# A SYSTEM FOR CREATING SIMPLE CHARACTER BEHAVIOURS

Stefan M. Grünvogel
Stephan Schwichtenberg
Laboratory for Mixed Realities
Institute at the Academy of Media Arts Cologne
Am Coloneum 1, D-50829 Köln, Germany
E-mail: {gruenvogel, schwichtenberg}@lmr.khm.de

**KEYWORDS**
character animation, scripting behaviour, dynamic motion model, augmented reality

**ABSTRACT**

We introduce a real-time character animation system which is currently used in an augmented reality environment for the fast creation of simple character behaviour. By placing and manipulating commands on a timeline, the overall choreography of the characters' movement is created. The movement of the character is controlled by subtasks which model reactive behaviour and control the dynamic motions model for the production of the animation.

## 1. INTRODUCTION

Creating character animation within an augmented reality environment is a relatively new topic. We are developing such an animation system for the augmented reality project mqube (http://www.mqube.de). The aim of this project is to build a prototype of a multi-user environment, where several people work together to create the stage set and to place the lights on a miniaturised stage. It is also important for the stage set creators to get an impression on how an actor would look like on the stage under dynamic change of the light and the properties. Figure 1 shows the augmented view of an user on the miniaturized stage. In the foreground a virtual character can be seen walking along a user defined path. In the background the physical stage set and static (physical) puppets can be seen, which by then were used for representing actors. The users need an easy-to-use interface to create fast and simple animations of characters on the stage.

Nevertheless the requirements for the character animation system are clear in this context and resemble the tasks for scripting character animation sequences in games. It has to be possible to create, choose and delete different characters. For each character there should be an easy way to create and edit simple animations. It is not important for the user/editor to manipulate niceties of the animations. But he or she should have the possibility to choose between large building blocks for the animation e.g. let the character walk along a given path and wave with its arms at a certain time. Furthermore a character should react on the properties on the stage automatically, e.g. jump over obstacles which occur on his path.

In this article we will describe the underlying character animation system of the augemented reality system. Because the user interface is still under development and usability tests are pending, these topics will be reported in an upcoming article.

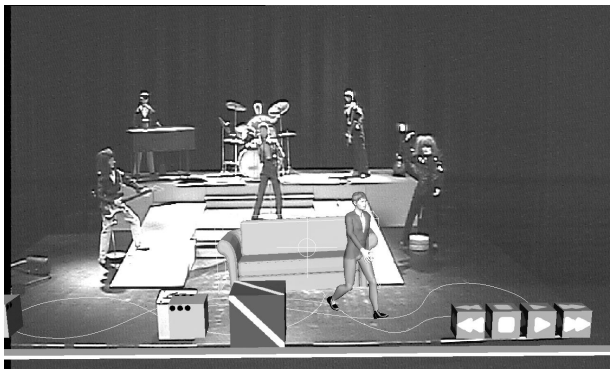

Figure 1: Augmented user view of the miniaturized stage. Picuture © by Fraunhofer FIT.

In Section 2 related work on character animation systems is summarised. In the following sections we present our approach in a top-down manner, starting with an overview on the systems architecture in Section 3. In Section 4 how the editing process is performed at the highest level. Then in 5 we explain the representation of virtual characters within the system and the way complex animations (choreographies) are created and edited. Section 6 explains the conversion from abstract choreographies to concrete motion models which create the final animation. Finally Section 7 concludes with a summary on the results and ends with some remarks on future and ongoing work.

## 2. RELATED WORK

There are still only a few approaches in literature where the interaction with virtual characters in augmented reality is examined. A general overview about augmented reality is given by Azuma (Azuma, 1997). In (Torre *et al.*, 2000) a system is described where interaction techniques between real and virtual humans are explored. A public domain checkers simulator is used to control the movement of a virtual character. In (Balcisoy *et al.*, 2001) a virtual character was used within this system for rapid prototyping in a mixed reality environment.

In Tamuras' RV-Boarder guards (Tamura, 2000) non-human virtual characters are created as opponents in an augmented reality shoot-em-up game.

The two systems mentioned above create the behaviour and the animation of the characters by an underlying system auto-

matically. In our augmented reality project, the mixed reality environment is used itself to edit the movements and the behaviour of the character.

A system for authoring complex scenes and animations within a virtual environment can be found e.g. in Balaguer et. al. (Balaguer & Gobbetti, 1995), (Balaguer & Gobbetti, 1996).

For the creation of the movement and the behaviour of virtual characters (Badler *et al.*, 1993) specify a three layer system by their Jack system. On the lowest level motion is described by the bio-mechanical simulation of the character and at the highest level the behaviour of the character is controlled by a parallel transition network.

Perlin and Goldberg also define with their Improv-System (Perlin & Goldberg, 1996) a multi layer architecture. At the lowest level, single movements of the character and the transitions between the animations are given. To create complex behaviour of characters, scripts are used to define the things each object (the character and other entities) is capable of doing and used to define what can be done with it. Our description of human movements at the lowest level resembles their approach for the lowest level (cf. (Grünvogel, 2003) for a discussion on the differences).

In (Sannier *et al.*, 1999) and (Kalra *et al.*, 1998) the real-time animation system VHD is presented which allows users to control the walking of a character with simple commands like *walk faster*.

The idea of building intelligent characters by creating multiple layers of different interaction with its environment can be found in a more general context in Brooks subsumption architecture (Brooks, 1991).

The Unreal Tournament Editor (Epic Games, 2003) is an example of a scripting environment for a current professional game engine. There *Action* commands are used to create so called *ScriptedSequences* of animation and behaviours. The editing process is made within a graphical user interface.

We also follow a multi-layer approach for the generation of character behaviour. At the lowest level we describe single movements by dynamic motion models. The term motion model is lent by Grassia (Grassia, 2000), who used motion models to build a script based system for the offline creation of character animation. In (Grünvogel, 2003) dynamic motion models are introduced for the real-time creation of character animation in interactive environments. We take these results to build our character animation system presented in this paper.

## 3. SYSTEM ARCHITECTURE

Figure 2 shows the hierarchical structure of the character animation system. If we follow the diagram top down it shows how abstract commands which are passed to the system are transformed into more and more low level commands, resulting finally in animation data which is put into the rendering engine.

At the top of the hierarchy lies the *Manager* which is the interface to the user interface component. The creation, selection and manipulation of characters is triggered by the user interface component of the AR-System by sending commands to the *Manager*. The *Manager* creates and deletes the geometrical

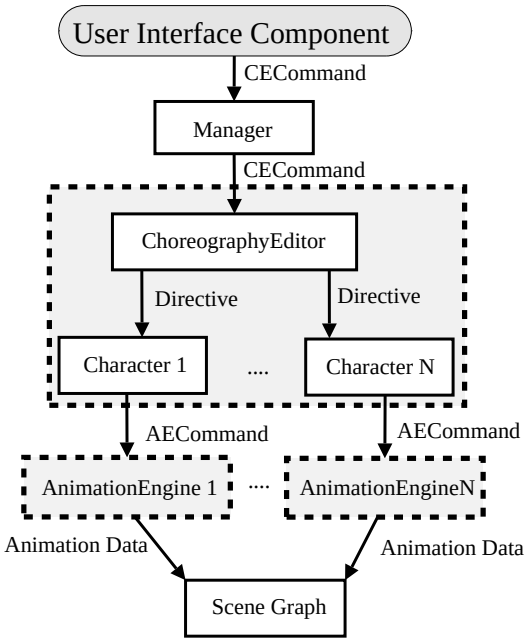


Figure 2: The System Architecture.

and the logical entities of a character, while at the lower levels the choreography editor and animation engine are independed of its geometrical representation.

The choreography editor consists of the *ChoreographyEditor* object together with its *Characters* and controls the behaviour of all the characters in the scene. For each character there is an *AnimationEngine* which is responsible for the creation of the animation data. The animation data is finally send to the scene graph of the rendering engine.

Within the *ChoreographyEditor* a character is represented by a *Character* object. The *ChoreographyEditor* is responsible for the creation, manipulation and deletion of *Characters*. The commands the *Manager* receives for changing a choreography script of a character are send to the *ChoreographyEditor* and then translated into control commands (*Directives*) for the corresponding *Character*. Each *Character* also has a link to an *AnimationEngine*. If a choreography is played, the *Character* controls the animation of the character by sending commands (*AECommands*) to the *AnimationEngine*.

The *Manager* and each animation engine are separately synchronised by a time controller component. This component also transforms the systems time of the AR-system into the internal time (simulation time). Within the character subsystem the simulation time is discretized, currently by 30 frames per second. The simulation time can be moved forward or backward or with arbitrary speed relative to the systems time.

## 4. THE EDITING PROCESS

In this section we describe how the editing process is carried out with the help of the *Manager*. In general the creation of the animation can be compared with the creation of complex animations with non-linear editing tools, like the Trax-Editor in Maya or the NLE in Kayadaras Filmbox. But there is a dif-

ference to these products to which we will come in a moment.

The general philosophy is that there is a global timeline for all characters and a timeline for each character. The behaviour of the characters is ruled by so called *subtasks* which may start and can be layered at an arbitrary point on the timeline and produce the animation of the character in the end. But in comparison to the products mentioned above, these subtasks are *not* fixed pieces of animations which produce always the same result. Instead subtasks represent simple behaviour of characters, like walking along a given path or waving with hands. The resulting animation of such a subtask may change during the execution of the subtask. For example suppose that at a certain time on the timeline the character was given a path which is lying on the flat ground of the stage and was advised to walk along the path by the corresponding subtask. If the choreography is played, i.e. the time moves on, this results in a character walking along a fixed path. Now the user spools back in time and restarts the animation. Then while the character is walking along his path, the user puts an obstacle (e.g. sofa) on the path. As the character reaches the obstacle, it has to decide if he just can walk over the obstacle, or if it has to jump over it, because it is too high. The corresponding subtask takes this decision and in the latter case, the choreography is changed by the character on its own and the whole timing of the animation may be changed.

A typical editing process would look like the this: Start at time 0 with a path the character should follow. Then the time is spooled forward 10 second and the subtask *wave* is send to the *Manager*. Thus the character still follows his path starting to wave at this time mark. At 15 seconds we pause the animation again and change the style of the walk movement to 'happy' by sending the corresponding command to the *Manager*. The result is a short animation, where a character starts to follow a given path, after 10 seconds starts to wave with his hands and while walking changes the style of the walk movement after 15 seconds.

It is possible that there are conflicts between different subtasks. E.g. the character could be in the middle of a pathfollowing animation and gets the command to sit down. Walking and sitting can in general not be executed simultaneously, because both movements need the same parts of the body. Thus there could be three different methods for the character to deal with the problem: ignore the sit command, execute the sit command as soon as it is possible (e.g. if the character has reached the end of his path) or abort the path following subtask and execute the sit subtask immediately. Usability tests will show us in the future, which default behaviour is preferred by the users of the application.

The commands the *Manager* receives from the user interface component are called CECommand and consist of four parts

- The *character ID* indicates the character for which this command is determined.
- The *sub task ID* indicates the sub task.
- The *command* for this subtask, like *start*, *stop* or *delete*
- A list of *parameters* which are needed to further describe the subtask.

With this simple interactive approach to create choreography scripts by moving forward and backward in time and sending commands to the character while observing the current state of the animation complex animations are created fast and easily.

## 5. CHARACTER REPRESENTATION

### 5.1 Anatomy of the *Character*

A character within the choreography editor component is represented by a *Character* object. The *ChoreographyEditor* is the interface between the *Manager* and the *Characters* (cf. Figure 2). If the *Manager* receives the command to create a new character, it sends the new build *AnimationEngine* to the *ChoreographyEditor* which creates the corresponding *Character*. The *ChoreographyEditor* also interprets the CECommands received from the *Manager*, creates appropriate commands (*Directives*) for the *Character*s and sends these commands to the *Character*.

Within the choreography editor a *Character* can be seen as an abstract representation of a character neglecting the actual appearance (e.g. textures, mesh). The *SubTask*Manager holds a varying set of *SubTasks* (cf. Figure 3). These are the implementation of reflexive behaviour like Brooks' subsumption architecture's level 0 (cf. (Brooks, 1991)) exhibiting a fixed behavioural pattern in response to given stimuli. The *SubTaskManager* controls the creation, manipulation and deletion of *SubTasks*. The *Character* also holds a reference to the *AnimationEngine*. As mentioned above, the *AnimationEngine* is responsible for the low level creation of the movement of the character. For executing their behaviour, the *SubTasks* control the animation of the character by sending commands (*AECommands*) to the *AnimationEngine*.

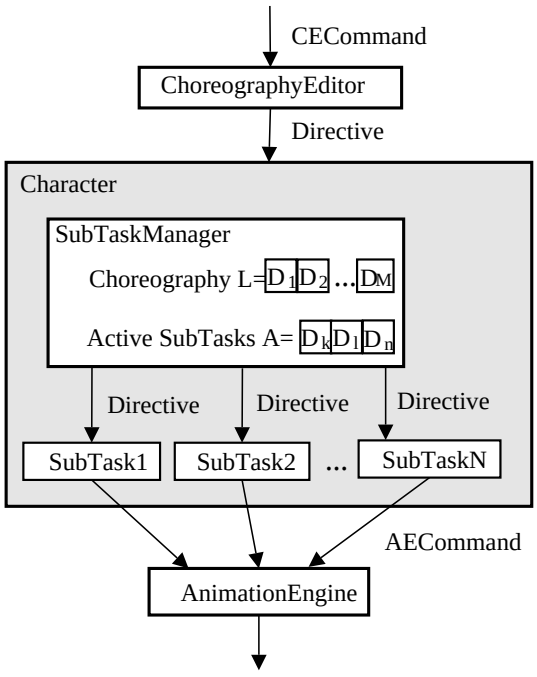


Figure 3: Structure of a *Character*.

Each *Character* has two modes, *play* and *edit* which can be set by the *ChoreographyEditor*. In the *play* mode, the *Character* reacts to every change of the current time on the global timeline, i.e. its active *SubTasks* send commands to the *AnimationEngine* to control the animation of the character at this current frame. In the *edit* mode the *Character* is able to receive new *Directives* from the *ChoreographyEditor* and changes the choreography of the character.

### 5.2 Scripting Choreographies

The time-dependent behaviour of a *Character* is ruled by the *Directives* which are created by the *ChoreographyEditor* by interpreting *CECommands*. *Directives* are commands used to create new subtasks at a current time or to change the state of an active subtask. A *Directive* $D$ can be represented as a quadruple $D = (t, S, C, < p_1, \ldots, p_n >)$ where

- $t$ is the time on the timeline where the directive is interpreted by the *SubTask*Manager.
- $S$ is the addressed *SubTask*.
- $C$ is a specific command for the $S$ which may change the state of the *SubTask*.
- $< p_1, \ldots, p_n >$ is a list of parameters for the command $C$.

The *Directives* are kept in a ordered list $L = \{D_1, \ldots, D_n\}$ where $D_i = (t_i, S_i, C_i, < p_1^i, \ldots, p_{n_i}^i >)$ are directives such that $t_i \leq t_{i+1}$ (cf. Figure 3). This list represents the choreography of the character and works like a script. The choreography can be changed by inserting new *Directives* into the list, modifying a *Directive* or deleting a *Directive* from this list. Currently it is not necessary to make a plausibility check if one adds, modifies or deletes a *Directive*. If a directive wants to modify a *SubTask* which is not existing at this specific frame, then this directive is ignored.

### 5.3 Playing Choreographies

The choreography of a character can be played by moving the current time $t_c$ on the timeline forward or backward. According to the current time $t_c$ on the timeline, the *SubTask*Manager holds the set $A$ of *SubTasks* which are active at this specific time (cf. Figure 3). First every change of the current time $t_c$ is send to every active *SubTask*. The *SubTask* eventually update their state and send (if necessary) commands to the *AnimationEngine*.

For every increase of the current simulation time $t_c$ the *SubTask*Manager checks if one of these *SubTask* has reached its goal and finished its task. This *SubTask* is taken out of the set of active *SubTasks* and deleted. Then the *SubTask*Manager interprets all those directives $D_i$ with $t_i = t_c$. Depending on the meaning of the *Directive*, two actions can follow. First the *Directive* could mean that a new *SubTask* has to be created. If the new *SubTask* can be created without colliding with another currently active *SubTask*, this *SubTask* is put into the list of active subtasks $A$. Second if the command $C_i$ of the *Directive* addresses an active *SubTask*, the *Directive* is passed to this *SubTask*. Then the *SubTask* checks the command $C_i$ and eventually changes its state according to the command.

If the current simulation time $t_c$ decreases which corresponds to a rewind on the timeline, we update the *Chacter* by playing the whole choreography $L$ from the beginning to this new time $t_c$. This has to be done, because the environment may have been changed such that subtasks get influenced from events backwards in time. To accelerate this procedure, we disconnect in this case the simulation of the characters movement from the rendering engine. Currently we do not have very complicated or long choreographies, thus this approach works still in real-time. For more complex scenarios with *SubTask* needing a lot more computer power, another approach has to be chosen.

## 6. REALISATION OF SUBTASKS

### 6.1 Two Layer Model

The structure of our character animation systems resembles Brooks' subsumption architecture. It is build upon two layers, where the lower layer models the simplest and atomic parts of character movements. The basic movements of the character are described by dynamic motion models (Grünvogel, 2003) which are described below. The upper layer lying above the motion models are the *SubTasks*, modelling reactive behaviour with the help of one or more motion models. We will now describe these layers and how they interact with each other.

### 6.2 Dynamic Motion Models

In (Grünvogel, 2003) we describe the dynamic motion models in detail, but we will here restate the major features.

The *AnimationEngine* controls the dynamic motion models. Dynamic motion models are models for movements like walking, waving with hands or jumping. They can change their movements according to some stimuli but have no planning component for complex tasks. Therefor the motion model *walk* lets the character walk straight ahead, but to follow a path the character has to be steered along the path, which is done within a *SubTask*. Each motion model can have a set of parameters which are motion model specific, e.g. for a walk motion model there are parameters describing the style of the movement (happy, sad etc.) or the speed, for a waving motion model the is the choice between left or right arm waving. In contrast to Grassias approach (Grassia, 2000), the parameters of a dynamic motion model can be changed in real-time during the execution of the motion model.

Dynamic Motion Models are implemented as state machines. All motion models have three states in common: RESET, START and STOP. In the RESET state the motion model is inactive, in START and STOP state it is active and produces animation data. After the motion model has carried out its task, it automatically switches into STOP mode, where the character is brought into a neutral pose.

The animations of the motion models are created by combining short animation clips (so called base motions) with clip operators. Clip operators are used to manipulate (e.g. time warp, loop) and combine (e.g. blend) clips. For a detailed discussion

on how these clip operators are used within motion models to create dynamic animation cf. (Grünvogel, 2003).

### 6.3 SubTasks

The motions of the dynamic motion models are only influenced by *AECommands* or by geometric objects in the virtual environment. The *AECommands* are used for changing the individual characteristics of the resulting motion. The virtual environment where the geometric representation of the character is acting puts geometrical constraints upon the movement.

*SubTasks* are used to solve more general goals than motion models, like following a path or another character. These goals are characterised by the fact that they can not be fulfilled by starting only a motion model with a given set of parameters. Instead the goals of the *SubTask* can be reached by controlling motion models while they are executed. Thus depending on the current situation of the character the *SubTasks* sends *AECommands* to the motion model to change its behaviour. The *SubTask* can react up to a certain degree upon obstacles to the goal. But at the layer of the *SubTasks* planning in the sense that the character uses beliefs about the situation and the consequences of movements to search for a solution in a more abstract space is not intended. This could be implemented in a layer upon the *SubTasks*.

## 7. CONCLUSION AND FURTHER WORK

We have introduced a character animation subsystem for the use in an augmented reality environment. Simple Animations are created by placing commands on the timeline of a virtual character. These commands are interpreted by the choreography editor and turned into *SubTasks*. *SubTasks* are models for reactive behaviour and create their animation by controlling dynamic motion models. Dynamic motion models implement basic motions, where the characteristics of the motions can be changed in real-time.

At the moment the final design of the user interface within the augmented reality system and usability tests are still pending. We hope that these tests will bring us more insight, which features are still missing in the character animation system. Another interesting point of research is to put an additional layer upon the given two layers choreography editor and animation engine for the implementation of planning and learning processes.


## ACKNOWLEDGEMENT

This work was supported by the German Ministry of Education and Research (BMBF Grant 01 IR A04 C: mqube - Eine mobile Multi-User Mixed Reality Umgebung, www.mqube.de).


# References


Azuma, Ronald T. 1997. A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, **6**(4), 355–385.

Badler, Norman I.; Phillips, Cary B., & Webber, Bonnie Lynn. 1993. *Simulating Humans: Computer Graphics and Control*. Oxford University Press.

Balaguer, Jean-Francis, & Gobbetti, Enrico. 1995. Sketching 3D Animations. *Computer Graphics Forum*, **14**(3), 241–258.

Balaguer, Jean-Francis, & Gobbetti, Enrico. 1996. 3D User Interfaces for General-Purpose 3D Animation. *IEEE Computer*, **29**(8), 71–78.

Balcisoy, Selim; Kallmann, Marcelo; Torre, Remy; Fua, Pascal, & Thalmann, Daniel. 2001. Interaction Techniques with Virtual Humans in Mixed Environments. *In: International Symposium on Mixed Reality, Yokohama, Japan*.

Brooks, Rodney A. 1991. Intelligence Without Representation. *Artificial Intelligence*, **47**, 139–159.

Epic Games. 2003. *Unreal Tournament 2003*. Atari. http://www.unrealtournament2003.com.

Grassia, F. Sebastian. 2000. *Believable Automatically Synthesized Motion by Knowledge-Enhanced Motion Transformation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh.

Grünvogel, Stefan M. 2003. Dynamic Character Animation. *International Journal of Intelligent Games & Simulation*, **2**(1), 11–19.

Kalra, Prem; Magnenat-Thalmann, Nadia; Moccozet, Laurent; Sannier, Gael; Aubel, Amaury, & Thalmann, Daniel. 1998. Real-Time Animation of Realistic Virtual Humans. *IEEE Computer Graphics and Applications*, **18**(5), 42–57.

Perlin, Ken, & Goldberg, Athomas. 1996. Improv: A System for Scripting Interactive Actors in Virtual Worlds. *Computer Graphics*, **30**, 205–218.

Sannier, Gael; Balcisoy, Selim; Magnenat-Thalmann, Nadia, & Thalmann, Daniel. 1999. "VHD: A System for Directing Real-Time Virtual Actors. *The Visual Computer*, **15**(7/8), 320–329.

Tamura, Hideyuki. 2000. Real-Time Interaction in Mixed Reality Space: Entertaining Real and Virtual Worlds. *In: Proc. Imagina 2000*.

Torre, Rémy; Fua, Pascal; Balcisoy, Selim; Ponder, Michal, & Thalmann, Daniel. 2000. Augmented Reality for Real and Virtual Humans. *Pages 303 – 308 of: CGI 2000*. IEEE Computer Society Press.