STEP: A Scripting Language for Embodied Agents

Zhisheng Huang, Anton Eliëns and Cees Visser

Intelligent Multimedia Group
Division of Computer Science, Faculty of Sciences
Vrije Universiteit Amsterdam, The Netherlands
{huang,eliens,ctv}@cs.vu.nl

Summary. In this chapter we propose a scripting language, called STEP, for embodied agents, in particular for their communicative acts like gestures and postures. Based on the formal semantics of dynamic logic, STEP has a solid semantic foundation, in spite of a rich number of variants of the compositional operators and interaction facilities on worlds. STEP has been implemented in the distributed logic programming language DLP, a tool for the implementation of 3D web agents. In this chapter, we discuss principles of scripting language design for embodied agents and several aspects of the application of STEP.

1 Introduction

Embodied agents are autonomous agents which have bodies by which the agents can perceive their world directly through sensors and act on the world directly through effectors. Embodied agents whose experienced worlds are located in real environments, are usually called *cognitive robots*. *Web agents* are embodied agents whose experienced worlds are the Web; typically they act and collaborate in networked virtual environments. In addition, *3D web agents* are embodied agents whose 3D avatars can interact with each other or with users via Web browsers [11].

Embodied agents usually interact with users or each other via multimodal communicative acts, which can be verbal or non-verbal. Gestures, postures and facial expressions are typical non-verbal communicative acts which contribute to the representation of avatars as life-like characters. In general, specifying communicative acts for embodied agents is not easy; they often require a lot of geometric data and detailed movement equations for the specification of gestures.

In this chapter we propose the scripting language STEP (Scripting Technology for Embodied Persona), in particular for communicative acts of embodied agents. At present, we focus on aspects of the specification and modeling

of gestures and postures for 3D web agents. However, STEP can be extended for other communicative acts like facial expressions or speech, and other types of embodied agents, like cognitive robots. Scripting languages are to a certain extent simplified languages which ease the task of programming and development. One of the main advantages of using scripting languages is that the specification of communicative acts can be separated from the programs which specify the agent architecture and mental state reasoning. Thus, changing the specification of communicative acts does not require to re-program an agent.

The avatars of our 3D web agents are built in the Virtual Reality Modeling Language (VRML) or X3D, the next generation of VRML. These avatars have a humanoid appearance. The humanoid animation working group[1] proposes a specification, called H-anim specification, for the creation of libraries of reusable humanoids in Web-based applications as well as authoring tools that make it easy to create humanoids and animate them in various ways. H-anim specifies a standard way of representing humanoids in VRML. We have implemented the proposed scripting language for H-anim based humanoids in the distributed logic programming language DLP[5].[2]

DLP is a tool for the implementation of 3D intelligent agents[12].[3] In this chapter, we discuss how STEP can be used for embodied agents. STEP introduces a Prolog-like syntax, which makes it compatible with most standard logic programming languages, whereas the formal semantics of STEP is based on dynamic logic[9]. Thus, STEP has a solid semantic foundation, in spite of a rich number of variants of the compositional operators and interaction facilities on worlds.

2 Principles

We designed the scripting language primarily for the specification of communicative acts for embodied agents; we have separated the external-oriented communicative acts from internal changes of the mental states of embodied agents because the former involves only geometric changes of the body objects and the natural transition of the actions, whereas the latter involves more complicated computation and reasoning. Of course, a question is: why not use the same scripting language for both external gestures and internal agent specification? Our answer is: the scripting language is designed to be a simplified, user-friendly specification language for embodied agents, whereas the formalization of intelligent agents requires a powerful specification and programming language. It is not our intention to design a scripting language with fully-functional computation facilities, as found in programming languages like Java, Prolog or DLP. A scripting language should be interoperable

[1] http://h-anim.org
[2] http://www.cs.vu.nl/∼eliens/projects/logic/index.html
[3] http://wasp.cs.vu.nl/wasp

with a fully powered agent implementation language, but offer a rather easy way for authoring. Although communicative acts are the result of the internal reasoning of embodied agents, they do not need the expressiveness of a general programming language. However, we do require that a scripting language should be able to interact with the mental states of embodied agents in some ways, which will be discussed in more detail later.

We consider the following design principles for a scripting language.

Principle 1: Convenience

As mentioned, the specification of communicative acts, like gestures and facial expressions usually involves a lot of geometric data, like ROUTE statements in VRML or movement equations in computer graphics. A scripting language should hide these geometric difficulties, so that even the authors who have limited knowledge of computer graphics can use it in a natural way. For example, suppose that authors want to specify that an agent turns his left arm forward slowly. This can be specified as:

```
turn(Agent, left_arm, front, slow)
```

It should not be necessary to specify it as follows, which requires knowledge of a coordinate system, rotation axis, etc.

```
turn(Agent, left_arm, rotation(1,0,0,1.57), 3)
```

One of the implications of this principle is that embodied agents should be aware of their context; they should be able to understand what certain indications mean, like the directions 'left' and 'right', or the body parts 'left arm', etc.

Principle 2: Compositional Semantics

Specification of composite actions based on existing components, for example an action of an agent which turns his arms forward slowly, can be defined in terms of two primitive actions, turn-left-arm and turn-right-arm:

```
par([turn(Agent, left_arm, front, slow),
    turn(Agent, right_arm, front, slow)])
```

Typical composite operators for actions are the sequence action *seq*, parallel action *par*, and repeat action *repeat*, which are used in dynamic logic [9].

Principle 3: Re-definability

Scripting actions (e.g. composite actions) can be defined in terms of other actions explicitly. The scripting language incorporates a rule-based specification system, where scripting actions can be defined by their own set of rules.

These defined actions can be re-used for other scripting purposes. For example, if we have defined two scripting actions *run* and *kick*, then a new action *run_then_kick* can be defined in terms of *run* and *kick*:

```
run_then_kick(Agent)=
   seq([run(Agent), kick(Agent)]).
```

which can be specified in a Prolog-like syntax:

```
script(run_then_kick(Agent), Action):-
   Action = seq([run(Agent),kick(Agent)]).
```

Principle 4: Parametrization

Scripting actions can be adapted to be other actions; actions can be specified in terms of how they cause changes over time to each individual *degree of freedom*, as proposed by Perlin and Goldberg in [16]. For example, suppose that we define a scripting action *run*: we know that running can be done at different paces. It can be done 'fast' or 'slow'. It should not be necessary to define run actions for particular paces. We can define the action 'run' with respect to a degree of freedom 'tempo'. Changing the tempo for a generic run action should be enough to achieve a run action at different paces. Another method of parametrization is to introduce variables or parameters in the names of scripting actions, which allows for a similar action with different values. In particular, agent names and their relevant parameters are specified as variables in script libraries, by which the same scripting actions can be re-used for different embodied agents under different situations by different authors. It would significantly improve the reusability of scripting actions for the purpose of productivity. This is one of the reasons why we introduce a Prolog-like syntax in STEP.

Principle 5: Interaction

Scripting actions should be able to interact with the world, including objects and other agents. More exactly, scripting actions can perceive the world, even embodied agents' states, in order to decide whether or not the current action should be continued, or replaced by other actions. This kind of interaction can be achieved by the introduction of high-level interaction operators as defined in dynamic logic. The operator 'test' and the operator 'conditional' are an example of operators that facilitate the interaction between actions and states.

These five principles are a guideline for the design of the scripting language STEP. The principle of convenience implies that STEP uses some natural-language-like terms for references. The principle of compositional semantics states that STEP has a set of built-in action operators. The principle of re-definability suggests that STEP should incorporate a rule-based specification

system. The principle of parametrization justifies that STEP introduces a Prolog-like syntax. The principle of interaction requires that STEP is based on a more powerful meta-language.

3 The Scripting Language STEP

In this section, we discuss the general aspects of the scripting language STEP. We propose the reference systems for STEP first.

3.1 Reference Systems

The reference system of STEP consists of three components: Direction Reference, Body Reference, and Time Reference.

Direction Reference

The direction reference system in STEP is based on the H-anim specification: the initial humanoid position should be modeled in a standing position, facing in the +Z direction with +Y up and +X to the humanoid's left. The origin $\langle 0, 0, 0\rangle$ is located at ground level, between the humanoid's feet. The arms should be straight and parallel to the sides of the body with the palms of the hands facing inwards towards the thighs.

Based on the standard pose of the humanoid, we can define the direction reference system as sketched in figure 1. The direction reference system is based on these three dimensions: front vs. back which corresponds to the Z-axis, up vs. down which corresponds to the Y-axis, and left vs. right which corresponds to the X-axis. Based on these three dimensions, we can introduce a more natural-language-like direction reference scheme, for example, turning left-arm to 'front-up', is to turn the left-arm such that the front-end of the arm will point to the up front direction. Figure 2 shows several combinations of directions based on these three dimensions for the left-arm. The direction references for other body parts are similar. These combinations are designed for convenience and are discussed in Section 2. However, they are in general not sufficient for more complex applications. To solve this kind of problem, we introduce interpolations with respect to the mentioned direction references. For instance, the direction 'left_front2' is referred to as one which is located between 'left_front' and 'left', which is shown in Figure 2. Natural-language-like references are convenient for authors to specify scripting actions, since they do not require the author to have a detailed knowledge of reference systems in VRML. Moreover, the proposed scripting language also supports the original VRML reference system, which is useful for experienced authors. Directions can also be specified to be a four-place tuple $\langle X, Y, Z, R\rangle$, for example $rotation(1, 0, 0, 1.57)$.

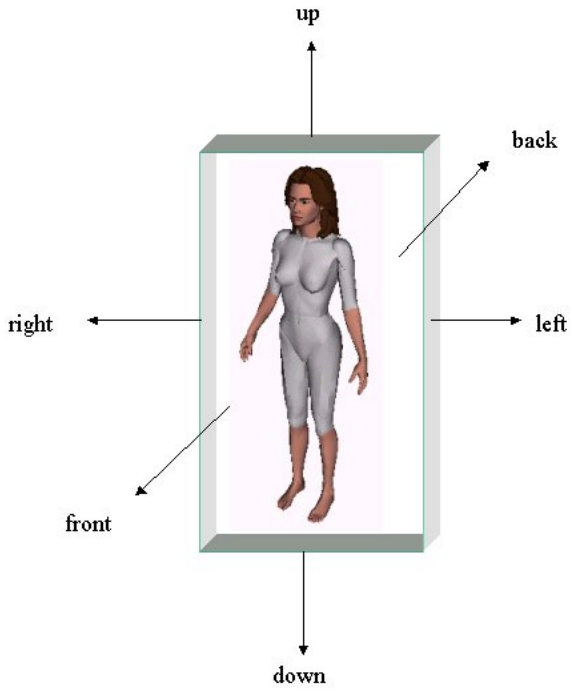

Fig. 1. Direction Reference for Humanoid

Body Reference

According to the H-anim standard, an H-anim specification contains a set of *Joint nodes* that are arranged to form a hierarchy. Each Joint node can contain other Joint nodes and may also contain a *Segment node* which describes the body part associated with that joint. Each Segment can also have a number of *Site nodes*, which define locations relative to the segment. Sites can be used for attaching accessories, like hat, clothing and jewelry. In addition, they can be used to define eye points and viewpoint locations. Each Segment node can have a number of *Displacer nodes*, that specify which vertices within the segment correspond to a particular feature or configuration of vertices.

Figure 3 shows several joints of humanoids. Turning body parts of humanoids implies the setting of the corresponding joint's rotation. Moving the body parts means the setting of the corresponding joint's position. For instance, the action 'turning the left-arm to the front slowly' is specified as:

```
turn(Agent, l_shoulder, front, slow)
```

Based on the H-anim specification, all body joints are contained in a hierarchical structure. Accordingly, the direction reference of a body joint in STEP is measured relative to the default rotations of its ancestor joints in the

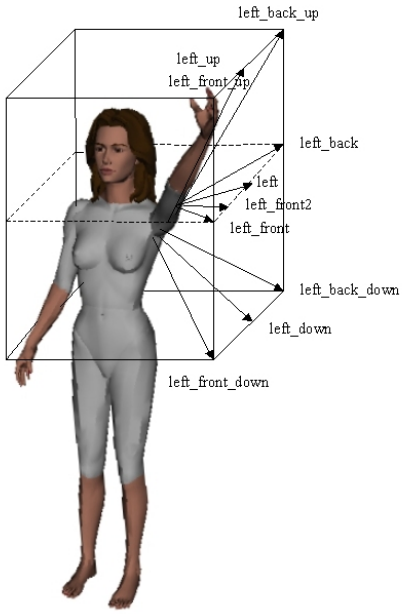

Fig. 2. Combination of the Directions for Left Arm

hierarchy. For instance, Figure 4(a) shows the posture of the left elbow joint to the direction 'front' relative to the default posture of the avatar. However, when the left shoulder joint or one of its parents joints, point to the direction 'front', the left elbow joint pointing to 'front' results in a posture in which the left hand points to the direction 'up', as shown in Figure 4(b). In practice, this kind of direction reference does not cause difficulties for authoring, for the correct direction can be obtained by reducing the directions of its ancestor body parts to be the default ones. Therefore, STEP is well suited for a forward kinematics system. Moreover, we would like to point out that STEP can also be used to solve inverse kinematics problems. That will be shown in Section 4.

Time Reference

STEP has the same time reference system as VRML. For example, the action *turning the left arm to the front in 2 seconds* can be specified as:

```
turn(Agent, l_shoulder, front, time(2, second))
```

This kind of explicit specification of duration in scripting actions does not satisfy the parametrization principle. Therefore, we introduce a more flexible time reference system based on the notions of beat and tempo. A *beat* is a time interval for body movements, whereas the *tempo* is the number of beats

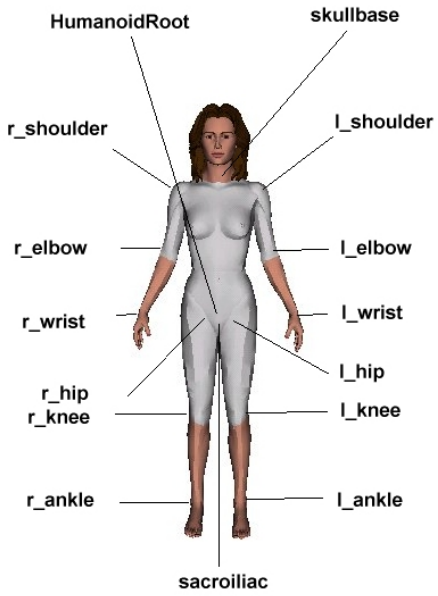

Fig. 3. Typical Joints for Humanoid

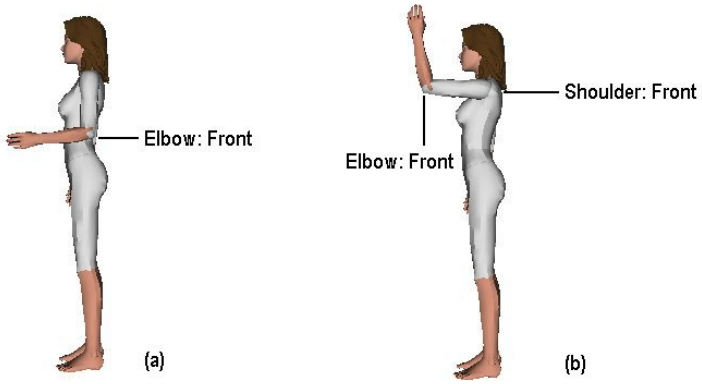

Fig. 4. Elbow joint in different situations

per minute. By default, the tempo is set to 60, i.e. a beat corresponds to a second. However, the tempo can be changed. Moreover, we can define different speeds for body movements, for example, the speed 'fast' can be defined as one beat, whereas the speed 'slow' can be defined as three beats.

3.2 Primitive Actions and Composite Operators

Turn and move are the two main primitive actions for body movements. Turn actions specify the change of the rotations of the body parts or the whole body over time, whereas move actions specify the change of the positions of the body parts or the whole body over time. A turn action of a body part is defined as follows:

```
turn(Agent,BodyPart,Direction,Duration)
```

where `Direction` can be a natural-language-like direction like 'front' or a rotation value like 'rotation(1,0,0,3.14)', and `Duration` a speed name like 'fast' or an explicit time specification, like 'time(2,second)'.

A move action of a body part is defined as:

```
move(Agent,BodyPart,Direction,Duration)
```

where `Direction` can be a natural-language-like direction, like 'front', a position value like 'position(1,0,10)', or an increment value like 'increment(1,0,0)'. The turn and move actions of the whole body are defined as follows:

```
turn_body(Agent,Direction,Duration)
move_body(Agent,Direction,Duration)
```

Typical composite operators for scripting actions are:

- Sequence operator 'seq': the action `seq([`$Action_1$`, ...,`$Action_n$`])` denotes a composite action in which $Action_1$, ..., and $Action_n$ are executed sequentially:

  ```
  seq([turn(agent,l_shoulder,front,fast),
  turn(agent,r_shoulder,front,fast)])
  ```

- Parallel operator 'par': the action `par([`$Action_1$`, ...,`$Action_n$`])` denotes a composite action in which $Action_1$, ..., and $Action_n$ are executed simultaneously.
- Non-deterministic choice operator 'choice': the action `choice([`$Action_1$`, ...,` $Action_n$`])` denotes a composite action in which one of the $Action_1$, ..., and $Action_n$ is executed.
- Repeat operator 'repeat': the action `repeat(Action, T)` denotes a composite action in which the *Action* is repeated T times.

3.3 STEP and Dynamic Logic

STEP is based on dynamic logic[9] and allows for arbitrary abstractions using the primitives and composition operators provided by the logic. In dynamic logic, there is a clear distinction between an action and a state. Semantically, a state represents the properties at a particular moment, whereas an action consists of a set of state pairs, which represent a relation between two states. Thus, there are two sub-languages in dynamic logic: a sub-language for actions and a sub-language for states. The latter is called the meta language of dynamic logic. Let a be an action represented in the action sub-language, and ψ and ϕ the property formulas represented in the meta language. In dynamic logic, a formula like

$$\psi \rightarrow [a]\phi$$

means that if the property ψ holds, then the property ϕ holds after doing the action a. The formula above states a relation between the pre-condition ψ and the post-condition ϕ for the action a.

A scripting language based on the semantics of dynamic logic is well suited for the purpose of intelligent embodied agents. As discussed previously, the scripting language is primarily designed for the specification of body language and speech for embodied agents. In this framework, the specification of external-oriented communicative acts can be separated from the internal states of embodied agents because the former involves only geometric changes of the body objects and the natural transition of the actions, whereas the latter involves more complicated computation and reasoning.

Dynamic logic has several primitive action operators: '$\alpha;\beta$' means that α is executed before β; '$\alpha \cup \beta$' means that either α or β is executed nondeterministically; 'α^*' means that α is executed a finite, but nondeterministic number of times; and $p?$ means to proceed if p is true, else fail. Based on these primitive action operators, some typical actions are relatively easy to define [9], for example:

if p then α *else* β	*as* $(p?;\alpha) \cup (\neg p?;\beta)$
while p *do* α	*as* $(p?;\alpha)^*;\neg p?$
repeat α until p	*as* $\alpha(\neg p?;\alpha)^*;p?$
IF $p \rightarrow \alpha \parallel q \rightarrow \beta$ FI	*as* $(p?;\alpha) \cup (q?;\beta)$

Therefore, based on the formal semantics of dynamic logic, STEP has a solid semantic foundation, in spite of a rich number of variants of the compositional operators. Refer to [15] for more details of the semantics issues about STEP.

3.4 High-level Interaction Operators

When using high-level interaction operators, scripting actions can directly interact with internal states of embodied agents or with external states of worlds. These interaction operators are based on a meta language which is

used to build embodied agents, say, in the distributed logic programming language DLP. In the following, we use lower case Greek letters ϕ, ψ, χ to denote formulas in the meta language. Similar to those in dynamic logic, STEP has the following higher-level interaction operators:

- test: `test(`ϕ`)`, check the state ϕ. If ϕ holds then skip, otherwise fail.
- execution: `do(`ϕ`)`, make the state ϕ true, i.e. execute ϕ in the meta language.
- conditional: `if_then_else(`ϕ`,`$action_1$`,`$action_2$`)`.
- until: `until(action,`ϕ`)`, perform action until ϕ holds.

The above-mentioned action operators are sufficiently powerful to define a number of variants of scripting actions. In particular, the execution operator '*do*' is used to access certain computation and interaction capabilities from the meta language level. In DLP and Prolog, the predicate '*is*' is for the evaluation of arithmetic expressions. Accordingly, actions which involve the '*do*' operator and the predicate '*is*', like $do(N\ is\ sqrt(S))$, can be used to perform computations in STEP. Actions with the '*do*' operator in combination with the VRML/X3D EAI predicates in DLP, like $do(getPosition(Agent, X, Y, Z))$ and $do(setRotation(Object, X, Y, Z, R))$, can be used to interact with virtual worlds. The same patterns of actions in combination with the available communication predicates at the meta language level can be used to achieve certain communication facilities between embodied agents. We will discuss some details how these capabilities can be achieved in Section 4. Before doing so, we will describe a brief example of how a number of temporal relations can be defined in terms of the parallel action operator '*par*' and the sequential action operator '*seq*' by means of the execution operator '*do*'. As discussed in [1, 2], there are 13 possible temporal relations between two actions, that is, $before$, $meets$, $overlaps$, $starts$, $during$, $finishes$, $equals$, and their inverse relations. All these 13 possible temporal relations can be defined in STEP[15], for example:

```
before(A1,A2)= seq([A1, do(random(N)), wait(N),A2])
meets(A1,A2)= seq(A1,A2)
overlaps(A1,A2)= par([A1,seq([duration(A1,T1),do(random(R)),
                    do(N is T1*R), wait(N), A2])])
starts(A1,A2)= par([A1,A2])
```

where $duration(A, T)$ calculates the duration T for the action A, which can be defined recursively on the sub-actions of A. $wait(N)$ is a special action which does nothing but just waiting for N seconds. The action $wait(N)$ can be defined as $seq([do(T\ is\ N * 1000), do(sleep(T))])$.[4] See [15] for more details with respect to the expressiveness of STEP and its semantics.

We have implemented the scripting language STEP in the distributed logic programming language DLP. See [13] for implementation issues of STEP. Based on STEP, we have also implemented XSTEP[14], the XML-based markup language for embodied agents.

[4] Because the predicate *sleep* in DLP requires milliseconds.

4 Examples

In this section, we discuss several examples how STEP can be used to define scripting actions for embodied agents. The first two examples 'walk' and 'run' describe general examples of body movements of embodied agents. The third example 'look at ball' and 'run to ball' describe actions which demonstrate the interaction between agents and virtual worlds. Finally, in the fourth example 'touch', we discuss how STEP can be used to solve some inverse kinematics problems for embodied agents. The first two examples demonstrate how users can use STEP easily. The third and the fourth examples require some knowledge of 3D geometry. They are designed for professional users.

4.1 Walk and its Variants

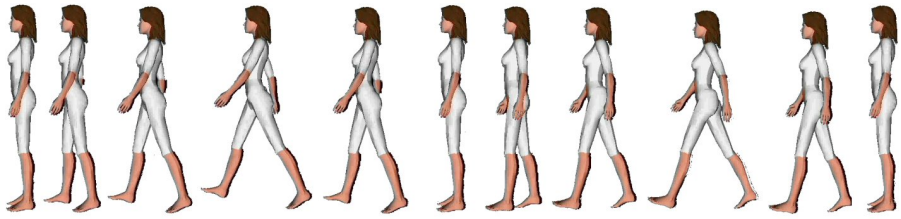

Fig. 5. Walk

A walking posture can be expressed as a movement which consists of the following two main activities: an action in which the left-arm/right-leg move forward while the right-arm/left-leg move backward, and an action in which the right-arm/left-leg move forward while the left-arm/right-leg move backward. The main poses and their linear interpolations are shown in Figure 5. The walk action can be described in the scripting language as follows:

```
script(walk_pose(Agent), Action):-
   Action = seq([par([
       turn(Agent,r_shoulder,back_down2,fast),
       turn(Agent,r_hip,front_down2,fast),
       turn(Agent,l_shoulder,front_down2,fast),
       turn(Agent,l_hip,back_down2,fast)]),
    par([turn(Agent,l_shoulder,back_down2,fast),
       turn(Agent,l_hip,front_down2,fast),
       turn(Agent,r_shoulder,front_down2,fast),
       turn(Agent,r_hip,back_down2,fast)])]).
```

As shown below, a walk step can be described as a parallel action which consists of the walking posture and the moving action (i.e. changing position):

```
script(walk_forward_step(Agent),Action):-
  Action= par([walk_pose(Agent),
           move(Agent,front,fast)]).
```

The step length can be a concrete value. For example, for a 0.7 meter step size, it can be defined as:

```
script(walk_forward_step07(Agent),Action):-
   Action= par([walk_pose(Agent),
          move(Agent,increment(0.0,0.0,0.7),fast)]).
```

Alternatively, the step length can also be a variable:

```
script(walk_forward_step0(Agent,StepLength),Action):-
  Action = par([walk_pose(Agent),
      move(Agent,increment(0.0,0.0,StepLength),fast)]).
```

Therefore, walking forward N steps with a particular *StepLength* can be defined as follows:

```
script(walk_forward(Agent,StepLength,N),Action):-
   Action = repeat(walk_forward_step0(Agent,StepLength),N).
```

As mentioned above, animations of the walk action based on these definitions are simplified and approximated ones. As analysed in [7, 20], a realistic animation of walk motions of human figure involves many computations which rely on a robust simulator where forward and inverse kinematics are combined with automatic collision detection and response. It is not our intention to use the scripting language to achieve a fully realistic animation of the walk action, because they are seldom necessary for most web applications. However, we would like to point out that there does exist the possibility to accommodate some inverse kinematics to improve the realism by using the scripting language.

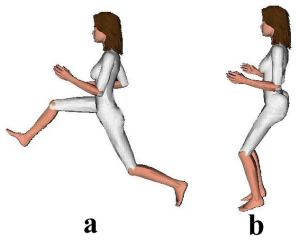

Fig. 6. Poses of Run

4.2 Run and its Deformation

As a first approximation, the action 'run' is similar to the action 'walk', however, with bending arms and legs. The latter would make the legs look like lifting from the ground, which is an important difference between the action 'walk' and the the action 'run'[19]. The run pose is shown in Figure 6a. As we can see from the figure, the left lower-arm points to the direction 'front-up' when the left upper-arm points to the direction 'front_down2' during the run action. Considering the hierarchies of the body parts, we should not use the primitive action $turn(Agent, l\_elbow, front\_up, fast)$ but the primitive action $turn(Agent, l\_elbow, front, fast)$, because the direction of the left lower-arm should be defined relative to the direction of its parent body part, i.e. the left arm (more exactly, the joint l_shoulder). This kind of re-direction does not impose major difficulties for authoring, because the correct direction can be obtained by reducing the directions of its parent body parts to be the default ones. As we can see in Figure 6b, the lower-arm actually points to the direction 'front'.

Based on the action 'walk', the action 'run_pose' can be defined as an action which starts with a run pose as shown in Figure 6b and then repeat the action 'walk_pose' for N times:

```
script(basic_run_pose(Agent), Action):-
 Action=par([turn(Agent,r_elbow,front,fast),
      turn(Agent, l_elbow, front, fast),
      turn(Agent, l_hip, front_down2, fast),
      turn(Agent, r_hip, front_down2, fast),
      turn(Agent, l_knee, back_down, fast),
      turn(Agent, r_knee, back_down, fast)]).

script(run_pose(Agent,N),Action):-
    Action = seq([basic_run_pose(Agent),
      repeat(walk_pose(Agent),N)]).
```

Therefore, the action running forward N steps with a particular $StepLength$ can be defined in the scripting language as follows:

```
script(run(Agent, StepLength,N),Action):-
 Action=seq([basic_run_pose(Agent), walk_forward(Agent,StepLength,N)]).
```

In practice, the action 'run' may have many variants. For instance, the lower-arm may point to different directions; they don't necessarily point to the direction 'front'. Therefore, we may define the action 'run' with respect to certain degrees of freedom. Here is an example to define a degree of freedom with respect to the angle of the lower arms to achieve the deformation.

```
script(basic_run_pose_elbow(Agent,Elbow_Angle),Action):-
   Action = par([
     turn(Agent,r_elbow,rotation(1,0,0,Elbow_Angle),fast),
     turn(Agent,l_elbow,rotation(1,0,0,Elbow_Angle),fast),
```

Fig. 7. Run

```
      turn(Agent,l_hip,front_down2,fast),
      turn(Agent,r_hip,front_down2,fast),
      turn(Agent,l_knee,back_down,fast),
      turn(Agent,r_knee,back_down,fast)]).

script(run_e(Agent,StepLength,N,Elbow_Angle),Action):-
   Action = seq([basic_run_pose_elbow(Agent,Elbow_Angle),
                 walk_forward(Agent, StepLength, N)]).
```

4.3 Interaction with Virtual Worlds

In this section we want to show how the interaction between embodied agents and virtual worlds can be achieved by using the high-level interaction operators. Consider a situation in which there are several agents and a ball. The position of the ball is always changing because other agents may kick the ball. We want to design the script actions for embodied agents so that they can always look at the ball and run to the ball no matter where the ball is located.

In the following, we suppose that the meta language of the scripts is DLP. Other languages can be used following the same strategy. Using DLP's VRML/X3D predicates, we can manipulate 3D objects in virtual worlds. For example, given the current position of the embodied agent and the ball, we can always calculate the new rotation of the agent so that it will look at the ball. By using the high-level interaction operator *do* with the built-in operators in the meta language we can define the script action 'look at ball' and other relevant actions.

First we want to define a scripting action 'turn_to_direction' which transforms a source direction vector into a destination direction by means of particular vector processing predicates. We know that the result of a vector cross product of two vectors v_1 and v_2 is a normal vector, i.e. a vector that is perpendicular to the original vectors v_1 and v_2. Such a normal vector defines the axis of the rotation and the corresponding angle θ between these two vectors can be calculated by the following formula:

$$cos\theta = \frac{v_1 \cdot v_2}{|v1| \times |v2|}$$

Therefore, a scripting action 'turn to direction' can be defined as:

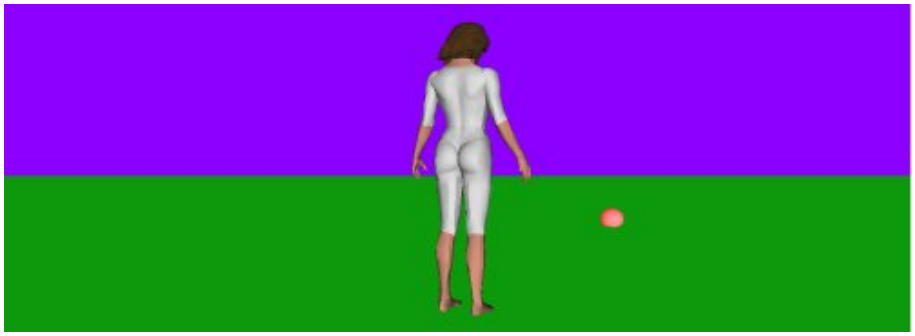

Fig. 8. Look at Ball

```
script(turn_to_direction(Object,SrcVector,DestVector),Action):-
  Action = seq([
      do(vector_cross_product(SrcVector,DestVector,vector(X,Y,Z),R)),
      do(setRotation(Object,X,Y,Z,R))]).
```

where the predicate $vector\_cross\_product(S, D, V, R)$ calculates the cross product V of the vector S and the vector D, as well as the angle R between the two vectors.

In general, embodied agents turn to the ball along the XZ plane, therefore we can ignore the Y-parameters. The Y-parameters are useful only when we want to calculate a rotation for the agent's head so that it can look down to the ball. H-anim avatars always face to the $+Z$ direction by default. Thus, the source vector is $\langle 0, 0, 1\rangle$. The destination vector can be calculated from the positions of the agent and the ball. Therefore, the scripting action 'look_at_position' can be defined as follows:

```
script(look_at_position(Agent,X1,_Y1,Z1),Action):-
  Action = seq([do(getPosition(Agent,X,_Y,Z)),
                do(Xdif is X1-X),
                do(Zdif is Z1-Z),
                turn_to_direction(Agent,vector(0.0,0.0,1.0),
                  vector(Xdif,0.0,Zdif))]).
```

Based on the scripting action 'look_at_position', the scripting action 'look_at_ball' can be easily defined as follows:

```
script(look_at_ball(Agent,Ball),Action):-
  Action = seq([do(getPosition(Ball, X1,Y1,Z1)),
                look_at_position(Agent,X1,Y1,Z1)]).
```

In the following, we want to define a script action $run\_to\_ball(Agent, Ball, N)$ so that the agent can continually run to the ball in N steps. Similarly we use the do-operator to obtain the current position of the agent and the ball first, from which we can calculate the increments of the positions in X and Z dimensions.

```
script(run_to_ball(Agent,Ball,Steps),Action):-
 Action = seq([do(getPosition(Agent,X,_,Z)),
               do(getPosition(Ball, X1,_,Z1)),
               do(StepLengthX is (X1-X)/Steps)),
               do(StepLengthZ is (Z1-Z)/Steps)),
               run_steps(Agent, increment(StepLengthX,0.0,
                          StepLengthZ),Steps)]).
```

The scripting action $run\_steps(Agent, Increment, N)$ describes an action in which the agent changes its position in N steps. This action can be defined as a recursive action:

```
script(run_steps(Agent,increment(X,Y,Z),1),Action):-
  Action = par([run_pose(Agent),
                move(Agent,increment(X,Y,Z),fast)]).

script(run_steps(Agent,increment(X,Y,Z),Steps),Action):-
  Action = seq([par([run_pose(Agent),
                     move(Agent,increment(X,Y,Z),fast)]),
                do(Steps1 is Steps - 1),
                run_steps(Agent,increment(X,Y,Z),Steps1)]).
```

4.4 Touch: An Inverse Kinematics Problem

A typical inverse kinematics problem is the calculation of the rotations of arms and wrists of embodied agents so that their hands can touch an object. As discussed in [20], many research efforts deal with this kind of problems. Finding solutions to this kind of inverse kinematics problems usually involves complex computations, like solving differential equations or applying particular non-linear optimizations [3, 20]. As discussed above, we can use high-level interaction operators to access the computational capabilities of the meta language in order to find the solutions by using the same methods which have been proposed in the literature. However, adopting these analytical and numerical methods to solve inverse kinematics problems may cause some performance problems for web applications. Therefore, one of our concerns is to find an acceptable trade-off between performance and realistic animations.

To illustrate this, we will discuss a 'touch' example in more detail to show how the scripting language STEP can be used to solve some real-time inverse kinematics problems with a satisfying performance result. To simplify the problem, embodied agents are designed to behave like this: they will touch an object by using their hands if the object is reachable, otherwise they will point their hands in the direction of the object. In addition, we will ignore the upper and lower limits of the rotations of the shoulder and elbow joints. In particular, we assume that the elbow joint has enough degrees of freedom for an appropriate solution.

This simplified 'touch' problem can be described as: given an agent *Agent* and a position $\langle x_0, y_0, z_0 \rangle$ of an object, try to set the rotations of the joints of

the shoulder and the elbow so that the hand of the agent can touch exactly the position if the position is reachable. Suppose that the length of the upper arm is u, the length of the forearm is f, and the distance between the shoulder center $\langle x_3, y_3, z_3\rangle$ and the destination position $\langle x_0, y_0, z_0\rangle$ is d. The position $\langle x_0, y_0, z_0\rangle$ is reachable if and only if $d \leq u + f$ if we ignore the upper and lower limits of the joint rotations. From the cosine law we know that if the object is reachable, then α, the angle between the upperarm and the forearm, can be calculated from:

$$\alpha = arc\ cos(\frac{u^2 + f^2 - d^2}{2uf})$$

Furthermore, if v is the direction vector which points to the destination position from the shoulder center, v_0 the default direction vector of the arm, and v_1 the destinating direction vector of the upperarm (Figure 9), then the angle β between the vector v and v_1 is given by:

$$\beta = arc\ cos(\frac{u^2 + d^2 - f^2}{2ud})$$

if the object is within the agent's reach.

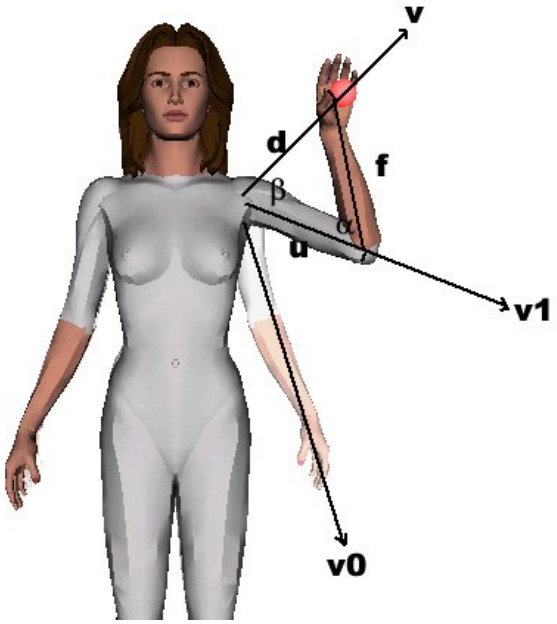

Fig. 9. Inverse Kinematics of Touch

If the position is not reachable, then $\alpha = \pi$ and $\beta = 0$ so that the arm will point to the direction of the destination position. Moreover, if $d \approx 0$, then the

destination position is close to the shoulder center. In this case, we set $\alpha = 0$ and $\beta = 0$. We can define a scripting action to realize the functions for α and β as follows:[5]

```
script(getABvalue(Agent,position(X0,Y0,Z0),Hand,A,B),Action):-
  Action = seq([
            getDvalue(Agent,position(X0,Y0,Z0),Hand, D),
            get_upperarm_length(Agent,L1),
            get_forearm_length(Agent,L2),
            do(D1 is L1 + L2),
            if_then_else(sign(D1-D)>sign(0.001-D),
                seq([do(cosine_law(L1,L2,D,A)),
                     do(cosine_law(L1, D, L2, B))]),
                seq([do(A is 1.57*(1+sign(D-0.001))),
                     do(B is 0.0)]))]).
```

The predicate *getDvalue* is an action which calculates the distance D between the shoulder center and the destination touch position for an agent. Suppose that the destination position $\langle x_0, y_0, z_0 \rangle$ is relative to the coordinate system of the agent body at which the agent is positioned in the default position and orientation of H-anim avatars, namely, it faces to the $+Z$ direction at the position $\langle 0, 0, 0 \rangle$. The action *getDvalue* can be defined by obtaining the positions of the shoulder. In the following, we will define a 'touch' action for relative positions first. We call the 'touch' action for agents with arbitrary position and arbitrary orientation *a 'touch' action for an absolute position*. We will show how the 'touch' action for absolute positions can be based on a 'touch' action for relative positions.

The cross product $v_0 \times v$, i.e. a normal vector $n = \langle x_n, y_n, z_n \rangle$, can be considered as a normal vector for v_0 and v_1, which defines the plane in which the arm turns from its default rotation to a destination rotation. This means that we require that the vector v_1 is in the same plane as the vectors v and v_0 so that the arm will turn close to the destination position via a shortest path. The angle γ between v_0 and v, can be calculated with the vector predicates, like those that are used in the last example. Thus, the rotation for the elbow joint is $\langle x_n, y_n, z_n, \pi - \alpha \rangle$, and the rotation for the shoulder joint is $\langle x_n, y_n, z_n, \gamma - \beta \rangle$.

The vector v can be calculated by using the following script, considered that the destination position is a relative one.

```
script(getVvalue(Agent,position(X0,Y0,Z0),Hand,V),Action):-
  Action = seq([
    get_shoulder_center(Agent,Hand, position(X2,Y2,Z2)),
    do(direction_vector(position(X2,Y2,Z2),position(X0,Y0,Z0),V))]).
```

where the predicate get_shoulder_center gets the position of the shoulder center, and the predicate direction_vector obtains a direction vector of the two

[5] The predicate $getABvalue(Agent, position(X0, Y0, Z0), Hand, A, B)$ means that for the agent $Agent$ and the destination position $(X0, Y0, Z0)$ of the $Hand$, the value of α is A, and the value of β is B.

positions. It is easy to define these two predicates at the STEP level. However, the predicate direction_vector is already available in DLP in order to obtain a better performance.

Now, we define the scripting action 'touch' for relative positions with the left hand as follows:

```
script(touch(Agent, position(X0,Y0,Z0),l),Action):-
  Action = seq([
    getABvalue(Agent,position(X0,Y0,Z0),l,A,B),
    do(R1 is 3.14-A),
    getVvalue(Agent,position(X0,Y0,Z0),l,V),
    get_arm_vector(Agent,l,V0),
    do(vector_cross_product(V0,V,vector(X3,Y3,Z3),C)),
    do(R2 is C-B),
    par([turn(Agent,l_shoulder,rotation(X3,Y3,Z3,R2),fast),
         turn(Agent,l_elbow,rotation(X3,Y3,Z3,R1),fast),
         turn(Agent,l_wrist,rotation(X3,Y3,Z3,-0.5),fast)])]).
```

Although we do not calculate the rotation for the wrist joint, we can adjust the rotation of the wrist joint based on the same normal vector, so that the hand can rotate a little bit to the position to achieve more realism. The 'touch' action with the right hand can be defined similarly.

Finally, we can define the 'touch' action for absolute positions in terms of the 'touch' action for relative positions, by the translation of the absolute position into a relative position, based on the agent's current position and orientation.

```
script(touch_absolutePosition(Agent,position(X1,Y1,Z1), Hand),Action):-
  Action = seq([do(getPosition(Agent,X,Y,Z)),
            do(getRotation(Agent, X2,Y2,Z2,R)),
            do(X3 is X1-X),
            do(Y3 is Y1-Y),
            do(Z3 is Z1-Z),
            do(R1 is -R),
            do(position_rotation(position(X3,Y3,Z3),
                  rotation(X2,Y2,Z2,R1),position(X4,Y4,Z4))),
            touch(Agent,position(X4,Y4,Z4), Hand)]).
```

where the predicate $position\_rotation(P1, R, P2)$ gets the new position $P2$ for a given position $P1$ after the rotation R.

Several touch situations based on this scripting action are shown in Figure 10. The tests show that STEP does not cause serious performance problems for this kind of inverse kinematics problem. Currently the computation time for each touch action is less than 50 milliseconds on a PC with a 500 mhz CPU and 128 MB memory, a low-end computer nowadays, under Windows NT running standard processes. There is still much room for improvement at the STEP level and the DLP meta language level. Therefore, the scripting language STEP can be used to achieve certain real-time inverse kinematics effects with a satisfying realism without serious performance problems.

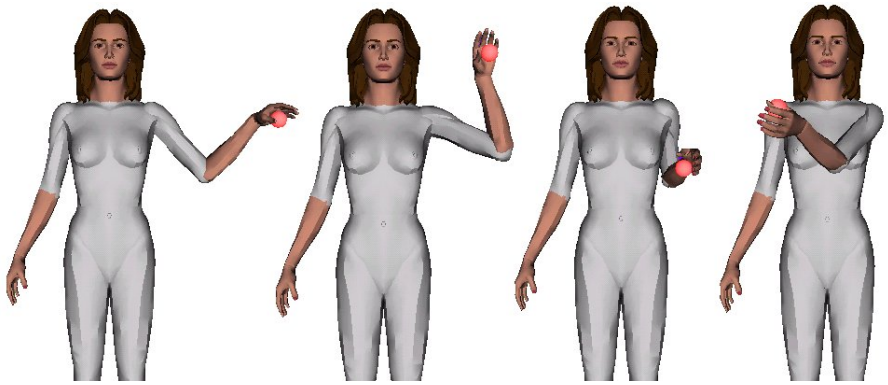

Fig. 10. Touch a Ball

5 Related Work

Our work is close to Perlin and Goldberg's Improv system. In [16], Perlin and Goldberg propose Improv, which is a system for scripting interactive actors in virtual worlds. STEP is different from Perlin and Goldberg's in the following aspects: First, STEP is based on the H-anim specification, i.e. VRML-based, which is convenient for Web applications. Secondly, we separate the scripting language from the agent architecture. Therefore, it is relatively easy for users to use the scripting language. Also, *Signing Avatar*[6] has a powerful scripting language. However, we wish to state that our scripting language is based on dynamic logic, and has powerful abstraction capabilities and support for parallelism.

In [4], Badler et al. discuss the system of Parameterized Action Representation (PAR), which provides an approach to use natural language instructions for virtual agents. Similar to STEP, PAR supports the specification of complex actions for agents, like pre- and post-conditions. PAR uses a special syntax and has many built-in notions. STEP is more similar to Funge's CML (Cognitive Modeling Language) [8] [7], although CML is not VRML/X3D and H-anim based. STEP shares many common operators with CML, like sequences, test, non-deterministic choice, and conditional operators. CML is an implementation of complex actions within the situation calculus, an action logic based on first order predicate logic. However, STEP is based on dynamic logic in which there is a clear distinction between a sub-language for actions and a meta language for states. As discussed before, such a distinction is very useful for both the development of fully-functional intelligent agents by using a meta language, and the realization of scripting actions by using a scripting language for a better performance.

[6] http://www.signingavatar.com
[7] http://www.dgp.toronto.edu/~funge/cml.html

In [17], Prendinger et al. are also using a Prolog-based scripting approach for animated characters but they focus on higher-level concepts such as affect and social context. In [18], Prendinger et al. discuss the systems MPML and SCREAM, an approach of scripting the bodies and minds of life-like characters.

XSTEP shares a number of interests with the VHML (Virtual Human Markup Language) community[8], which is developing a suite of markup language for expressing humanoid behavior, including facial animation, body animation, speech, emotional representation, and multimedia. We see this activity as complementary to ours, since our research proceeds from technical feasibility, that is how we can capture the semantics of humanoid gestures and movements within our dynamic logic, which is implemented on top of DLP.

6 Conclusions

In this chapter we have proposed the scripting language STEP for embodied agents, in particular for their communicative acts, like gestures and postures. STEP will be extended for other communicative acts like facial expressions or speech. We have discussed several principles of scripting language design for embodied agents. These principles are justified by a number of typical examples of how the scripting language STEP can be used. The first two examples 'walk' and 'run' show that STEP can be used for authors to design a rich number of variants of scripting actions. The third example demonstrates the capabilities of STEP for applications which involve interactions with virtual worlds. The fourth 'touch' example discusses the possibilities of using STEP for real-time agents with inverse kinematics. The experiments show that STEP can be used for embodied agents and applications of inverse kinematics with a satisfying performance.

References

1. Allen, J.F., Maintaining Knowledge about Temporal Intervals, *Communications of the ACM* 26, 11, 832-843, November 1983.
2. Allen, J.F., Time and time again: The many ways to represent time, *Journal of Intelligent Systems* 6, 4, 341-356, July 1991.
3. Badler, N., Manoochehri, K., and Walters, G., Articulated figure positioning by multiple constraints, *IEEE Computer Graphics Applications*, 7(6), 28-38, 1987.
4. Badler N., Bindiganavale, R., Bourne, J., Palmer, M., Shi, J., and Schuler, W., A Parameterized Action Representation for Virtual Human Agents, Workshop on Embodied Conversational Characters, Lake Tahoe, California, 1998.
5. Eliëns, A., *DLP, A Language for Distributed Logic Programming*, Wiley, 1992.

[8] http://www.vhml.org

6. Eliëns, A., *Principles of Object-Oriented Software Development*, Addison-Wesley, 2000.
7. Faure, F., et al., Dynamic analysis of human walking, Proceedings of the 8th Workshop on Computer Animation and Simulation, Budapest, 1997.
8. Funge, J., *Making Them Behave: Cognitive Models for Computer Animation*, University of Toronto, 1998.
9. Harel, D., Dynamic Logic, *Handbook of Philosophical Logic*, Vol. II, D. Reidel Publishing Company, 497-604, 1984.
10. Harel, D., Kozen, D., and Tiuryn, J., *Dynamic Logic*, MIT Press, 2000.
11. Huang, Z., Eliëns, A., van Ballegooij, A., and de Bra, P., A Taxonomy of Web Agents, *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, IEEE Computer Society Press, 765-769, 2000.
12. Huang, Z., Eliëns, A., and Visser, C., Programmability of Intelligent Agent Avatars, *Proceedings of Agents'01 Workshop on Embodied Agents*, 2001.
13. Huang, Z., Eliëns, A., and Visser, C., Implementation of a Scripting Language for VRML/X3D-based Embodied Agents, *Proceedings of the 2003 Web 3D Conference*, ACM Press, 2003.
14. Huang, Z., Eliëns, A., and Visser, C., XSTEP: a Markup Language for Embodied Agents, *Proceedings of the 16th International Conference on Computer Animation and Social Agents*(CASA'2003), IEEE Computer Society Press, 2003.
15. Huang, Z., Eliëns, A., and Visser, C., Formal Semantics of STEP: a dynamic logic approach, Research report, Vrije Universiteit Amsterdam, 2003.
16. Perlin, K., and Goldberg, A., Improv: A System for Scripting Interactive Actors in Virtual Worlds, *ACM Computer Graphics*, Annual Conference Series, 205-216, 1996.
17. Prendinger, H., Descamps, S., and Ishizuka, M., Scripting affective communication with life-like characters in web-based interaction systems, *Journal of Applied Artificial Intelligence*, 16, 519-553, 2002.
18. Prendinger, H., Saeyor, S., Ishizuka, M., MPML and SCREAM: Scripting the bodies and minds of life-like characters. In: *Life-like Characters. Tools, Affective Functions and Applications*, ed. by Prendinger, H., Ishizuka, M. (Springer 2003). This volume.
19. Rohr, K., Towards Model-based Recognition of Human Movements in Image Sequences. *Computer Vision, Graphics, and Image Processing (CVGIP): Image Understanding*, 59(1), 94-115, 1994.
20. Tolani, D., Goswami, A., and Badler, N., Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs, *Graphical Models*, 62(5), 353-388, 2000.

Index