NICE project (IST-2001-35293)

Natural Interactive Communication for Edutainment

NICE Deliverable D3.7-2a
Multimodal Output Generation Module for the Second Prototype for H.C. Andersen

22 December 2004

Editor

LIMSI-CNRS : Jean-Claude Martin

Authors

NISLab : Andrea Corradini, Niels Ole Bernsen

Project ref. no.	IST-2001-35293
Project acronym	NICE
Deliverable status	Restricted
Contractual date of delivery	1 November 2004
Actual date of delivery	22 December 2004
Deliverable number	D3.7-2a
Deliverable title	Multimodal Output Generation Module for the Second Prototype for H.C. Andersen
Nature	Report
Status & version	Final
Number of pages	13
WP contributing to the deliverable	WP3
WP / Task responsible	LIMSI-CNRS
Editor	Jean-Claude Martin
Author(s)	Andrea Corradini, Niels Ole Bernsen
EC Project Officer	Mats Ljungqvist
Keywords	Multimodal output, non-verbal behaviour generation
Abstract (for dissemination)	This report, Deliverable 3.7-2a from the HLT project Natural Interactive Communication for Edutainment (NICE), describes the multimodal output generation module for the second prototype for H.C. Andersen.

Table of Contents

1 Introduction

This report, Deliverable 3.7-2a from the HLT project Natural Interactive Communication for Edutainment (NICE), describes the multimodal output response generation module (RG) for the second prototype for H.C. Andersen (HCA). The multimodal output generation module for the first prototype was described in D3.7-1 (Boye et al. 2003).

Multimodal output for PT2 is already partly described in D5.2a (Bernsen et al. 2004). The current report describes in more details how PT2's augmented rendering functionality is applied to PT2 non-verbal and combined verbal and non-verbal response planning. HCA can move about and interact with his environment as well as communicate with the user through spoken conversation and non-verbal gesture, body posture, facial expression and gaze. The described approach aims to make the virtual agent's appearance, voice, actions, and communicative behaviour convey the impression of a character with human-like behaviour, emotions, relevant domain knowledge, and a distinct personality. Multimodal output generation exploits parameterized semantic instructions from the conversation manager and splits the instructions into 1) synchronized text instructions to the text-to-speech synthesizer, and 2) behavioural instructions to the animated character (Corradini et al. 2004; Corradini et al. 2005).

Our animated character is built upon a hierarchy of bones that we refer to as a frame. The frame, together with a textured polygon mesh and skin weighting information, is represented as a skinned mesh (Figure 1). The skin weighting information specifies the influence the frame has on its mesh. The root frame node contains a transformation matrix relative to the world space. An animation that affects the root node affects the whole scene while an animation that affects a leaf node does not affect any other node. We use the frame to give the system the functionality of overloading animations for different body parts.

The animation system receives network commands and schedules animation events via the scheduler system. A valid command to concurrently carry out sequences of animations while synthesizing a sound file is an XML string whose syntax looks like:

```
<play>
        <sound> SOUND </sound>
        <animList Track= P1> T11,A11; .. </animList>
        <animList Track=P2> T21,A21; .. </animList>
</play>
```

The tag <sound> is utilized to play back the audio file SOUND either as positioned or global sound. The items <animListTrack=Pj> contain a list (or track) of animations A11,.. to be played in sequence in accordance with their start times T11,.. relative to a common time. To resolve conflicts that may occur while animations are rendered in parallel, each track is assigned a priority value Pj. If at any given time more animations affect, in different ways, the same node within the frame, the animation within the track with higher priority will prevail, i.e., be displayed. In general, an animation can be played only if an existing file that contains the specification of the actual behaviour has been loaded into memory by the rendering engine upon start-up. We refer to these animations as elementary animations or primitives.

From a technical point of view, it may be useful to distinguish response planning for non-communicative actions and communicative functions (section 2), on the one hand, and response planning for communicative actions on the other (section 3).

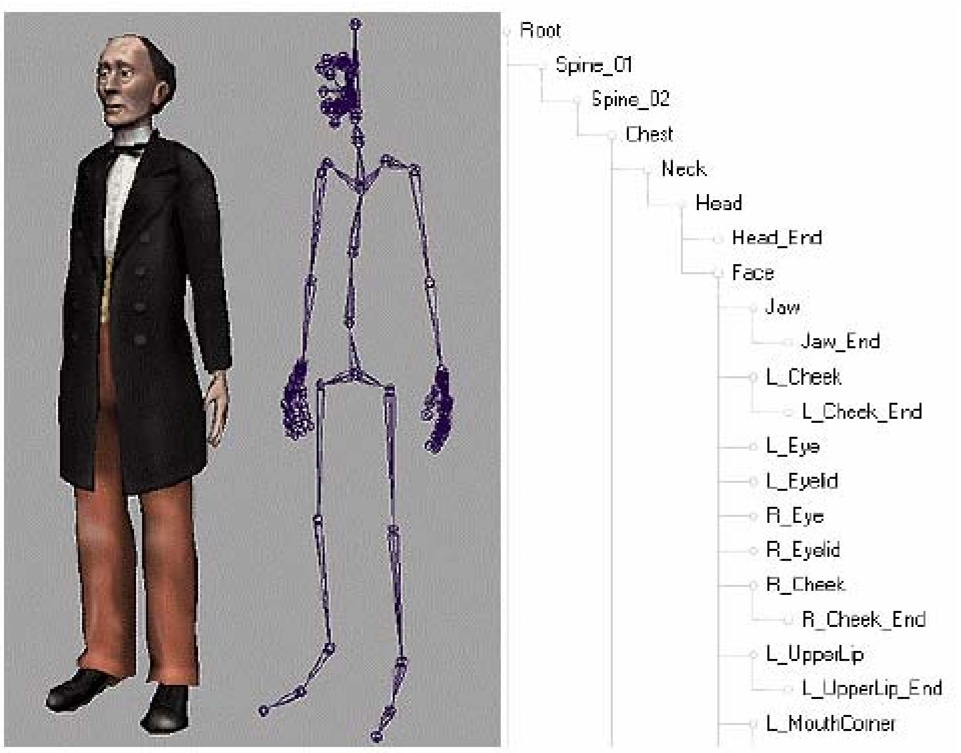

Figure 1: Skinned mesh of the HCA character.

2 Non-communicative action and communicative function

A non-communicative action output state refers to the situation in which HCA is not engaged in conversation with a user but goes about his normal work and life within his study. This may happen either at system start-up if there is no user around, or after the user stops conversation and walks away. In general, while in this state, the character does not produce spoken utterances in terms of full-form sentences. Yet, the system is capable of playing back, e.g., footsteps when HCA walks, or music sounds when he, say, enacts a dance. Sound files in wav format and mp3 are stored to allow for that capability.

In our implementation, we employ a hierarchical two-tiered approach that supports the designer in creating those output states through scripts. Lower-level scripts define sets of elementary animations along with their temporal specification and sets of custom animations. Elementary animations are behaviours that can be rendered by the animation engine by just one command and that belong to the core of the application as default animations. Custom animations are sequences of such elementary behaviours.

In the following an excerpt from our software that show both elementary animations (their names are self-explanatory) and how custom behaviours are generated using them as building blocks:

```
NOD             LASTS       3000        MSECS
HEADUP          LASTS       3000        MSECS
HEADDOWN        LASTS       3000        MSECS
TILTHEADLEFT    LASTS       3000        MSECS
TILTHEADRIGHT   LASTS       3000        MSECS


SEQUENCE SEQ_STARTMUSIC
        SEQ_GOTOCENTER THEN WAIT 4000 MSECS
        PLAY SOUND FROM POOL 2
        GO TO Desk
        WAIT 2000 MSECS
        SET CAMERA LookAtActorCamera
        WAIT 2000 MSECS
        TURN TO CAMERA FROM POOL 1
        WAIT 2000 MSECS

PARALLEL PAR_DANCESTEP1
        TRACK WITH PRIORITY 1
        ANIMATION HEADUP        AFTER 0    MSECS PLAYS 100 PERCENT AT SPEEDRATE 0.5
        ANIMATION HEADDOWN      AFTER 2000 MSECS PLAYS 100 PERCENT AT SPEEDRATE 0.5

        TRACK WITH PRIORITY 2
        ANIMATION ARMSHOLDINGOBJECT   AFTER 0    MSECS PLAYS 50 PERCENT AT SPEEDRATE 1.0
        ANIMATION RESTARMS            AFTER 3000 MSECS PLAYS 100 PERCENT AT SPEEDRATE 1.0
```

As one can see, we have two kinds of possible custom behaviours: sequences of elementary ones and overlapped animations. Recursive calls of sequences are allowed as showed in the sequential SEQ_STARTMUSIC which calls the other sequence SEQ_GOTOCENTER from within it. Sequences of elementary and/or custom animations can thus be used to define complex behaviours. To make the character perform a richer variety of behaviours, we make use of placeholders within the sequences for animations and other elements to be selected at run-time. In more detail, we define several sub-sets of behaviours, objects, locations, cameras and sounds, assign them a name, and select them in a non-deterministic way, one each time the placeholder refers to that particular subset. In the example above, for example, the sequence animation SEQ_STARTMUSIC is capable to play several different sounds according to the music file chosen at run time given that the sound is specified to be chosen from a certain pool of sounds by the command PLAY SOUND FROM POOL 2. The same applies in the choice of the camera, just few lines later in the behaviour. Each time the behaviour is rendered a run-time random function selects the camera view from a sub set of camera definitions.

We also use syntactic rules to define and provide appropriate transitions among animations in order to produce believable and smooth interactive character behaviour.

An example is the following excerpt from a file, which specifies a transition such as:

```
ANIMATION SEQ_WALKTOWINDOW CANNOT BE FOLLOWED BY [SEQ_WALKTOWINDOW,ARMSONDESK]
```

The use of commands such as GO TO <LocationName> gives HCA own locomotion capabilities and makes the behaviours more life-like and not restricted to a single location. Currently, we still have the issue of obstacle avoidance. In fact, if HCA is sent to a location but he is hindered by an object in his path to get there, he gets stuck in that state and cannot move further without external help from the user via the use of the keyboard. Scripts are then defined on top of behaviours and their transactions. Here an excerpt from our code:

```
SCRIPT 0

SEQ_WALKTODESK THEN WAIT 4000 MSECS
SEQ_STUDYATDESK THEN WAIT 3000 MSECS
SEQ_LOOKAROUND THEN WAIT 4000 MSECS
SEQ_RESET THEN WAIT 4000 MSECS
```

We use again syntactic rules to define and to provide appropriate transitions among scripts. So, for example, the rule:

```
SCRIPT 0 CANNOT BE FOLLOWED BY [0-6,9]
```

is a possible rule that states that script numbered with 0, if chosen at run time and after being realized, cannot be followed by scripts 9 and all those from number 0 through 6. In that case, the Character Module, in which the decision about which script to run resides, is constrained in its choice of the next script yet it is guaranteed that script transitions occur smoothly without abrupt movements between the end of a script and the start of the following one.

Figure 2 depicts a couple of screen shots of some non-communicative actions showing HCA looking out of the window and bending over his desk within the study. The same scene of HCA bending over into his desk is shown with different camera view that is chosen randomly at run-time.

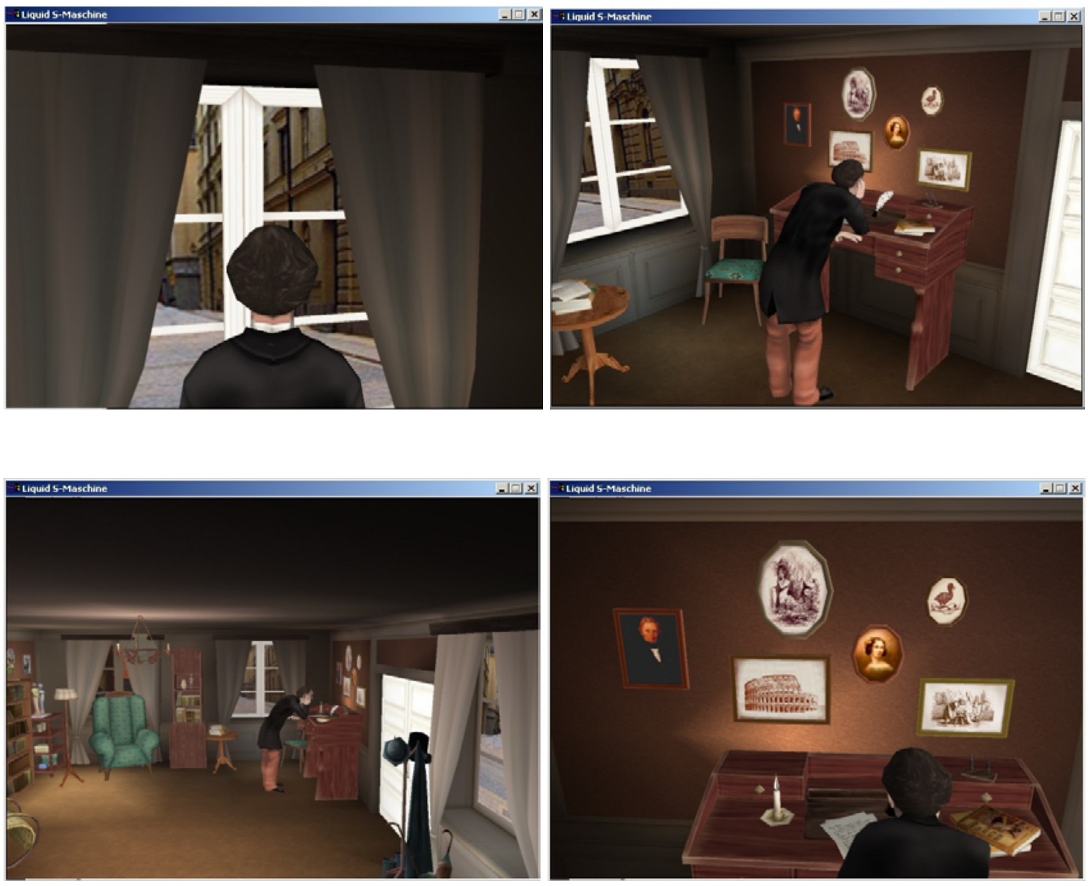

Figure 2: screenshots for non-communicative actions: (from left to right, top to down) HCA looking outside the window, HCA leaning over his desk from three different camera views.

While in the communicative function output state, we have made our character show his awareness of being spoken to or otherwise addressed by the user, by employing a rather ‘neutral’ and general set of animations or sequences thereof. The relative ‘neutrality’ of these behaviours is imposed by the technical limitations which prevent us from processing the user’s input incrementally in real time. This means that he cannot react, *while* being addressed, to parts of the user input which, given his personality, should otherwise make him react emotionally or cognitively.

Much like Steve (Rickel and Johnson 2000), HCA uses gaze, deictic and body orientation as a cue to his attentional focus. For example, when an object is gestured by the user and detected by the Gesture Interpretation module (GI), a message sent by the GI drives HCA to gaze and orient his body towards the gestured object. This fast feedback using natural modalities aims at giving the user the understanding that the system detected the gesture. Regardless the communicative aspects and/or function, the RG listens to messages coming from the GI generated in the case of any object selection by the user for the Hans Christian Andersen character to turn to the object signalled. An example of such a behaviour is given in Figure 3.

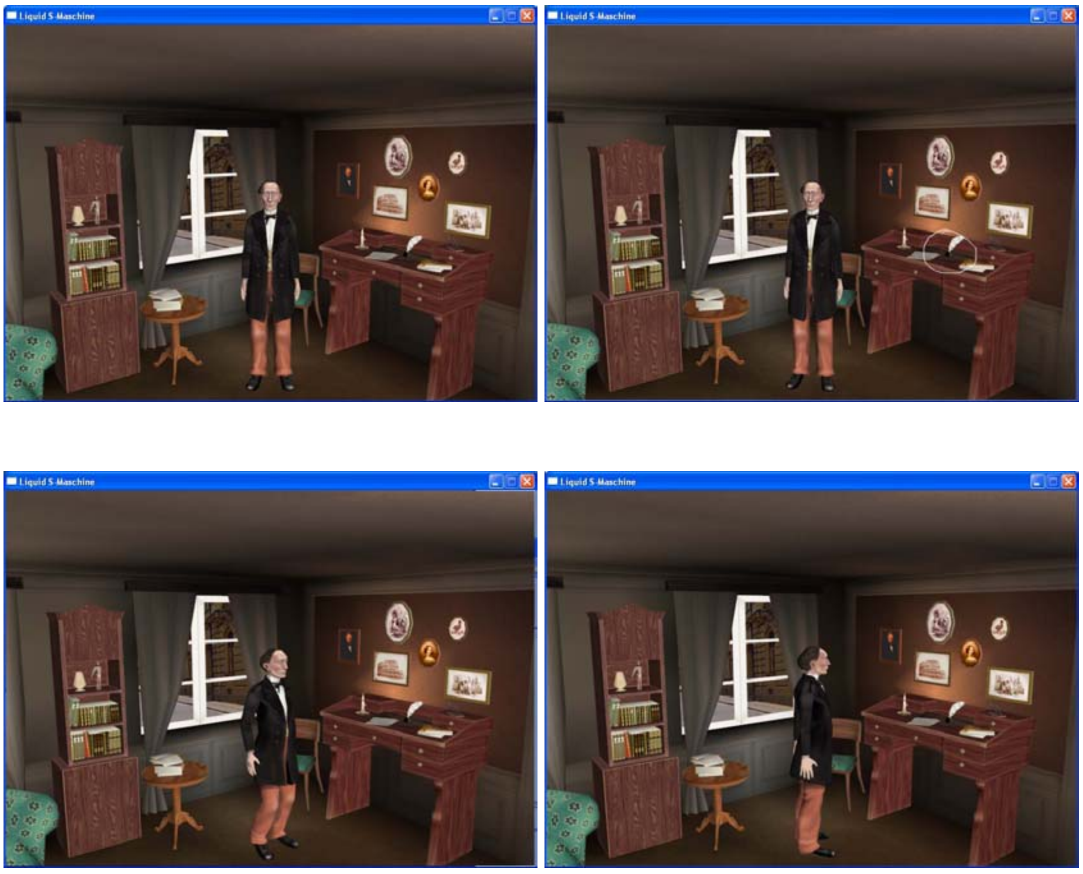

Figure 3: the character turned to the selected object.

3 Communicative action

When conversation is going on between the user and HCA and it is HCA's turn, the Character Module looks for HCA's conversation contribution within a knowledge base that stores many predefined sentences along with encoded non-verbal behaviours, semantic classes, and the system's domain ontology. Numerous, ever-expanding in number, canned templates guarantee broad domain coverage but also require manual maintenance and have a limited variability by design. Each template is a compact representation of a predefined spoken output with embedded start and end tags for non-verbal behaviours and placeholders to text values to be filled in at run-time. The following is an example of behavioural template:

Now, tell me [g0] your [/g0] {EMOTION ADJ_2} opinion about {FAIRYTALE}

Here, elements within square brackets starting with numbered *g* letters represent onset and offset of non-specified non-verbal parts of the template. Elements within curly parentheses, like *FAIRYTALE,* are placeholders for text-to-speech values to be filled in using input value information. The other elements within curly parentheses, starting with the string *EMOTION*, are TTS values as well but these are tied to emotional values. In the present example, *ADJ_2* indicates a set of emotional value/text pairs from which the verbal realization for the appropriate text has to be retrieved. Both TTS variable values and non-verbal behavioural elements are initially un-instantiated. The binding of non-verbal behaviour to gesture and TTS variables to text occurs at run-time rather than being hard-coded, enabling a sentence to be synthesized at different times with different accompanying non-verbal elements and/or words.

To have what we call *verbal differentiation* due to emotional state, we use a template for each different emotional state, so e.g. the template above is associated with a set of emotions but for instance not with an angry emotion. In that case we have the template specified for an angry emotion and with different distribution of graphical behaviours attached. The emotion calculator passes the emotional state to the RG during execution and this latter chooses the template according to the current state.

The pair of tags that marks start and end of any non-verbal behavioural element supplies implicit timing information for speech and gesture during rendering. In the behavioural template above, tags *[g0]* and *[/g0]* indicate that an animation may co-occur with uttering the spoken text '*your'* around which they are wrapped. A certain gesture is selected for insertion in place of *g0* depending on the semantic class(es) of the text surrounded by the placeholders. Tables that map semantic categories onto non-verbal behaviours are maintained. Let us assume that a *POINT* animation is selected to expand the non-verbal behaviour *g0*, while the textural placeholders *EMOTION ADJ_2* and *FAIRYTALE* are expanded to *valuable* and *the Princess and the Pea,* respectively. The behavioural template is then converted into the surface language string:

Now, tell me [POINT] your [/POINT] valuable opinion about the Princess and the Pea

We have been implementing a strategy different from the one in PT1 to deal with such surface representation. The RG still replaces non-verbal behaviour references with bookmarks that can be dealt with by a text-to-speech component. Then, the entire string containing the TTS bookmarks is sent to the TTS, which synthesizes the verbal output. Any time a bookmark is encountered, the TTS fires an event and calls on the Response generator to create the XML string representation of the corresponding animation. In PT2 however, we first parse the surface string for the TTS module to create wav files of text enclosed within animation bookmarks and determine its temporal duration. During parsing, the surface string is broken down into sequential segments of either audio-only segments, animation-only segments, or parallel audio and video segments.

Three XML strings would be generated when parsing the surface string of our previous example:

```
1) <play> <sound> SOUND_1 </sound> </play>
2) <play>
      <sound> SOUND_2 </sound>
      <animList Track=0> 0, POINT; </animList>
   </play>
3) <play> <sound> SOUND_3 </sound> </play>
```

Here, *SOUND_1* contains the synthesized text *now tell me, SOUND_2* the text *your*, and eventually the verbal synthesis for *valuable opinion about the Princess and the Pea* is stored into *SOUND_3*. The animation *POINT* is stretched over a time period equivalent to the duration of the sound file *SOUND_2*. To have HCA point to an object or location we first have to make him turn to that location/object and then perform a pointing gesture.

Once all XML segments are created, they are sequentially sent to the graphical animation engine that automatically coordinates playback of sound and non-verbal behaviour rendering for each of them. This approach is suitable for short behavioural templates because it requires the data to be analyzed twice: first parsing the template to create single wav files, then go through it again to break it down into single segments to send to the animation engine. Long templates are a technical issue wrt. this approach. Thus, we prefer to break down as many of them as possible as sequences of shorter ones.

The approach is, nevertheless, technically more challenging than the one followed in PT1, but we wish to mention the problems that occur in fine-tuning the duration of single animations for each of them to last exactly as long as the sound files they play along with, independently of the machine that runs the application. These technical problems are particular visible in the rendering of synchronized visemes and speech to be played back.

In addition to elementary animations, more complex non-verbal behaviours can be created, combined, sequenced, assigned a name, and stored by the RG. To that end, we have employed a generative approach based on a layered composition of primitives similar to that described in the previous sub-section.

Since the rendering engine can play only elementary animations, the Response Generator has to break any user-defined animation, let's call it *parent animation* for the sake of clarification, down into its primitive components and create XML representation strings for each of them. If any of these components is by itself a compound animation, it has to be recursively decomposed as well, until only primitives are used to express the parent animation. Sequentiality, parallelism, and partial overlapping of existing animations to create new behaviours can be tuned by setting appropriate values for the temporal items in the XML representation.

Currently, we store some 300 output templates (some 1200 if the emotion is taken into account), many of which are no-variable stories to be told by HCA, and 150 different non-verbal primitives. Templates have been designed by hand and, similarly to non-verbal behaviours, were partly inspired by analysis of data from recordings of an actor impersonating HCA and interacting with kids in a children's theatre class in the fairy-tale writer's hometown Odense, Denmark.

Table 1 shows an excerpt from a conversation between a child user and HCA. At turn #9, HCA starts meta-communication due to the absence of (gestural and spoken) user input over a certain period of time. The gestures occurring in that dialogue can vary, depending on the mapping from the semantic categories onto non-verbal behaviours. The CM may also decide to leave some of them un-instantiated. Thus, e.g., at turn #8 less than three gestures are attached to the output.

Table 1. Excerpt from conversation between HCA and a child

turn #7

Child's spoken input: *ehm how old are you now?*

HCA output: *I am 55 years old. [g0] And I am in a good shape [/g0] occasionally I ride on horses and go for long walks and [g1] when I travel [/g1] I even sometimes hike*

Animations selected:

g0 = ARMS_CROOKED

g1 = JUMP

turn #8

Child's spoken input: *okay*

HCA output: [*g0][/g0][g1][/g1][g2][/g2] my shoe size is 48 [g3][/g3]*

Animations selected:

g0 = RAISE_EYEBROWS

g1 = LAUGHING

g2 = LOOK_AT_DOWN

g3 = SMILE

turn #9

Child's spoken input: NONE

HCA output: *silence is gold but silver is very beautiful*

Animations selected: NONE

turn #10

Child's spoken input: *what are you doing now?*

HCA output: *I am doing many different things in my study. I am [g0] writing, [/g0] [g1] thinking, [/g1] [g2] reading, [/g2] singing, dancing, and so on.*

Animations selected:

g0 = LEFTARM_CIRCULAR_MOVE

g1 = RAISE_EYEBROWS

g2 = FURROWED_EYES

4 References

Bernsen, N. O., Charfuelàn, M., Corradini, A. and Mehta, M. (2004). NICE Deliverable D5.2a. Second Prototype Version of Conversation Management and Response Planning for H.C. Andersen., 29 October.

Boye, J., Gustafson, J., Wirén, M., Mehta, M., Corradini, A., Bernsen, N. O., Fredriksson, M. and Martin, J.-C. (2003). NICE Deliverable D3.7-1. Multimodal output generation module for the first prototype., 17 October.

Corradini, A., Fredriksson, M., Mehta, M., Königsmann, J., Bernsen, N. O. and Johanneson, L. (2004). Towards believable behavior generation for embodied conversational agents. Workshop on Interactive Visualisation and Interaction Technologies (IV&IT 2004) in conjunction with the International Conference on Computational Science 2004 (ICCS 2004) Krakow, Poland, June 7-9. http://www.nis.sdu.dk/publications/2004/RG_NICE_IV&IT04_26.4.04-F.pdf

Corradini, A., Mehta, M., Bernsen, N. O. and Charfuelan, M. (2005). Animating an Interactive Conversational Character for an Educational Game System. Intelligent User Interfaces (IUI'2005) San Diego, California, January 9 - 12.

Rickel, J. and Johnson, W. L. (2000). Task-oriented collaboration with embodied agents in virtual worlds. Embodied Conversational Agents. J. S. Cassell, J.; Prevost, S.; Churchill, E., Cambridge, MA: MIT Press: 95 - 122.http://citeseer.nj.nec.com/article/rickel00taskoriented.html