Scripting Facial Expressions for X3D/VRML-based Embodied Agents

Zhisheng Huang, Anton Eliëns, and Cees Visser

Department of Computer Science
Vrije University Amsterdam,
De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

Abstract

This paper proposes a scripting approach to facial animations for embodied agents, in particular for X3D/VRML-based web agents. The use of a scripting technology to control facial expressions has the advantage that it results in highly flexible components; not only can scripts be re-used to define a wide range of facial animation scenarios, but they can also be applied with appropriate parameterization for different application contexts and different agents, despite the fact that agents may have different 3D facial geometrical data. The proposed approach has been implemented as an extension of STEP, a scripting language for embodied agents, by using the distributed logic programming language DLP. In the DLP language framework, a facial model as well as the animation engine are web-based and animation results can be viewed locally in a browser. In addition, facial animations in STEP can be generated on-the-fly, which is a requirement for web-based embodied agents that need to interact with users and other agents in real time.

CR Categories: H.5.1 [Multimedia Information Systems]: Animations; H.5.3 [Group and Organization Interfaces]: Web-based interaction; I.2.11 [Distributed Artificial Intelligence]: Intelligent agents

Keywords: agents, distributed logic programming, H-Anim, VRML/X3D, STEP, facial animation

1 Introduction

In recent years there has been a growing interest in embodied conversational agents (ECAs), either in the form of animated humanoid avatars or, more simply, as a 'talking head'. Their graphical representations are often based on sequences of 2D cartoon drawings or 3D facial and body models[Gratch et al. 2002; Heylen et al. 2001; Poggi et al. 2000]. In particular, web-based embodied conversational agents become appealing, because they’re increasingly more lifelike and realistic [Prendinger and Ishizuka 2003].

X3D/VRML has become one of the standard technologies for 3D web applications. X3D/VRML-based avatars, in particular H-Anim1 humanoid avatars are well suited as avatars for web-based embodied agents. However, facial animation and speech support are a must for such embodied agents in order to give them a more realistic appearance. Furthermore, web-based embodied conversational agents must have a sufficiently comprehensive repertoire of facial expressions for some degree of believability. The convenience of facial animation authoring thus becomes an important issue.

Applying a modular scripting technology to facial animations makes authoring easier in several aspects. First of all, it provides users with a high-level animation system interface without the need for extensive programming activities. Moreover, using a modular scripting framework for facial expressions results in components that can be easily combined and re-used. There are at least three levels of reusability:

- **Reusable by composition**: scripts can be re-used to define more complicated actions by combining them with other actions without changing the original scripts;

- **Reusable by parameterization**: scripts can be re-used to define variants of particular facial expressions by changing the parameterization of scripts;

- **Reusable by interaction**: although embodied conversational agents may have different 3D facial geometrical data, scripts can be applied in different application contexts with different agents, by obtaining the necessary data at runtime.

In this paper, we propose a scripting approach to facial animation for embodied agents with all of the three levels of reusability. Scripts can not only be re-used to define variants of facial expressions, but they can also be applied with appropriate parameterization for different application contexts. We have realized the proposed approach as an extension of STEP, a scripting language for embodied agents[Huang et al. 2003b; Huang et al. 2003c], by using the distributed logic programming language DLP. The DLP language has been used for the implementation and development of different types of intelligent agents2 [Huang et al. 2001; Huang et al. 2002] because it is a high level declarative language and an object-oriented development tool. The integration of facial animations in STEP makes DLP a powerful tool for the implementation of intelligent embodied agents, because the combination of the two languages offers a unique system which can be used for the high-level modeling of intelligent behaviours as well as the specification of animations.

In this paper, we will show that DLP and its scripting language STEP are convenient for authoring facial animations. In this framework, the facial animation engine can process the facial objects efficiently with a satisfactory realism, although occasionally by some tradeoffs between realism and performance. In this paper, we also briefly discuss implementation issues of facial animation in STEP. The structure of this paper is as follows. In order to make this paper self-contained, Section 2 is a brief introduction to the distributed logic programming language DLP and the scripting language STEP. Section 3 discusses the general requirements of facial animation for embodied agents and how STEP can meet these requirements in the DLP framework. Section 4 describes an MPEG-4 FAP-based facial model for H-Anim avatars and their scripting actions in STEP. Section 5 shows several examples how STEP can be used to define a number of variants of facial expressions. Section 7 discusses some related work. Finally, Section 8 gives some conclusions and indications for future work.

*email: huang@cs.vu.nl

1email: eliens@cs.vu.nl

‡email: cvt@cs.vu.nl

1http://www.h-anim.org

2http://wasp.cs.vu.nl
2 DLP and STEP

2.1 DLP

The distributed logic programming language DLP [Eliëns 1992; Eliëns 2000] combines logic programming, object-oriented programming and parallelism. DLP has been used as a tool for web agents, in particular for 3D web agents [Huang et al. 2001; Eliëns et al. 2002; Huang et al. 2002]. 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.

In DLP, an object is designed as a set of rules and facts, which consist of a list of formulas built from predicates and terms (variables or constants). DLP is an extensible language: special-purpose requirements for particular application domains can easily be integrated in the existing object-oriented language framework. DLP has been extended with a run-time library for VRML EAI and a TCP/IP library for distributed processing. Typical EAI predicates for the manipulation of virtual worlds in DLP are the get/set predicates, for example:

\[
\text{getPosition}(\text{Object} .X, Y, Z),
\]

gets the position \((X, Y, Z)\) of the Object in the virtual world, and

\[
\text{setPosition}(\text{Object} .X, Y, Z),
\]

sets the position [Huang et al. 2001].

2.2 STEP

STEP (Scripting Technology for Embodied Persona) is a scripting language for embodied agents, in particular for their non-verbal acts like gestures and postures [Huang et al. 2003b]. The design of STEP was motivated by the following principles: convenience, compositional semantics, re-definability, parameterization, and interaction. See [Huang et al. 2003a] and [Huang et al. 2003b] 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. Moving the body part means the setting of the corresponding joint to a new position. Turn and move actions are expressed as follows:

\[
\text{turn(Agent, BodyPart, Direction, Duration)}
\]

\[
\text{move(Agent, BodyPart, Position, Duration)}
\]

where BodyPart refers to a body joint in the H-Anim specification, like l_shoulder, r_elbow, etc., Direction denotes a rotation, which can be a term like rotation(1,0,0,-1.57), or ‘front’, a natural-language like term. Position defines a position, which can be an item like position(1,0,0), or ‘front’, a natural-language like term. Duration specifies the time interval for an action, which can be an explicit time specification, like time(2,second), or a natural-language like term, like ‘fast’, or ‘slow’. For instance, an action such as \text{turn(humanoid.l_shoulder, front, fast)} indicates turning the humanoid’s left arm to the ‘front’ direction fast. The meaning of natural language-like terms such as ‘front’ and ‘fast’ are defined by the ontology component of STEP.

Scripting actions can be composed by using the following composition operators: the sequence operator ‘seq’, the parallel operator ‘par’, and the repeat operator \text{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 \text{do(\phi)}, which executes a goal \phi at the meta language level, and the conditional \text{if_else(\phi,Action1,Action2)}, which executes Action1 if state \phi 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 scripting language STEP and its meta-language DLP can call each other by their interaction operator. In STEP, the do-operator \text{do(\phi)} is used to call a goal \phi in DLP, whereas in DLP the formula \text{do_script(\text{Script})} is used to call a script in the STEP animation engine.

3 Facial Animation Requirements for Embodied Agents

In order to obtain some degree of realism, face models of avatars usually consist of a few thousand vertices and polygons. Many research efforts deal with facial animation engines for web-based avatars. The performance of facial animation engines and network communication overhead are often a major design issue. Java-based approaches make facial animation engines and face models web-enabled: after the engine and avatars have been loaded from a web site, facial expressions and other animations can be generated locally. Although this solves the bandwidth problem to a certain extent, java-based approaches may introduce their own performance problems when processing a large number of vertices or polygons that describe the facial models. Therefore, many animation solutions require their own special-purpose facial animation player, either as a DLL or a plug-in.

To summarize the most important requirements for facial models and facial animations with respect to web-based embodied agents:

- **Realism**: Facial models should have an appropriate number of vertices and polygons for the lifelike appearance of avatars.
- **Web-infrastructure**: Distributed virtual worlds in general, and interacting animated avatars in such worlds in particular should be able to overcome the Internet bandwidth problem.
- **Performance**: Facial animation engines should be capable to generate facial animations on-the-fly for real-time interaction between users and agents.
• **Convenience for authoring:** The system should provide a convenient and efficient way for authoring a range of facial expressions. In particular, authors need no knowledge of facial muscle models for authoring.

• **Naturalness:** Facial animations should be realistic. For instance, muscle models in the facial animation engine improve the naturalness of facial animations.

• **Reusability:** Facial expressions can be re-used for different application contexts and different avatars as discussed above.

Using the DLP language framework, STEP can deal with avatars that consist of a few thousand vertices/polygons in their facial models for some degree of lifelike appearance [Huang et al. 2003d]. The facial model as well as the animation engine are web-based in the DLP language framework since all DLP programs are compiled to Java classes. Content manipulation is done at the client side, therefore bandwidth issues are restricted to loading a particular animation or interactive authoring tool.

4 Facial Object Hierarchy in STEP

4.1 Face Models

MPEG-4 FAPs (facial animation parameters) offer a convenient way to reference facial animation points. We have developed VRML/X3D based face models with MPEG-4 FAPs 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 eyelid 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

\[
\text{turn(Agent.L_eyeball, rotation(0,1,0,0.37), fast)}
\]

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.

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, 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 6. Several facial expressions based on such a mouth model are shown in Figure 3.

Figure 1: Face Object Hierarchy in H-Anim

Figure 2: Mouth Object with MPEG-4 FAPs
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 'fap 8.1' 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, FAPs (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 [Gachery and Magenmuth-Thalmann 2001]. Waters’ muscle model [Waters 1987] 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 FAPs and FATs and incorporated Waters’ muscle model for improved realism and a good performance [Huang et al. 2003d].

4.2 Primitive Face Actions in STEP

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

- \text{getFAP}(\text{Agent}, \text{FAP}, \text{position}(X, Y, Z))$: retrieves the position \((X, Y, Z)\) for the specified FAP of the agent.
- \text{setFAP}(\text{Agent}, \text{FAP}, \text{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.

- \text{position}(X,Y,Z): \text{position}(X,Y,Z) = <X, Y, Z>.
- \text{increment}(X,Y,Z): \text{increment}(X,Y,Z) = <X_1 + X, Y_1 + Y, Z_1 + Z>.
- \text{change}(X,Y,Z): \text{change}(X,Y,Z) = <X_0 + X, Y_0 + Y, Z_0 + Z>.
- \text{change}(X,Y,Z, \text{FAPU}): \text{change}(X,Y,Z, \text{FAPU}) = <X_0 + X \times \text{unit} \text{(FAPU)}, Y_0 + Y \times \text{unit} \text{(FAPU)}, Z_0 + Z \times \text{unit} \text{(FAPU)}>.

Assume that the initial value of a FAP is \((X_0, Y_0, Z_0)\), the current value is \((X_1, Y_1, Z_1)\), and the unit of a FAPU is \text{unit} \text{(FAPU)}. A setFAP action with the format \text{position}(X,Y,Z) sets the FAP to the absolute value \((X, Y, Z)\). Of course, a facial operation with this format is only appropriate for avatars with identical 3D geometrical data.

A setFAP with the format \text{change}(X,Y,Z) sets the FAP by increasing the initial value of vertices with the value \((X, Y, Z)\). 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 \text{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 \text{change} format with an additional facial animation parameter unit (FAPU).

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

4.3 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 \text{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:

- \text{moveFAP}(\text{Agent}, \text{FAP}, \text{Position}): gets the initial FAP value of the agent. We use this action to reset an FAP to the default value.
- \text{resetShape}(\text{Agent}, \text{Object}): resets the initial \text{Object} values of the agent. We use this action to reset a facial object to its default shape.
- \text{setFAPs}(\text{Agent}, \text{FAPChange}): set a number of FAPs simultaneously. FAPChange is a list of \text{change} specifications: \([\text{fap}(\text{FAP}_1, \text{Change}_1), ..., \text{fap}(\text{FAP}_n, \text{Change}_n)]\). The script action \text{setFAPs}(\text{Agent}, \text{FAPChange}) is semantically equivalent to the action \text{par}(\text{setFAP}(\text{Agent}, \text{FAP}, \text{Change}_1), ..., \text{setFAP}(\text{Agent}, \text{FAP}, \text{Change}_n))\), however, the former uses only one execution thread to achieve the parallel action, while the latter uses multiple threads.
- \text{moveFAPs}(\text{Agent}, \text{FAPChange}, \text{Speed}): set a number of FAPs simultaneously with the specified \text{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.
- \text{moveFAPs}(\text{Agent}, \text{FAPChange}, \text{Intensity}, \text{Speed}): The action \text{moveFAPs}(\text{Agent}, \text{fap}(\text{FAP}_1, \text{Change}_1 \times \text{Intensity}), ..., \text{fap}(\text{FAP}_n, \text{Change}_n \times \text{Intensity}), \text{Speed}).

5 Examples

5.1 Emotion Expressions

MPEG-4 FAP 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

\text{facialExpression}(\text{Agent}, \text{Expression}, \text{Intensity})

to represent the facial expression for \text{Agent} with a particular \text{Intensity}.

The expression \text{joy} can be described [Osterrann 2002] as follows: \text{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:

\text{script(facialExpression(\text{Agent}, \text{joy}, \text{Intensity}), \text{Action})}:=
\text{Action} = \text{par}([\text{relax} \text{(Agent, eyebrows)}, \text{mouthShape} \text{(Agent, joy, \text{Intensity})}]).

The expression \text{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:
Figure 3: Facial Expressions

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:

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('8.4',change(0,0.002,0)),
        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)])).

5.2 Synchronization with Speech

The synchronization of facial animation and speech, in particular lip synchronization, is an important aspect of realistic facial animations. Implementation issues of speech in STEP are discussed in [Huang et al. 2003e]. In this paper, we describe how lip synchronization is realized in STEP built-in scripts.

First we define viseme scripts in terms of the action setFAPs. For example, in MPEG-4, the viseme 1 corresponds to the phonemes p, b, m. Thus, we define the scripts as follows:

script(viseme(Agent, 1), Action):-
    Action = setFAPs(Agent, [
        fap('8.1',change(0,-0.001,0.001)),
        fap('8.2',change(0,0.0017,0.001))]).

......

script(phoneme(Agent, p), Action):-
    Action = viseme(Agent, 1).

script(phoneme(Agent, b), Action):-
    Action = viseme(Agent, 1).

script(phoneme(Agent, m), Action):-
    Action = viseme(Agent, 1).

......

We also define a script action lipmove in which the agent continuously has different mouth movements based on a list of phonemes. Moreover, the script action lipmove should only perform actions when the speech engine is active. The current state of the speech engine is given by the field stepspeech and has three possible values: start, stop, and wait. The speech engine state can be obtained by calling the predicate get_field.

The scripting action lipmove processes the list of phonemes recursively and checks the speech engine state as follows:

script(lipmove(_Agent,[]), Action):-
    Action = [].

script(lipmove(Agent,[H|T]), Action):-
    Action = seq([do(get_field(stepspeech, status, Status)),
                  if_then_else(Status=start, seq([phoneme(Agent,H),
                                                  lipmove(Agent,T)]),
                  if_then_else(Status=wait(S),
                                  seq([wait(S),lipmove(Agent,[H|T])]),skip))]).

The script lipmove gets the current speech engine state first and checks its value. If the value is 'start', the corresponding phoneme action is performed recursively. If the speech engine state is 'wait', the script waits for the specified time, after which it continues the action lipmove. If the speech engine state is 'stop', the

Note that [H|T] is the standard Prolog list notation, indicating the list composed of the head H and the tail list T.
current action is skipped. Finally, if the phoneme list is empty, the action lipmove terminates.

5.3 Eye movements: Interaction with virtual worlds

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 something, 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 [Huang et al. 2003b], 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:

\[
\text{script(look_at_object(Agent, Object), Action):=}
\text{Action = seq([do(getPosition(Object,X,Y,Z)),}
\text{ look_at_position(Agent,position(X,Y,Z))])).}
\]

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.

\[
\text{script(look_at_position(Agent, Position), Action):=}
\text{rotatingEyeballValue(Agent, position(X1,Y1,Z1), Rotation):=}
\text{get_eye_center(Agent,position(X,Y,Z)),}
\text{ Xdif is X1-X,}
\text{ Ydif is Y1-Y,}
\text{ Zdif is Z1-Z,}
\text{ vector_cross_product(vector(0,0,1),}
\text{ vector(Xdif,Ydif,Zdif), vector(X,Y,Z),R),}
\text{ Rotation = rotation(X,Y,Z,R)).}
\]

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

\[
\text{script(gaze_at_object(Agent, Object, Intensity, Time),}
\text{ Action):=}
\text{Action = par([look_at_object(Agent, Object),}
\text{ facialExpression(Agent, joy, Intensity, Time)]).}
\]

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.

6 Implementation

We have implemented the facial animation engine and the speech control component for STEP in the distributed logic programming language DLP. The architecture of STEP consists of the following components:

- **Action library**: The action library is a collection of scripting actions, which can be user defined or system built-in.
- **STEP ontology**: The STEP ontology component defines the semantics of the STEP reference system.
- **STEP kernel**: The STEP kernel is the central controlling component of STEP. It translates scripting actions into executable VRML/X3D EAI commands based on the semantics of the action operators and the ontological definitions of the reference terms. The STEP kernel and the STEP ontology component are application independent.
- **STEP application interface**: The STEP application interface component offers the interface operators for both users and applications. The scripting actions can be called by using these application interface operators. They can be java-applet-based, java-script-based, or XML-based.
- **STEP facial animation engine**: The STEP facial animation engine translates the primary facial animation actions into executable VRML/X3D EAI commands.
- **STEP speech control**: The STEP speech control component interprets the STEP speech actions.

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. The STEP architecture and its interface for embodied agents is shown in Figure 6.
Figure 5: Variants of Gaze and Stare

Figure 6: STEP Architecture and Embodied Agents

Figure 7: STEP Animation and Speech Authoring Tool
7 Related Work


Breton et al. propose a real time facial animation engine for VRML-based avatars in [Breton et al. 2001]. 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 [Gachery and Magnenat-Thalmann 2001], 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 [Pelachaud and Bilvi 2003]. The examples discussed in this paper focus the scripting approach for the specification of gaze behaviour and other eye movements with emotional expressions.

8 Conclusions

In this paper, we have proposed a scripting approach to facial expressions for embodied agents, in particular, for X3D/VRML-based web agents. Using a modular scripting technology for describing facial expressions make animation scripts highly reusable. Scripts can be re-used to define a wide range of facial expressions and can also be applied with appropriate parameterization for different application contexts and different agents. The proposed approach has been implemented in STEP, a scripting language for embodied agents, using the distributed logic programming language DLP. Facial animation in STEP can be generated on-the-fly, which has a satisfactory performance for web-based embodied agents which need to interact in real-time with users and other agents.

No doubt that there is still a lot of further work left. For instance, more efficient script for lip synchronization would significantly improve the facial expressions. More comprehensive scripts for gaze and other eye movements would improve the intelligence of embodied conversational agents. The work proposed in this paper offers in this respect a foundation for the scripting approach of facial animation and expressions.

References


