<td>0.126</td>
</tr>
<tr>
<td>mdev</td>
<td>0.306</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Of course this data highly depends on the location of the servers, but it still illustrates the advantage of having co-located servers (which is possible with OpenSim) for the inter-server communication. Furthermore, once these values have been measured for specific system configurations, they can used to compensate for timing issues between the different modules. In future work we want to automatically measure this continuously during a presentation and thus optimize the timing in real-time.

6.3.1 Latency

The round trip time (RTT) is a good metrics for the latency and can be actively measured by using network tools like *ping* or *traceroute*. We have summarized the resulting data for the two aforementioned network routes from our system in Table 2.

Fig. 10. Synchronization of speech and gestures in distributed virtual worlds. The crucial part is the network delay which must be estimated.

queue size, number of users, etc., the measurement was performed over a range of different conditions. Here we present some selected results from our extensive measurements. We have focused on the following two network routes for both virtual worlds SL and OpenSim (compare Fig. 2):

a) Inter-server: between the MPML3D server (hosted at a leased commercial dedicated server in the US) to the virtual worlds server (hosted at Linden Labs for SL and hosted on the same dedicated server as the MPML3D server for OpenSim), and

b) Server-client: between the virtual worlds server to the virtual worlds viewer (the official SL client, instantiated at our institute in Tokyo, Japan).

Other researchers compared the latency of visiting uncached and cached scenarios in SL [53].
server. Thus we need to pay attention to the average bandwidth between the MPML3D server and the SL region server depending on the number of bots. Here, again a direct connection (localhost) between MPML3D server and OpenSim server is advantageous.

7 CONCLUSIONS

We have presented the Multimodal Presentation Markup Language 3D (MPML3D), an XML-based scripting language for easy control of the verbal and non-verbal behavior of multiple bots (agents) in virtual worlds. The system is based on the MPML3D framework [13] and its predecessor MPML [22], [11], [12]. In 2008, MPML3D has been adapted and extended to accommodate interfaces to virtual worlds like Second Life and OpenSimulator. Initial brief reports and a preliminary language description can be found in two short papers [14], [15]. Here, in this article, we have described the following key achievements:

- Description and comparison of the technical features and capabilities of SL and OpenSim, including measurement of network performance and solutions to ensure responsive behavior and smooth interactions of agents.
- Complete language specification of MPML3D and examples demonstrating the expressiveness of the scripting language.

MPML3D is freely available for content creators [16], and it is released as an open source project for developers [17]. Videos of scenarios based on MPML3D can be watched on YouTube [49]. Some of them can be experienced on our own SL island called “NIIsland” (NI island) [54].

The developed system combines a powerful multimodal content scripting language for life-like animated agents (MPML3D) with the popular multi-user online environments of SL and OpenSim. Agents in metaverses can be used in numerous scenarios, including applications directed at entertainment, education, training, marketing, sales, or scientific research. The bots may act out dialogue scripts, welcome visitor avatars or give instructions to them, assume the role of (programmed) subjects in a user study (e.g. for testing simulations of ubiquitous computing systems [55]), as so on. In this way, MPML3D may greatly support endeavors to enrich virtual worlds with attractive content at low cost.

In our future work, we intend to utilize the open interface of OpenSimulator to further improve the visual quality and timing of verbal and non-verbal behavior of the agents. Additionally, new features are planned, such as deictic (referential) gestures, and the authoring of interactions of bots with objects in the virtual world (see [56] for an initial report). For instance, content creators might want to let bots perform some simple assembly tasks for demonstration purposes. Another challenge is to create content for the metaverse automatically. We already developed methods to automatically generate gestures and gaze [50], emotion [51], and multimodal dialogues [57] by simply inputting text. We hope that the integration of these techniques with our MPML3D scripting language can contribute to the fast growth and added value of the 3D Internet.

ACKNOWLEDGEMENTS

The second author was supported by a “Grand Challenge” Grant from the National Institute of Informatics. The authors would like to thank Alexis Selva-Binoche for his help on the network metrology, and Jeff Ames and Adam Johnson, two core developers of OpenSim, for their valuable information and fruitful discussions. Our special thanks go to Anette von Kapri for her contributions to the release of a stable and documented version of MPML3D.

REFERENCES


Sebastian Ullrich is a Ph.D. student at the RWTH Aachen University. During his studies, he spent one year at the National Institute of Informatics in Tokyo.

Arturo Nakasone is a Postdoctoral researcher and project manager at the National Institute of Informatics. He received his M.S. and Ph.D. degrees in information and communication technology from the Univ. of Tokyo. His research focuses on interactive storytelling and 3D virtual worlds (Second Life, OpenSim).

Mitsuru Ishizuka is a professor of the Graduate School of Information Science and Technology, Univ. of Tokyo. Previously, he worked at NTT Yokosuka Laboratory and Institute of Industrial Science, Univ. of Tokyo. During 1980–81, he was a visiting assoc. professor at Purdue University. He received his B.S., M.S. and Ph.D. degrees in electronic engineering from the Univ. of Tokyo. His research interests are in the areas of artificial intelligence, multimodal media with lifelike agents, and intelligent WWW information space. He is a member of IEEE, AAAI, Japanese Society for AI (formerly president), and IPS Japan, among others.