Under consideration for publication in Journal of Applied Artificial Intelligence

Realtime Animation of Interactive Agents: Specification and Realization

Alexis Heloir, Michael Kipp

DFKI, Embodied Agents Research Group

Campus D3 2, 66123 Saarbrücken, Germany

Tel. (+49)6 813 025 387 Fax. (+49)6 813 025 341

`firstname.surname@dfki.de`

Received January 2009

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. 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 also introduce new concepts for the high-level control of motion quality (spatial/temporal extent, power, fluidity). Finally, we describe 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 (Hartmann et al. 2005; Kopp and Wachsmuth 2004; Neff et al. 2008), recent research has identified three fundamental layers of processing which facilitates the creation of generic software components (Kopp et al. 2006; Vilhjalmsson et al. 2007): 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 unified interfaces.

In this article, we present a new realizer called EMBR[†] (Embodied Agents Behavior Realizer) and its control language EMBRScript. An embodied agents realizer has particularly demanding requirements: it must run at interactive speed, animations must be believable while complying with high-level goals and be synchronized with respect to 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 (Kopp et al. 2006; Vilhjalmsson et al. 2007), users work on the level of intent

[†] see also http://embots.dfki.de/EMBR

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 (Thiebaux et al. 2008). 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). We also show how to integrate the control of motion quality (spatial extent, temporal extent, power and fluidity) into our framework.

To sum up, the main contributions of this article 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

— New formulations for realizing motion qualities (spatial/temporal extent, power, fluidity) on individual motions, based on the concept of *nucleus* which subsumes stroke and independent hold of a gesture [‡].

[‡] An independent hold is the hold phase of a stroke-less gesture which usually carries the “meaning” of the gesture (Kita et al. 1998)

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. At the first split of the SAIBA framework, the Behavior Markup Language (BML) describes human movement at the level of abstract behavior (Kopp et al. 2006)(perform a pointing gesture, shrug, etc.)

The problem of behavior planning may be informed by the use of communicative function (De Carolis et al. 2004), linguistic analysis (Cassell et al. 2001), archetype depiction (Ruttkay and Noot 2005), or be learned from real data (Stone et al. 2004; Neff et al. 2008). The problem of realization involves producing the final animation which can be done either from a gesture representation language (Kopp and Wachsmuth 2004; Hartmann et al. 2005) or from a set of active motion segments in the realizer at runtime (Thiebaux et al. 2008; Gillies et al. 2008).

Kopp et al. (Kopp and Wachsmuth 2004) created an embodied agent realizer that provides the user with a fine grained constraint-based gesture description language (MURML) that lets the user precisely specify communicative gestures involving skeletal animation and morph target animation. This system allows the 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 (Thiebaux et al. 2008) 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 (Kopp et al. 2006) 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" (Thiebaux et al. 2008 p. 153). Moreover, even if Smartbody lets users import their own art assets, the only supported assets creation tool is Maya (commercial). As for rendering the authors claim to have full and partial intergration of multiple engines, e.g. Unreal 2.5 (full), Gamebryo, Ogre3D and Half-Life 2. The *BML Realizer*[§] (BMLR) is an open source project that uses the SmartBody system as an engine and Panda3D as a renderer. It therefore remedies the drawbacks of commercial tools from the Smartbody system. Existing commercial products like NaturalMotion[¶] or Golaem behavior Pack[‖] are focused on low level motion generation and do not provide a language capable of specifying the motions in an abstract way.

A more recently development is the *PIAVCA* framework (Gillies et al. 2008) for controlling responsive animation. *PIAVCA* provides a range of real time motion editing filters that can be combined to achieve complex animation effects, as well as authoring of control flows that make it possible to create both scripted and reactive responses to events. However, it seems that *PIAVCA* is more focused on generating reactive behavior: for instance, it doesn't offer an animation specification language. Furthermore, although it gives access to basic animation primitives like morph targets or joints, *PIAVCA* doesn't provide elaborate procedural animation routines like inverse kinematics but rather relies extensively on motion capture.

[§] http://cadia.ru.is/projects/bmlr
[¶] http://www.naturalmotion.com/
[‖] http://www.golaem.com/

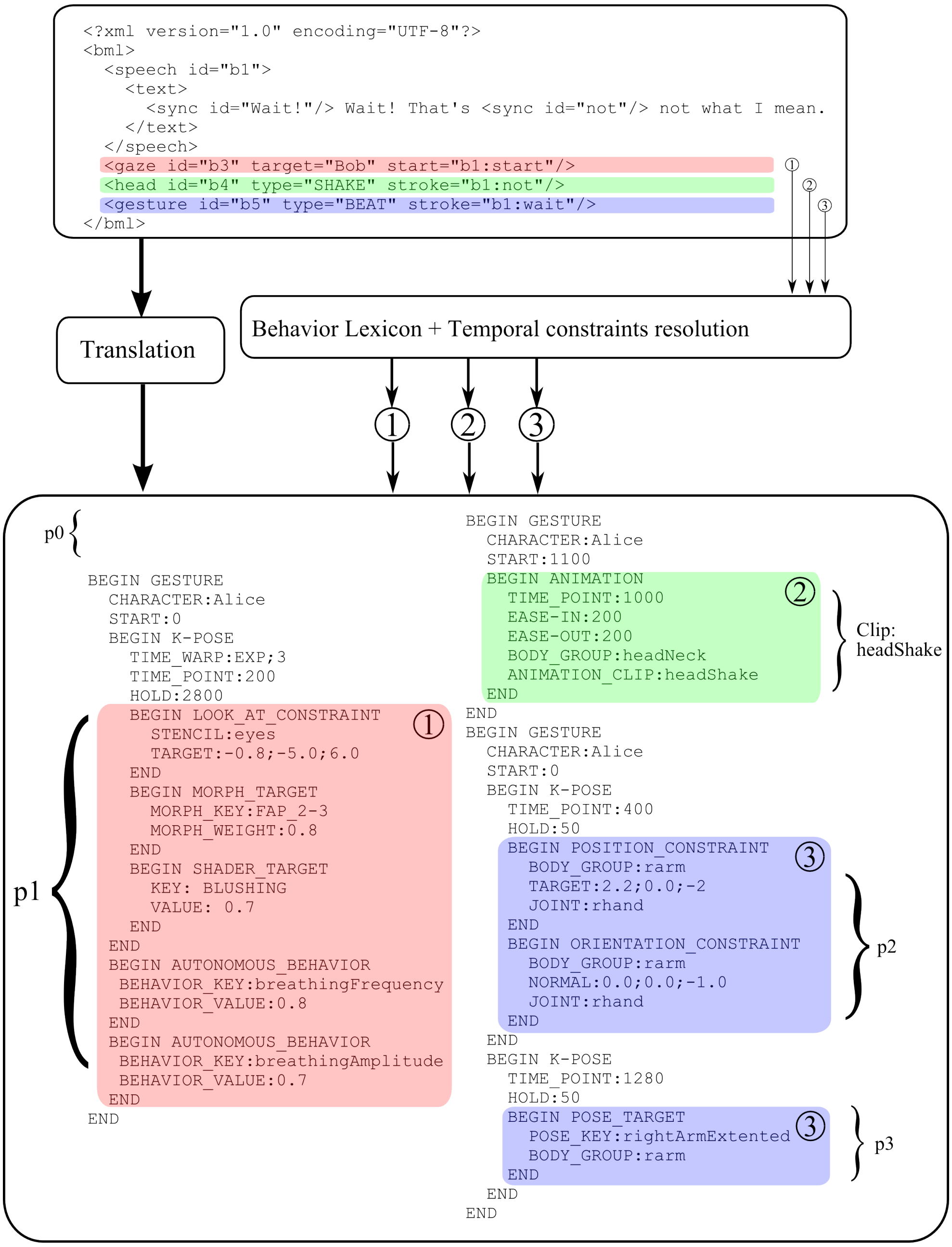
3. Animation Layer: EMBRScript

It has been proposed that an abstract behavior specification 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

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[††]. Note that the problem of translating BML

[††] BML examples are deliberately chosen to be similar to the ones used in (Thiebaux et al. 2008).



```
<?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).

temporal and spatial constraints to specific EMBRScript forumlations is an open problem that we are working on at the moment.

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 real time. 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. 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 basic human behaviors that are beyond conscious control. Examples are breathing, eye blinking, the vestibulo-ocular reflex, eye saccades and smooth pursuit, balance control and weight shifting. Such behaviors can be realized with automatic routines that are controlled with parameters like breathing frequency or blinking probability. In addition to a set of predefined autonomous behaviors EMBR also lets the users define and implement their own autonomous behaviors and associated control parameters. The pose description labeled *p1* in the EMBRScript sample 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 in and ease out* can be specified in the EMBR language with the two parameters of function family (TAN and EXP) and slope steepness (a real number). The first gesture described in the EMBRScript sample of Fig. 1 illustrates a possible usage of the `TIME_WARP` element.

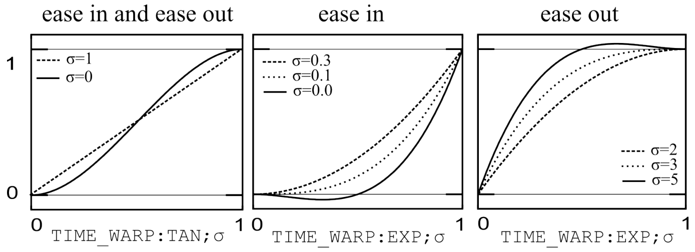


Fig. 2. Time warp profiles can be used to model *ease in*, *ease out* and *ease in and ease out*. EMBRScript offers two spline-based function families, `TAN` and `EXP`, where parameter σ roughly models steepness at $(0, 0)$. A profile may (intentionally) result in overshoot like in (Hartmann et al. 2005).

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. 6 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.

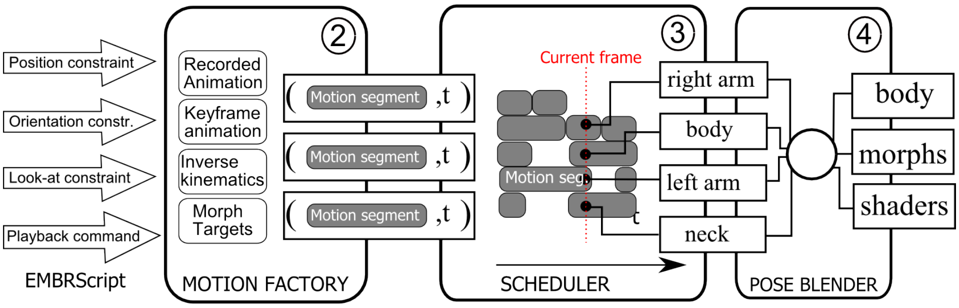


Fig. 3. The EMBR architecture

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[‡‡] over a period of time . For 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 (overlapping segments are interpolated according to their respective weights)

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

— else if segment contains a kinematic constraint, it is tagged with `priority` (will override all other segments)

— else segment is assigned weight 1 (overlapping segments are interpolated according to their respective weights)

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. Expressivity Parameters

Even a single human being displays an almost infinite variety in motion, depending on mood, emotion, style and many other factors. For character animation it is desirable to achieve variety by applying a small set of high-level parameters to the same motion. Prior work has come up with various parameters. (Hartmann et al. 2005), from now on called HPM, introduced spatial extent, temporal extent, fluidity, power and repetition, whereas EMOTE employed the effort dimension of Laban's movement analysis system (Chi et al. 2000). Both models use a similar set of parameters (see Table1).

expressivity	HPM	EMOTE
space used for gesturing	Spatial Extent	Shape, Effort (space)
timing, accelerations	Temporal Extent	Effort (time, weight)
trajectory, smoothness	Fluidity	Effort (flow, space)
weight, force	Power	Effort (weight ,flow)

Table 1. *Traditional principles of animation and their realisation in EMOTE and HPM*

We decided to select four of the HPM parameters but re-formulated their implementation as described in this section. The parameters are (all vary between -1 and 1):

— **Temporal extent**: How fast/slow a movement is.

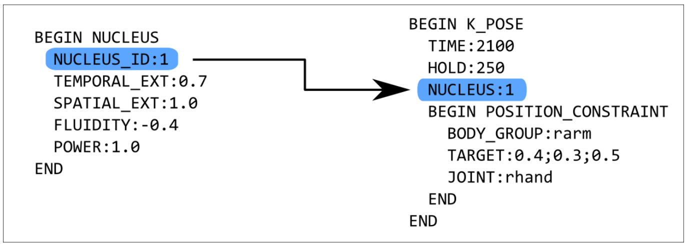
— **Spatial extent**: Size of the amplitude of a motion.

— **Fluidity**: Smoothness and continuity of the movement.

— **Power**: Sense of force and engagement of the movement.

We introduce the concept of *nucleus* to make these parameters applicable to a low-level language like EMBRScript. This is necessary for two reasons. First, we need to know where the *most expressive part* of a gesture is located in a sequence of poses (McNeill 1992), because usually only this part is modified. Second, in EMBRScript a single gesture could be defined across multiple sequences (e.g., one for each arm), or a sequence could contain multiple gestures. The *nucleus* construct is therefore used to mark the expressive part and to bind together the potentially many parts of a single nucleus.

Since we apply the expressivity parameters to a nucleus and not, as HPM do, to a whole character, we can modulate the expressiveness of an agent over time, for instance, to convey different emotions or context-dependent changes of conduct (e.g. due to status differences). Fig. 4shows how a nucleus is defined and parametrized, and then referenced from particular pose(s) belonging to that nucleus.



```
BEGIN NUCLEUS
  NUCLEUS_ID:1
  TEMPORAL_EXT:0.7
  SPATIAL_EXT:1.0
  FLUIDITY:-0.4
  POWER:1.0
END
```

```
BEGIN K_POSE
  TIME:2100
  HOLD:250
  NUCLEUS:1
  BEGIN POSITION_CONSTRAINT
    BODY_GROUP:rarm
    TARGET:0.4;0.3;0.5
    JOINT:rhand
  END
END
```

Fig. 4. The Nucleus embeds a set of expressivity parameters, it is bound to a K_POSE by ID

5.1. *Parameter implementation*

Our formulation of expressiveness is based on the modification of three different aspects of the animation

— the shape of the key poses

— the interpolation parameters between key poses

— the timing between key poses

We control the fluidity of an animation by modifying the interpolation style. For this, we implemented Kochanek spline interpolation (Kochanek et al. 1984) in quaternion space (details can be found in the appendix).

5.1.1. *Temporal Extent* The temporal extent parameter p_{temp} specifies how slow or fast a motion appears by varying the duration of the several gesture parts from key pose to key pose. A negative value reduces the duration, a positive value increases the amount of time needed to reach the target point. This time offset is obtained using a (logarithmic) function of the original duration d: $delay = log(d) * p_{temp} * 1.5$; The logarithm avoids too big delays or too drastic cuts. The 1.5 scaling value guaranties that P_{temp} values are kept within an intuitive range ($[-1, 1]$). The temporal extent can cause a pause after finishing the modified gesture since the overall duration of the motion is shortened for negative values or may lead to an overlap with the subsequent gesture for positive values. Also note that particular synchronization points with other modalities (most notably speech) that are located after the nucleus can become out-of-sync.

5.1.2. *Spatial Extent* Various methods have been proposed to systematically enlarge/shrink an existing gesture. The challenge is to preserve the overall shape of the gesture after the modification. For instance, if a gesture is simply scaled with regard to a fixed point without respecting the trajectory of the gesture the overall shape may be distorted (e.g. a circle to an ellipsoid). One may instead adjust single segments between key points which may cause problems at start and end points and may also distort the geometry.

Therefore, we decided to use a "gesture-centric" method: we compute the bounding box of the original key points k_i of both hands (for bihanded gestures). According to the value of the spatial extent parameter $p_{spatial}$ this box is scaled together with the

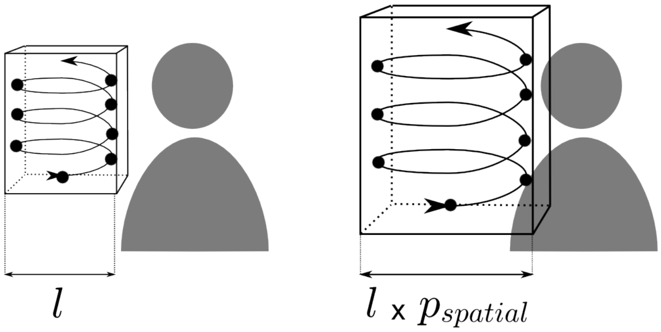


Fig. 5. Applying deformation over a precomputed bounding box is well fitted to gestures depicting 3D trajectories.

contained key coordinates around its centre c: $k'_i = k_i + ((k_i - c) * Scale_{morph} * p_{spatial}$ as depicted in Fig. 5. Again, the scalar $Scale_{morph}$ guaranties that $P_{spatial}$ values are kept within an intuitive range and depends on the agent's morphology. This method proved most useful for a number of different gestures, wheres the other approaches always failed for a specific geometric or symmetric gestures.

5.1.3. *Power* The power parameter p_{pow} varies the energy of the gesture, a forceful motion for positive values emphasizes the stroke. It adds a preparation key pose to generate a preparatory motion in the opposite direction of the motion and with a length of $0.1 * p_{pow}*$the length of the original motion. A 300ms hold at this position is added and the remaining motion is sped up by factor $1.0 + (p_{pow} * 0.2)$. The tension parameter for interpolation is set equal to p_{pow} leading to straight segments for high power and curves for low power. The bias parameter is used to create over- or undershoot at the end of strokes by setting it also equal to p_{pow} for all motion segments as depicted in Fig. 6.

5.1.4. *Fluidity* This parameter p_{fluid} affects continuity and holds of gestures. A high fluidity setting shortens or removes pauses by subtracting up to $300 * v_{fluid}$ ms from existing pauses and guarantees the continuity of the interpolation by setting continuity to zero if fluidity if greater or equal zero. A negative value causes the target joint to rest

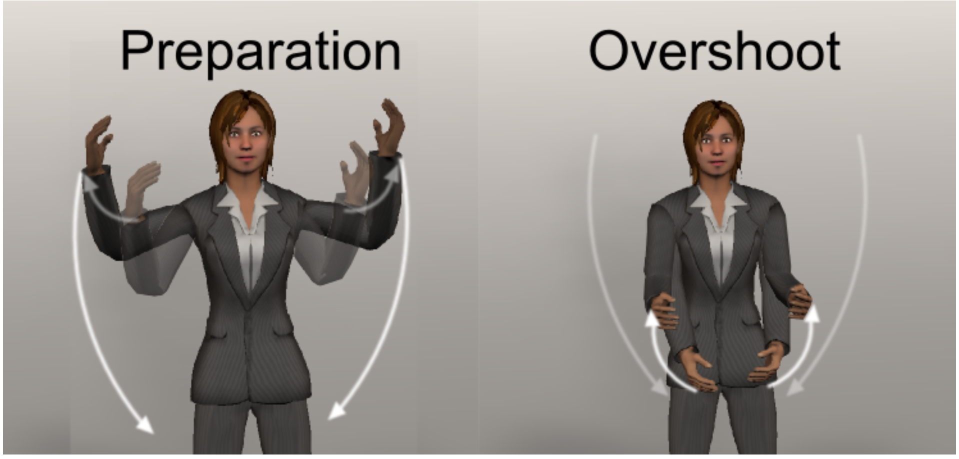


Fig. 6. A positive value for power adds a preparation phase before the stroke and some overshoot towards its end.

at every key position for $300 * v_{fluid}$ ms and adds corners in the interpolation process by setting continuity to $-p_{fluid}$.

To conclude, we introduced formulations for our four quality parameters based on the concept of the nucleus. Although this needs to be validated by a user study, our impression is that our gesture-centric approach to expressivity parameter realization makes the resulting motion look more natural.

6. 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 allowing other different characters to be animated. 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* for assets creation and on the free *Panda3D* engine 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

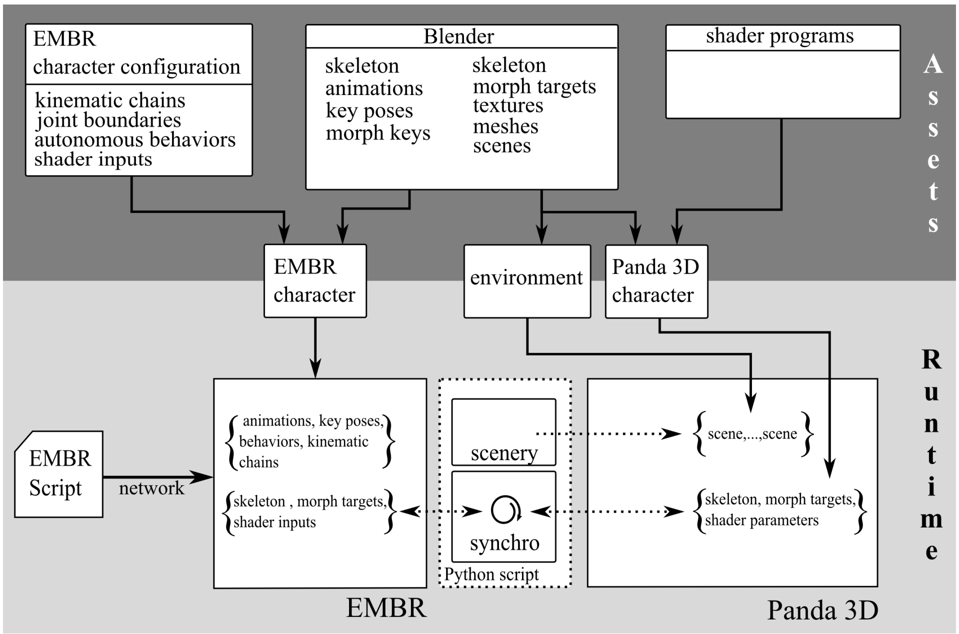


Fig. 7. The EMBR framework: Assets creation and runtime system.

— 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. 7: 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).

6.1. *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 file.* 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. 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 shading 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.

6.2. *Character Configuration File*

Since in the modeling 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 inherent features. The character configuration file is in charge of specifying the different properties and capabilities of the agent so that they can be exposed and accessed through the EMBRScript language. Currently, the character specification file support the following elements for one agent:

- a set of available pre-recorded animation clips (Sec. 3.1),
- a set of available morph targets (Sec. 3.2),
- a tree element describing the joint hierarchy,
- joint properties: name, boundaries and kinematic gain,
- joint groups: defined by a set of joint references,
- kinematic structures (set of joint references) and their associated algorithms (Sec. 3.1),
- autonomous behavior: defined by their animation(s) and triggering mechanism: periodic, randomized or event based (Sec. 3.3).

The character specification file not only lets a user describe the properties and capabilities of a new agent; it also provides a basic language capable of describing complex multimodal behaviors by combining basic animations using three basic relational predicates: *reverse*, *sequential* and *parallel*.

For instance, the animation triggered by the breathing behavior is a compound of a skeletal animation and a morph target based animation, played forward then backwards, as described in Fig. 8.

```
<morphAnim id="morph_inspire">
  <morphKey id="inflateTorso" value="0.0" time="0"/>
  <morphKey id="inflateTorso" value="0.6" time="1100"/>
</morphAnim>

<parallel id='inspire'>
 <animation id="spine_inspire">
 <animation id="morph_inspire">
</parallel>

<sequential id="breathing">
  <animation id="inspire"/>
  <reverse>
   <animation id="inspire"/>
  </reverse>
</sequential>
```

Fig. 8. The breathing animation is a compound of a skeletal animation and a morph target based animation, played forward then backwards.

The character configuration file follows an XML specification whose DTD is available online[§§].

6.3. *Runtime*

At runtime, EMBR is initialized with the EMBR character data (Fig. 7). First, it dynamically populates the *animation factory* with *motion segment* producers corresponding to the character's attributes and capabilities defined in the character configuration file. Second, it configures the EMBRScript parser. 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 controlling 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

[§§] http://embots.dfki.de/EMBR/characterConfiguration.dtd

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. Because gestures are defined using high level constraints and because these gestures are resolved at the last moment using real time motion generation methods, our system in particularly suited for interactive animation. However, users wanting to stick with preprocessed motion (handcrafted) or captured can also import and use their own asset.

7. Application in Procedural Gesture Synthesis

To demonstrate the capabilities of EMBR we outline its application in a gesture synthesis system (Kipp et al. 2007; Neff et al. 2008). 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 (Kipp et al. 2006). 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 (Kendon 2004; McNeill 1992) 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[¶¶] $s =< p_0, \ldots, p_{n-1} >$ 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

[¶¶] An expressive phase is either a stroke or an independent hold; every gesture phrase must by definition contain one and only one expressive phase (Kita et al. 1998).

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, 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 (Kipp 2001) 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.

8. 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. Although EMBRScript may sound like a scripting language, we see it as a general specification layer that abstracts away from concrete animation engine implementations. EMBRScript may therefore develop into a standardization discussion on how to control character animation in general. EMBR is now freely available to the research community[‖‖]. Finally, for overall integration purposes we want to develop a consistent way of describing the features and capabilities of embodied, extending 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). Authors would like to thank Pascal Pohl for his valuable contribution to the expressivity model and for his help during the writing process. EMBR's shaders management has been greatly enhanced by Stefan John.

[‖‖] get EMBR at http://embots.dfki.de/EMBR

Appendix

Adding tension, bias and continuity parameters to quaternion interpolation

Linear interpolation does not create convincing paths for limb motion and orientation, whereas cubic interpolation leads to more natural curves. To be able to influence these paths with our expressivity parameters we transfer the parameters introduced in (Kochanek et al. 1984), tension, continuity and bias, to the squad interpolation algorithm by Shoemake (Shoemake 1987)

$$Squad(q_i, q_{i+1}, a, b, h) = Slerp(Slerp(q_i, q_{i+1}, t), Slerp(b, a, t), 2t(1-t)) \tag{1}$$

In contrast to the normal squad algorithm, incoming and outgoing tangents are different. We influence the computation of this tangents in dependence of the parameters tension t, bias b and continuity c, all in $[-1.0, 1.0]$:

$$\begin{aligned} T_i^0 &= \frac{(1-t)\cdot(1-c)\cdot(1-b)}{2}\cdot\log(q_i^{-1}q_{i+1}) + \frac{(1-t)\cdot(1+c)\cdot(1+b)}{2}\cdot\log(q_{i-1}^{-1}q_i) \\ T_i^1 &= \frac{(1-t)\cdot(1+c)\cdot(1-b)}{2}\cdot\log(q_i^{-1}q_{i+1}) + \frac{(1-t)\cdot(1-c)\cdot(1+b)}{2}\cdot\log(q_{i-1}^{-1}q_i) \end{aligned} \tag{2}$$

With the modified tangents we compute the additional quaternions a and b for the squad interpolation:

$$\begin{aligned} a &= q_i \cdot \exp\left(-\frac{T_i^0 - \log(q_i^{-1}q_{i+1})}{4}\right) \\ b &= q_{i+1} \cdot \exp\left(-\frac{\log(q_{i-1}^{-1}q_i) - T_i^1}{4}\right) \end{aligned} \tag{3}$$

Below is a brief description of how the different parameters (tension, bias and continuity) influence the interpolated curve. A graphical representation is also depicted in Fig.9

— *tension* defines how sharply the curve bends between the key points. For 1.0 it reduces

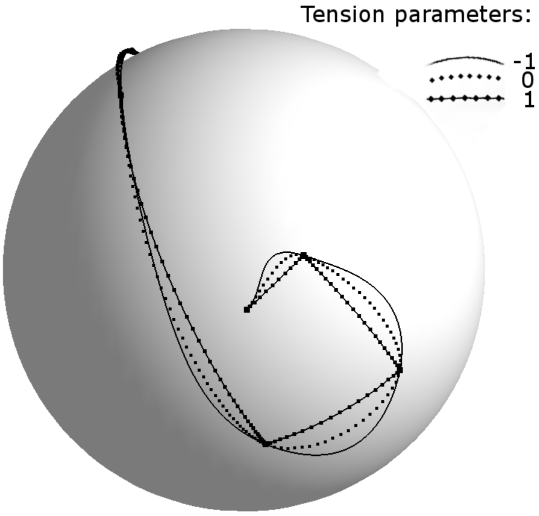


Fig. 9. Influense of the tension parameter $-1, 0, 1$ on the quaternionic interpolation spline visualized on the simplified $SO(3)$ hypersphere.

the length of the corresponding tangent vector to zero leading to linear interpolation and for -1.0 it doubles its length, resulting in more unstressed curves.

— *continuity* controls the angle between the incoming and outgoing tangents at key points. For 0.0 spline tangent continuity at the key points is preserved, for -1.0 and 1.0 the curve has acute corners to the in- or outside of the path.

— *bias* specifies the direction of the path at key points, for -1.0 it is completely determined by the incoming, for 1.0 by the outgoing tangent.

References

Cassell, J., H. Vilhjálmsson, and T. Bickmore (2001). BEAT: The Behavior Expression Animation Toolkit. E. Fiume (Ed.), *Proc. of SIGGRAPH 2001*, pp. 477–486.

Chi, D., M. Costa, L. Zhao and N. Badler (2000). The EMOTE model for effort and shape. Kurt Akeley (Ed.) *Proc. of SIGGRAPH 2000*, pp. 173–182.

De Carolis, B., C. Pelachaud, I. Poggi, and M. Steedman (2004). APML, a mark-up language for believable behavior generation. H. Prendinger and M. Ishizuka (Eds.) *Life-Like Characters.*

Tools, Affective Functions and Applications, pp. 65–84. Cognitive Technologies Series, Berlin: Springer.

Gillies, M., X. Pan and M. Slater (2008). PIAVCA: A Framework for Heterogeneous Interactions with Virtual Characters H. Prendinger, J. Lester and M. Ishizuka (Eds.) *Proc. of Intelligent Virtual Agents, Tokyo, Japan*, pp. 494–495.

Hartmann, B., M. Mancini, and C. Pelachaud (2005). Implementing expressive gesture synthesis for embodied conversational agents. G. Sylvie, N. Courty and J.F. Kamp (Eds.) *Proc. of Gesture in Human-Computer Interaction and Simulation, Berder Island, France*, pp. 188–199.

Kendon, A. (2004). *Gesture – Visible Action as Utterance.* Cambridge: Cambridge University Press.

Kipp, M. (2001). Anvil – a generic annotation tool for multimodal dialogue. P. Dalsgaard, B. Lindberg, H. Benner, and Z. Tan (Eds.) *Proc. of Eurospeech, Aalborg, Denmark*, pp. 1367–1370.

Kipp, M., M. Neff, and I. Albrecht (2007, December). An annotation scheme for conversational gestures: How to economically capture timing and form. JC. Martin, P. Paggio, P. Kuehnlein, R. Stiefelhagen and F. Pianesi (Eds.) *Journal on Language Resources and Evaluation 41* (3-4), pp. 325–339.

Kipp, M., M. Neff, K. H. Kipp, and I. Albrecht (2006). Toward natural gesture synthesis: Evaluating gesture units in a data-driven approach. C. Pelachaud, J.-C. Martin, E. André, G. Chollet, K. Karpouzis and D. Pelé (Eds.) *Proc. of Intelligent Virtual Agents, Marina Del Rey, USA*, pp. 15–28.

Kita, S., I. van Gijn, and H. van der Hulst (1998). Movement phases in signs and co-speech gestures, and their transcription by human coders. I. Wachsmuth and M. Fröhlich (Eds.), *Gesture in Human-Computer Interaction and Simulation, Bielefeld, Germany*, pp. 23–35.

Kochanek, D. and R. Bartels (1984). Interpolating splines with local tension, continuity, and bias control. *Proc. of SIGGRAPH 1984*, *18* (3), pp. 33-41.

Kopp, S., B. Krenn, S. Marsella, A. N. Marshall, C. Pelachaud, H. Pirker, K. R. Thórisson, and H. Vilhjalmsson (2006). Towards a common framework for multimodal generation: The behavior markup language. J. G. Carbonell and J. Siekmann (Eds.)*Proc. of Intelligent Virtual Agents, Marina Del Rey, USA*, pp. 205–217.

Kopp, S. and I. Wachsmuth (2004). Synthesizing multimodal utterances for conversational agents. N. Magnenat-Thalmann and D. Thalmann (Eds.) *Computer Animation and Virtual Worlds*, *15* (1), pp.39–52.

McNeill, D. (1992). *Hand and Mind: What Gestures Reveal about Thought.* Chicago: University of Chicago Press.

Neff, M., M. Kipp, I. Albrecht, and H.-P. Seidel (2008). Gesture modeling and animation based on a probabilistic recreation of speaker style. J. C. Hart (Ed.)*ACM Transactions on Graphics*, *27* (1), pp. 5:1-24

Ruttkay, Z. and H. Noot (2005). Variations in gesturing and speech by GESTYLE. N. Suzuki and C. Bartneck (Eds.) *Intl. Journal of Human-Computer Studies*, *62* (2), pp. 211–229.

Shoemake, K (1987) Quaternion Calculus and Fast Animation, Computer Animation: 3-D Motion Specification and Control. *SIGGRAPH 1987 course 10*, pp. 101–121.

Stone, M., D. DeCarlo, I. Oh, C. Rodriguez, A. Stere, A. Less, and C. Bregler (2004). Speaking with hands: Creating animated conversational characters from recordings of human performance. John C. Hart (Ed.) *ACM Transactions on Graphics*, *23* (3), pp. 506–513.

Thiebaux, M., S. Marsella, A. N. Marshall, and M. Kallmann (2008). Smartbody: Behavior realization for embodied conversational agents. L. Padgham, D. Parkes, J. M.üller and S. Parsons (Eds.) *Proc. of the intl. joint conf. on Autonomous agents and multiagent systems, Estoril, Portugal*, *7* (1) pp. 151–158.

Vilhjalmsson, H., N. Cantelmo, J. Cassell, N. E. Chafai, M. Kipp, S. Kopp, M. Mancini, S. Marsella, A. N. Marshall, C. Pelachaud, Z. Ruttkay, K. R. Thórisson, H. van Welbergen, and R. J. van der Werf (2007). The Behavior Markup Language: Recent developments and challenges. C. Pelachaud, J.-C. Martin, E. André, G. Chollet, K. Karpouzis and D. Pelé (Eds.) *Proc. of Intelligent Virtual Agents, Marina Del Rey, USA*, pp. 99–111..