Graceful Anytime Interruptability for Virtual Agents

Diplomarbeit im Fach Informatik

vorgelegt von

Klaus Brgmann

geb. am 21.November 1977 in Erlangen

angefertigt am

Institut für Informatik
Lehrstuhl für Graphische Datenverarbeitung
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Prof. Dr. Marc Stamminger

Betreuender Hochschullehrer: Prof. Dr. Marc Stamminger

Beginn der Arbeit: 1.Januar 2007

Abgabe der Arbeit: 2. Juli 2007

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work presents techniques useful for an integrated approach to generate highly natural animation for virtual characters in realtime. While there is a magnitude of tools and techniques to create and represent statically prepared animation data and apply it for non-interactive applications, higher requirements apply when motions have to be continuously adapted upon user feedback. An example for visual artifacts that are likely to occur are computer games, where animations are cut off at any point and replaced by other animations as soon as the desired mode of motion changes. Today's computer games generally do suffer little from those motion "pops" as the surprise effect as such is often an essential feature. However, applications that address more fine-grained visual reception of motion might loose their persuasiveness and thus miss their goal. A virtual martial arts instructor demonstrating motion sequences to real-life students for example needs to display naturally possible a physically possible choreography at least to be taken serious. The interdisciplinary research field of Intelligent Virtual Agents best characterizes the family of applications that have an urge to create a high naturalness of body expression. These applications aim at creating the illusion of life-likeliness for virtual avatars and thus might profit from the presented techniques in the first place. The remainder of this introduction describes the nature and requirements of Intelligent Virtual Agent applications along with giving some representative examples.

1.1 Embodied Conversational Agents

"Humans are biased to treat computers like real people", a conclusion of Reeves and Nass from their book "The Media Equation" [37], may serve as a motivating thought behind the efforts taken to make human-computer interaction more similar to human-human interaction. As long as computers are used as mere tools a command-based communication like apparent in many user interfaces might be a suitable choice. However, there has been a tendency to utilize them as more independent entities that serve as assistants for users. For example, a datebook application might offer a reminder function that extraordinarily addresses the user in case a certain appointment is likely to be forgotten referring to similar cases in the past. As applications get more intelligent, Human Computer Interaction (HCI) will benefit from more sophisticated ways of information interchange. Interaction between real people is communicative, broad in content, highly contextual, non-predetermined, communicative and behaviorally subtle [5]. Communication channels are multimodal and encode multiple levels of meaning.

Human Interaction is a highly effective and efficient way of interaction ([36], p5), which is why, in order to improve usability and intuitiveness of user interfaces, it is of interest to enrich HCI about modes of natural face-to-face conversation.

Embodied Conversational Agents pursue that goal by having life-like creatures ”apparently living on the screen” ([36], p4) function as a mediator of information transfer to and from the user. By displaying the visual appearance of a physical embodiment, they can utilize essential modes of human communication such as gesture and mimic, but also symbols like clothing and stature to express personality and attitude towards the matter of discussion.

However the user thereby is never meant to be deceived in a way that he really believes to talk to a living creature. It is assumed that most users would feel uncomfortable with interfaces that look too natural ([36], p461). The goal is rather to produce an illusion that makes the user be inclined to suspend his or her disbelief in the live of the agent for a certain time.

1.1.1 Applications

The applications benefiting from ECA interfaces include virtual e-learning and training environments, computer based therapeutical interfaces, interactive entertainment and product presentation platforms. In the following I give representative examples for the most important application fields.

1.1.1.1 Training and E-Learning

In learning applications Embodied agents can be used to create rich, face-to-face learning interaction. Just like the viewers eye is guided by the camera perspective in a motion picture, a virtual character can improve a learning environment by guiding the student's attention to the most essential aspects of a learning task [19]. By broadening the bandwidth of tutorial communication the capabilities of a learning environment to engage and motivate the learner ar increased. An example is Design-A-plant [24], an interactive learning application for the domain of botanical anatomy and physiology for children. The interface provided the opportunity to design plants by assembling plant parts and involved the cartoon-style character 'Herman the Bug', who would comment the learners actions and give hints and explanations about concepts and tasks.

1.1.1.2 Product Presentation and Customer appraisal

ECAs are also employed in commercial platforms like customer websites. There they serve in multiple functions. Virtual Max for example was a virtual dog character, living on the consumer website of a company for pet-related products, Petopia []. Being the front end of a dialogue system, where user questions are answered by matching them to a predefined catalogue. While helping customers in navigating the site, Jack would display dog-like behaviors and engage in creating a personal conversation atmosphere by telling things about his persona and life and encouraging the visitors to answer to his own questions. An ambition of that application therefore was not only making the online presence more appealing, but also to gather information about users.

1.1.1.3 Conversation Partners in Therapy

ECAs can be useful in computer-based therapeutical applications. Lisetti et al [25] presented an example for enrichment of therapeutical applications with an emotional agent interface. The underlying MOUE system comprises sensor input components gathering information about the users emotional state. The system offers the possibility to engage an ECA to assist users in understanding his/her emotional state by prompting simple questions about it and comparing the user's self-estimation with the gathered data. In parallel, the ECA interface might confirm the users emotion by displaying a similar facial expression. Further, the agent could be utilized to augment an internet-based therapeutical chat session systems by showing empathic expressions.

1.1.1.4 Interactive Entertainment

Of course, computer games have been featuring and essentially benefitting from virtual characters for a long time. By providing direct control on the actions of a character a personal identification of the user is established. Although in most recent years users are empowered to actually control an emotional display of those avatars, those are typically not interpreted by the program, but forwarded to other human players. Non-player characters often do show emotions during cinematic sequences, however those are prescripted and therefore generally not reactions to some emotional signal issued by the player (a prominent example are contemporary multiplayer role-playing games like World of Warcraft [42], where the main attraction consists mainly in building up in-game communities with other human players rather then of socially interacting with NPCs. A single player game involving emotion display as a central concept was presented by Paiva et al by the virtual fighting game 'FantasyA' [17], where the user was providing the emotional state for his avatar, whereas the actual actions of each of the opponents were determined by a model taking into account the own emotional state at that of the opponent. To win a fight, a human player had to estimate the emotional state of the opponent and guide his own avatar's offensiveness by giving emotional impulses accordingly.

1.1.2 Criteria for Life-likeliness

This section outlines a set of properties that ECAs should present in order to be acceptable as conversation partners for the user. The set refers to Barbara Hayes-Roth's "Seven Qualities of Life-Likeliness", stated in [36](p452).

Referring to their purpose, ECAs should seem to be conversational. That means, they should engage into communication proactively on the one hand, and react to conversational acts of the user properly on the other. For example, user questions should granted with a reply, user statements should be commented. If the user fails to take initiative, the agent should provide ways to keep the conversation going such as introducing a new topic spontaneously.

They also should seem intelligent. Even if the underlying dialog system has a limited amount of semantic resources, the agent should be present deeper knowledge in at least one area of expertise. This will make the agent valuable to the user and compensate for shortcomings in other fields of knowledge.

Individuality is another important property for ECA characters. Agents should be equipped with a personal set of attitudes, likes and dislikes, emotional dynamics and a personal history. They should

display distinct behaviors that ”reveal and express their personas” any various aspects. That will support the impression of a consistent personality and therefore help to increase life-likeliness.

Other important criteria for life-likeliness are social and empathic behavior. Socialness in this context means the property to keep track of conversation partners. For each user the agent talks to, a profile should be maintained on the agents side, such that for future encounters, he or she will remember the one and therefore demonstrates social awareness. Empathy means emotional behavior. An agent should display not only a perfect shape unique personality, but also be affective to some extend. For example, an insulting utterance of a compliment should evoke some personal emotional response.

Variability addresses non-repetitiveness. An agent should vary it’s behavior continuously. Even if conversation stagnates or similar situations of talk arise there should always be subtle variation. Generally, actions should impose random, normally distributed variability in the choice and the style of their actions.

Last, characters could seem to behave coherently, that is, actions that are performed and feelings to be shown should relate to each other and give a consistent picture of the processes happening ”inside” the agents mind.

This collection of desirable properties gives a hint about the interdisciplinary nature of the research field of Embodied Conversational Agents. Major challenges are the still vague knowledge about processes in the human mind, questions about conversational functions of co-verbal behaviors like gesture or gaze, but also technical issues like speech recognition, natural language processing and, not least, the synthesization of all channels that humans use to communicate with each other, one of them, gesture, being the in focus of this thesis.

The remainder of this work is organized in six parts. The first part outlines the requirements to and possible features of frameworks for ECA behavior modeling and control. The second part identifies requirements to engines supporting ECA applications. The third part gives an extended protrait of MPML3D, the scripting language driving the agent control framework extended by this work. The fourth part proposes a suite of techniques to produce highly natural animation of ECAs. The fifth part contains a technical description of the framework and engine (being the applicatory part of this work) and part six addresses future work and concludes the paper.

Chapter 2

ECA Behavior Modeling and Control

ECA control frameworks (ACF) are attempts to model and computationally realize ways of acting that can be seen in real humans. In any case, the communication output to be generated is desired to be life-like and expressive. Expressiveness in human communication is closely related to conversational intent and emotion of the speaker. To increase believability a framework always has to deal with character internal processes.

As the internal processes in real humans are an open research field, the main challenge in developing ACFs is not only that of implementing certain algorithms in an efficient way, but also to find out about the algorithms themselves. The objective is therefore closely related to problems targeted by system simulation. The system to be modeled thereby would be the mind of a living creature. The important factors influencing real human behavior generation and their interdependency, and the way they are influenced by sensual stimuli, are to be modeled and integrated into it. The produced behavior is then evaluated with regard to naturalness and the initial conversational intent. This is an iterative process that demands for a flexible implementation to quickly realize and test new models of behavior generation.

2.1 Agent Autonomy

A distinction to be made is that of complexity of the mind's model and the level of agent autonomy. As pointed out in section [], various goals can be pursued with ECAs. A multi-agent framework investigating social processes for example needs to model agents of increased autonomy. A system like that would allow outside influence to an agents "mind" on a high level of abstraction only. For example an author might specify some parameters representing the personality and current goals of a character, but leave action planning and realization to the system at runtime. The actual plot of an application like that would be hard to predict and the result would a document of decisions taken by the agents as the plot evolved and resulting interactions. In such a setting a content author would have no low level control on which actions are performed, as those decisions would be modeled into the agents' "minds".

For many applications operating ECAs however the main target is the impression generated at the user side. Those applications focus on displaying affective behavior in order to promote certain attitudes and to elicit emotional response. In that case, a more direct control on the actions of an agent

is needed. It's "mind" then would be consulted only to choose between predefined alternatives of actions, or to modify them in order to communicate certain emotions and thus improving the naturalness of behavior. The content author would thus have a direct control on which actions are performed at each turn during the talk.

2.2 Synthetic Minds

ECA control frameworks of either type realize a synthetic agent mind. Features that are typically to be integrated into such a model are dynamic properties like emotional states, mood and momentary intent on the one hand, and static properties like knowledge, beliefs and personality traits on the other. Various models for representing internal states are available from psychological research, such as the OCC model of emotion [30] or the OCEAN model [12], also refereed to as the "Five Factor Model", being a way to describe personality with five characteristic values.

2.2.1 Input

Agent behavior in not only dependent on internal issues. In form of an agents' dedicated goals those might of course give an input to what a talk is about. However the essential thing about communication is the reaction on communicative acts and the resulting unpredictability of the plot. Input to an agents mind can be given by any objects perceptible, which can be other agents and their acting as well as signals issued by the user. Also, environmental factors can be the origin for events affecting the agents mind.

An ACF therefore usually incorporates a variety of channels by which an agent's mind percepts the world it is "living" in. Those channels might deal with utterances of other agents or the user in the first place, but also account for other types of interaction like gesture and facial displays. Environment factors could be the virtual (or real) temperature and lighting conditions of a setting. It is the responsibility of the framework designer to decide which input channels are of interest for a ECA scenario and which can be neglected. Also, of course, technical or methodological insufficiency might hinder the integration of certain channels that were of interest.

To handle input, the model for an agent's mind also includes procedures on how internal states are modified according to outside world events.

2.2.2 Generating Output

Resembling natural conversation behavior, after the agents mind has received and processed input, an output is generated. To do so, each agent needs certain channels to convey conversational acts. Regarding natural conversations, these can be channels for uttering words, controlling the intonation of words, performing hand gestures, giving facial displays, performing eye gaze, expressive head movements and body postures. The mind determines the output dependent on communicative intentions and it's actual "state of mind" and issues actions to be performed to the output channels accordingly. Of course, other types of acting might be supported as well, for example locomotion or physical interaction with scene elements.

To create the illusion of a living creature an ACF always has to incorporate that feedback loop outlined above in some way, for natural conversation as such consists of feedback elicitation. With a framework producing emotional expressive behavior randomly or without a sound model of an agents mind, the impression of life-likeliness will not be coherent at some point and the agent's believability will suffer ([36], p460).

2.2.3 Controlling Behavior

Each ACF realizes some aspects of emotional attentiveness as found in human-human-conversation by performing certain behavior patterns. In order to easily experiment with and improve the incorporated behavior models, often languages are provided, which authors use to define rules for choices that the agents take. The structure of those languages varies with the type of decisions and the level of control that an author should be granted with. The next section gives examples for control frameworks and an outline of the languages they provide.

2.3 Agent Control Frameworks and Behavior Definition Languages

2.3.1 Improv

An influential control framework for virtual characters is the Improv system, presented by Perlin and Goldberg [Perlin+others.1996]. The system is strong for multi-agent environments, where each character acts autonomously, based on a set of decision rules, that are to be defined by authors via scripts. The aim of that system was not to convey specific content to the user, but to animate users to interact with the agents in a playful way. Being a research project, a goal was to provide tools for authors to drive their own application-dependent actors by specifying custom sets of scripts providing decision rules along with a set of actions that can be performed by the actor. The basic mechanism consists of choosing from acting alternatives. providing probabilistic weighting of alternatives. For example, by scripting

```
1 { choose from {"Rock" 0.5 "Paper" 0.3 "Scissors" 0.1}}
```

an author specifies probabilities for the individual actions, a concept that combines unpredictability and behavior variety with character-dependent preferences, both important requirements for life-like agents (see section 1.1.2).

Authors developing Improv characters do construct a hierarchy of cascading scripts, which realize a multilevel action and decision process. Levels are established by organizing scripts into groups with each group representing a set of competing alternatives. Each group contains behavior scripts that are to be performed on the same level of abstractness. To give an example, a very abstract layer could contain scripts that specify behavior plans for certain phases of a day (like morning, noon, afternoon, evening). A less abstract could then specify activity behaviors such as Resting, working, dining and conversing. The higher level scripts now specify precedences, which means they define, which lower level script has which probability of being carried out. The ”morning”-script for example could define a higher priority for resting than the one representing noon. On even lower levels of abstraction scripts would contain actual actions to be performed, such as telling a joke or walking towards some goal.

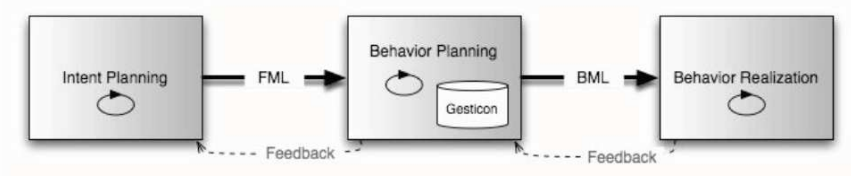

Figure 2.1: Conceptual SAIBA framework for multimodal behavior generation (from [22])

The layered action model will be evaluated iteratively and trigger distinct behaviors with respect to higher level script circumstances. To introduce personality for actors, Improv allows to specify and modulate arbitrary variables and have scripts make decisions dependent based on such. Improv provided two ways of interacting with the virtual world. One is the direct manipulation of scripts and individual agent preferences and the other was given in an VR user interface. Improv was created with a virtual theater in mind, a virtual space where experimental agent behavior patterns were easily to be incorporated.

2.3.2 BML and SAIBA

BML is a recent approach to unify a language for encoding multimodal behaviors (in the meaning of actual actions). It aims at providing a standardized interface for issuing action commands to a behavior realization module. The language is part of the conceptual SAIBA ([22]) framework, which is strong for defining a clear separation between stages of ECA-behavior realization. The framework proposes three main modules, as shown in figure 2.1, representing those stages. The first state, Intent Planning, thereby encapsulated the ”reasoning” part of the agents mind. That means, the current conversational intent of the agent is determined in that module. The second part, Behavior Planning, processes that intent and chooses sets of possible conversational actions those that are best suitable to realize the desired conversational effect. The Third part, the Behavior Realization module, encapsulated all functionality of animation and behavior display to the user. BML now is the language that defines communicative actions and thus mediates between the second and the third part. BML defines complex communicative act to be performed by an actor. For example, the structure

```
<bml>
    <head id="h1" type="nod" amount="0.4" />
    <face id="f1" type="eyebrows" amount="1.0" />
</bml>
```

defines a head-nod to be performed with an accompanying raise of the eyebrows. BML is based on XML such that in can be easily embedded into other XML-based languages. BML is limited on actions used in conversation, similar to those listed in section 2.2.2. The top-level elements are therefore closely related to parties of the body used to carry out some family of actions. Example tags are ¡head¿ for head movements, ¡face¿ for mimics, ¡gesture¿ for body movements with the upper body and ¡speech¿ specifying utterances. For most tags a type specifies the exact type of action desired and a set of parameters might follow. Actions within a BML structure are generally performed in parallel, however a mechanism to temporally align sub-actions, similar to MPML3D is incorporated. A salient

feature of the BML specification is the organizing of gesture motions into phases. This is especially useful to specify exact temporal coincidence of feature moments of several actions. The following structure demonstrates the temporal alignment feature

```
<bml>
    <gesture id="g1" type="beat"/>
    <head type="nod" stroke="g1:stroke"/>
    <gaze target="object1" start="g1:ready" end="g1:relax"/>
</bml>
```

, causing the stroke moment of the beat gesture to coincide with the head-nod.

SAIBA as such does not favor or prescribe any particular model for any of the stages, but leaves those open to individual approaches. However, by advocating a clear separation between Behavior Planning and Realization the SAIBA framework advocates shared use of resources and modules between research projects, which are often dealing with similar difficulties such as preparing large sets of animation footage. The complementing language to mediate between the first two parts of the SAIBA model, FML, specifies communicative function of acting. The separation of character behavior definition into two markup languages illustrates the multitude of layers, that real human decision making processes consist of.

Chapter 3

Animation Engine Requirements

Any framework that controls ECA actions needs modules that render the actual output that is presented to a user. Further, feedback channels have to be supported in order to make an application interactive. In this section I will give an outline of required or desired capabilities components.

3.0.3 Addressing human senses

To start with a general consideration a conversational user interface should address the human senses used in a natural conversation, which are hearing and the sense of sight. An ideal conversational interface would support both senses in both direction, that is, full audiovisual output and input.

However a more useful classification than that of human senses was given in section 2.2.2. Regarding output, today's engines and hardware are technically capable to produce behavior of remarkable expressivity. As for verbal utterances along with proper intonation, TTS systems are available that feature speed and pitch control allowing for a wide range of emotional expressivity. Hand gestures and body postures, as well as expressive head movements are managed satisfyingly with skeleton animation. Facial displays as well as facial animation for speech output can be handled by various approaches, depending on the level of realism required. For highly realistic characters, morphing and skeletal subspace deformation are suitable techniques to maintain the organic appearance of meshes during animation.

Regarding input from human users, techniques for recognizing and evaluating signals are investigated about for most channels. Besides speech recognition and analysis there are efforts to detect and classify hand gestures based on haptic hardware device input [20]. A overview to research projects and other efforts around hand and body motion recognition is available under [13]. Gaze-tracking technology is available both with head-mounted and so-called non-contact tracking devices. Gaze analysis is an active research field, however much contribution has been done concerning communicative functions of gaze behavior [43]. To gather emotional information about a user, even signals are taken into account which are not available in natural conversation, such as heart-frequency [41] or skin-conductivity [34].

3.0.4 Requirements regarding Gesture Motion Synthesis

As it is the goal of our conversational agents interface to produce as human-like behavior as possible, gesture has to be cared for to appear natural. As soft skinning and skeletal subspace deformation already provide the means to have limbs deform naturally, the main focus of this work is on animating the actual joint hierarchy properly, the configuration of the figure's skeleton. As there are rotational joints only throughout the human skeleton, only rotational transformations have to be dealt with. An exception is the global transformation of the root-joint, which comprises translation as well. (Although the techniques presented here are implemented for rotations primarily, all are applicable to translating transformations as well). The motion produced has to be smooth as it supposed to be the results of muscles accelerating limbs. C1-continuity of animation functions therefore is mandatory, except for motions that are stopped by obstacle collision, which is out of the scope of this work. The motions have to be physically possible, meaning they have to respect angle constraints of the human skeleton. Producing motions that exceed those constraints will disturb the illusion of realism or even elicit a feeling of disgust [P:unnatural bent elbow] from a viewer taking the excess for being intentional. Providing the motion is natural in terms of smoothness and angle constraints, the ultimate goal of course is making it expressive and matching that expressivity with the other modes of output in the sense of the plot author.

Besides those major requirements, human body motion has other features which are desirable to be targeted: General dynamic and Non-repetitiveness. A human figure is never completely static. Even when standing perfectly still, slight movements result from the center of weight being balanced by muscles under ongoing contraction. While sleeping, all muscles are supposed to be relaxed, however breathing and heartbeat will cause parts of the body to move or vibrate and that movement is propagated throughout the skeleton. That general dynamic is partly responsible for the non-repetitiveness of human movements. In reality it is nearly impossible for us to perform a motion twice without any variation. Besides the previously mentioned dynamics, continuous changes in the human physique are identified being a reason for that. For example, an arm gets tired when doing a movement, such that when performed for the second time the motion will be less energetic, unless that change is compensated for consciously. Complementary to that, the intended expressivity will alter the way of performing. For example, a gesture that is performed repeatedly and is associated with a verbal utterance, will vary in motion style depending on the emotional content of the individual words.

3.0.5 Examples

The most traditional technique employed for virtual character animation is that of controlling animatable degrees of freedom by motion functions given by keyvalues with associated keytimes together which some interpolation scheme. The motion is therefore represented in the time-domain, which makes it easy to author with state of the art animation tools. As this form of representation has the widest support regarding file formats (H-Anim, [14]), animation interfaces (MPEG4/BAP) authoring tools (3D Studio MAX, MAYA), realtime gesture animation relied on mere choosing from a catalogue of static prepared motions. In order to provide parametrization for gesture motions, approaches are often based on that kind of representation. An influential approach is the aforementioned Verbs and Adverbs by Charles Rose. As it is the predecessor of the approach of this work, that technique is

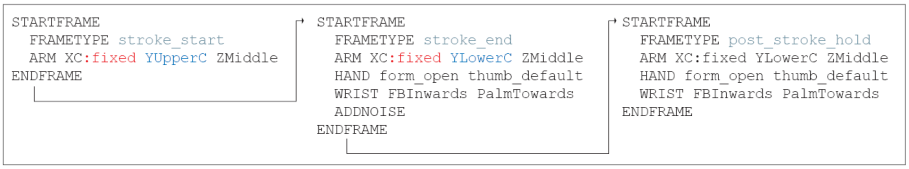

Figure 3.1: A pointing gesture, SAIBA internal representation(from [15])

explained in more detail in section 5.1.2.1.

A different form of motion representation was chosen by Ken Perlin. Perlin [32] approaches the problem of producing realtime parameterizable character animation by driving limb animation via sinus curves, thus representing motions as amplitudes and phases in the time domain. To specify a particular motion, a touple of three vectors is given for each animated joint of the figure. The first two vectors specify the minimum and maximum angle of rotation around the three axes respectively, whereas the third chooses the interpolant function for each axis. For example, the line

```
{15 0 5} {-15 0 -5} {c1 0 s1} RChest
```

results in the rotation around the x-axis of the joint named 'RChest' to be constrained to the interval $[-15, 15]$ and $[-5, 5]$ for the z-axis rotation. $c1$ and $s1$ are periodic input signals evaluated at each frame. To introduce some randomization of motion, a set of noise-signals are provided to be used as interpolant functions.

Perlins representation of motion provides many opportunities for motion customization. By specifying and modulating a bias to the input signal functions for each joint individually, motion style can be varied slightly and different emotive states can be expressed. For example, bias applied to knee joints as given by the lines

```
{0 0 0} {130 0 0} {s1 0.2 bias 0 0} RKnee
{-45 0 0} {45 0 0} {s1i 0.7 bias 0 0} Rankle
```

"will give the bearing knee a little extra kick at the time it swings forward". By continuously adjusting those bias values during animation, the motion style is be adjusted smoothly and changes in mood can be expressed.

Transitions between motions are realized with weights. Every animation is associated with a weight, and a transition is performed by simultaneously decreasing one actions's weight while increasing the weight of another action, using s-shaped interpolation functions. To prevent unnatural movements, transitions are only allowed at certain intervals of a motion and with the two blended motions being temporally aligned.

A more recent approach that focusses more on actual conversational body motion is presented Hartmann et al [15]. The work extends the GRETA agent architecture [16]. The technique focusses on expressive hand gestures, therefore addressing the arm skeleton joints only. The motion representation is done by specifying the position of the wrists along with a hand configuration. Position is given in coordinates relative to dedicated regions of performing (gesture spaces) and the temporal profile is given with regard to typical motion phases (gesture phases), both concepts defined by McNeill [27]. An example script describing a pointing gesture can be seen in picture , where the lines starting with

FRAMETYPE specify temporal alignment with motion phases, those starting with ARM determine the relative position of the wrist and the lines beginning with HAND define the hand configuration. The actual arm motion is realized with a Kochanek-Bartels-spline used to animate the wrist position, from which a Inverse Kinematic module computes actual joint orientations, whereas the hand configuration is given by plain forward kinematic DOF configurations.

To realize parametrization, a set of motion characteristics is defined, which are Overall Activation, Spacial Extent, Temporal Extent, Fluidity, Power and Repetition. Those parameters are chosen to cover a wide range of expressivity without referring to any specific emotions, similar to what the EMOTE framework does. By doing so, a separation between behavior planning and realization is introduced, paralleling the modular approach proposed by the SAIBA framework 2.3.2. When the actual motion function is generated, the motion parameters are used to move the control points of the spline spacially as well as temporally, for each parameter applying an individual technique. The Spacial Extend parameter for example is applied by scaling the coordinated frame of the associated gesture space, which results in limbs to sweep wider or narrower areas and thus motions to take place in variable size volume. The Fluidity parameter on the other hand is applied by varying the continuity parameter of Kochanek-Bartels-splines [21] used in the interpolating component.

Arm movements are specified by the essential part of the motion only, an approach that is adapted in this work. Working with spline curve motion control, smooth transitions between gestures are realized in a straightforward way by connecting the individual gestures' curve segments to form a continuous spline curve.

An important characteristic of that approach is that motions are sourced from abstract descriptions, which facilitates the generation of new motions while leaving more responsibility about the produced motion to the underlying model of motion synthesis (based on McNeill's work in this case). Further, the decoupling of arm and hand movement results in a greater variety of available basis motions.

Chapter 4

MPML3D

MPML is a family of languages addressing the needs of researchers to script behavior of computer controlled entities not only for virtual environments, but also physically represented agents like robots. It was first introduced by Ishizuka in 1999 and has undergone several incarnations in related research projects. One of the variants developed was MPML 3.0 being a scripting language for interactive plot in virtual presentations carried out by ECAs. The language included tagging structures for synchronization of multimodal communicative actions. As reaction on user feedback was still cumbersome to be integrated, in 2006 the syntax was extended with tags to incorporate anytime interruptibility, relying on a reactive framework suggested by Nischt et al, resulting in MPML3D [28]. Parallel to revisiting the salient features of the syntax as well as completing the control framework implementation, the language has been extended about in 2007 about agent state parameters, which serve as way to script personalized and emotional behavior for the employed entities (agents) without having to deal with high-level programming languages. MPML3D is based on XML, which provides ease of use for authors on the one hand and a clear document structure and freely available parsing tools on the other.

As MPML3D in it's current state is most suitable to exploit the animation techniques presented in this work, an introduction to the language is given in the following sections.

4.1 Outline

The MPML3D syntax allows an author to specify a scene in which a plot is taking place. This includes the description of static scene elements as well as acting entities, which will perform actions that make up the actual plot.

As the plot may be nonlinear, it is given not in a single sequence of acting instructions, but in initially independent segments of content. Those segments are called 'tasks' and define complex actions meant to be performed by acting entities "in one piece". In communicative presentation plots these tasks contain snippets of dialogue which consist of complex communicative acts.

To provide means for non-linearity, transitions between those segments are defined. The transitions to be taken (as well as the timing) are determined according to the evolving of the plot. Segments of content can either be concatenated in a fixed manner, or they can follow each other based on decisions. On top of that, decisions to change from a currently performed sequence to some other can be defined.

Figure 4.1: MPML3D presentation agents in traditional Japanese (virtual) Environment

The paradigm of MPML3D does not prescribe for particular types of acting entities. The language as such is an instrument to trigger actions to be carried out by scene entities according to certain events. Events are issued by actions or by changes in entity states. Which types of action are available and which states are existing for a certain entity type is dependent on the actual MPML3D player. Each player might expose its own types of entities with their own capabilities. Furthermore, each player can implement its own channels for user feedback which in turn might change dedicated entity states.

The actual modes addressed by the ”Multimodal Presentation Markup Language” are application dependent and may include output channels (actions) as well as input channels (user feedback). Of course, as the target of our research are interactive presentations with ECAs, the MPML3D player developed at the NII supports human-like virtual characters and actions for communicative acts like speech, gaze and gesture.

4.2 Actions

Actions are the basic construct for any activities that can be carried out by scene entities. Action can depict a concrete action such as a body movement or a speech act, in which case the tag’s content is always a command interpreted by the player:

```
1 Action> yuki.speak("Hello, my name is Yuki.") </Action>
```

’yuki’ is thereby the identifier of an entity, associated with the entity type ’human’, which in turn is registered to accept a command ’speak’. The String-parameter of course specifies the sentence to be uttered.

To allow more complex communicative acts than mere speech, MPML3D allows the aggregation of actions via so-called container-actions, which are ’Parallel’, ’Sequential’ and ’Selected’. A Parallel element will cause the contained actions to be started by the player simultaneously. For example, the structure

```
<Parallel>
    <Action> yuki.speak("Let's talk about this tomorrow.") </Action>
    <Action> yuki.gesture("Invite") </Action>
</Parallel>
```

will cause agent yuki to open her arms while speaking. Similar, actions contained in a 'Sequential' tag will be performed sequentially. By nesting Container action tags, the author can script entire dialogs involving multiple entities with each performing multiple concrete actions in parallel. Of course this would not introduce any interactivity to the plot. To do so, MPML3D provides Perceptions.

As temporal alignment between for example speech and gaze is a very important feature of human conversation, MPML3D provides "subaction synchronization" for concrete agent activities. Within a parallel action tag, each concrete action might refer to the others to be started and stopped on certain sub-events. For a speech action these are the single words of the utterance. To give an example, the structure

```
<Parallel>
    <Action name="yukisQuestion"> yuki.speak("Now what do you think, ken?") </Action>
    <Action name="yukiLookAtKen" startOn="yukisQuestion[4].begin"> yuki.gaze(ken) </Action>
</Parallel>
```

will result in the action named 'yukiLookAtKen' being started on the begin event of the fourth subaction of the action named 'yukisQuestion' (which with some player might be the fourth word in the sentence).

4.3 Tasks and Perceptions

Perceptions are comparable to event listeners. They are attached to certain states of agents and will be triggered on a specific event to occur on that state. For example, the line

```
<Perception name="yukiSaysTomorrow"> onEvent(yuki, "saysWord", "tomorrow") </Perception>
```

describes a perception that will be triggered as soon as the agent yuki utters the word 'tomorrow', i.e., the state "says" of human entity "yuki" matches the String parameter "tomorrow". To introduce interactivity, perceptions can be designed to start certain pieces of the presentation, called 'tasks'.

"Tasks" are special type of action container. Actually, they behave like sequential action containers, with the exception that each task can be started upon perceptions to be triggered. For example, a task that might be specified by the structure

```
<Task startOn="yukiSaysTomorrow">
    <Parallel>
        <Action> ken.speak("yuki! Why not talk about it today?") </Action>
        <Action> ken.gesture("Shrug") </Action>
    </Parallel>
<Task>
```

will be started as soon as the perception name 'yukiSaysTomorrow' triggers. Tasks are the only actions that an author may explicitly start by script commands. Tasks are the major parts that a plot consists of, similar to chapters of a book. Tasks may represent single utterances or a long dialogue with multiple turns between agents. All that is happening in a presentation scripted via MPML3D is tasks being performed and preemptively stopping each other.

4.3.1 Task managers

Of course, starting tasks upon perceptions might cause multiple tasks to be performed concurrently. To prevent tasks that might interfere with each other from being executed in parallel, tasks are associated with a task manager. A task manager maintains a priority queue of tasks currently being started and thus ready to perform, a mechanism similar to a process scheduler in an OS. Each task is equipped with a priority value which decides if it will interrupt the currently executed task or not. When a lower priority task is interrupted, it is stopped and waits for all higher priority tasks to be finished before it will be started again. To enable independent performing of tasks for multiple scene entities, the framework provides a task manager for each entity along with the global scene task manager.

Task Managers allow an author to specify layers of behavior for agents. For example, a pattern of general human activities like sleeping, eating and working might be realized by a low priority task.

4.4 State Parameters

Whereas each entity type usually comes with a built-in set of state variables, author may define additional entity state parameters to realize specific agent behaviors. For example, a personality profile like the OCEAN model 2.2 might be represented via float parameters. Those parameters can then be considered by perceptions or script functions when making decisions about scene flow.

```
<StateParameter name="extraversion" type="float" value="0.8"/>
```

The above statement will register a state parameter representing the personality property 'extraversion' of an agent. That property can later be queried by the condition element of a perception and the character might behave different accordingly.

As state parameters can be altered during a presentation, time-variant entity properties like emotional states can be represented as well. This way the author could control an agents behavior according to his or her mood and choose between several options for some part of the dialogue. As another feature, state parameters can be directly wired to motion controlling variables (so called 'adverbs' which are explained in detail in section 5.1.2.1 of this work).

```
<StateParameter name="arousal" type="float" min="0.0" max="1.0" default="0.3">
    <BindParameter type="gesture" name="beat" slot="energy"/>
</StateParameter>
```

The above statement for example wires the entity state parameter 'arousal' to the adverb 'speed' of the gesture 'beat'. When that gesture is started later during a running plot, the actual amount of arousal of the agent will be reflected in the style the motion is carried out.

Agent state parameters can be changed by a dedicated action command. By specifying

```
<Action>yuki.setStateParameter("arousal", 0.9, 2.0)<Action>
```

within an action construct, the author will cause the emotional 'arousal' state of agent yuki to reach a value of 0.9 after 2.0 seconds.

As author-defined entity state parameters enable to incorporate personality and mood-dependent behavior, they are the essential new features to the language in order to make MPML3D-controlled presentation agents seem more emotionally (and thus personally) involved with the presentation topic

and the user. Their usage is available to those researchers not familiar with high-level programming languages and therefore provide a means to quickly incorporate emotional models to a broader audience.

4.5 User Feedback

Using state variables and the concept of perceptions, plot attentiveness to user feedback can be modeled in a straightforward way. By concept, each signal that is captured from a user is to be stored to state variables of the special entity type that represents the user within the scene. Whichever input channel the MPML3D player might support, the user entity stores the values of interest extracted from these channels in dedicated state variables, which are accessible for perceptions while the plot evolves. For example, a speech recognition system might store each word extracted from user utterances to a variable 'saysWord' and a perception could listen for certain words. Another examples are the extraction of the emotional state of the user from certain biosignals or gesture detection. By doing so, agent behavior can be scripted to be attentive to user affect and thus resemble an essential feature of human face-to-face conversation for ECAs.

4.6 Scope of MPML3D

The goal of MPML3D is not to provide or prescribe any specific model of conversational behavior generation, but presents a general but easy-to-use interface to realize arbitrary behavior models within interactive conversational plots. The specific goal MPML is designed for of course are interactive presentations. Presentations typically have a predefined storyline, which might be adapted to the needs and interests of the user, but not in the overall content. The layered task model and the preempting functionality realized by task managers clearly anticipated that fact by supporting a merely linear main plot, which could be interrupted by subsidiary tasks to resume afterwards. Typically those subsidiary tasks were used to regain the attention and engagement of the user, in case his/her interest shifted away from the presentation. To incorporate secondary motion typically a layer with lower priority tasks was scripted, which contained only idling actions, such as slight changes of the pose or breathing. However scene flow is not required to be that linear using MPML3D. The Task and Perceptions model provides the means to freely switch between task, similar to states of a finite state machine defining conditional transitions between states. Decisions about scene flow can be made on the basis of author scene state variables and can also be supported by an embedded scripting functionality supporting javascript and other languages in order to realize more complex decisions. The scope of MPML3D is therefore not limited to Interactive presentations, but addresses conversation-style HCI-applications in general. Compared to other scripting languages for ECAs MPML3D provides low-level control on agents, in the meaning that speech acts can and must be prepared by authors explicitly. MPML3D is thus suitable to script dialogues and multiagent settings with focus on conversation, but less to realize autonomous agent behavior.

Chapter 5

Gesture Parametrization, Transitions and Blending - Extending a Standard animation Engine

In this fourth part of my work, I describe how features desirable for control frameworks for highly realistic ECAs as characterized in section 3 can be realized by an animation engine.

The following sections presents techniques to address the aforementioned issues. Section 5.1 describes how to realize parametrization in order to control motion expressiveness and to reduce repetitiveness. Section 5.2 proposes a generalized approach to spontaneous switching between gestures and section 5.3 discusses ways of improving motion output and restrictions to performing several animations at once.

5.1 Gesture Parametrization

Making ECA gesture parameterizable improves the output in two ways. First, it offers the possibility to perform the same gesture several times with some variation, increasing the naturalness of motion. Working with a limited set of animation footage, this becomes crucial in order to prevent the audience to be amused or the user's attention on the subject to fade. The more important gain in parameterizable motions however is the widened range of expressivity that is opened up. Not only the choice of a certain gesture might control the express some internal state of the agent. The user thus might recognize some motion he has watched before, but notice a change on a more subtle level: The beat carried out with the hand has become a bit stronger, expressing the increasing arousal of the character. Continuous parametrization of motion accounts for continuous state parameters of an agent and is therefore the choice for highly natural gesture synthesis.

5.1.1 Dimensionality of Body Expression

As pointed out in section 1.1.2, a believable conversational interface agent is desired to have its own internal states and to communicate those to the user. Gesture parametrization therefore is a necessity resulting from some parametrization existing in the agents mind. Although those states are supposed

to affect each other they can be seen as dimensions of state-space, in which the current state of an agent is represented as a vector whose components are the individual state variables. Which states are to be modeled for a control framework depends on the aim of the application. Possible internal states could be of psychological nature like personality, mood and emotion state, but also physical states such as weariness. Further, propositional gestures are another reason the introduce gesture parameters. A pointing gesture for example might accept the direction which to point to as a parameter. In that case, the variable refers to some object anticipated by the agent and can be considered as an internal state as well. An animation engine should therefore provide the means to have gestures performed according to parameters that reflects the internal state of the agent.

5.1.2 Realization of Parametrization

How (for example) some emotion is to be communicated by gesture is an open research topic. Many approaches rely on mere choosing the appropriate co-verbal movement from a catalog of predefined animations. In this work however, the aim is to empower a researcher to define his own parameters for both gestures and internal agent states, and to freely choose a mapping from one to the other.

In order to do so, I adapt the approach ”Verbs And Adverbs” by Charles Rose et al [38], where samples of predefined animation data are blended to produce new of motions. In the following I will first explain how that approach works and then describe my adaption for conversational gesture animation.

5.1.2.1 Verbs And Adverbs

V&Adv is a method to generate new motions for virtual characters from existing ones on a frame-by-frame interactive basis. The V&Adv concept introduces a graph containing nodes that represent available basic motions, called “verbs”, and legal transitions between them. For each basic motion, several variations (motion samples) exist, representing nuances of motion styles, which are called “adverbs”. For each verb an expression space is defined, which the adverbs represent the dimensions of. The adverb space of each verb is populated with the available sample motions, each of which is individually positioned according to it’s expression profile. In addition to smooth transitions between verbs, the technique offers a way to produce interpolated motion output based on that adverb space. Although it requires for motion variants to be similar in anatomy, it allows for samples with differing keyframe spacing as well as differing overall duration. Unlike other approaches, it is not based on the frequency domain. It is based directly on keyframes in the time-domain, and thus integrates processing of non-periodic motions in a straight-forward way.

Similar to this work, V&Adv focuses on skeleton animation. It works with a reduced human joint hierarchy resulting to 46 degrees of freedom (DOF) in total. A verb thereby consists of a set of motion functions, one for each DOF, which are treated independently from each other.

To realize parametrization, Rose and al utilize a set of sample motions for each verb. Those sample motions are blended using weights that depend on the actual input parameters of the motion, the so called ”adverbs”. An adverb describes a continuum between two extremes, such as happy and sad, knowledgable and clueless, but also more physical variations like uphill and downhill (for a walking movement). Each adverb is a variable that stands for a certain dimension of expressivity. Each motion

can be equipped with arbitrary adverbs, which then span the expression-(or adverb-)space for that motion. Each motion sample is annotated with a vector that characterizes it's expressiveness with regard to the available adverbs by specifying a position in the adverb space. The sample motions thus populate the adverb space.

5.1.2.2 How motions are blended

Verbs and Adverbs employs uniform cubic B-Splines curves as a representation for the DOF-functions of the samples. The key concept of the blending mechanism now is such that not the individual sample functions are evaluated first in order to blend their results afterwards. For each degree of freedom and each spline-control point the multidimensional interpolation function is precomputed considering the values of the respective control point in each sample motion. When the animation is performed, that interpolation function is evaluated for a number of control points and the result is a blended motion curve. Finally sampling from that curve using the current time index yields the final value for the DOF.

The interpolating function is a combination of a simple approximation method and a refinement to interpolate the actual sample position values. The approximating function thereby describes a hyperplane placed in the adverb space according to an ordinary least squares solution to best fit the variations across the sample values. The refining is done by employing radial basis functions $R_i(p)$ for each sample value, consisting of a dilated cubic B-Spline $B(d/\alpha)$, with the dilation factor $(1/\alpha)$ individually chosen to limit the support radius of each basis to twice the distance to the nearest other sample motion. Being j the index of the DOF, k the index of the B-spline control point and l the index of the adverb, the interpolation function for control point b_{jk} is given by equation (5.1),

$$b_{jk}(p) = \sum_{i=1}^{NumExamples} r_{ijk} R_i(p) + \sum_{i=1}^{NumAdverbs} a_{jkl} A_l(p) \tag{5.1}$$

where $A_l(p)$ are linear basis functions and a_{jkl} are the calculated coefficients for the approximating hyperplane.

Although the authors stress that the representation of sample motion functions is independent from the interpolation technique, advantages of the approach arise from the usage of uniform B-splines. First, the interpolated motion functions can be used as long as adverb settings do not change. Furthermore, using cubic splines causes a maximum of only four control points (of each sample motion) to be of interest (per DOF) at a given time.

To produce meaningful results, Verbs and Adverbs imposes some restrictions to the sample-animations footage. The motions have to be similar in structure. For example, a set of walking motions must start on the same foot, have the same arm swing phase. Generally speaking, the phases of each sample motion must refer to the phases of the other sample motions of a verb. To account for these phases during animation, each sample defines a set of keytimes stating it's realization of those phases. Similar to the B-spline control points, these keytimes are interpolated to obtain a blended phase profile. That profile is then used to compute a generic time index t that reflects the current phase and time factor within that phase for all samples. Equation (5.2) shows how the generic time index is computed from global animation time T and interpolated Keytimes K_i.

$$t(T) = ((m-1) + \frac{T - K_m}{K_{m+1} - K_m}) * \frac{1}{NumKeyTimes - 1} \tag{5.2}$$

The generic time is thus a value in the interval $[0, 1]$ and is used as input parameter for the interpolated B-spline curves.

Although Verbs and Adverbs is applicable for other motion representations than B-Spline curves, the approach requires all sample-functions to have the same number of coefficients. Without this precondition, the interpolation functions for control points cannot be determined at initialization time. In contrast to that, our application field targets authors that want to benefit from footage prepared with attention to subtle motion features. To order equal numbered and structurally aligned keyframes across sample motions for a verb is a strong limitation for an animation artist, who tend to successively insert and delete keyframes wherever a motion needs refinement. Another possibility is the ex post insertion of missing keyframes. However, matching semantically related, but temporally separated keyframes of samples to determine for missing ones again requires the expertise of an animator and is hard to automatize.

5.1.2.3 Adaption of the Original Approach - Parametrization

Verbs and Adverbs has been chosen as a starting point for this work, because it targets similar animation goals. Motion functions have to be smooth and computed in realtime according to continuously changing parameter settings. It requires only a minimal set of motion samples compared to other approaches [Wiley and Hahn, Guo and Roberge]. It moreover allows an author to locally refine the adverb space population by inserting new samples at arbitrary positions and allows for interpolating points outside the convex hull of sample positions. Also, the approach is based on motion represented in the time-domain, which facilitates non-periodic movements which are typical for conversational gestures. However, some restrictions of V&Adv are not acceptable for the application of this work, such that I adapt the original technique in several ways.

First, V&Adv assumes that motion samples share the same number of coefficients. This facilitates the use of the same set of interpolated control points as long as adverb values remain unchanged and thus reduces computational cost. However, when using samples with a differing number of control points, preprocessing is necessary and might result in cumbersome adjustments to the animation footage. Automatic insertion of missing control points might result in faulty inter-sample control point associations. Therefore, as computing an interpolated function as proposed in [38] is not feasible, a more basic approach has been chosen.

In this approach not the individual control points are interpolated, but the results of the sample functions. The representation of animation footage is therefore completely independent from the interpolation technique. According to Verbs and Adverbs, the sample motions are located in the adverb space. Then, dependent on current adverb settings a weight is computed for each sample motion, which applies to all DOF-functions respectively. Being evaluated for some t, the values of all sample motions are summed according to that (normalized) weight distribution, such that the result will reside within the convex hull of the sample motions results. By doing so, unnatural joint configurations are avoided.

To compute the individual contribution of the sample motions, we apply inverse distance weighting according to Shepard's Method [39] on the sample locations in the adverb space. Shepard's Method was originally presented by Donald Shepard in a 1968, being an interpolation scheme for 2-dimensionally scattered sample-data. As noted in the paper, it can easily be extended to higher dimensional space and a basic variant of it is adapted here to interpolate between sample locations of arbitrary dimensionality. In the following, a short outline of the applied technique is given.

Shepard's method faces the problem of providing an interpolating function for scattered data points. That means, for some domain which a finite number of irregularly spaced sample points D_i with associated values z_i is given, a continuous function is to be found to estimate the values of arbitrary locations P within that domain. This is not only useful to compute intermediate data points, but also for computing extremes or gradients. The original algorithm comprises a set of stages, each of which refines the shape of the function.

The first stage considers the distance d_i of data points to the interpolated location only (equation (5.3)). This results in certain shortcomings concerning direction dependent influence of nearby data points.

$$f_1(P) = \begin{cases} [\sum_{i=1}^{N} (d_i)^{-2} z_i] / [\sum_{i=1}^{N} (d_i)^{-2}] & \text{if } d_i \neq 0 \text{ for all } D_i \\ z_i & \text{if } d_i = 0 \text{ for some } D_i \end{cases} \tag{5.3}$$

A second stage limits the number of considered data points to numbers derived from the dimensionality of the sample domain. As the number of samples in our application is very limited, we skip this particular improvement.

An important improvement is consideration of direction of data points in relation to P. This stage introduces a directional weighting term is computed for each data point D_i, that considers "shadowing" effects between data point pairs D_{i_1}, D_{i_2} dependent on direction as seen from and distance to P. Other extensions of the function aim at improving slope at locations close to sample data points, computational error and the simulation of barriers. As we use the computed weights for blending motion samples which hide certain shortcomings of the interpolation function, for our approach we are satisfied with the two explained stages.

5.2 Transitions between Gestures

5.2.1 Desired features of gesture motion synthesis

Human gesture motion apparent as co-verbal actions enhancing conversational bandwidth in human-human interaction has certain qualities, that are to be produced by a animation engine for ECAs. Some important are continuity, non-repetitiveness, spontaneity, expressivity, temporal alignment (to speech) and spacial alignment. The two features that are timing issues, temporal alignment and spontaneity, are those addressed by this section and the transition technique presented here. Temporal alignment refers to the fact that gesture is synchronized with speech in most cases. To have the desired effect, motions have to fit exactly to the phonemes they are supposed to support. If that synchronization is bad, the effect might either be just zero, or it might stress the wrong passage in a sentence and have the

opposite effect in the worst case. Spontaneity refers to the same fact, but emphasizes that a change in mind can happen spontaneously during a conversation, and therefore speech output along with gesture are amenable to anytime interruptions.

5.2.1.1 Motion types

As the gesture animations are applied to the entire skeleton each, all body motions (not only hand gestures) are displayed with the same mechanism. A ruff classification of motion types refers to what the essential part of the gesture is. I refer to those gestures as a *postures* when the essential part of the motion is a static phase, one with the expression given by the pose of the skeleton. When a certain moment in a motion sequence can be identified as the climax, I refer to them as *culmination gestures*. Beat gestures [9] are the prominent example for this type. The third type I call *motion gestures*, which the expressive content is represented by a motion as such. An example for this third type is a rolling hand movement to express some kind of development [9].

This classification makes is useful as the types offer varying processing possibilities: Postures may be represented by motions to and from the essential configuration only or, if the transition synthesis is satisfying, as a mere configuration. In both cases the duration of the hold-phase can freely be prolonged as desired. Culmination gestures offer the possibility to perform only a short interval based on keyframed animation and compute inter-gesture transition to following gestures dynamically, as will be presented in this section. Finally, motion gestures often have a periodic nature such that they are likely to be performed in repetitions. This, of course, opens the way to arbitrary concatenations of that motion cycle with itself, resulting in seamless motion if the footage is prepared accordingly.

5.2.1.2 Prior State

The state of the animation engine prior to this work provided the simple concatenation of gestures. To ensure C0-continuity of DOF-functions (i.e. no jumps in joint-motion), gestures always had to be performed completely, which led them back to the normal joint configuration with arms hanging besides the torso. Although the animation footage declared phases of motion already, those where only used to determine if the essential part of the motion was finished yet when a gesture was stopped prematurely. The presented techniques take advantage of phase annotation in several ways. On the one hand it is used to align motion sample for blending as described in section 5.1. The other way is presented in section 5.2.3.

5.2.1.3 Long Term Goal

An effect that can be watched in people using many gestures is that of *gesture flow*. Single gestures are hardly to be segmented from the stream of motion and the communicative content of the individual movements merges into each other. The production of such a gesture flow for interactive realistic ECA applications is the long-term goal of this work. The concatenation of essential gesture phases with smooth transit motion inserted in between is the first step in that direction.

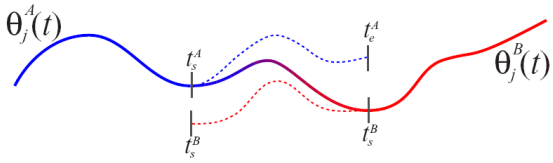

Figure 5.1: Linear fade between motion functions (from [38])

5.2.2 Verbs and Adverbs - Transitions

V&Adv focuses a wide variety of motions including periodic ones like walking and dancing as well as movements performed with arms only while the character is standing. In Verbs and Adverbs, the verb graph defines at which phases of an animation transitions may happen. Motions are transited between each other by determining a dedicated phase where transitions are legal and by performing a linear interpolation from one motion output to the other. Figure 5.1 illustrates that technique, where a linear fade between the motion functions $\theta_j^A(t)$ and $\theta_j^B(t)$ is performed. Provided that the phases are defined properly, this ensures a general smoothness of motion, but does not particularly care about motion expressiveness on a fine-grained level during a transition. An essential property of this procedure is that both of the motions that is transited between are contributing to the motion and thus a mixed expression might be visible during transition time.

Our application deals with the more specific class of motions found in co-verbal gesture behavior. The goal of motions is therefore less a mechanical one (like with walking), but to express certain concepts along with speech and to give hints on the performer's attitude. Especially the latter being fulfilled by only subtle variances in movement. For human gesture is undesirable if not disturbing to have a such mix of expressive "statements" during a transition as produced by simple blending of verbs is not satisfactory in our case.

An even more critical limitation to the Verbs and Adverbs technique is the aforementioned legal transition phases. For conversational behavior spontaneity is an essential feature. Being responsive to user feedback means that the actions to be performed by an agent might change at any point in time, just as the mind of a human might change on unexpected happenings during a talk. The controlling framework might request the animation engine to break the current gesture and return to some idling pose or start a new gesture immediately. For an application that aims for a maximum of naturalness in conversational behavior, it is not acceptable to wait for some legal transition phase to be reached before transiting.

For these reasons I advocate a different approach based on dynamic transition curve generation.

5.2.3 Dynamic Transition Curves

5.2.3.1 Outline

In the technique presented here, the typical transition situation is a gesture that signals the end of it's performing phase and the engine finds another gesture queued to be carried out subsequently. As only

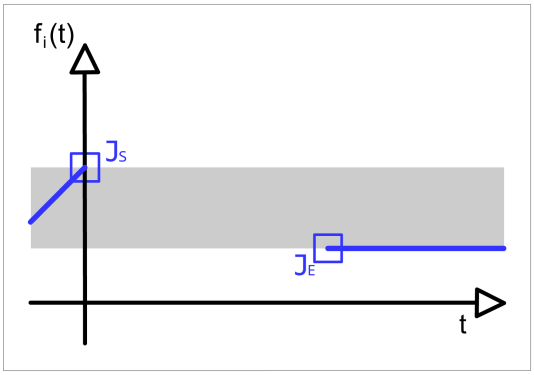

Figure 5.2: Motion function pieces and junction points

the performing phases are of interest, the end of that phase of the predecessor animation has to be connected to the beginning of that of the succeeding one. This is done by constructing a transitional motion curve to gain a smooth transducing movement in between.

The ”end” and ”beginning” of an animation are hereby considered to be configurations of the animated skeleton, named uppercase C as follows. The evaluation of the motion functions of all DOF of an animation at a given time t results in some configuration $C(t)$. The footage of each animation defines a start and end times for it's phases. Analog to Verbs and Adverbs I handle all DOF-functions independently. This works well for translation vector components and rotations represented as euler angles. However, as componentwise linear interpolation of rotations represented as quaternions (as is the case in our framework) does not produce the desired results (see section 5.2.3.3), rotation-related DOF are grouped in four-tuples and processed according to quaternion algebra. In the following i will speak about channels or ”channel groups” - which refer to translations or or rotations - and state formulas *as if* DOF were treated independently. Explanations about special rules applying to quaternions are given in a dedicated section (section 5.2.3.3).

Whenever a transition is required, two skeleton configurations C_S and C_E are apparent, which a transition is to be found between. To construct transition curves, I chose bezier curves of grade 3. This is a tradeoff between computational efficiency and smoothness of motion at the begin and ending point of the transition, referred to as the junction points J_S and J_E from now. J_S and J_E are associated with C_S and C_E, but refer to a temporal context, whereas configurations describe a mere state of the skeleton. An example scenario of a transition situation for some DOF-motion function $f_i(t)$ is shown in picture 5.2, where the two function pieces address the same degree of freedom, but belong to different animations. The curve is fit in similar to a Bezier-spline segment between two interpolated control points of a bezier spline. I.e. the first and the last control point b_0 resp. b_3 of the fitted curve coincide with the junction points J_S and J_E of the connected animations. To gain C1-continuity, the intermediate control points b_1 and b_2 are positioned in a way that the first derivation of the animations at the junction points is resembled by the transition curve 5.3. As I account for the first derivation of a animation when computing a transition in this approach, a skeleton configuration always provides velocity information, too. To refer to the value at a junction point, I will use C^0 and to refer to the first derivation of it, C^1.

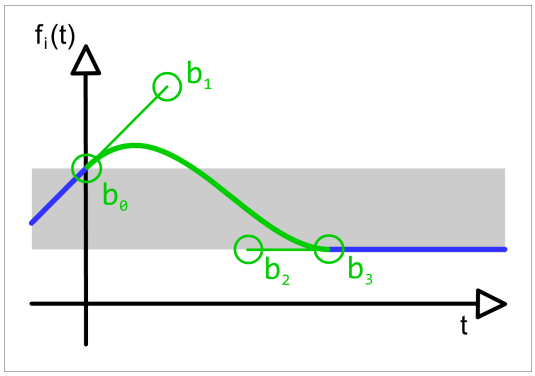

Figure 5.3: Fit in bezier curve with C1-continuity

Figure 5.4: Unnatural joint configuration due to wrong transition timing

5.2.3.2 Transition Timing

Given any pair of animations to be transduced across, only the configurations as such are given, but not the time that may pass until the succeeding animation is reached. In order to maintain C1-continuity, any transition duration greater than zero is possible. The animation engine therefore has to compute a duration that will make the transition look natural and does not destroy the characteristics of the motion. A wrong duration might produce an intermediate motion speed that contradicts to that of the connected gestures, it might result in unnatural movements and can easily lead to impossible joint configurations (picture 5.4).

In my approach, I compute the time necessary for a transition by the distance between the junction points. For rotations, the angular distance between the two orientations is considered. Therefore the duration for a transition curve reflects the actual distance to be bridged as well as the velocity at the two junction points. Particularly, the duration d_i for some DOF is computed by

$$d_i = \frac{((b_1 - b_0) + (b_2 - b_1) + (b_3 - b_2)) + (b_3 - b_0)}{2} * s_d \tag{5.4}$$

with the $b_j, j = 0...3$ being the control points positioned for uniform curve parametrization and s_d being a general duration scaling constant. This formula computes the average between the upper and the lower bound of the curve length given by the curves control polygon. As a result, the motion during transition will resemble the speed of the connected gestures. On the other hand, if a gap is small but the velocity quaternions differ in rotation axis and/or sign, the duration will be prolonged accordingly. A more advanced variant of this algorithm accounts for the mass of the joints as well. Accounting for heavier bones (like forearm and tibia) afford more time to change angular velocity, the duration time computation can be extended to formula (5.5).

$$d_i^- = \sqrt{d_i * l_i} * s_d^- \tag{5.5}$$

with l_i being the length of joint i. By doing so, the finger joints might return to their idling orientation far quicker then for example the upper arm joint.

The main idea behind transition timing is producing a curve with proper shape for each joint. In particular, this aims at not exceeding any constraint intervals imposed by the nature of the human skeleton. These constraint interval is represented by the unknown maximum and minimum value I_{max} and I_{min} for each DOF. As our skeleton representation does not provide information about such, the curve generation algorithm is designed to implicitly estimate those constraints when determining proper durations. A hint on the natural constraint interval is given by the interval given by the two junction point values C_{Si} and C_{Ei}, which can be assumed to always be completely contained in the prior one. Working on bezier curves of grade three, a reparametrization (according to some duration) affects the intermediate control point values (which have to be repositioned to maintain C1 continuity) and therefore affects the extent that the curve might exaggerate interval $[C_S, C_E]$. By choosing the duration properly the probability for an exaggeration of the natural constraint interval $[I_{max}, I_{min}]$ is minimized. More details on that are given in section 5.2.4.1.

Of course, individual durations cannot be applied for each joint, as a transition must always have the same duration for the whole skeleton. The calculations stated above are used to determine the longest duration time d_{max} needed throughout the hierarchy for a given transition. d_{max} will then be used for all channels, such that all of them will "arrive" at the junction configuration at the same time. The individual durations calculated however are not discarded. They are important because they provide a good estimation of the optimal transition curve shape of each joint. A variant of the curve generation algorithm making use of those individual durations is $C1ALS$ pointed out in section 5.2.3.4.

Once the duration for a transition is determined, the intermediate control points b_1 and b_2 can be computed. Formula (5.6) states how intermediate control points are positioned. Thereby multiplying by d_{max} compensates for the reparametrization of the bezier curve.

$$b_1 = C_S^0 + \frac{C_S^1}{3} * d_{max} \tag{5.6}$$

$$b_2 = C_E^0 - \frac{C_E^1}{3} * d_{max} \tag{5.7}$$

5.2.3.3 Quaternion Bezier curves

Quaternions are a way to represent rotations in three-dimensional space. Quaternions are made up of four components, three of which represent a rotational axis and one specifying the rotation angle. Each quaternion can be taken like a position in four-dimensional space, however, to make them useful for representing rotations, a special algebra applies to quaternions (that are actually an extension of complex numbers). Quaternions are a compact way to represent and calculate orientations. Especially interpolation between two orientations (an operator called "slerp") provides a more robust means to generate intermediate orientations and produces more "intuitively" correct results compared to interpolation of euler angles.

For these reasons we decided to use quaternions and quaternion algebra for rotation animation and interpolation. This has the advantage that when blending orientations that lie close to each other, linear interpolation between the quaternion components can be used, whereas for transition curve computation, a convenient representation especially for rotation derivatives is available.

Due to the specific quaternion-algebra (see ([29], p58) for a good introduction), the four "degrees of freedom" can not be treated independent from each other. The four channels specifying a joint rotation are always processed as a group. That means, that the control-points of a transitional curve for a joint rotation resemble quaternions as well. We now have to realize the same curve properties (c1-continuity) at the junction points to rotations. We therefore have to account for orientation and angular velocity of a joint and place (or better orient) the curve control quaternions such that steadiness of angular velocity is maintained.

To determine the intermediate control points for a quaternion transition curve, the angular velocity at the junction points is calculated. This velocity is represented as quaternions as well. To obtain the intermediate control points, the border control points are multiplied with the respective derivation quaternion scaled to a third of it's actual rotation angle.

Bezier curve with quaternion control points can be evaluated in a similar way to vector-based control points. However, difficulties arise when blending quaternion control points using Bernstein-polynomials. In general linear interpolation of quaternions does not produce linear interpolation of rotation angles ([29], p.98). To solve for that problem, spherical linear interpolation is combined with classic DeCasteljau evaluation, where interpolation is carried out only on two control points at a time. Equation (5.8) shows the modified DeCasteljau-algorithm, where b_i are the initial control points representing quaternions and slerp means spherical linear interpolation between two orientations. A more detailed introduction on that topic can be found in [40].

$$\begin{aligned} b_i^0 &= b_i \\ b_i^j &= slerp(b_i^{j-1}, b_{i+1}^{j-1}, t) \qquad i = 0, ..., n-j \end{aligned} \tag{5.8}$$

5.2.3.4 Variants of Curve Construction

A number of variants for transition curve generation has been implemented. Five variants are available for testing in the Gesture Trigger tool as follows.

C0:

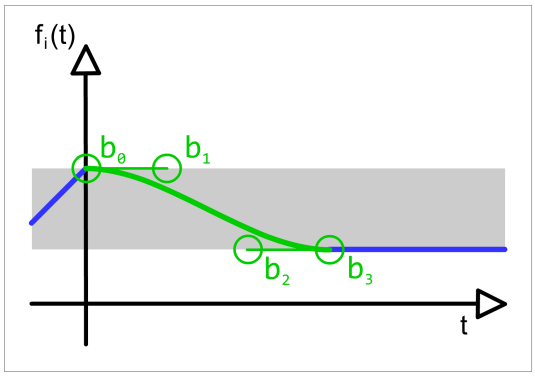

Figure 5.5: Transition curve with C0 continuity

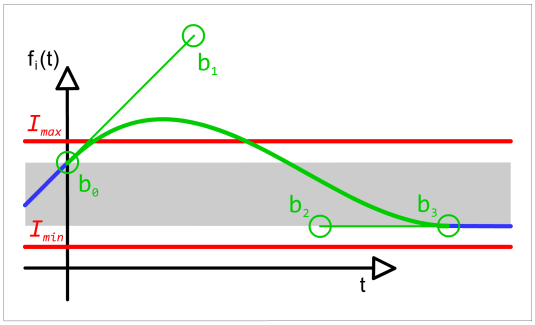

Figure 5.6: Transition curve exceeding constraint intervals

This variant uses a constant transition duration and makes intermediate control points coincide with the border control points of the curve, resulting in zero gradient at the junction points 5.5. This variant ensures that the transitional values will lie within the interval defined by the junction point values and therefore no natural joint constraints are violated. However, the animation produced will contain hard stops or changes in motion direction at the begin and end of transitions. Further, The flattened shape of the transition curves will give a robot-like motion impression.

C1:

This variant guarantees C1 continuity at the junction points by placing the intermediate control points according to formula. Like C0, this algorithm applies a constant transition time and therefore might easily produce out-of range angles (see figure 5.6). This variant ensures smooth motion throughout transitions. However, timing issues are not considered, which is why natural joint constraints are easily violated.

C1A:

This algorithm works similar to C1, but estimates an optimal transition time for each joint rotation by considering the angle between the junction point orientations (equation (5.4)). It then uses the maximum required duration as overall transition duration. This variant provides good results in terms of both a natural impression of motion and natural joint constraints for most gesture animations. However in certain cases, for example when junction point value differences are small while the

velocities are large, a too short transition duration might be computed for heavy limbs to be reoriented for the following gesture.

C1AL:

This variant works similar to C1A, but also considers the length of each joint for computing the required duration (equation (5.5)). This simulates the fact that heavier limbs have a higher inertia and thus need more time for reorientation. The drawback of the previous variant has been solved with this variant by considering limb length. Still, this algorithm applies the maximum duration to all DOF transition curves which might result in exceeding natural constraint interval bounds, especially for short joints like those of the fingers.

C1ALS:

The last variant performs the same duration calculation as variant C1AL, but takes a different approach to curve generation. This variant focusses on maintaining the curve shape characteristics resulting from the optimal transition duration estimated for individual joints. To do so, the applied transition curve is a compound of two functions $tr_0(t)$ and $tr_1(t)$, each of which is a piecewise function consisting of a bezier part and a linear part. The bezier part of either function thereby covers only the fraction definition interval given by $f_i = d_i/d_{max}$. Each bezier curve regularly adapts to one junction point on one side and to the function $L(t)$ on the other, where $L(t)$ is the function that linearly interpolates between the junction points. $tr_0(t)$ and $tr_1(t)$ are thus defined by equation (5.9) and (5.10).

$$tr_0(t) = \begin{cases} F(t/f_i) & \text{if } t < f_i \\ \text{L(t)} & \text{else} \end{cases} \tag{5.9}$$

and

$$tr_1(t) = \begin{cases} \text{L(t)} & \text{if } t < 1 - f_i \\ F((t - 1 + f_i)/f_i) & \text{else} \end{cases} \tag{5.10}$$

where t represents a normalized time parameter running from 0 to 1 and $L(t)$ is given by

$$L(t) = (1 - t) * C_S^0 + t * C_E^0 \tag{5.11}$$

. $L(t)$ therefore covers the remainder of the interval for each of the two functions. To compute the final transition function $T(t)$, those two functions are again blended linearly.

$$T(t) = (1 - t) * tr_0(t) + t * tr_1(t) \tag{5.12}$$

Figures 5.7 and 5.8 show the example scenario for the two functions.

The result is a transition curve that spans the required overall transition duration, but resembles the in most cases smaller value exaggerations resulting from the estimated optimal transition durations d_i of each joint while still maintaining C1-continuity at the junction points.

This last variant realizes the two essential requirements of C1-continuity and a low probability of natural constraint interval violation. The motion produced gives good results in terms of natural

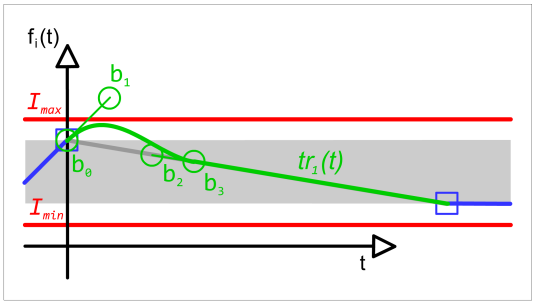

Figure 5.7: Transition functions with bezier part at the begin of the transition.

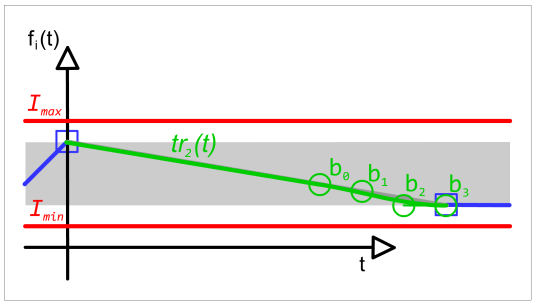

Figure 5.8: Transition functions with bezier part at the end of the transition.

motion impression, but, like that variant C0, produces flattened curve shapes, which gives a similar robotic impression of motion, especially for shorter joints.

Recapitulating, variants C1AL and C1ALS give the best results, where C1AL is to prefer for naturalness of motion impression, and C1ALS for better natural DOF constraint adherence.

5.2.3.5 Anytime transitions

By now I have explained transitions connecting the predefined performing phases of two gesture animations only. To account for instantaneous changes in the ”mind” of an agent, we need to apply these transition curves at any time.

In principle transition curves as pointed out above can be applied at any point in time of a motion. This works well for motions that are performed by the upper part of the body and with the hands moving not to the backside of the torso. This is the case for the magnitude of conversational gestures. However, there are limitations to applying transition curves blindly, which are outlined in the next paragraphs.

5.2.4 Limitations to this approach

The result will always be a smooth movement that seamlessly connects two pieces of animation. The flaws of the technique described here become apparent when effects like collision and angle constraints are considered.

5.2.4.1 Physical Skeleton DOF Constraints

Traditionally keyframed animation data usually describes, how joints are positioned at a given time in the animation. It's the responsibility of the animator to ensure that the resulting motion is natural and that no impossible skeleton configurations are produced. Even if the authoring tool provides facilities to ensure DOF-constraints, they are meant as an aid for the animator but the constraints are not exported along with the keyframe data. When using that pure animation data as input to our engine, the transition generation algorithm might result in configurations that violate natural joint angle constraints. Although this is not likely to happen, the scenario is described here. The constraints applied by an authoring tool usually specify an interval of legal values for each DOF. For example, the elbow joint was allowed to adopt angles between 0 degrees (stretched out) and 160 degrees on the x-axis, and no bending on the other axes. Any value outside of a constraint interval that might be issued by some animation controller is clamped to that interval before the actual output is generated. An early version of my transition generation algorithm worked on straight lines (bezier curves of grade 1) between the junction points. This ensured ensured legal configurations, provided that the junction points lied within the natural constraints interval (which is assumed). Later versions of the algorithm add C1-continuity at the junction points, which results in possible exaggerations of constraint intervals (see figure 5.6). The transition is triggered while the prior gesture is nearly finished, meaning the arms are on their way back to the idling posture besides the torso. The elbow joint is still bent but rotating (at a high angular velocity) towards the stretched out constellation with the upper arm joint. When applying a transition at this point in time, the high velocity might cause the algorithm to create a transition curve that exceeds the natural bending limitation of the elbow. This effect may result from a bad duration calculation but also from the animation footage. Usually, when a human joint reaches its bending maximum, some damping will occur to prevent sudden peaks of force being imposed to the joint mechanics. A professional animator simulates this damping by slowing down the rotation speed well before reaching a maximum configuration. This approach relies on the animation data to be prepared that way, such that the described exaggeration is unlikely to happen.

5.2.4.2 Collisions

A known problem in motion synthesis for skeleton animation are collisions between joints of a hierarchy. For human characters specifying proper bending constraints does not prevent limbs to touch each other. In animation, again it's the responsibility of the animator to not have limbs pass through each other. As for conversational agents, the magnitude of motions is insusceptible to these effects. Most of the motions consist of arm movements on the figure's front side with the hands apart from each other. Therefore calculating collisions between limbs is not considered in this work. However, there are some motions at the backside of the character, such as folding hands on the back or unobtrusively hiding a hand while doing something with the other hand. When transiting from those poses to gestures carried out at the front side, the hand is likely to move through the torso of the figure, a most undesirable effect. To prevent that, this technique comprises a simple extension mechanism, which is explained in the next paragraph.

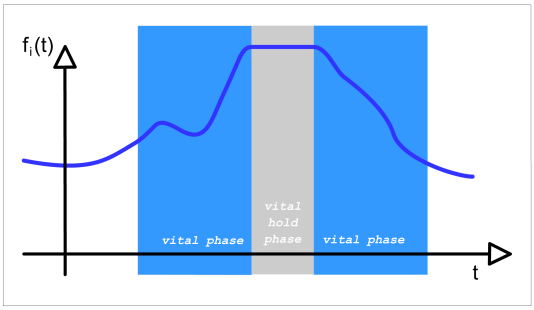

Figure 5.9: Motion function with vital hold phase

5.2.5 Vital phases

Some postures do not welcome anytime interruptibility. It's those that will require a specific movement in order to be established, such as the movement of folding the arms on the chest or, as pointed out before, those with arms at the backside. When transiting from or to such a pose directly using my approach, the arms will most likely pass through each other of the torso and create a visual disturbance.

A solution to that are vital phases. While performing common concatenation of those critical gestures, the performing phase can easily be chosen in a way that it entirely covers the time at which the hand is in a critical area. Transition curves will then be applied not before skeleton reaches a transitable configuration, such that collisions will not occur. For anytime transition however that does not work. I therefore introduced so called *vital* phases for gestures. When a phase is declared vital, the transition generation is suspended until the next non-vital phase is reached. This will allow the skeleton to "return" from a critical configuration on the way it is meant to return according to the animation footage.

Although this suspending breaks the claim of anytime interruptibility, it only accounts for natural circumstances: A human that takes on a more complicated pose cannot return from that pose as easily as he might wish. Therefore, the vital phases only reflect natural restrictions to anytime interruptibility.

An improvement to vital phases are the so called *vital hold* phases. Hold phases are times in which the character holds a certain configuration. Usually this is a posture as described in section 5.2.1.1. As no movement is taking place during hold phases, they could be skipped without producing any motion discontinuity. Normally it is not desirable to skip those phases, as they have a designated communicative function and convey a message. However, when a gesture with critical configurations is interrupted, it is not the intention of the agent to convey that message any more. It is desirable to return from the critical configuration as soon as possible, which is why in that case hold phases should be cut short. Phases that are declared as "vital hold" are therefore skipped by the animation engine, as soon as such a gesture is interrupted. Vital hold phases do only make sense in the middle of a sequence of vital phases, but not at the start or end of it. Picture 5.9 shows a schematic view of a gesture with a vital hold phase, and 5.10 shows the result after removing the vital hold phase due to a transition request.

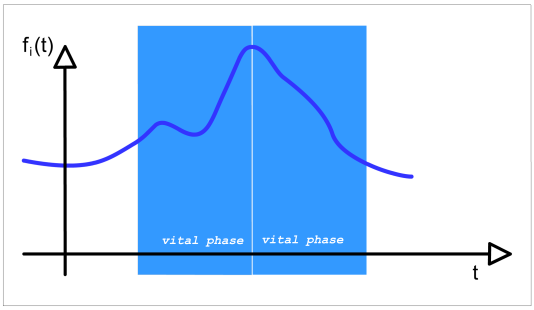

Figure 5.10: Motion function after vital hold phase removal

5.3 Mixing Animations

This section discusses possibilities and limitations of approaches to mix several body movements in order to increase conversational capabilities of an animated character.

5.3.1 Doing Multiple Things at Once

The human motorical system allows us to perform several physical actions with different functions at once. For example, we might walk towards some destination whilst operating a handheld device. In a technical sense difficulties arise when the limbs performing those disjunct operations do hinder each other in flexibility and degrees of freedom are limited. For example, waving a hand towards someone while indicating some centimeter distance using the fingers of the same hand will diminish or destroy the desired conversational effect for both actions. In that example, two mutually exclusive configurations of the hand joints are apparent. However, of course, using the two arms to do several independent conversational acts at the same time is quite common. For example, one arm might support a posture while the other might be used for performing a beat gesture. This supports the impression of naturalness, as it is an untypical behavior to leave one arm completely relaxed while using the other for an expressive act. More likely, the unused arm would support the used one, or strive for a pose that stresses the fact that it is not used (like putting an arm to the backside, while supporting a verbal remark with the other).

5.3.2 Related Work

Perlin and Goldberg introduce simultaneous performance of multiple motions with the animation engine of their Improv framework [33]. They realize concurrency by applying a layered action model. Analogous to image compositing methods, where pixels from either image plane are drawn in a back-to-front-manner, possibly masking each other or blending according to weights (alpha-values), these layers represent motions, that issue output values for DOF-channels and that might overlap and overwrite each other in controlling those channels. In each layer therefore, the motions compete with each other to be performed, whereas layers are applied to the DOF-channels successively, simply replacing previous values. Authors are entitled to define a grouping for the available motions, such that all motions that might semantically conflict with each other are in the same group, whereas motions that

might gracefully coexist are in different layers. For example, an author might group walking motions into a lower layer and hand gestures into a higher layer. While the walking motions specify motion functions for all DOF of the skeleton, hand gestures only affect those at the arms. A character with active walking motion therefore can keep on walking when requested to scratch it's head with the hand. The values issued to arm-joint DOF by the walking animation are simply overwritten by the scratching animation output.

5.3.2.1 Drawbacks to Motion Mixing Approaches

Motion mixing approaches like the one described before may result in plausible simultaneous motion performing and provide sufficient control to an author to determine for mixability-restrictions. A fact about that technique, however, is that motion is considered to be local (gestures). That suggests, that groups of DOF of a skeleton might be addressed independently by several animations. But, as each limbs motion changes the figure's center of weight and propagates forces to all limbs in a hierarchy and therefore has to be negotiated by the whole skeleton, that independence does not correspond to physical reality. To give an example, a walking motion might be animated in a well-balanced way including arm-swinging. When stopping that swinging for one arm an performing a hand raising motion, the figure would intuitively be expected to compensate for the asymmetric change in weight distribution in some way. Without showing such compensating motion features, naturalness suffers.

For cartoon style ECAs that might not be of any disturbance, because the expressive aim with such is rather to be striking then natural. But, as we focus on realistic characters with a high naturalness of motion, we refrain from mixing primary motions. We leave it in the responsibility of the animator to provide well-balanced motion footage that applies motion control to the entire limb hierarchy.

5.3.3 Subtle Secondary Motion Blending

Motion mixing can however be successfully be applied for realistic characters when dealing with secondary motion, which means motion that does not carry any communicative intent but that refers to physical necessities. If that motion is subtle enough, it might be applied along with any primary motion without affecting the results.

Human posture is never completely static. Even when standing completely still we perform slight swaying movements that refer to the muscular structures balancing the bodies weight while continuously spending energy on contracting. Other dynamic necessities are heartbeat and breath which cause vibration in the human body. When watching motion picture, we therefore expect to see that motion even for characters not moving. When it is omitted, an effect called "moving hold" can result [23], which is the impression of a character (or the whole movie)to suddenly freeze. Professional computer-animators therefore care for always keep some parts of the body in a slight motion.

To account for that, we apply motion mix to blend a subtle swaying motion with whatever primary motion is performed. This swaying movement is authored by an animator and prepared the same way as other body expressions. However, it is not available as a gesture to be performed explicitly. The motion is initialized at program start for any human entity in the scene and is preformed repeatedly until the presentation ends.

5.3.4 Non-Repetitiveness using Noise

Perlin also introduced randomization of motion in order to reduce repetitiveness [32] using noise functions [31]. This technique is promising in order to improve an animation engine for ECAs, as it used here. In [32], it is applied on joint rotations. The approach faces the difficulty of enforcing certain constraints like keeping the supporting foot at ground level during a walking motion by simple measures like displacing the whole figure according to the relative translation of that foot. That displacement is computed after randomization is applied to joint orientation.

For our application however, positional constraints (for the feet) are not possible by mere displacement, as there are two feet to be considered. Applying random rotational increment to for example leg joints will thus move the feet around, unless Inverse Kinematic constraints were adopted. For conversational gesture it is still an option to limit randomization to joints that no constraints apply to (for example arm joints). This however requires individual annotation of a characters joint hierarchy. For these reasons we keep that option in mind for future extensions of the MPML3D framework that might feature character locomotion or physical interaction among scene entities.

For example, any animation might include the hip-joint to be shifted to some direction. The correct placement for the feet is thereby guaranteed by the animation footage to be consistent with a flat and static ground as well as the figure not moving.

Chapter 6

Implementation

6.0.5 Outline

This part describes the exemplary implementation complementing this thesis. It comprises extensions to the MPML3D framework implementation under development at the National Institute of Informatics Tokyo and under the supervision of Helmut Prendinger. The framework realizes a player for MPML3D-script files. It consists of three major parts, which are shown in figure 6.1.

First, there is the reactive framework (represented by package net.monoid.agents) performing the execution of the plot as specified by the script (net.monoid.agents.core). This part is attached with a plot-generation facility, which parses the input MPML3D document (net.monoid.agents.mpml) and establishes the interdependencies between activities, perceptions and scene entities (net.monoid.agents.author).

Second, the application-specific part (represented by package jp.ac.nii.application) provides all necessary extensions for entity types that are to be used by a specific application of MPML3D (jp.ac.nii.application.entities). Each entity type specifies it's dedicated activities and native state variables that are available to the author. This module also comprises the actual JAVA-application. It hosts the entry class and manages OpenGL-context acquisition and windowing issues.

Third, the engine part manages the audiovisual output of the application. It contains classes for internal representation of media resources as well as for processing them for output. This module performs the animation and drawing of character models (among other things) and therefore is the part of the software where the majority of extensions related to this work are contained.

Besides these main packages, several support libraries are used by the main modules. Those include *util* (net.monoid.util), *math* (net.monoid.math), *games* (net.monoid.games) and *opengl* (net.monoid.opengl).

The software is written in JAVA entirely. It makes use of the programming interface and standard packages provided with revision 6.0 of the language. External packages used are the *jogl-library* ([1])for retaining *OpenGL*-support and the swing-library for GUI control elements. In case speech resources have to be generated for a presentation, the framework also needs access to the Text-to-Speech application *LoquendoTTS* [2], contacted via the JAVA Remote Method Invocation (RMI) protocol. That server also accesses another external software, *Lipsync* [3], which provides speech-aligned lip animation.

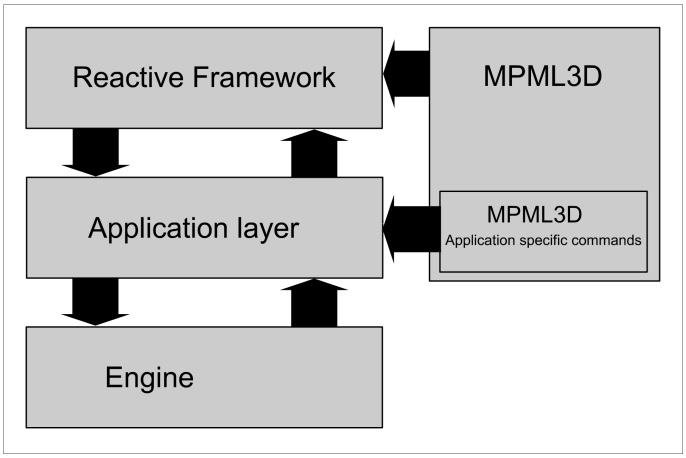

Figure 6.1: Major parts of the MPML3D-player framework

6.1 Reactive Framework and Application module

6.1.1 Activities

This reactive framework operates a network of activities that constitute the scene plot. Each activity represents some action or compound of actions to be carried out by the scene entities. Compositions of activities form hierarchies related to each other which in turn forms the plot network. Activities can be started and stopped, they hold references to associated parent and child activities and give notification about activity-related events.

Activities are realized via a set of classes derived from abstract class *AActivityBase*. The most important classes are *DeferredActivity*, which holds an actual (concrete) action to be performed by an entity, and a number of classes realizing container activities like *SequentialActivity* and *ParallelActivity*.

When an activity is started or stopped, it will forward operational commands to it's children according to it's specific purpose. A *SequentialActivity* object for example holds a list of child activities to be performed sequentially. When started, it starts the first activity in the list. As soon as a child notifies it has finished, it will start the next activity from the list, and, if the last child has finished, notify it's parent about it's own successful finishing. Unlike that, a *ParallelActivity* for example will start or stop all of it's children at once. It will then collect notifications of all it's children and notify having finished to it's parent when every child has finished.

The leaves of the activity hierarchy tree are classes derived from *AConcreteActivity*. Concrete activities do not hold any child activities, but represent actual actions to be performed. After being started and until they finish (or are stopped) they are updated with the time passed at each frame and change the actual output by addressing the engine module accordingly.

To get updated, concrete activities enqueue themselves in an activity channel belonging to the associated entity object.

6.1.2 Activity Channels

Activity channels are pipelines for concrete activities that are supposed to be performed. Each entity type owns it's special set of channels, reflecting several types of actions that it can perform concurrently. A human entity for example holds different channels for gesture and speech output. Activities enqueued in a channel will be performed sequentially in most cases. Activity channels are realized by classes deriving the class *AActivityChannel* and are not to be confused with animation channels that refer to degrees of freedom of an animated figure.

6.1.3 Links

There are three types of links that may connect activities. First there are links between perceptions and tasks (see below), realized by class *StartOnPerceptionLink*. Second, tasks can be scripted such that the will start other tasks when they end or begin, realized by class *OnEventStartTaskLink*. The third link used to perform subaction synchronization, as explained in section 4.2, is realized by class *SyncLink*. All those classes are listener classes implementing the interface *IActivityObserver*. When a link is constructed, an instance of that interface is registered as an event listener at the activity of interest and obtains a reference to the activity that is to be started or stopped.

6.1.4 Perceptions and Tasks

As explained in section 4.3, perceptions are listeners to events generated by scene entity states. Perceptions are realized by classes deriving class *APerceptionConcreteActivity*. Perceptions are a key concept for realizing an entirely event-based reactive behavior control framework. Perception objects are started similar to activities. Each perception holds references to the state it is supposed to watch. Like concrete activities, each perception is updated at each frame and checks that state. When the adequate state event occurs, it dispatches it's own "occurred" event to the global scene event queue, which in turn might effect some tasks to be started.

Tasks, as mentioned in section 4.3 are the main parts of a scripted plot. Tasks are special sequential activity containers holding a priority value and are realized by class *TaskActivity*. Each object of that class holds a reference to a task manager that it will enlist itself to as soon as it is started. Task managers (class *TaskManager*) decide upon task precedence as explained in section 4.3.1. When a task manager decides for a certain task to be performed, it causes the child activity of the task to be started.

6.1.5 Generating the Plot from a script

After an MPML3D document was parsed, the activity network is established in two major stages. In the first stage the individual activity hierarchies (i.e. tasks) will be constructed. Thereby, XML tags are processed by the reactive framework, whereas action commands are forwarded to the application specific entity classes (application module) which implement the interface *ICommandFactory*. Each entity type thus can be requested to generate concrete activities according to those action commands. In the second stage, connections between activities are established via activity observer objects (links) (section 6.1.3).

6.2 Animation Engine Enhancements

Section 6.2 first describes the basic animation engine features without my extensions and then describes the extensions themselves. For convenience reasons, present class names are given, even where different names were used initially.

6.2.1 Engine Overview

The engine part of the MPML3D player implementation developed at the NII includes the basic facilities to handle graphical output like static and animated meshes, materials, textures and illumination using gouraud shading provided as a standard feature of graphics hardware, retrieving hardware acceleration via the OpenGL programming interface, provided for JAVA by the Jogl library [1]. Other features of the engine are sound output (used for speech actions), timers and basic scene camera management. The most elaborate part of course are the classes to manage and process the virtual agent 3D models. As those are character models of high realism (compared to other ECA applications) and as a high realism of animation is also a major requirement for their graphical representation, state of the art animation techniques like soft-skinning and skeletal subspace deformers are featured. For the same reasons, morph targets are supported in order to display facial expressions like smile or astonishment, but also to move the lips synchronous to speech output of a character.

6.2.2 Character Model File Formats

For storage of character model data we use several XML based file-formats which represent a characters subspace hierarchy and all meshes attached to it. A brief outline of the different formats will be given now.

The starting point for a characters graphical representation is the model file. A model contains two elements below the root element which contain a listing of skeletal and mesh subspaces and a list of geometry nodes respectively. The listed subspaces, or joints, are both such that position meshes within the model-space and such representing the actual bones of the skeleton hierarchy. Bone related subspaces hold references to their parent by which the whole bone hierarchy is represented in the model file. The second list contains 'Mesh' elements which each represent one mesh to be drawn. Mesh elements hold a reference to a file providing the actual mesh data, a reference to some subspace associated with the mesh and a set of modifiers. In order to be animated, each mesh refers to at least one bone related subspace, to which the mesh will be 'skinned'. Additionally, each mesh might refer to a list of morph targets, used for local deformations mainly at the character's face mesh to animate lips while speaking.

Each mesh is represented by an xml file type called *mesh*. Each mesh contains lists of vertex positions and texture coordinate pairs. For each of these lists a mapping is specified which is used to associate positions and texture coordinate pairs with each other to generate the actual set of vertices. This storage technique is used to reduce storage cost for vertices that are used more than once with varying texture coordinates. Lastly a list of indices specifies the triangles to be rendered, which also contains a reference to a material file for the mesh.

Analogous to the model representation animations are provided to the engine using an xml-based format. An *animation* file specifies the duration, the phases and for each animated degree of freedom

a list of keytimes with associated key-values. The phases relate to the animation phases explained in section 5.2. Because some degrees of freedom share the same set of keytimes, those DOF are grouped to animation channels. For joint hierarchy animations, these channels usually refer to the translation of rotation of a certain joint. Each animation file containing joint transformation channels thus specifies the configuration of the entire skeleton at each time during animation display.

6.2.3 Animation Data Processing

While the framework constructs the plot of a presentation, animation sources are loaded according to the gesture actions specified throughout the MPML3D document. Classes derived from interface *ResourceProvider* parse the XML files and store the animation data in intermediate classes implementing interface Animation. Then, a class specific for processing skeleton animation data, *GestureAnimatorLoader*, interprets the parsed animation data and attaches it to the animatable channels of the associated character model. The result of this operation is an instance of class *GestureAnimator*, derived from abstract class *AJointHierarchyAnimator*, and is referenced by the *HumanGestureActivity*-objects to be used when the animation is displayed. Class *AJointHierarchyAnimator* implements the interface *IAnimator*, which represents animations that are currently performed. The abstract class holds a timer that determines the actual progress of animation. It also stores references to the channels of the animated model.

When the framework starts a gesture activity, the respective *HumanGestureActivity*-object will rewind the timer of it's animator object and register itself in it's associated activity channel. For gesture animations, this is a modified sequential activity channel, which means that a character can only perform one gesture at a time. This fact refers to the circumstances stated in section 5.3.2.1. While registered, the activity object will be updated at each frame with the time that has elapsed, and it will forward this time to the animator instance by calling the *animate* function. This animate function updates the current configuration of each animated joint and applies those settings to the character model. To compute the current configuration, linear interpolation is performed between the two key values relating to the keytimes that clasp around the current time index. This is performed for each DOF independently.

When the character model is to be drawn, it will transform the vertex of its geometry according to the actual configuration of the skinned joints, which, performed frame-by-frame, results in the actual display of gesture motion.

6.2.4 Gesture Parametrization Enhancements

To incorporate Gesture Parametrization as described in section 5.1, the class hierarchy has been extended, starting from *AJointHierarchyAnimator* deriving classes. The principle of creating new motions from existing samples is realized by the class *VerbGestureAnimator*, which is basically a wrapper for several *GestureAnimator* objects representing the sample motions available for a gesture. It therefore replaces the wrapped class as the *AJointHierarchyAnimator* object referenced by the gesture activity.

To handle action parameters, *VerbGestureAnimator* stores an array of float variables that hold the actual values of the individual adverbs, representing the position in adverb space for which the new

motion is to be generated. The class also holds weights for each sample motion. Whenever the adverb values are set (or changed), the class will update the weights for the sample animations. When the *animate* function is called, the contributions of all sample animators whose weights are not zero are added, and the result is applied to for each animatable channel of the model.

Class *VerbGestureAnimator* is initialized by the dedicated loader class *VerbGestureAnimatorLoader*, which wraps the loader objects associated with the wrapped sample motions.

A dedicated file of another XML based format called *gestures* stores a list of all available gestures for a model. This file is loaded along with the model data at initialization time. The list specifies the available sample motions and supported dimensions of the adverb space. For each sample motion, an input file containing the actual animation is given. Further, each sample motion is tagged with its position in that adverb space. To add new gestures or enhance dimensionality of existing gestures, authors have to edit this file.

Adverb values might be set via action command parameters, or they might be wired to entity state parameters (section 4.4). To account for adverb changes that might occur during a gesture is performed, a gesture activity is registers itself at the associated entity for being adverb change listener as soon as it is started.

6.2.5 Gesture Transition Enhancements

In order to introduce gesture transitions as described in chapter 5.2 several classes have been added. First, to manage transitions between activities, the agents framework has been equipped with a special activity channel class, the *TransitionActivityChannel*. This class behaves similar to a *SequentialActivityChannel*, but considers more activity states returned on calls to method updateConcrete() than *FINISHED* and *RUNNING*. When managed by this channel class, a concrete activity class like the *HumanGestureActivity* might signal preparedness to be turned into a transit activity. A gesture activity might do so in two cases: First, when it regularly reaches it's finishing phase. In this case the transition activity channel will check if there are further gestures waiting to be performed and generate a transition animation. Second, a gesture activity might be explicitly stopped by some event and signal that it immediately needs to be transited immediately. The channel will then again arrange a transition to a follow-up gesture or, if none is at hand, perform a preemptive transition to the normal body posture of the character. This behavior refers to the feature of anytime interruptibility for body expressions described in section 5.2.

The transition activity channel carries out transitions as follows. As the channel is handling activities which in turn drive animator objects, the channel has to insert a new activity objects in order to have an intermediate animator object operate the skeleton hierarchy. To do so, the previous activity (the one that has been interrupted) is cloned and the clone is equipped with a new animator that performs the actual transit motion. The reference to the previous activity is discarded by the channel, which is particularly needed in case the same activity is the one that is transited to. This might well be the case if a gesture is performed repeatedly.

To calculate the actual transition animation, two new classes derive from the the *AJointHierarchyAnimator* class. The *BezierAnimator* class provides animation channels represented by beziercurves, whereas the *TransitionGestureAnimator* is derived from that class and introduces adverb handling similar to class *VerbGestureAnimator*. When a transition is to be computed, an instance of *Tran-*

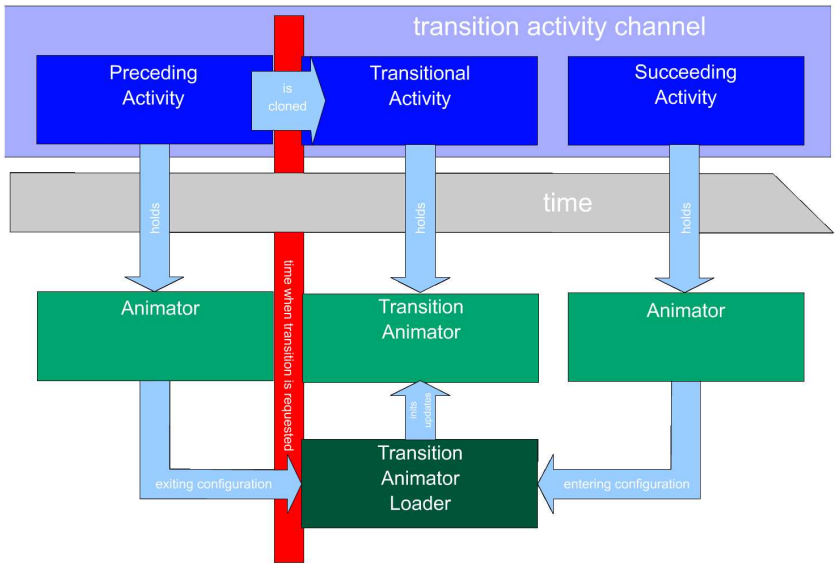

Figure 6.2: Transition activity channel state when a transition is generated

sitionGestureAnimator is initialized by an instance of loader class *TransitionGestureAnimatorLoader*. That loader class dedicated to manage transition animators does not only set the initial control points of the transition curves for each animated channel, but also updates the curves when adverb changes occur. To set the curves' control points, the loader class refers to both the animator of the preceding as well as the succeeding gesture animator. Figure [] illustrates the relationships between objects when a transition is performed. In that moment, the loader retrieves the current configuration from the preceding and the future configuration of the succeeding animator. As the latter configuration might still vary with adverb changes, the transit activity is registered as an adverb change listener for the bound adverbs of the succeeding activity.

6.2.6 Subtle Background Motion

Subtle background motion as described in section 5.3.3 is realized by simply applying a second animation to the skeleton. This animation is represented the same way as the regular gesture and posture animations, however the values are applied to the animated channels accumulative, that is , the results are added for each channel. As only slight displacements are added, the anatomy of the primary motion is not destroyed in its expression. For phases where there is no primary motion performed on the skeleton (when no gesture is performed or the current gesture is in a hold phase), the the impression of freezing (or *moving hold*) is prevented. The entity type human administers this background motion by employing an extra sequential activity channel which a single gesture activity, representing the motion is queued and repeatedly performed. This activity is started at initialization time of the human entity.

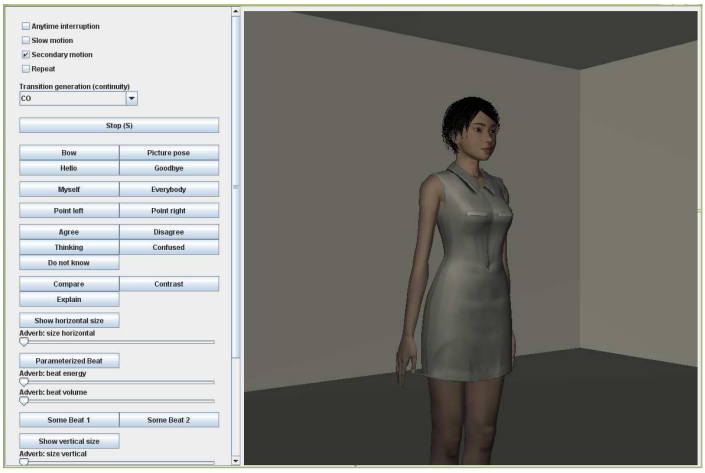

Figure 6.3: Gesture Trigger demo application GUI

6.3 Demonstration Applications

6.3.1 Gesture Trigger

The *Gesture Trigger* application presents a way to test and examine the quality of gesture animation output. It is designed to issue action commands to the control framework directly, without a wrapping MPML3D-script. Gesture performance is triggered by simply pressing buttons, motion parameters are adjusted by sliders. This section gives a short introduction to the available options.

The viewport on the right side shows a single agent. My mouse dragging within the viewport area, the camera can be rotated with the center of view residing at the agent. The mouse wheel can be used to zoom in and out. Pressing the 'R'-key will restore the initial camera perspective.

The GUI on the left side offers a set of gestures to be triggered by pushing buttons. By pressing a button the associated gesture is enqueued to the activity channel of the agent, and performed as soon as any previous gesture is finished. As described in chapter 5.2, a gesture is finished with the end of it's performing phase. Therefore, if a follow-up gesture is enqueued, a transition is generated. The checkbox *Anytime Interruption* will change this behavior. With that option activated, any running gesture will be interrupted immediately when a new gesture is triggered. This will happen, even if the same gesture is scheduled again. If the animation is currently in a transition, that transition will be interrupted and replaced by a new transition. Also, the running gesture can always be stopped immediately by pushing the stop button or hitting the *S*-key. In that case a transition to the normal pose will be performed. The GUI also contains a dropdown list where the transition generation algorithms variant (see section 5.2.3.4) can be chosen.

Some gestures expose animation parameters. Those can be controlled using the sliders next to the gestures buttons. To explore, how adverb changes are managed even during transitions, the *Repeat* option can be activated. By doing so, a gesture will be performed in cycles with appropriate transitions between the recursions.

The *Slow motion* option should be activated to take a closer look at the animation quality and to de-

tect shortcomings of the individual algorithms. Last, the option *Secondary motion* can be deactivated to get an idea of the moving hold effect outlined in section 5.3.3.

The slider named *Transition duration scale* at the very bottom can be used to scale the duration applied for transitions. By experimenting with this slider, one can detect the destructive effects, that a wrong timing can have on animation naturalness.

6.3.2 MPML3D Player

The *MPML3D Player* application is the dedicated way of experiencing the MPML3D language and framework in action. This application will just be started by specifying an MPML3D-script as a command line parameter, and perform that script with all user feedback channels addressed within the script, as far as hardware support is available. The basic demonstration scripts run on a JAVA virtual machine supporting revision 1.6 of the language. Several demonstration scripts are included in the software bundle. Those include short simple dialogues (for example T2DSatohSensei.mpml.xml), a full-feature presentation utilizing gaze-tracking hardware (MP3PodIncEyeTracker.mpml.xml) and a long demo for emotion-rich dialogue scripting via MPML3D, the script PowerDemo.JapaneseRoom.mpml.xml. All provided MPML3D-scripts can be found in subfolder src/player/res/scripts in the folder codebase or packed and into the deployed jar-archives.

6.3.3 Deployment

All demonstration applications are provided on the CD in folder *deploy*, where dedicated batch-files are deposited for startup. Those batch-files can be edited to change parameters for screen resolution as well as libraries used. A reference for the markup language MPML3D and documentary about the MPML3D-framework and the provided interfaces for developers can be found in folder *doc*. The player can be invoked by running the startup batch-file *niiplayer.bat* with the name of a script-file as a single argument.

Chapter 7

Conclusions and Future Work

7.1 Future Work

7.1.1 A More Accurate Subaction Synchronization

Timing is a very important issue when aiming at enriching mere text with emotional expression. This refers both to the speed of and pauses during spoken utterances themselves and to the co-verbal performance of body expression such as gaze or gesture. For a beat gestures for example it is essential that the time it reaches it's climax matches exactly with the word that is to be emphasized. When working with high quality speech output, even a synchronization with the accentuated syllable of that word was desirable. If timing is not accurate, the motion will support a different word in the sentence at therefore have an effect opposite to the intended one. Often event-based frameworks support a synchronization that starts sub-actions on the occurrence of other subactions. For the MPML3D-framework is one example for that concept. It means that a gesture can be started at a certain word of a sentence. However that gives no control on when the performing phase or the climax of that motion will happen. Authors that want to accurately synchronize actions with speech, need to do cumbersome fine-tuning to get the desired effects. A solution to that were a mechanism that realized an event queue with event prediction on utterance level. The time until the occurrence of a certain word (or syllable) could so be estimated and a gesture to be aligned according to it's climax could be started in advance. Although this would contradict to the pure event-based nature of the framework, it would provide the means for a synchronization of actions according to their key moment, not to their begin.

7.1.2 A More Generic Approach to Gesture Synthesis

Currently gesture motion for ECA is provided as a catalogue to be sampled from. The variety of motions depends strongly on the application, however all motions are predefined and are scheduled according to some expression associated with them. This inherently limits the diversity of gesture display to the set of gestures provided by the framework.

This work showed that encoding the essential (performing) phase of an arm movement only is often sufficient and transition movements between those phases or idling postures can be computed synthetically. A more sophisticated engine could engage an Inverse Kinematic System and represent gesture motions by the position and speed of the hand relative to the body in combination with the

finger configuration, rather than storing forward kinematic DOF-functions. This would result in a more meaningful encoding on the one hand and open more combinations. For example a beat carried out with only the index finger stretched out has a more instructing sense than the same beat with a flat hand.

7.2 Conclusions

This work gave a brief introduction on Emotional Conversational Agents and the kind of applications they might be a valuable part of. It also highlighted projects in which ECAs approve to be an essential feature to be an essential contribution to Human Computer Interaction.

This work presented a way to process traditionally keyframed animation footage to be used for realtime interactive emotional co-verbal conversational expressive gesture animation output including parameterizibility and anytime smooth transitions between motions.

The approach taken here is generally applicable with regard to the fat the the technique presented is not only applicable for human, but also human-like or any kind of skeleton where motion expressivity is gained from limb animation. The approach is specific with regards to the fact that only conversational body actions performed with the upper-body benefit from the presented technique to the full extend, whereas other motions need special treatment (like vital phase consideration). However, the strong side of the technique is that new animations can be added and integrated into a catalogue with only minimal annotation requirements. Only phases have to be specified in order to get appropriate transition results.

An ECA control framework was presented providing authors with little experience in high level programming languages a feasible way to realize ECA emotionally expressive behavior generation models of moderate complexity in interactive dialogue and presentation applications in the first place, but not exclusively.

The algorithms presented combine techniques presented first by Rose et al, Shepard and Shoemake, but also anticipate the ongoing research in the field of Embodied Conversational Agents. The particular technique presented is not only applicable for interactive presentation applications, but for any project aiming at highly natural gesture animation output.

Bibliography

[1]

[2]

[3]

[4] Yasmine Arafa, Kaveh Kamyab, and Ebrahim Mamdani. Towards a unified scripting language. Lessons learned from developing CML & AML. In Prendinger and Ishizuka [36], pages 39–63.

[5] N. Badler, J. Allbeck, L. Zhao, and M. Byun. Representing and parameterizing agent behaviors, 2002.

[6] Klaus Bruegmann, Helmut Prendinger, Marc Stamminger, and Mitsuru Ishizuka. Graceful anytime interruptibility for virtual agents. In *ACE '07: Proceedings of the international conference on Advances in computer entertainment technology*, pages 284–285, New York, NY, USA, 2007. ACM Press.

[7] Berardina De Carolis, Catherine Pelauchaud, Isabella Poggi, and Mark Steedman. APML: Markup language for communicative character expressions. In Prendinger and Ishizuka [36], pages 65–85.

[8] Justine Cassell. More than just another pretty face: Embodied conversational interface agents. *Communications of the ACM*, 43(4):70–78, 2000.

[9] Justine Cassell. Nudge nudge wink wink: Elements of face-to-face conversation for embodied conversational agents. In J. Cassell, J. Sullivan, S. Prevost, and E. Churchill, editors, *Embodied Conversational Agents*, pages 1–27. The MIT Press, Cambridge, MA, 2000.

[10] Justine Cassell, Hannes Vilhjálmsson, and Tim Bickmore. BEAT: the Behavior Expression Animation Toolkit. In *Proceedings of SIGGRAPH-01*, pages 477–486, 2001.

[11] N. E. Chafai, C. Pelachaud, D. Pele, and G. Breton. Gesture expressivity modulations in an ECA application. In *Proceedings 6th International Conference on Intelligent Virtual Agents (IVA-06)*, Springer LNAI 4133, pages 181–192, 2006.

[12] Iulia Dobai, Leon Rothkrantz, and Charles van der Mast. Personality model for a companion aibo. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 438–441, New York, NY, USA, 2005. ACM Press.

[13] 2007.

[14]

[15] B. Hartmann, M. Mancini, and C. Pelachaud. Implementing expressive gesture synthesis for embodied conversational agents. 2005.

[16] Björn Hartmann, Maurizio Mancini, and Catherine Pelachaud. Formational parameters and adaptive prototype instantiation for mpeg-4 compliant gesture synthesis. In *CA '02: Proceedings of the Computer Animation*, page 111, Washington, DC, USA, 2002. IEEE Computer Society.

[17] Kristina Höök, Adrian Bullock, Ana Paiva, Marco Vala, Ricardo Chaves, and Rui Prada. Fantasya and sentoy. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 804–805, New York, NY, USA, 2003. ACM Press.

[18] Mitsuru Ishizuka and Helmut Prendinger. Describing and generating multimodal contents featuring affective lifelike agents with MPML. *New Generation Computing*, 24:97–128, 2006.

[19] Lewis L. Johnson, Jeff W. Rickel, and James C. Lester. Animated pedagogical agents: Face-to-face interaction in interactive learning environments. *International Journal of Artificial Intelligence in Education*, 11:47–78, 2000.

[20] Patrick Kelly and Saied Moezzi. Visual computing laboratory. *IEEE MultiMedia*, 02(1):94–99, 1995.

[21] D. H. U. Kochanek and R. H. Bartels. Interpolating splines with local tension, continuity, and bias control. 18(3):33–41, July 1984.

[22] S. Kopp, B. Krenn, S. Marsella, A.N. Marshall, C. Pelachaud, H. Pirker, K.R. Thórisson, and H.H. Vilhjálmsson. Towards a common framework for multimodal generation: the Behavior Markup Language. In *Proceedings 6th International Conference on Intelligent Virtual Agents (IVA-06)*, Springer LNAI 4133, pages 205–217, 2006.

[23] John Lasseter. Tricks to animating characters with a computer. *SIGGRAPH Computer Graphics*, 35(2):45–47, 2001.

[24] James C. Lester, Brian A. Stone, and Gary D. Stelling. Lifelike Pedagogical Agents for Mixed-Initiative Problem Solving in Constructivist Learning Environments. *User Modeling and User-Adapted Interaction*, 9(1-2):1–44, 1999.

[25] C. Lisetti, F. Nasoz, C. LeRouge, O. Ozyer, and K. Alvarez. Developing multimodal intelligent affective interfaces for tele-home health care. *Int. J. Hum.-Comput. Stud.*, 59(1-2):245–255, 2003.

[26] Michael Mateas and Andrew Stern. A Behavior Language: Joint action and behavioral idioms. In Prendinger and Ishizuka [36], pages 19–38.

[27] D. McNeill. *Hand and Mind - What gestures reveal about thought*. The University of Chicago Press, Chicago, USA, 1992.

[28] Michael Nischt, Helmut Prendinger, Elisbeth André, and Mitsuru Ishizuka. MPML3D: a reactive framework for the Multimodal Presentation Markup Language. In *Proceedings 6th International Conference on Intelligent Virtual Agents (IVA-06)*, Springer LNAI 4133, pages 218–229, 2006.

[29] Rick Parent. chapter Interpolation of Rotations Represented by Quaternions, page 98. 2002.

[30] H. Van Dyke Parunak, Robert Bisson, Sven Brueckner, Robert Matthews, and John Sauter. A model of emotions for situated agents. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 993–995, New York, NY, USA, 2006. ACM Press.

[31] Ken Perlin. An image synthesizer. In *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-85)*, pages 287–296. ACM Press, 1985.

[32] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

[33] Ken Perlin and Athomas Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 205–216, New York, NY, USA, 1996. ACM Press.

[34] R. W. Picard and J. Scheirer. The galvactivator: A glove that senses and communicates skin conductivity. In *9th International Conference on Human-Computer Interaction*, pages 1538–1542, New Orleans, August 2001.

[35] Rui Prada, Marco Vala, Ana Paiva, Kristina Hk, and Adrian Bullock. Fantasya - the duel of emotions.

[36] Helmut Prendinger and Mitsuru Ishizuka, editors. *Life-Like Characters. Tools, Affective Functions, and Applications*. Cognitive Technologies. Springer Verlag, Berlin Heidelberg, 2004.

[37] Byron Reeves and Clifford Nass. *The media equation: how people treat computers, television, and new media like real people and places*. Cambridge University Press, New York, NY, USA, 1996.

[38] Charles Rose, Bobby Bodenheimer, and Michael F. Cohen. Verbs and adverbs: multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[39] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM Press.

[40] Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, New York, NY, USA, 1985. ACM Press.

[41] Gillian M. Wilson and M. Angela Sasse. Listen to your heart rate: counting the cost of media quality. pages 9–20, 2000.

[42]

[43] Shumin Zhai. What's in the eyes for attentive input. *Commun. ACM*, 46(3):34–39, 2003.

Erklärung

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Ich bin damit einverstanden, daß die Arbeit veröffentlicht wird und daß in wissenschaftlichen Veröffentlichungen auf sie Bezug genommen wird.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Graphische Datenverarbeitung, wird ein (nicht ausschließliches) Nutzungsrecht an dieser Arbeit sowie an den im Zusammenhang mit ihr erstellten Programmen eingeräumt.

Erlangen, den 2. Juli 2007

(Klaus Brgmann)