EMBR
A Realtime Animation Engine for Interactive Embodied Agents

Alexis Heloir and Michael Kipp

DFKI, Embodied Agents Research Group
Campus D3 2, 66123 Saarbrücken, Germany
firstname.surname@dfki.de

Abstract. Embodied agents are a powerful paradigm for current and future multimodal interfaces, yet require high effort and expertise for their creation, assembly and animation control. Therefore, open animation engines and high-level control languages are required to make embodied agents accessible to researchers and developers. In this paper, we present EMBR, a new realtime character animation engine that offers a high degree of animation control via the EMBRScript language. We argue that a new layer of control, the *animation layer*, is necessary to keep the higher-level control layers (behavioral/functional) consistent and slim, while allowing a unified and abstract access to the animation engine, e.g. for the procedural animation of nonverbal behavior. We describe the EMBRScript animation layer, the architecture of the EMBR engine, its integration into larger project contexts, and conclude with a concrete application.

1 Introduction

Turning virtual humans into believable, and thus acceptable, communication partners requires highly natural verbal and nonverbal behavior. This problem can be seen from two sides: creating intelligent behavior (planning context-dependent messages) and producing corresponding surface realizations (speech, gesture, facial expression etc.). The former is usually considered an AI problem, the latter can be considered a computer graphics problem (for nonverbal output). While previous embodied agents systems created their own solutions for transitioning from behavior planning to graphical realization (cf. [1–3]), recent research has identified three fundamental layers of processing which facilitates the creation of generic software components [4, 5]: intent planner, behavior planner and surface realizer. This general architecture allows the implementation of various embodied agents *realizers* that can be used by the research community through a unified interface.

In this paper, we present a new realizer called EMBR[1] (Embodied Agents Behavior Realizer) and its control language EMBRScript. An embodied agents realizer has particularly demanding requirements: it must run at interactive speed,

[1] see also http://embots.dfki.de/projects.html

animations must be believable while complying with high-level goals and be synchronized with multiple modalities (speech, gesture, gaze, facial movements) as well as external events (triggered by the surrounding virtual environment or by the interaction partners), it must be robust and reactive enough to cope with unexpected user input with human-like responses. The system should provide the researchers with a consistent behavior specification language offering the best compromise between universality and simplicity. Finally, all the components of such a system should be open and freely available, from the assets creation tools to the rendering engine. In the terminology of the SAIBA framework [4, 5], users work on the level of intent planning and behavior planning and then dispatch high-level behavior descriptions in the behavior markup language (BML) to the realizer which transforms it into an animation. Because the behavior description is abstract, many characteristics of the output animation are left for the realizer to decide. There is little way to tune or modify the animations planned by existing realizers [6]. To increase animation control while keeping high-level behavior descriptions simple, we propose an intermediate layer between: the *animation layer*. The animation layer gives access to animation parameters that are close to the actual motion generation mechanisms like spatio-temporal constraints. It thus gives direct access to functionality of the realizer while abstracting away from implementation details.

On the one hand, the animation layer provides users with a language capable of describing fine-grained output animations without requiring a deep understanding of computer animation techniques. On the other hand, the concepts of this layer can be used as building blocks to formally describe behaviors on the next higher level (BML).

To sum up, the main contributions of this paper are:

- Introducing a new, free behavior realizer for embodied agents
- Presenting a modular architecture for realtime character animation that combines skeletal animation, morph targets, and shaders
- Introducing a new layer of specification called the *animation layer*, implemented by the EMBRScript language, that is based on specifying partial key poses in absolute time

In the following, we will first review related work, then describe the animation layer and EMBRScript. We then explain EMBR's modular architecture and conclude with a concrete application and future work.

2 Related Work

In the terminology of the SAIBA framework, the nonverbal behavior generation problem can be decomposed into behavior planning and realization. The problem of behavior planning may be informed by the use of communicative function [7], linguistic analysis [8], archetype depiction [9], or be learned from real data [10, 3]. The problem of realization involves producing the final animation which can be done either from a gesture representation language [2, 1] or from a set of active motion segments in the realizer at runtime [6].

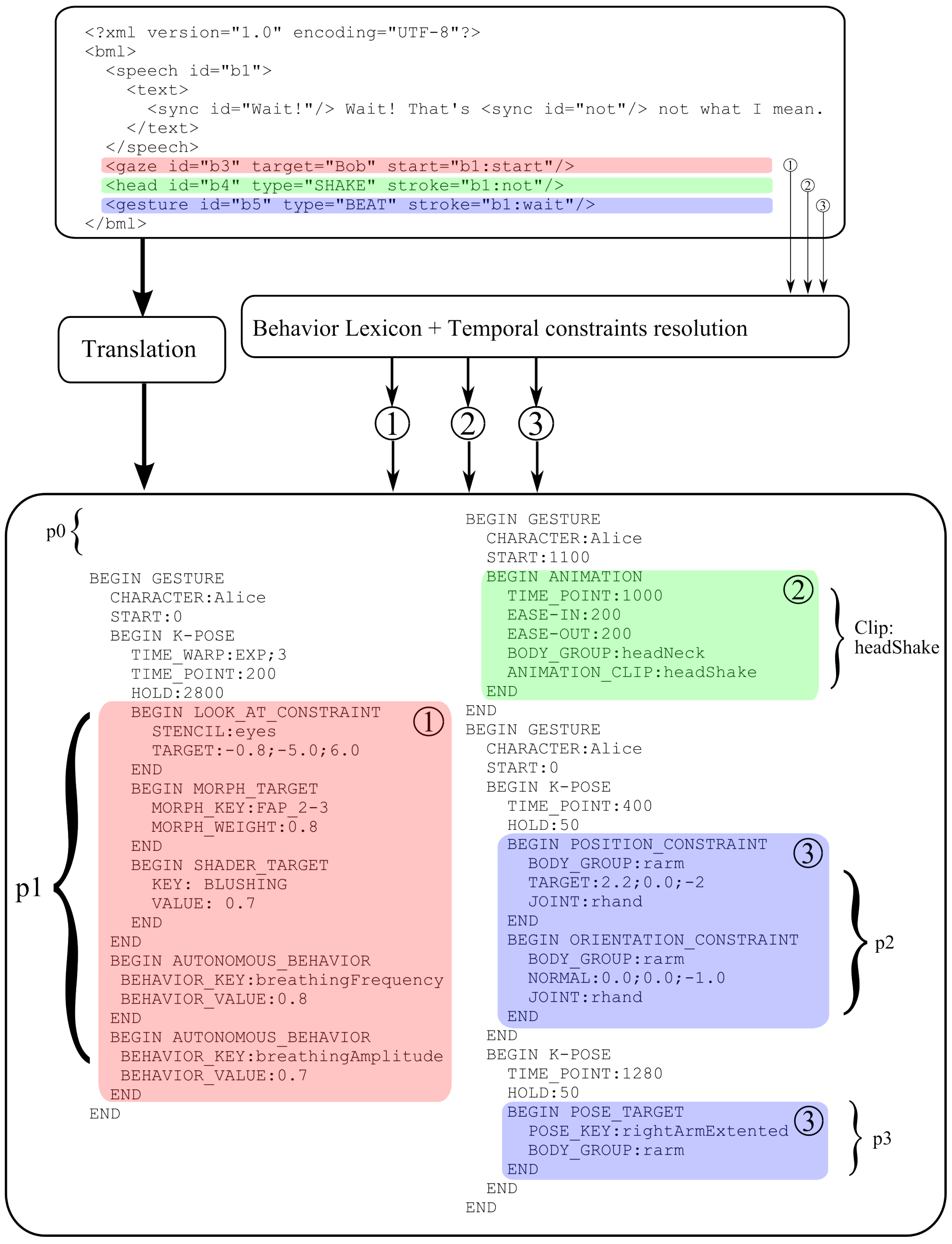
Kopp et al. [2] created an embodied agent realizer that provides the user with a fine grained constraint-based gesture description language (MURML) that lets him precisely specify communicative gestures involving skeletal animation and morph target animation. This system allows a user to define synchronization points between channels, but automatically handles the timing of the rest of the animations using motion functions extracted from the neurophysiological literature. Their control language can be regarded to be on the same level of abstraction as BML, being, however, much more complex with deeply nested XML structures. We argue that a number of low-level concepts should be moved to what we call the animation layer.

The *SmartBody* open-source framework [6] relates to our work as a freely available system that lets a user build its own behavior realizer by specializing generic animation segments called *motion controllers* organized in a hierarchical manner. Motion controllers have two functions: they generate the animation blocks and manage motion generation (*controllers*) as well as blending policy, scheduling and time warping (*meta-controllers*). As *SmartBody* uses BML [4] as an input language, it must tackle both the behavior selection and the animation selection problem. Although extending the controllers to tailor animation generation is feasible, there is currently no easy way to modify the behavior selection as "each BML request is mapped to skeleton-driving motion controllers" [6]. Moreover, even if Smartbody lets users import their own art assets, the only supported assets creation tool is Maya (commercial) and used rendering engine is Unreal (also commercial). The *BML Realizer*[2] (BMLR) is an open source project that uses the SmartBody system as an engine and Panda3D as a realizer. It therefore remedies the drawbacks of commercial tools from the Smartbody system.

3 Animation Layer: EMBRScript

It has been proposed that an abstact behavior specificaiton language like BML should be used to communicate with the realizer. Such a language usually incorporates concepts like relative timing (e.g. let motions A and B start at the same time) and lexicalized behaviors (e.g. perform head nod), sometimes allowing parameters (e.g. point to object X). While we acknowledge the importance of this layer of abstraction we argue that another layer is needed that allows finer control of animations without requiring a programmer's expertise. We call this layer the *animation layer*. It can be regarded as a thin wrapper around the animation engine with the following most important characteristics:

- specify *key poses* in time
- use absolute time
- use absolute space
- avoid deeply nested specification structures



```
<?xml version="1.0" encoding="UTF-8"?>
<bml>
  <speech id="b1">
    <text>
      <sync id="Wait!"/> Wait! That's <sync id="not"/> not what I mean.
    </text>
  </speech>
  <gaze id="b3" target="Bob" start="b1:start"/>
  <head id="b4" type="SHAKE" stroke="b1:not"/>
  <gesture id="b5" type="BEAT" stroke="b1:wait"/>
</bml>
```

```
BEGIN GESTURE
  CHARACTER:Alice
  START:0
  BEGIN K-POSE
    TIME_WARP:EXP;3
    TIME_POINT:200
    HOLD:2800
    BEGIN LOOK_AT_CONSTRAINT
      STENCIL:eyes
      TARGET:-0.8;-5.0;6.0
    END
    BEGIN MORPH_TARGET
      MORPH_KEY:FAP_2-3
      MORPH_WEIGHT:0.8
    END
    BEGIN SHADER_TARGET
      KEY: BLUSHING
      VALUE: 0.7
    END
  END
  BEGIN AUTONOMOUS_BEHAVIOR
   BEHAVIOR_KEY:breathingFrequency
   BEHAVIOR_VALUE:0.8
  END
  BEGIN AUTONOMOUS_BEHAVIOR
   BEHAVIOR_KEY:breathingAmplitude
   BEHAVIOR_VALUE:0.7
  END
END
```

```
BEGIN GESTURE
  CHARACTER:Alice
  START:1100
  BEGIN ANIMATION
    TIME_POINT:1000
    EASE-IN:200
    EASE-OUT:200
    BODY_GROUP:headNeck
    ANIMATION_CLIP:headShake
  END
END
BEGIN GESTURE
  CHARACTER:Alice
  START:0
  BEGIN K-POSE
    TIME_POINT:400
    HOLD:50
    BEGIN POSITION_CONSTRAINT
      BODY_GROUP:rarm
      TARGET:2.2;0.0;-2
      JOINT:rhand
    END
    BEGIN ORIENTATION_CONSTRAINT
      BODY_GROUP:rarm
      NORMAL:0.0;0.0;-1.0
      JOINT:rhand
    END
  END
  BEGIN K-POSE
    TIME_POINT:1280
    HOLD:50
    BEGIN POSE_TARGET
      POSE_KEY:rightArmExtented
      BODY_GROUP:rarm
    END
  END
END
```

Fig. 1. EMBRScript sample (bottom box): The script describes realizations of the behavior specified in the original BML script (top box).

We incorporate the functionality of this layer in a language called *EMBRScript* (see Fig. 1 for a sample script). EMBRScript's main principle is that every animation is described as a succession of *key poses*. A key pose describes the state of the character at a specific point in time (`TIME_POINT`), which can be held still for a period of time (`HOLD`). For animation, EMBR performs interpolation between neighboring poses. The user can select interpolation method and apply temporal modifiers. A pose can be specified using one of four principal methods: skeleton configuration (e.g. reaching for a point in space, bending forward), morph targets (e.g. smiling and blinking with one eye), shaders (e.g. blushing or paling) or autonomous behaviors (e.g. breathing). Sections 3.1 to 3.3 describe each of these methods in detail. Since the animation layer is located between behavior planning (BML) and realizer (animation engine), one can implement a BML player by translating BML to EMBRScript, as depicted in Fig. 1[3].

3.1 Skeleton configuration

Animating virtual humans using an underlying rigid skeleton is the most widely used method in computer animation. In EMBRScript, a skeleton configuration can be described in two different ways: (1) using forward kinematics (FK): all angles of all joints are specified, usually pre-fabricated with the help of a 3D animation tool, (2) using inverse kinematics (IK): a set of constraints (e.g. location of wrist joint, orientation of shoulder joint) are passed to an IK solver to determine the pose. In Fig. 1 the pose description labeled *p2* in EMBRScript defines a key pose using two kinematic constraints on the right arm: a position constraint and a partial orientation, both defined in Cartesian coordinates. In EMBR, kinematic constraints modify parts of the skeleton called `BODY_GROUPS` which are defined in terms of skeleton joints. Pose description *p3* referes to a stored pose in the engine's pose repository. The animation description *Clip:headShake* refers to a pre-fabricated animation clip (which is treated as a sequence of poses) also residing in the engine's repository.

3.2 Morph targets and shaders

The face is a highly important communication channel for embodied agents: Emotions can be displayed through frowning, smiling and other facial expressions, and changes in skin tone (blushing, paling) can indicate nervousness, excitement or fear. In EMBR, facial expressions are realized through morph target animation, blushing and paling are achieved through fragment-shader based animation. In EMBRScript, the `MORPH_TARGET` label can be used to define a morph target pose and multiple morph targets can be combined using weights. Like with skeletal animation, the in-between poses are computed by interpolation. In Fig. 1, pose *p1* in the EMBRScript sample defines a weight for a morph target key pose which corresponds to the basic facial expression of anger in MPEG-4.

[2] http://cadia.ru.is/projects/bmlr

[3] BML examples are deliberately chosen to be similar to the ones used in [6].

For skin tone, one defines a SHADER_TARGET together with an intensity, like the blushing pose in *p1*.

3.3 Autonomous behaviors

Autonomous behaviors are very basic human behaviors that are beyond conscious control. Such autonomous behavior include breathing, eye blinking, the vestibulo-ocular reflex, eye saccades and smooth pursuit, balance control and weight shifting. Although we don't want to completely describe such behavior in EMBRScript, we still want to specify relevant parameters like breathing frequency or blinking probability. However, we don't want the EMBRScript language to be restricted to a set of predefined autonomous behavior parameters but rather let the users define and implement their own autonomous behaviors and associated control parameters. The pose description labeled *p1* in the EMBRScript sample depicted in Fig. 1 shows how a user can modify autonomous behavior parameters like breathing frequency and amplitude.

3.4 Temporal variation and interpolation strategies

Human motion is rarely linear in time. Therefore, procedural animations derived from interpolation between poses must be enhanced with respect to temporal dynamics. Therefore, EMBR supports time warp profiles that can be applied on any animation element and correspond to the curves depicted in Figure 2. Time warp profiles conveying *ease in*, *ease out* and *ease it and ease out* can be specified in the EMBR language with a combination of two parameters: function family and slope steepness. The first gesture described in the EMBRScript sample of Fig. 1 illustrates a possible usage of the TIME_WARP element.

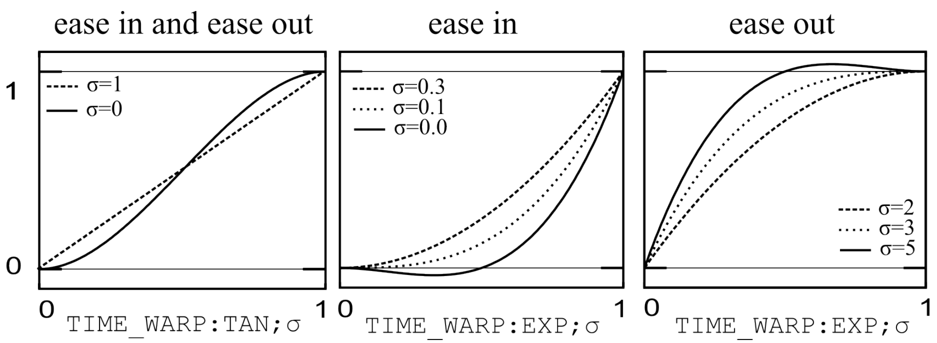


Fig. 2. Time warp profiles can be used to model *ease in*, *ease out* and *ease in and ease out*. EMBRScript offers two spline-based function families, TAN and EXP, where parameter σ roughly models steepness at $(0, 0)$. A profile may (intentionally) result in overshoot (also used by [1]).

3.5 Immediate, high-priority execution

An agent may have to respond to an interruptive event (like dodging an incoming shoe). In order to specify behaviors which require immediate, high-priority execution, EMBRScript provides a special `TIME_POINT` label: `asap`. A behavior instance whose time stamp is `asap` is performed as soon as possible, overriding existing elements.

4 EMBR Architecture

The EMBR engine reads an EMBRScript document and produces animations in realtime. In practice, this means that EMBR must produce a skeleton pose for every time frame that passes. This process is managed by a three-component pipeline consisting of the motion factory, the scheduler and the pose blender (Fig. 3). This processing is independent of the concrete rendering engine (cf. Sec. 5 to see how rendering is managed).

To give an overview, EMBR first parses the EMBRScript document which results in a sequence of commands and constraints. The *motion factory* gathers and rearranges constraints according to their timestamp and type in order to create a set of time-stamped *motion segments*. Motions segments are sent to the *scheduler* which sorts out at regular intervals a set of motion segments whose timestamp matches the current time. Relevant poses are sent to the *pose blender*. The pose blender merges all input poses resolving possible conflicts and outputs a final pose.

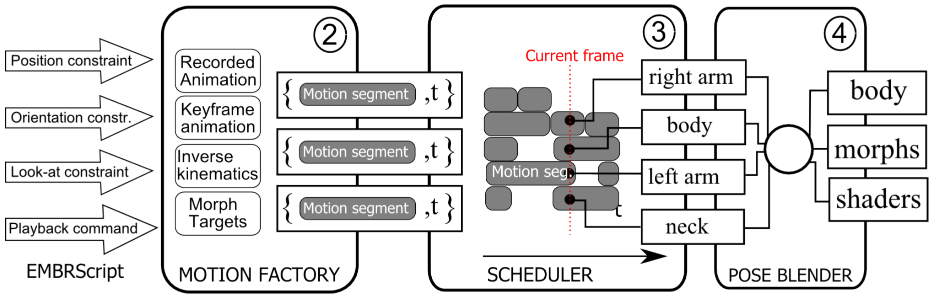


Fig. 3. The EMBR architecture

4.1 Motion Factory

The motion factory produces the building blocks of the animation called *motion segments* from the key poses specified in EMBRScript. A *motion segment* represents an animation for part of the skeleton[4] over a period of time . For

[4] More precisely: for part of the pose which includes morph targets and shaders.

instance, a motion segment may describe a waving motion of the right arm or the blushing of the face. Each motion segment contains an instance of a specialized *actuator* which drives the animation. The actuator's type depends on the motion generation method and the relevant pose component (recorded animation playback, skeleton interpolation, morph target weight and shader input interpolation). Motion segments are controlled in terms of absolute time and the timing can be warped (see Sec. 3.4) to model e.g. ease-in and ease-out.

4.2 Scheduler and Pose Blender

The *scheduler* manages incoming motion segments, makes sure that active segments affect the computation of the final pose and removes obsolete segments. For each time frame the scheduler collects all active segments and assigns a weight according to the following algorithm:

– if segment is terminating (fade out), assign descreasing weight from 1 to 0
– else if segment contains a kinematic constraint, it is tagged with `priority`
– else segment is assigned weight 1

The pose components from the motion segments are merged in the *pose blender* according to their weights using linear interpolation (we plan to allow more blending policies in the future). For kinematic constraints it is often critical that the resulting pose is not changed (e.g. *orientation* and *hand shape* of a pointing gesture), therefore the pose is tagged `priority` and overrides all others.

5 Integrating EMBR Into Larger Projects

An open character animation engine is only useful if it can easily be integrated into a larger project context and if it is possible to extend it, specifically by adding new characters. For EMBR, one of our goals was to provide a framework whose components are freely available. Therefore, we rely on the free 3D modelling tool *Blender*[5] for assets creation and on the free *Panda3D* engine[6] for 3D rendering. This section describes the complete pipeline from assets creation to runtime system. To sum up our goals:

– components for assets creation and rendering are freely available
– modifying existing characters is straightforward
– creating a new agent from scratch is possible
– use of alternative assets creation tools or renderers is possible

The EMBR framework is depicted in Fig. 4: It can be characterized by an assets creation phase (top half of figure), a runtime phase (bottom half), and data modules connecting the two (boxes in the middle).

[5] http://www.blender.org
[6] http://panda3d.org

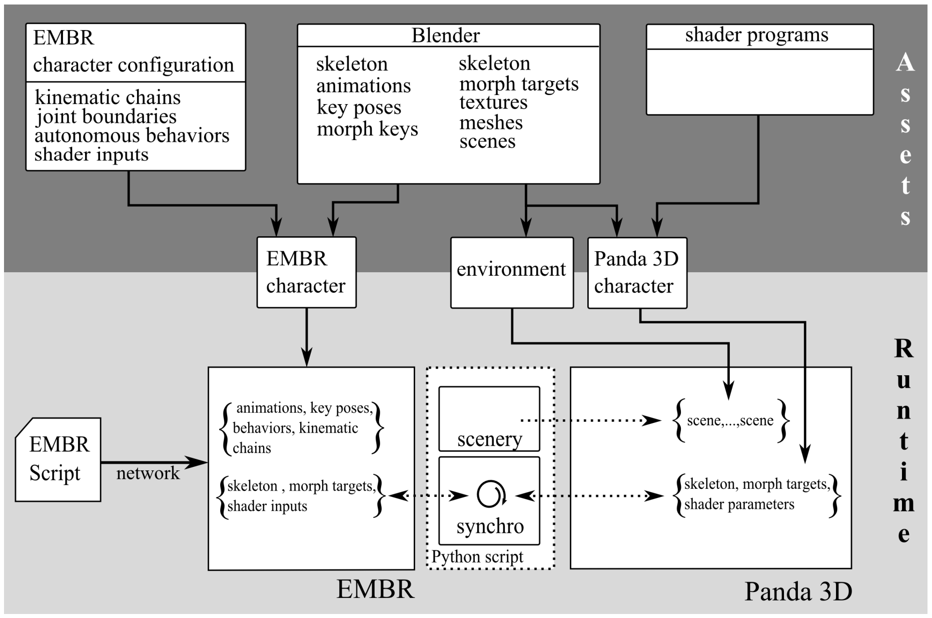


Fig. 4. The EMBR framework: Assets creation and runtime system.

5.1 Assets creation

When creating a new character, two mandatory steps are involved: creating 3D assets in a 3D modelling tool (Blender) and specifying the *EMBR character configuration*. Optionally, shader programs can be designed. In the 3D modelling tool, one first creates static resources: the character's mesh, skeleton, mesh-skeleton rigging, and textures. For facial animation, one usually creates a set of morph targets. Finally, one creates a repertoire of skeletal animations. Since in the modelling stage the user is free to choose joint names and skeleton topology, the EMBR character configuration file must be created to inform EMBR about the character's kinematic chains (e.g. left/right arm) and joint limits. Autonomous behaviors (Sec. 3.3) can also be defined here. Finally, the user may create a set of programmable shaders, e.g. for changing skin tone at runtime (blushing/paling) or for improved rendering. Shader programming is highly dependant on the rendering engine. For instance, only the Cg[7] programming language is currently supported by Panda3D. Developers can expose parameters from the shader program and control them through EMBRScript. Shader input parameters must be declared in the EMBR character configuration file. Once the character is ready, export scripts package the data from Blender for later usage by the EMBR engine and the Panda3D renderer.

[7] http://developer.nvidia.com/page/cg_main.html

5.2 Runtime

At runtime, EMBR is initialized with the EMBR character data (Fig. 4). First, it dynamically populates the *animation factory* with *motion segment* producers corresponding to the character's attributes and capabilities (see Section 4). Second, it configures the EMBRScript parser according to the character's attributes and capabilities. EMBR uses the Panda3D rendering engine for interactive display of the character. Panda3D provides a Python scripting interface which we use for synchronizing EMBR with Panda3D and for contolling the runtime system. During the lifespan of a character inside an EMBR session, two instances of the characters exist: one stored in EMBR representing the poses that result from processing EMBRScript, another one stored in Panda3D representing the display-optimized version of the character. Our Python synchronizer ensures that the Panda3D character is always in the same the state as the EMBR character.

6 Application in Procedural Gesture Synthesis

To demonstrate the capabilities of EMBR we outline its application in a gesture synthesis system [11, 3]. The system produces coverbal gestures for a given piece of text using statistical profiles of human speakers. The profiles are obtained from a corpus of annotated TV material [12]. The gesture annotations can be considered a low-dimensional representation of the high-dimensional original motion, if the latter is seen as a frame-wise specification of all joint angles. In gesture synthesis the low-dimensional representation facilitates planning of new motions. However, at the final stage such low-dimensional representations have to be translated back to a full motion. In this section, we describe one example of such a translation: from gesture annotation to EMBRScript commands.

The annotation of gesture is performed by the hierarchical three-layered decomposition of movement [13, 14] where a gestural excursion is transcribed in terms of phases, phrases and units. Our coding scheme adds positional information at beginning and end of strokes and independent holds. The transcription can be seen as a sequence of expressive phases[8] $s =< p_0, \ldots, p_{n-1} >$ where each phase is an n-tuple $p = (h, t_s, t_e, p_s, p_e)$ specifying handedness (LH, RH, 2H), start/end time, start/end pose. This description can be used to recreate the original motion which is useful for synthesizing new gestures or for validating how faithfully the coding scheme describes the form of the gesture.

For the translation to EMBRScript we separate the pose vector s into two *channels* for LH and RH, obtaining two pose vectors s_{LH} and s_{RH}. Each vector is then packaged into a single `GESTURE` tag (cf. Fig. 1). For each pose start/end information, a respective key pose is defined using positional constraints. Note that even two-handed (2H) gestures are decomposed into the described LH and RH channels. This is necessary to model the various possibilities that arise when 2H gestures are mixed with single handed gestures in one g-unit. For instance,

[8] An expressive phase is either a stroke or an independent hold; every gesture phrase must by definition contain one and only one expressive phase [15].

consider a sequence of $< 2H, RH, 2H >$ gestures. There are now three possibilities for what the left hand does between the two 2H gestures: retracting to rest pose, held in mid-air or slowly transition to the beginning of the third gesture. Packaging each gesture in a single gesture tag makes modeling these options awkward. Using two channels for RH, LH allows to insert arbitrary intermediate poses for a single hand. While this solution makes the resulting EMBRScript harder to read it seems to be a fair trade-off between expressivity and readability.

Using this straightforward method we can quickly "recreate" gestures that resemble the gesture of a human speaker using a few video annotations. We implemented a plugin to the ANVIL annotation tool [16] that translates the annotation to EMBRScript and sends it to EMBR for immediate comparison between original video and EMBR animation. Therefore, this translation can be used to refine both coding schemes and the translation procedure. A coding scheme thus validated is then an ideal candidate for gesture representation in procedural animation systems.

7 Conclusion

We presented a new realtime character animation engine called EMBR (Embodied Agents Behavior Realizer), describing architecture, the EMBRScript control language and how to integrate EMBR into larger projects. EMBR allows fine control over skeletal animations, morph target animations, shader effects like blushing and paling and autonomous behaviors like breathing. The EMBRScript control language can be seen as a thin wrapper around the animation engine that we call the *animation layer*, an new layer between the *behavior layer*, represented by BML, and the realizer. While the behavior layer has behavior classes (a pointing gesture, a head nod) for specifying form, and allows for time constraints to specify time, the animation layer uses channels, spatial constraints and absolute time to control the resulting animation. The latter is therefore much closer to the animation engine while abstracting away from implementation details. We showed how to use EMBR in conjunction with gesture coding to visually validate the coding scheme. Thus encoded gestures can then be used to populate procedural gesture synthesis systems.

For the future we plan to make EMBR freely available to the research community. We also intend to work on extending the BML layer, e.g. providing descriptors of gesture form. We will use EMBRScript to prototype these extensions. Finally, for overall integration purposes we want to develop formal descriptions of embodied agents characteristics and capabilities, taking into account existing standards like h-anim.

Acknowledgements

This research has been carried out within the framework of the Excellence Cluster Multimodal Computing and Interaction (MMCI), sponsored by the German Research Foundation (DFG).

References

1. Hartmann, B., Mancini, M., Pelachaud, C.: Implementing expressive gesture synthesis for embodied conversational agents. In: gesture in human-Computer Interaction and Simulation. (2006)
2. Kopp, S., Wachsmuth, I.: Synthesizing multimodal utterances for conversational agents. Computer Animation and Virtual Worlds **15** (2004) 39–52
3. Neff, M., Kipp, M., Albrecht, I., Seidel, H.P.: Gesture modeling and animation based on a probabilistic recreation of speaker style. ACM Trans. on Graphics **27**(1) (2008) 1–24
4. Kopp, S., Krenn, B., Marsella, S., Marshall, A.N., Pelachaud, C., Pirker, H., Thórisson, K.R., Vilhjlmsson, H.: Towards a common framework for multimodal generation: The behavior markup language. In: Proc. of IVA-06. (2006)
5. Vilhjalmsson, H., Cantelmo, N., Cassell, J., Chafai, N.E., Kipp, M., Kopp, S., Mancini, M., Marsella, S., Marshall, A.N., Pelachaud, C., Ruttkay, Z., Thórisson, K.R., van Welbergen, H., van der Werf, R.J.: The Behavior Markup Language: Recent developments and challenges. In: Proc. of IVA-07. (2007)
6. Thiebaux, M., Marsella, S., Marshall, A.N., Kallmann, M.: Smartbody: Behavior realization for embodied conversational agents. In: Proc. of AAMAS-08. (2008) 151–158
7. De Carolis, B., Pelachaud, C., Poggi, I., Steedman, M.: APML, a mark-up language for believable behavior generation. In: Life-like Characters. Tools, Affective Functions and Applications. (2004)
8. Cassell, J., Vilhjalmsson, H., Bickmore, T.: BEAT: The Behavior Expression Animation Toolkit. In Fiume, E., ed.: Proc. of SIGGRAPH 2001. (2001) 477–486
9. Ruttkay, Z., Noot, H.: Variations in gesturing and speech by GESTYLE. Int. Journal of Human-Computer Studies **62** (2005) 211–229
10. Stone, M., DeCarlo, D., Oh, I., Rodriguez, C., Stere, A., Less, A., Bregler, C.: Speaking with hands: Creating animated conversational characters from recordings of human performance. In: Proc. ACM/EUROGRAPHICS-04. (2004)
11. Kipp, M., Neff, M., Albrecht, I.: An annotation scheme for conversational gestures: How to economically capture timing and form. Journal on Language Resources and Evaluation **41**(3-4) (December 2007) 325–339
12. Kipp, M., Neff, M., Kipp, K.H., Albrecht, I.: Toward natural gesture synthesis: Evaluating gesture units in a data-driven approach. In: Proc. of the 7th International Conference on Intelligent Virtual Agents (IVA-07), Springer (2007) 15–28
13. Kendon, A.: Gesture – Visible Action as Utterance. Cambridge University Press, Cambridge (2004)
14. McNeill, D.: Hand and Mind: What Gestures Reveal about Thought. University of Chicago Press, Chicago (1992)
15. Kita, S., van Gijn, I., van der Hulst, H.: Movement phases in signs and co-speech gestures, and their transcription by human coders. In Wachsmuth, I., Fröhlich, M., eds.: Proc. of GW-07, Berlin, Springer (1998) 23–35
16. Kipp, M.: Anvil – a generic annotation tool for multimodal dialogue. In: Proc. of Eurospeech. (2001) 1367–1370