Facial Expressions for Embodied Agents in STEP

Zhisheng Huang, Anton Eliëns, and Cees Visser
Department of Computer Science, Vrije University Amsterdam,
De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
email: {huang,eliens,ctv}@cs.vu.nl

Abstract

STEP is a scripting language for embodied agents. STEP has been implemented in DLP, an object-oriented parallel logic programming language. This paper discusses how the scripting approach can be extended with facial expressions for embodied agents in STEP. The scripting language STEP can serve both as a high-level specification language for knowledge representation and reasoning, and as a lower-level programming language for animation and interaction of embodied agents. In addition, the STEP scripting framework provides a comprehensive set of sensors and effectors by means of which agents can perceive their world or can take appropriate actions, respectively.

1. Introduction

The distributed logic programming language DLP [1, 2] combines logic programming, object-oriented programming and parallelism. DLP has been used as a tool for web agents, in particular for 3D web agents[3, 4, 5]. The use of DLP as the language for a web-based virtual agent platform is motivated by the following language characteristics: object-oriented Prolog, VRML EAI (external authoring interface) extensions, and distributed processing facilities.

DLP incorporates multi-threaded object-oriented programming concepts, which make it a useful tool for programming. The language accepts the syntax and semantics of logic programming languages like Prolog. It is a high-level declarative language suitable for the construction of distributed software architectures in the domain of artificial intelligence. In particular, it's a flexible language for rule-based knowledge processing.

STEP (Scripting Technology for Embodied Persona)[1] is a scripting language for embodied agents[6]. The design of STEP was motivated by the following principles: convenience, compositional semantics, re-definability, parameterization, and interaction. See [6] and [7] for details. The principle of convenience implies that STEP uses natural-language-like terms for 3D graphical references. The principle of compositional semantics means that STEP has a set of built-in action operators. The principle of re-definability implies that STEP incorporates a rule-based specification system. The principle of parameterization justifies that STEP introduces a Prolog-like syntax. The principle of interaction requires that STEP is based on a more powerful meta-language, like DLP.

STEP is a scripting language for H-Anim based embodied agents. According to the H-Anim standard, an H-Anim specification contains a set of Joint nodes that are arranged to form a hierarchy. Turn and move operations are the two main primitive actions for body movements in STEP. Turning body parts of humanoids implies the setting of the corresponding joint's rotation. Scripting actions can be composed by using the following composition operators: the sequence operator 'seq', the parallel operator 'par', and the repeat operator $repeat(Action, T)$. When using high-level interaction operators, scripting actions can directly interact with the internal state of embodied agents or with the external state of worlds. These interaction operators are based on a meta-language which is used to build embodied agents. Typical higher-level interaction operators in STEP are the do-operator $do(\phi)$, which executes a goal ϕ at the meta language level, and the conditional $if\_then\_else(\phi, Action1, Action2)$, which executes $Action1$ if state ϕ holds, otherwise executes $Action2$.

The scripting language STEP is primarily designed for the specification of a body language for virtual agents. In this framework, the specification of external-oriented communicative acts can be separated from the internal states of virtual agents because the former involves only geometrical changes of the body objects and the natural transition of the actions, whereas the latter involves more complicated computations and reasoning. In the DLP+VRML platform, we usually use STEP to define the animation actions for body movements and use its meta-language DLP for reasoning, planning, and agent modeling. The predicates in the

1 http://step.intelligent-multimedia.net/

scripting language STEP and its meta-language DLP can call each other by their interaction operator. In STEP, the do-operator $do(\phi)$ is used to call a goal ϕ in DLP, whereas in DLP the formula $do\_script(Script)$ is used to call a script in the STEP animation engine.

This paper discusses how the scripting approach can be extended with facial expressions for embodied agents in STEP. In Section 2 we give an overview of the facial models and facial animation primitives in STEP. Section 3 describes how the presented animation primitives can be used to express MPEG-4 facial expressions. In Section 4 we discuss examples of eye movements in STEP and show that the interaction between perception and actions can be specified in an efficient way for embodied agents. Section 5 discusses the implementation briefly. Section 6 concludes the paper.

2. Facial Expressions in STEP

MPEG-4 FAPs (facial animation parameters) offer a convenient way to reference facial animation points. We have developed VRML/X3D based face models according to an MPEG-4 FAP like model for H-Anim avatars. A well calibrated facial model is defined by only one indexed face set and usually consists of a few thousand vertices / polygons. However, in a number of cases it's more appropriate to define particular facial objects (like eyeballs) as independent objects with their own indexed face set. The motivation is not only because of performance considerations, but also for a more convenient way of authoring. For example, in order to make an avatar gaze at a particular object, it is more convenient to handle the corresponding eyeball movements independently by setting the rotations of the eyeballs; this way, eyeballs can be manipulated like any other body joint in the H-Anim specification: in STEP, we can use a scripting action like

$$turn(Agent, l\_eyeball, rotation(0, 1, 0, 0.37), fast)$$

to turn the left eyeball to the left. Therefore, we design facial models as an object hierarchy that corresponds to the H-Anim specification. Figure 1 shows a face object hierarchy with its H-Anim specification.

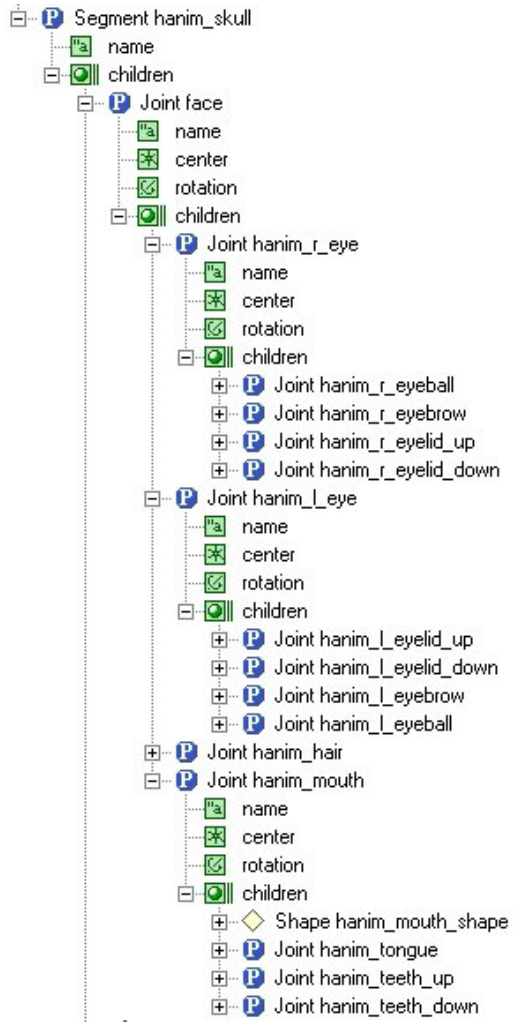

Figure 1. Face Object Hierarchy in H-Anim

STEP can manipulate MPEG-4 FAP-based facial models, irrespective of whether it is based on a well-calibrated realistic model or a less detailed mouth object model, like Figure 2. However, experiments have shown that even an approximate mouth model like Figure 2 can achieve a satisfactory realism with a good performance. We discuss the performance issue further in Section 5. Several facial expressions based on such a mouth model are shown in Figure 3

Although MPEG-4 FAP-based facial models are convenient to reference certain points at faces, they are not convenient to specify the relationship between multiple facial points for group movement of facial expressions. For example, 'fap8.1' refers to the middle point of the outer upper-lip contour. Moving the point 'fap8.1' up should also result in a position change of its neighboring points or FAPs, like 'fap2.2', the middle point of the inner upper-lip contour. In MPEG-4, FATs (facial animation tables) are usually used to define how a model is spatially deformed as a function of FAP amplitudes. The design of a good FAT for a realistic MPEG-4 based animation is a time-consuming activity [8]. Waters' muscle model [9] is a well-known model to specify the relationship between facial points when they are animated. Waters' muscle model is intuitive and computationally cheap. We implemented our face models with MPEG-4

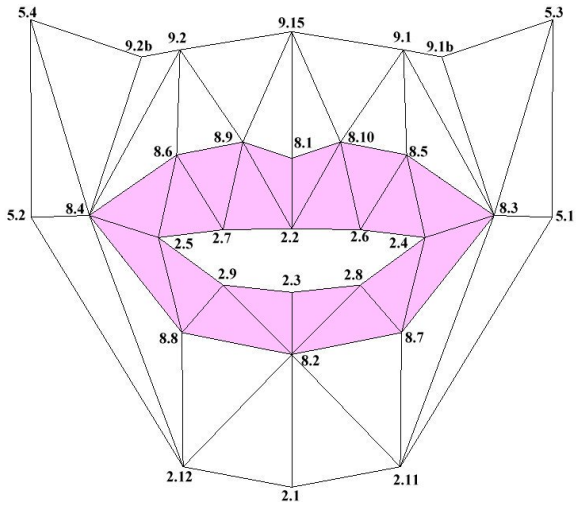

Figure 2. Mouth Object with MPEG-4 FAPs

FAPs and FATs and incorporated Waters' muscle model[9] for improved realism and a good performance[10].

The mouth object is the most important part of a facial model, because it is crucial for facial expressions and lip-speech synchronization. A well calibrated mouth object should cover at least almost the bottom half of the head, so that it can simulate muscle movements and cheek rotations when avatars talk. A less calibrated mouth object may consist of only its relevant MPEG-4 FAPs and some neighboring vertices, as is shown in Figure 2.

STEP can manipulate MPEG-4 FAP-based facial models, irrespective of whether it is based on a well-calibrated realistic model or a less detailed mouth object model. Several facial expressions based on such a mouth model are shown in Figure 3.

2.1. Primitive Face Actions in STEP

The primitive operations for facial animation in STEP are $getFAP$ and $setFAP$:

- $getFAP(Agent, FAP, position(X, Y, Z))$: retrieves the position $\langle X, Y, Z\rangle$ for the specified FAP of the agent.
- $setFAP(Agent, FAP, Position)$: sets the position for the FAP of the agent, neighboring points/vertices are also affected according to the facial animation table.

The following four $setFAP$ predicate formats can be used to modify positions. The first two formats are primarily intended for general body manipulations. The latter two formats are used for facial animations.

- position(X,Y,Z): $position(X, Y, Z) = \langle X, Y, Z\rangle$.
- increment(X,Y,Z): $increment(X, Y, Z) = \langle X_1 + X, Y_1 + Y, Z_1 + Z\rangle$.
- change(X,Y,Z): $change(X, Y, Z) = \langle X_0 + X, Y_0 + Y, Z_0 + Z\rangle$.
- change(X,Y,Z, FAPU): $change(X, Y, Z, FAPU) = \langle X_0 + X \times unit(FAPU), Y_0 + Y \times unit(FAPU), Z_0 + Z \times unit(FAPU)\rangle$.

Assume that the initial value of a FAP is $\langle X_0, Y_0, Z_o\rangle$, the current value is $\langle X_1, Y_1, Z_1\rangle$, and the unit of a $FAPU$ is $unit(FAPU)$. A $setFAP$ action with the format $position(X, Y, Z)$ sets the FAP to the absolute value $\langle X, Y, Z\rangle$. Of course, a facial operation with this format is only appropriate for avatars with identical 3D geometrical data.

A $setFAP$ with the format $change(X, Y, Z)$ sets the FAP by increasing the *initial* value of vertices with the value $\langle X, Y, Z\rangle$. Standard avatars have the same initial default model. For example, for H-Anim avatars, the face should be modeled with the eyebrows at rest, the mouth closed and the eyes open. Changing the FAP value relative to its initial value results in almost the same effect if the 3D geometrical data, in particular the height of avatars don't differ too much, e.g. they represent all adults. We have found that facial operations with this $change$ format have the reusability property for all adult avatars in a satisfying way. In order to reuse facial actions with different geometrical data, say, apply a script which is originally designed for adult avatars to child avatars, we can use the $change$ format with an additional facial animation parameter unit (FAPU).

The facial action with an $increment(X, Y, Z)$ specification sets the FAP by increasing the current values of the vertex by $\langle X, Y, Z\rangle$. Again, it is easy to see that a facial action with this incremental specification cannot be reused for different face models.

2.2. Initialization and Parallelism

Facial animations involve time-dependent parallel actions. We need several facial actions which combine setFAP operations with parallelism and interpolation. Furthermore, facial actions based on the $change$ format have a strong reusability property, therefore we need a facility in the facial animation engine to obtain the initial data of a model. The following script actions for initialization, parallelism, and interpolation are defined:

- $fae\_init(Agent)$: the facial animation engine retrieves the initial data of the face model of the agent. This action is necessary for other facial actions with the '$change$' format.
- $getFAPInit(Agent, FAP, Position)$: gets the initial FAP value of the agent. We use this action to reset an FAP to the default value.

- $resetShape(Agent, Object)$: resets the initial $Object$ values of the agent. We use this action to reset a facial object to its default shape.
- $setFAPs(Agent, FAPChange)$: set a number of FAPs simultaneously. FAPChange is a list of '$change$' specifications: $[fap(FAP_1, Change_1), ..., fap(FAP_n, Change_n)]$. The script action $setFAPs(Agent, FAPChange)$ is semantically equivalent to the action $par([setFAP(Agent, FAP_1, Change_1), ...])$, however, the former uses only one execution thread to achieve the parallel action, while the latter uses multiple threads.
- $move\_FAPs(Agent, FAPChange, Speed)$: set a number of FAPs simultaneously with the specified $Speed$. During the corresponding time interval, the facial animation engine performs an interpolation between the current values and the destination values to achieve a smooth animation.
- $move\_FAPs(Agent, FAPChange, Intensity, Speed)$ corresponds to the action $move\_FAPs(Agent, [fap(FAP_1, Change_1 \times Intensity), ..., fap(FAP_n, Change_n \times Intensity)], Speed)$.

3. Emotion Expressions

MPEG-4 FAP2 defines six primary facial expressions: joy, sadness, anger, fear, disgust, and surprise. In this subsection, we show how these six facial expressions can easily be defined as built-in STEP scripts. Facial expressions can have different intensities. We use the predicate

$$facialExpression(Agent, Expression, Intensity)$$

to represent the facial expression for $Agent$ with a particular $Intensity$.

The expression joy can be described [11] as follows: *the eyebrows are relaxed, the mouth is open and the mouth corners pulled back toward the ears*. We define the scripts based on the description above as:

```
script(facialExpression(Agent,joy,Intensity),
  Action):-
    Action = par([relax(Agent, eyebrows),
      mouthShape(Agent, joy, Intensity)]).
```

The expression joy is defined as a parallel action of relaxing eyebrows with a corresponding mouth shape. We define the scripting action of relaxing eyebrows as an action that resets the eyebrows to their initial default shape as follows:

```
script(relax(Agent, eyebrows), Action):-
  Action = par([resetShape(Agent, l_eyebrow),
    resetShape(Agent, r_eyebrow)]).
```

The script action to set the 'joy' mouthshape is defined as:

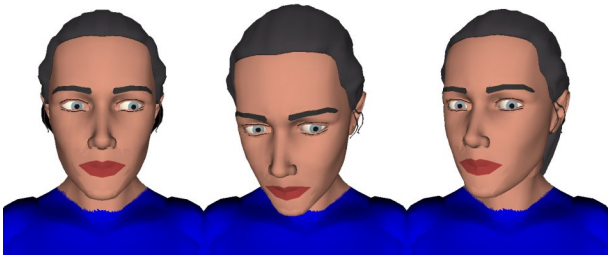

Figure 4. Looking at objects

```
script(mouthShape(Agent, joy, Intensity), Action):-
  Action = move_FAPs(Agent, [
    fap('8.2',change(0,-0.004,0)),
    fap('8.3',change(0.001,0.008,0)),
    fap('8.4',change(-0.001,0.008,0))],
       Intensity, fast).
```

In the following, we show that these scripts can be re-used for other facial expressions. For example, the script of relaxing eyebrows can be re-used in the definition of $disgust$. The expression $disgust$ is described as: *the eyebrows and eyelids are relaxed. The upper lip is raised and curled, often asymmetrically*. Thus, the facial expression $disgust$ can be defined by the following scripts:

```
script(facialExpression(Agent, disgust, Intensity),
       Action):-
  Action = par([relax(Agent, eyebrows),
    relax(Agent, eyelids),
    mouthShape(Agent, disgust, Intensity)]).


script(mouthShape(Agent, disgust, Intensity),
       Action):-
  Action = move_FAPs(Agent,[
            fap('8.1',change(0,0.0027,0)),
            fap('8.2',change(0,-0.001,0)),
            fap('2.4',change(0,-0.002,0)),
            fap('2.8',change(0,-0.004,0)),
            fap('8.3',change(-0.002,-0.003,0))],
               Intensity, fast).
```

Similarly, we can define other facial expressions. The neutral face and its six primary facial expressions as defined by STEP built-in scripts are shown in Figure 3. Based on these primary expressions, a range of expressions can be defined for other purposes. For example, if we need a script action in which the agent smiles for a particular time interval, we can use the definition:

```
script(facialExpression(Agent, Expression,
          Intensity, Time), Action):-
  Action = seq([facialExpression(Agent,
                   Expression, Intensity),
                wait(Time),
                resetShape(Agent, mouth)]).
```

4. Eye movements: An example of the interaction between perception and action

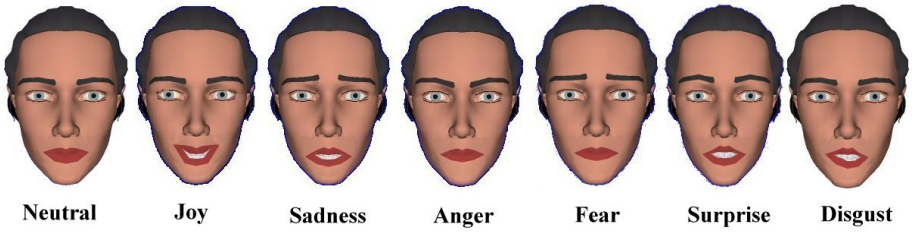

Figure 3. Facial Expressions

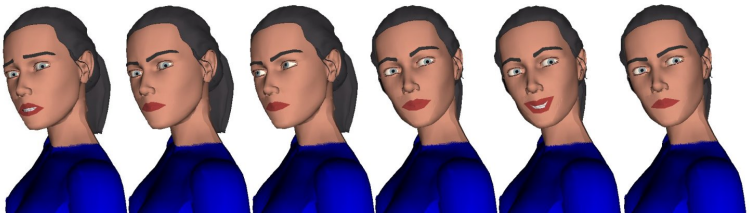

Figure 5. Variants of Gaze and Stare

Eye movements always play an important role for generating facial animations with emotional expressions for embodied agents. Gaze is a typical eye action in which an agent looks steadily at someting, i.e. at other agents or virtual objects. Stare is an eye action in which an agent looks steadily at something with emotional expressions like wonder, anger, and fear. The specification of facial expressions not only requires how facial geometrical data should be manipulated, but also involves several aspects of virtual objects, because positions and orientations of objects may change.

In [6], we show how STEP can be used to specify actions which involve inverse kinematics by calling high-level interaction operators to access the computational capabilities in the meta-language. Similarly, we can use the interaction operators to obtain changing data of virtual worlds for these kind of eye movements. In this example, we will briefly outline how STEP is used to define facial expressions and eye movements when interacting with virtual worlds.

First of all, we define an action 'look_at_object' which specifies eye movements as: the agent can move the eyes to look at an object while keeping the position and orientation of the body and head unchanged. In addition, if necessary, the agent may rotate the head or body. The action 'look_at_object' can be defined in terms of a more primitive eye action 'look_at_position' and call the high-level interaction operator 'do' as follows:

```
script(look_at_object(Agent, Object),
        Action):-
  Action = seq([do(getPosition(Object,X,Y,Z)),
            look_at_position(Agent,
                position(X,Y,Z))]).
```

The action 'look_at_position' is defined as a set of scripts in which the agent may only move the eyeballs if the rotation values of the eyeballs are within a particular range, otherwise the agent proceeds with an alternative action which rotates the head or body.

```
script(look_at_position(Agent, Position),
          Action):-
  rotatingEyeballValue(Agent, Position, Rotation),
  within_eyeball_limit(Rotation), !,
  Action = par([turn(Agent, l_eyeball,
              Rotation, fast),
    turn(Agent, r_eyeball, Rotation, fast)]).

script(look_at_position(Agent, Position), Action):-
  Action = par([reset(Agent, eyeballs),
  rotatingHeadOrBody(Agent,Position)]).
```

We can always call predicates in the meta-language to obtain the results of a computation. The next example illustrates how the rotation values of the eyeballs can be computed at the meta-language level:

```
rotatingEyeballValue(Agent, position(X1,Y1,Z1),
           Rotation):-
  get_eye_center(Agent,position(X,Y,Z)),
  Xdif is X1-X,
  Ydif is Y1-Y,
  Zdif is Z1-Z,
  vector_cross_product(vector(0,0,1),
    vector(Xdif,Ydif,Zdif),
    vector(X,Y,Z),R),
  Rotation = rotation(X,Y,Z,R).
```

Now, we can define the action 'gaze' with the emotional expression 'joy' as:

```
script(gaze_at_object(Agent, Object,
        Intensity, Time), Action):-
  Action = par([look_at_object(Agent, Object),
```

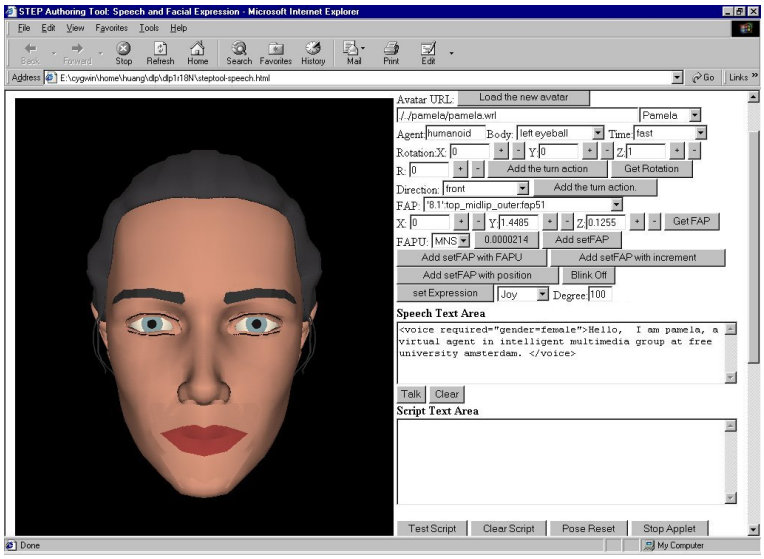

Figure 6. STEP Animation and Speech Authoring Tool

```
facialExpression(Agent, joy, Intensity,
    Time)]).
```

Similarly, we can define the action 'stare' with other emotional expressions. Figure 4 shows eye movements with different head postures for looking at objects with different positions and Figure 5 shows variants of 'gaze' and 'stare' with emotional expressions when rotating the eyes and head.

So far, the presented scripting characteristics of STEP in general and the facial animation capabilities in particular illustrate how several aspects of a single agent can be handled. However, the comprehensive set of EAI sensors and effectors as provided by the DLP / STEP framework not only allows for modeling single agent behavior, but also allow agents to perceive their (changing) world and to react to particular behaviors of participating agents in an orthogonal manner. This holds for both parallel virtual environments and distributed virtual worlds [5].

5. Implementation

We have implemented the facial animation engine and a speech control component for STEP in the distributed logic programming language DLP.

The avatars of VRML/X3D-based embodied agents are displayed in a VRML/X3D browser and are controlled by DLP applets. Applets interact with virtual environments via the VRML/X3D External Authoring Interface. STEP is designed to interoperate with DLP applets and can be called by embodied agents via the STEP interface component. We also have implemented several interactive authoring tools for facial animation and speech in STEP. A screenshot of the authoring tool is shown in Figure 6. Online demonstrations are available at the STEP website: http://step.intelligent-multimedia.net

6. Conclusions

Breton et al. propose a real time facial animation engine for VRML-based avatars in [12]. They developed a muscle based animation on Waters' first muscle model. Our approach uses MPEG-4 FA parameterization for authoring, however, with Waters' muscle model for a more intuitive animation. In [8], Gachery and Magnenat-Thalmann describe different interactive methods for designing facial models based on MPEG-4 facial animation parameters. They propose a web-based MPEG-4 facial animation system which supports the interaction between users and avatars. They use standard facial animation tables (FATs) to define how a model is spatially deformed as a function of the amplitude of FAPs. Our approach is different from theirs with respect to the structure of FATs with muscle models. Moreover, our scripting approach makes authoring easier and more convenient.

Pelachaud and Bilvi propose a model of gaze behaviour for embodied conversational agents in [13]. The examples discussed in this paper focus on a scripting approach for the specification of gaze behaviour and other eye movements with emotional expressions.

In this paper, we have discussed how a scripting approach can be extended with facial expressions in STEP. Facial animations in STEP can be generated on-the-fly and have a satisfactory performance for web-based embodied agents which need to interact in real-time with users and

other agents. Moreover, the scripting language STEP can serve as a high-level specification language for knowledge representation and reasoning, as well as a lower-level programming language for interaction and animation of embodied agents. As shown in the gaze example in this paper, these features significantly improve the interaction between perception and actions for embodied agents.

References

[1] Anton Eliëns. *DLP, A Language for Distributed Logic Programming*. Wiley, 1992.

[2] Anton Eliëns. *Principles of Object-Oriented Software Development*. Addison-Wesley, 2000.

[3] Zhisheng Huang, Anton Eliëns, and Cees Visser. Programmability of intelligent agent avatars. In *Proceedings of Agents'01 Workshop on Embodied Agents*, 2001.

[4] Anton Eliëns, Zhisheng Huang, and Cees Visser. A platform for embodied conversational agents based on distributed logic programming. In *Proceedings of AAMAS 2002 WORKSHOP: Embodied conversational agents - let's specify and evaluate them*, 2002.

[5] Zhisheng Huang, Anton Eliëns, and Cees Visser. 3D agent-based virtual communities. In *Proceedings of the 2002 Web 3D Conference*, pages 137–143. ACM Press, 2002.

[6] Zhisheng Huang, Anton Eliëns, and Cees Visser. STEP: a scripting language for embodied agents. In *Life-like Characters, Tools, Affective Functions and Applications*. Springer-Verlag, 2003.

[7] Zhisheng Huang, Anton Eliëns, and Cees Visser. Implementation of a scripting language for VRML/X3D-based embodied agents. In *Proceedings of the 2003 Web 3D Conference*, pages 91–100. ACM Press, 2003.

[8] S. Gachery and N. Magnenat-Thalmann. Designing MPEG-4 facial animation tables for web applications. MIRALab, University of Geneva, 2001.

[9] K. Waters. A muscle model for animating three-dimensional facial expression. *Computer Graphics (SIGGRAPH'87)*, 21(4):17–24, 1987.

[10] Zhisheng Huang, Anton Eliëns, and Cees Visser. Face models for h-anim based avatars. In *Research report (in preparation)*. Vrije University Amsterdam, 2004.

[11] J. Ostermann. Face animation in mpeg-4. In *MPEG-4 Facial Animation: The Standard, Implementation and Applications*, 2002.

[12] G. Breton, C. Bouville, and D. Pele. Faceengine: a 3D facial animation engine for real time applications. In *Proceedings of the 2001 Web3D Conference*. ACM Press, 2001.

[13] C. Pelachaud and M. Bilvi. Modelling gaze behaviour for conversational agents. In *Proceedings of the 4th International Working Conference on Intelligent Virtual Agents(LNAI 2792)*, 2003.