STEP: A Scripting Language for Embodied Agents

Zhisheng Huang, Anton Eliëns and Cees Visser
Vrije University Amsterdam, The Netherlands
{huang,eliens,ctv}@cs.vu.nl

Abstract

In this paper 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 logics, STEP has a solid semantic foundation, in spite of a rich number of variants of the compositional operators and interaction facilities on the worlds. STEP has been implemented in the distributed logic programming language DLP, a tool for the implementation of 3D web agents. In this paper, we also discuss principles of scripting language design for embodied agents and several aspects of the application of STEP.

Keywords: embodied agents, virtual environments, VRML, avatars, humanoids, H-anim, 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 [5].

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

In this paper 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, speech, and other types of embodied agents, like cognitive robots. Scripting languages are to a certain extent simplified languages which ease the task of computation and reasoning. 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 other mental state reasoning. Thus, changing the specification of communicative acts doesn't require to re-program the agent.

The avatars of 3D web agents are built in the Virtual Reality Modeling Language (VRML). These avatars are usually humanoid-like ones. 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 [1].[2]

DLP is a tool for the implementation of 3D intelligent agents [6].[3] In this paper, 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 those in dynamic logic [4]. Thus, STEP has a solid semantic foundation, in spite of a rich number of variants of the compositional operators and interaction facilities on the worlds.

2 Principles

We design the scripting language primarily for the specification of communicative acts for embodied agents. Namely, we separate external-oriented communicative acts from internal changes of the mental states of embodied agents because the former involves only geometrical changes of the body objects and the natural tran-

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

sition 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's not our intention to design a scripting language with fully-functional computation facilities, like other programming languages as Java, Prolog or DLP. A scripting language should be interoperable 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 have not necessarily the same expressiveness of a general programming language. However, we do require that a scripting language should be able to interact with mental states of embodied agents in some ways, which will be discussed in more detail later.

We consider the following principles of the design for a scripting language.

Principle 1: Convenience

As mentioned, the specification of communicative acts, like gestures and facial expressions usually involve a lot of geometrical data, like using ROUTE statements in VRML, or movement equations, like those in the computer graphics. A scripting language should hide those geometrical difficulties, so that non-professional authors can use it in a natural way. For example, suppose that authors want to specify that the agent turns his left arm forward slowly. It can be specified like this:

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

It should not be necessary to soecify as follows, which requires knowledge of a coordination 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. Namely, 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, like:

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

Typical composite operators for actions are sequence action, parallel action, repeat action, which are standardly used in dynamic logics [4].

Principle 3: Re-definability

Scripting actions (i.e., composite actions), can be defined in terms of other defined actions explicitly. Namely, the scripting language should be a rule-based specification system. Scripting actions are defined with their own names. These defined actions can be re-defined for other scripting actions. 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)=
[script(run(Agent)), script(kick(Agent))].
```

which can be specified in a Prolog-like syntax:

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

Principle 4: Parametrization

Scripting actions can be adapted to be other actions. Namely, actions can be specified in terms of how these actions cause changes over time to each individual *degree of freedom*, which is proposed by Perlin and Goldberg in [9]. For example, suppose that we define a scripting action *run*: we know that running can be done at different paces. It can be a 'fast-run' or 'slow-run'. We should not define all of the 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 with 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. That is one of the reasons why we introduce Prolog-like syntax in STEP.

Principle 5: Interaction

Scripting actions would not simply just continually take effects on the world. They should be able to interact with, more exactly, perceive the world, including embodied agents' mental states, to decide whether or not it would continue the current action, or change to other actions, or stop the current action. This kind of interaction modes can be achieved by the introduction of high-level interaction operators, like those in dynamic logic. The operator 'test' and the operator 'conditional', which are useful for the interaction between the actions and the states.

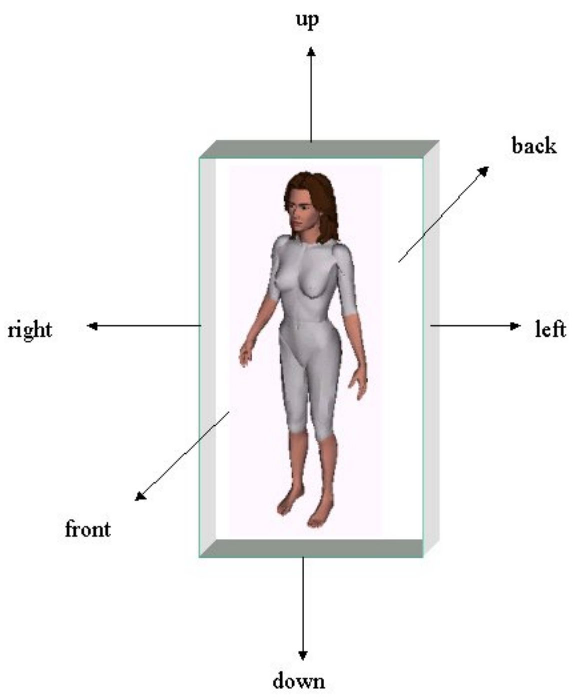

Figure 1: Direction Reference for Humanoid

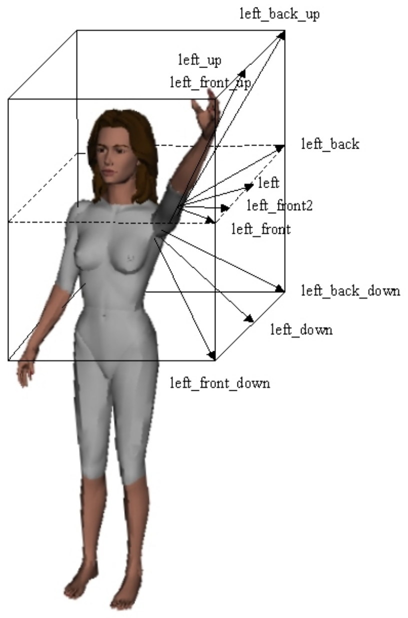

Figure 2: Combination of the Directions for Left Arm

3 Scripting Language STEP

3.1 Reference Systems

3.1.1 Direction Reference

The reference system in STEP is based on H-anim specification: namely, 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, say, 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 orginal VRML reference system, which is useful for experienced authors. Directions can also be specified to be a four-place tuple $\langle X,Y,Z,R\rangle$, say, $rotation(1,0,0,1.57)$.

3.1.2 Body Reference

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 typical joints of humanoids. Therefore, turning body parts of humanoids implies the setting of the relevant joint's rotation. Body moving means the setting of the HumanoidRoot to a new position. For instance, the action 'turning the left-arm to the front slowly' is specified as:

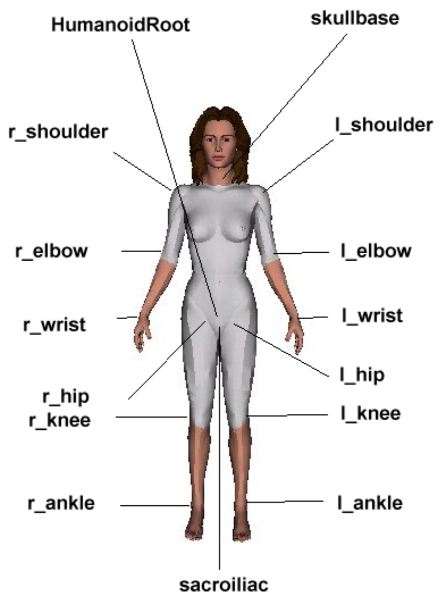

Figure 3: Typical Joints for Humanoid

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

3.1.3 Time Reference

STEP has the same time reference system as that in 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 per minute. By default, the tempo is set to 60. Namely, a beat corresponds to a second by default. However, the tempo can be changed. Moreover, we can define different speeds for body movements, say, 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 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)', $Duration$ can be a speed name like 'fast' or an explicit time specification, like 'time(2,second)'.

A move action 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)'.

Here are typical composite operators for scripting actions:

- Sequence operator ',': we use a comma separated list to define a composite sequence action, which is like,

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

- Parallel operator 'parallel': the action $parallel([Action_1, ..., Action_n])$ 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 Parametrization and Scoping

Change of tempo is one of the most useful temporal deformation methods. Tempo changes apply to different scopes, either globally, i.e. its change effects all threads, or locally, i.e. its change effects only one thread. In this paper, we use the latter, namely, tempo with a local scope.

3.4 High-level Interaction Operators

When using high-level interaction operators, scripting actions can directly interact with internal states of embodies agents or with external states of worlds. These interaction operators are based on a meta language which is used to build embodied agents, say, 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. Examples of several higher-level interaction operators:

- test: $test(\phi)$, check the state ϕ. If ϕ holds then skip, otherwise fail.
- execution: $do(\phi)$, make the state ϕ true, i.e. execute ϕ in the meta language.

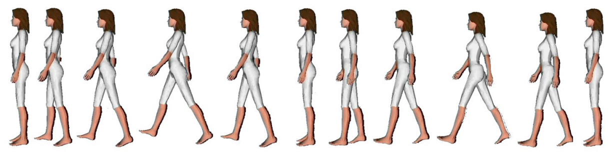

Figure 4: Walk

- conditional: $if\_then\_else(\phi, action_1, action_2)$.
- until: $until(action, \phi)$: take action until ϕ holds.

We have implemented the scripting language STEP in the distributed logic programming language DLP. We discuss the implementation issues in the paper [7].

4 Examples

4.1 Walk and its Variants

A walking posture can be simply expressed as a movement which exchanges the following two main poses: a pose in which the left-arm/right-leg move forwardly while the right-arm/left-leg move backward, and a pose in which the right-arm/left-leg move forwardly while the left-arm/right-leg move backward. The main poses and their linear interpolations are shown in Figure 4. The walk action can be described in the scripting language as follows:

```
script(walk_pose(Agent), ActionList):-
   ActionList =[parallel([
       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)]),
    parallel([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)])].
```

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

```
script(walk_forward_step(Agent),ActionList):-
  ActionList=[parallel([script_action(walk_pose(Agent),
           move(Agent,front,fast)])].
```

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

```
script(walk_forward_step07(Agent),ActionList):-
   ActionList=[parallel([script_action(walk_pose(Agent),
          move(Agent,increment(0.0,0.0,0.7),fast)])].
```

Alternatively, the step length can also be a variable like:

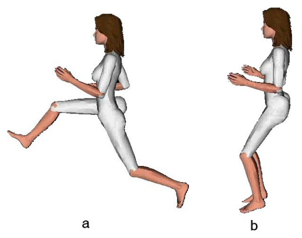

Figure 5: Poses of Run

```
script(walk_forward_step0(Agent,StepLength),ActionList):-
  ActionList =[parallel([script_action(walk_pose(Agent),
      move(Agent,increment(0.0,0.0,StepLength),fast)])].
```

Therefore, the walking forward N steps with the *StepLegnth* can be defined as follows:

```
script(walk_forward(Agent,StepLength,N),ActionList):-
   ActionList =[repeat(script_action(
        walk_forward_step0(Agent,StepLength)),N)].
```

As mentioned above, the animations of the walk based on those definitions are just simplified and approximated ones. As analysed in [3], a realistic animation of the walk motions of human figure involves a lot of the computations which rely on a robust simulator where forward and inverse kinematics are combined with automatic collision detection and response. We do not want to use the scripting language to achieve a fully realistic animation of the walk, 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. See the paper [7] for details.

4.2 Run and its Deformation

The action 'run' is similar to 'walk', however, with a bigger wave of the lower-arms and the lower-legs, which is shown in Figure 5a. 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. Consider the hierarchies of the body parts, we should not use the primitive action $turn(Agent, l\_elbow, front\_up, fast)$ but the primitive action $turn(Agent, l\_elbow, front, fast)$, for the direction of the left lower-arm should be defined with respect to the default direction of its parent body part, i.e., the left arm (more exactly, the joint l_shoulder). That kind of the re-direction would not cause big difficulties for the authoring, for 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 5b, the lower-arm actually points to the direction 'front'.

The action 'run_pose' can be simply defined as an action which starts with a basic run pose as shown in Figure 5b and then repeat the action 'walk_pose' for N times as follows:

```
script(basic_run_pose(Agent), ActionList):-
 ActionList=[parallel([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),ActionList):-
    ActionList =[script_action(basic_run_pose(Agent)),
      repeat(script_action(walk_pose(Agent)),N)].
```

Therefore, the action running forward N steps with the *StepLegnth* can be defined in the scripting language as follows:

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

Actually, the action 'run' may have a lot of variants. For instances, the lower-arm may point to different directions. They would not 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 on the waving angle of the lower arms to achieve the deformation.

```
script(basic_run_pose_elbow(Agent,Elbow_Angle),ActionList):-
   ActionList = [parallel([
     turn(Agent,r_elbow,rotation(1,0,0,Elbow_Angle),fast),
     turn(Agent,l_elbow,rotation(1,0,0,Elbow_Angle),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_e(Agent,StepLength,N,Elbow_Angle),ActionList):-
   ActionList =[script_action(
      basic_run_pose_elbow(Agent,Elbow_Angle)),
      script_action(walk_forward(Agent, StepLength, N))].
```

5 Conclusions

In this paper we have proposed the scripting language STEP for embodied agents, in particular for their communicative acts, like gestures and postures. Moreover, we have discussed the principles of scripting language design for embodied agents and several aspects of the application of the scripting language.

Our work is close to Perlin and Goldberg's **Improve** system. In [9], 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, thus, VRML-based, which is convenient for Web applications. Secondly, we separate the scripting language from the agent architecture. Therefore, it's relatively easy for users to use the scripting language.

STEP shares a number of interests with the VHML(Virtual Human Markup Language) community[4], 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.

We are also working on the development of X–STEP , the XML-based STEP markup language [8], so that the definitions of scripting actions can be in separated XML-based text files. Based on that, embodied agents can exchange their information of their scripting actions more efficiently over the Web.

References

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

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

[3] F. Faure, et al., Dynamic analysis of human walking, Proceedings of the 8th Workshop on Computer Animation and Simulation, Budapest, 1997.

[4] D. Harel, Dynamic Logic, *Handbook of Philosophical Logic*, Vol. II, D. Reidel Publishing Company, 1984, 497-604.

[5] Z. Huang, A. Eliëns, A. van Ballegooij, P. de Bra, A Taxonomy of Web Agents, *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, IEEE Computer Society, 765-769, 2000.

[6] Z. Huang, A. Eliëns, and C. Visser, *Programmability of Intelligent Agent Avatars*, *Proceedings of Agents'01 Workshop on Embodied Agents*, 2001.

[7] Z. Huang, A. Eliëns, and C. Visser, STEP: a Scripting Language for Embodied Agents(full version), `http://wasp.cs.vu.nl/step/paper/script.pdf`

[8] Z. Huang, A. Eliëns, and C. Visser, *X–STEP : an XML-based STEP Markup Language for Embodied Agents*, in preparation, 2002.

[9] K. Perlin, and A. Goldberg, Improv: A System for Scripting Intereactive Actors in Virtual Worlds, *ACM Computer Graphics*, Annual Conference Series, 205-216, 1996.

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