XSTEP: An XML-based Markup Language for Embodied Agents

Zhisheng Huang, Anton Eliëns and Cees Visser

Intelligent Multimedia Group
Division of Mathematics and Computer Science
Vrije University Amsterdam
De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
{huang,eliens,ctv}@cs.vu.nl

Abstract

In this paper we propose a XML-based markup language for embodied agents, called XSTEP, based on the scripting language STEP. XSTEP is XML-encoded STEP. STEP is developed on the formal semantics of dynamic logic, and has been implemented in the distributed logic programming language DLP, a tool for the implementation of 3D web agents. In this paper, we discuss the issues of markup language design for embodied agents and several aspects of the implementation and application of XSTEP.

Keywords: embodied agents, avatars, humanoids, H-anim, STEP, XSTEP, XML

Contents

1 Introduction **2**

2 Design of Markup Languages for Embodied Agents **2**
2.1 What XSTEP wants to have . . . 2
2.2 What XSTEP does not want to become . . . 3
2.3 A Recommended Reference System for XSTEP . . . 4
2.3.1 Direction Reference . . . 4
2.3.2 Body Reference . . . 6
2.3.3 Time Reference . . . 6

3 XSTEP:XML-encoded STEP **8**
3.1 Actions Operators . . . 8
3.2 High-level Interaction Operators . . . 8
3.3 Example: Walk and its Variants . . . 9

4 XSTEP: Components and Implementation **11**
4.1 Components of XSTEP . . . 11
4.2 Implementation of XSTEP . . . 11

5 Conclusions **11**
5.1 Comparison . . . 12
5.2 Further work . . . 12

6 Appendix A: XSTEP DTD **14**

7 Appendix B: XSTEP XSL: translation of XSTEP into STEP **17**

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* [19]. *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 non-verbal or verbal. Gestures, postures and facial expressions are typical non-verbal communicative acts. One of the main applications of embodied agents are virtual presenters, or alternatively called *presentation/conversation agents*. These agents are designed to represent users/agents in virtual environments, like virtual meeting spaces, or virtual theaters, by means of hypermedia tools as part of the user interface.

These kinds of applications appeal for human markup languages for multimedia presentations. These markup languages should be able to accommodate the various aspects of human-computer interaction, including facial animation, body animation, speech, emotional representation, and multimedia. In [7], we outline the requirements for a software platform supporting embodied conversational agents. These requirements encompass computational concerns as well as presentation facilities, providing a suitably rich environment for applications deploying conversational agents.

The avatars of 3D web agents are typically built in the Virtual Reality Modeling Language (VRML)[1]. These avatars are usually humanoid-like ones. The humanoid animation working group[2] 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 STEP for H-anim based humanoids in the distributed logic programming language DLP [3, 5]. DLP is a tool for the implementation of 3D intelligent agents [13, 14].

STEP introduces a Prolog-like syntax, which makes it compactable with most standard logic programming languages, whereas the formal semantics of STEP is based on those in dynamic logic [10]. 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.

In this paper, we propose a XML-based markup/scripting language for embodied agents, called XSTEP, based on the scripting technology STEP. Thus, XSTEP is XML-encoded STEP. In this paper, we discuss the issues of markup language design for embodied agents and several aspects of the implementation and application of XSTEP.

This paper is organized as follows: Section 2 discusses the general requirements on a markup language for embodied agents, and examine in what extend the scripting language STEP would satisfy these requirements. Section 3 proposes the language XSTEP, namely, the XML-encoded STEP, and discuss the examples. Section 4 discusses the components of XSTEP and its existing implementation. Section 5 compares XSTEP with other markup/scripting languages, discusses the future work, and concludes the paper.

2 Design of Markup Languages for Embodied Agents

2.1 What XSTEP wants to have

We consider the following requirements for the design of the markup language for embodied agents.

Temporary The specification of communicative acts, like gestures and facial expressions usually involve the changes of geometrical data with time, like ROUTE statements in VRML, or movement equations, like those in the computer graphics. A markup language for the presentation of embodied agents should be designed to base on a solid temporal semantics. A good solution is to use existing temporal models, like those in temporal logics or dynamic logics. The scripting language STEP, thus, the markup language XSTEP, is

[1] http://www.vrml.org
[2] http://www.h-anim.org

based on the semantics of dynamic logics. The typical temporal operators in STEPare: the sequential action *seq* and the parallel action *par*. Thus, in XSTEP, we have the corresponding tags <par> and <seq>.

Agent-orientation Markup languages for embodied agents should be different from those markup language for general multimedia presentation, like SMIL. The formers have to consider the expressability and capability of their targeted agents. However, It's not our intention to design a markup language with fully-functional computation facilities, like other programming languages as Java, DLP or Prolog, which can be used to construct a fully-functional embodied agents. 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 transition of the actions, whereas the latter involves more complicated computation and reasoning. The markup language is designed to be a simplified, user-friendly specification language for the presentation of embodied agents instead of for the construction of a fully functional embodied agent. A markup/scripting language should be interoperable with a fully powered agent implementation language, but offer a rather easy way for authoring. This kind of interaction modes can be achieved by the introduction of high-level interaction operators, like those in dynamic logic. The typical higher level interaction operators are: the operator 'do' and the operator 'conditional'. In XSTEP, these two operators are presented as two markup tags <do> and <if_then_else> respectively, which are discussed in the section 3.

Prototypability The presentation of embodied agents usually consists of some typical communicative acts, say, a presentation with greeting gesture. The specification of the greeting gesture can also be used for other presentation. Therefore, a markup language for embodied agents should have the re-usability facilities. XML-based markup languages offer a convenient tool for the information exchange over the Web. Thus, an inline hyperlink in the markup language is an easy solution for this purpose. That would lead to the design of prototypability of markup languages, like the internal/external prototypes in VRML. The scripting language STEP is designed to 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. XSTEP uses the similar strategy like STEP for the prototypability. One of the advantages of this kind rule-based specification is *parametrization*. 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 [18]. 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 STEP introduces Prolog-like syntax. Thus, XSTEP also uses the similar method of the parametrization like that in STEP. Just like those in Prolog, an action rule is said to be general if there exists unbounded variable on it. An instantiated rule is one in which there exists no unbounded variable. Similarlly in Prolog, both in STEP and XSTEP, a variable is denoted by a name starting with a upper-case character, whereas a constant is denoted by a name starting with a lower-case character.

Ontological-relevance XSTEP is designed for the purpose of convenience, in the sense that it can be used for non-professional authors, and the XSTEPcodes are interoperable across the Web. The principle of the convenience implies that the use of the flexible terms, most are natural-language-like, for its reference systems. A good solution to maintenance of the interoperability is to make XSTEP ontological-relevance. So-called *Ontology* is a description of the concepts or bodies of knowledge understood by a particular community and the relationships between those concepts. In the existing version of XSTEP, we recommend a typical ontological specification as its reference system. This typical ontology claim is based on H-anim specification, which is discussed in details in Subsection 2.3.

2.2 What XSTEP does not want to become

XSTEP is designed to be a markup language for the presentation of embodied agents. Naturally, XSTEP is considered to be one which includes a lot of functionality on the relevant specifications, like on those for 2D/3D avatar, multimedia, and agents. There is a lot of work have been done on these areas. Most of them are quite mature already. XSTEP does not want to overlap these existing work. These language/specification can be embedded into XSTEPin some degrees. Here are several examples:

- **2D/3D graphical markup languages** The specification of the scalable vector graphics (SVG)[3] is a typical XML-based language for describing two-dimensional graphics. SVG drawings can serve as XML-based 2D avatars for embodied agents. The X3D[4], the new generation of VRML, is a typical XML-based language for 3D object specification. X3D can be used a tool for the design of XML-based 3D avatars. XSTEP codes can be used to manipulate these avatars specified by SVG/X3D. Moreover, these SVG/X3D codes can also be embedded into XSTEPcodes. Thus, it is not necessary for XSTEP to overlap the functionality of the languages SVG and X3D.

- **XML-based multimedia markup languages** The Synchronized Multimedia Integration Language (SMIL)[5] is a typical XML-based multimedia specification language, which integrates streaming audio and video with images, text or any other media type. Again, XSTEP does not want to replace/overlap the functionality of the language SMIL. The SMIL codes can also embedded into XSTEP ones.

- **Humanoid markup languages** H-anim[6] is typically used to be a specification for humanoid based on VRML. The body references in H-anim are well suitable to be used as an ontology of the body parts for 3D embodied agents. Therefore, the body reference based on h-anim becomes a typical ontological specification in STEP and XSTEP. That would be disucssed in details in Subsection 2.3. The X3D extension of H-anim can be considered a typical XML-based Humanoid markup specification. The HumanMarkup specification (HumanML)[7] is another example to represent human characteristics through XML.

- **Agent specification languages** Embodied agents can be constructed by means of different ways. They can be directly built by programming languages like Java, prolog, or DLP. Agents specification languages offer indirect ways to build embodies agents, in the sense that they are built with high-level abstraction. The Foundation for Intelligent Physical Agents (FIPA)[8] produces standards for heterogeneous and interacting agents and agent-based systems. XSTEP does not want to replace any existing work on agent specification languages. The existing version of XSTEP is able to interact with the internal states of embodies agents which are directly built via the high-level interact operators/tags. The approach to interact with embodied agents built by agent specification languages is one of the further work for XSTEP.

2.3 A Recommended Reference System for XSTEP

H-anim is a typical specification for 3D avatars based on VRML. It can serve as a point of departure for the design of the reference system in XSTEP. In this paper, we recommend this typical ontological specification for XSTEP.

2.3.1 Direction Reference

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 for non-professional authors. 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

[3] http://www.w3.org/Graphics/SVG/
[4] www.web3d.org/x3d.html
[5] http://www.w3.org/AudioVideo/
[6] http://www.h-anim.org
[7] http://www.oasis-open.org/committees/humanmarkup/index.shtml
[8] http://www.fipa.org/

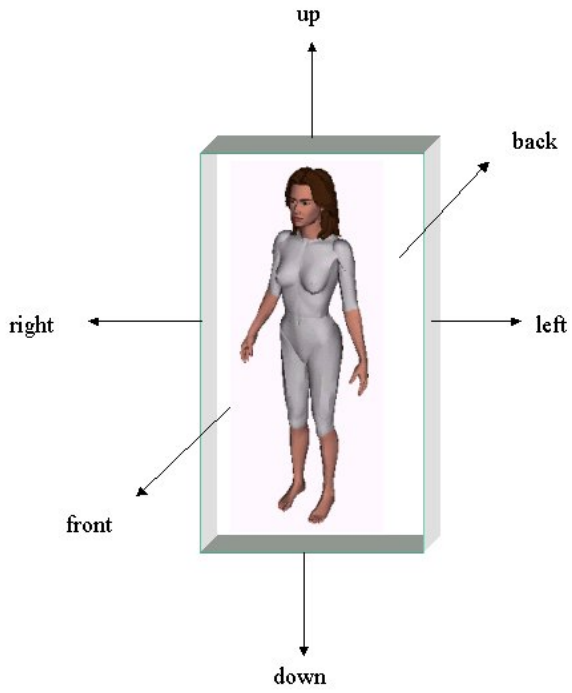

Figure 1: Direction Reference for Humanoid

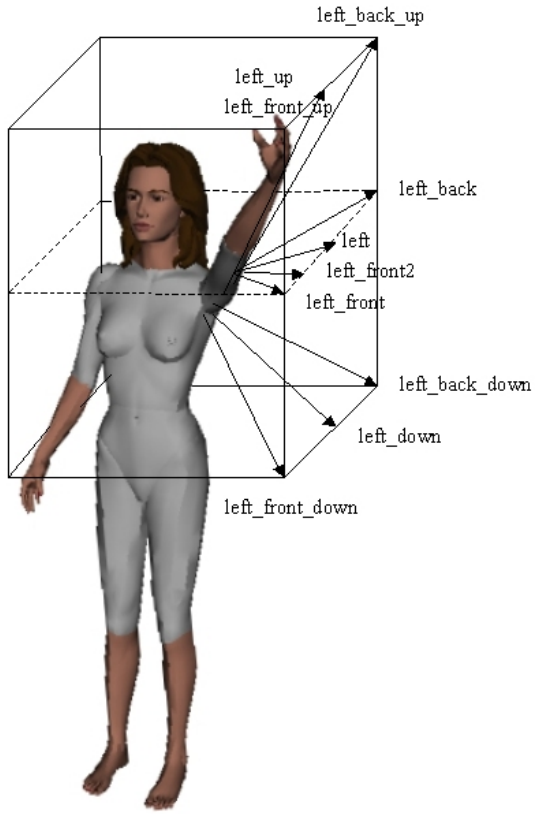

Figure 2: Combination of the Directions for Left Arm

for authors to specify scripting actions, which does not require the author have a detailed knowledge of reference systems in VRML. Moreover, STEP 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, $rotation(1, 0, 0, 1.57)$. Thus, the directions consists of a complex composited types, which cannot be simplely represented as an attributes in an XML elements. Therefore, in XSTEP, the directions are represented either the tug 'dir', like <dir value="front"/> or the tug 'rotation', like <rotation x="1" y="0" z="0" r="1.57"/>. In XSTEP, the elements of the directions are defined in DTD as follows:

```
<!ELEMENT dir EMPTY>
<!ATTLIST dir
value %Direction; #REQUIRED>

<!ELEMENT rotation EMPTY>
<!ATTLIST rotation
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED
r CDATA #REQUIRED>
```

2.3.2 Body Reference

An H-Anim specification contains a set of *Joint nodes* that are arranged to form a hierarchy. 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:

```
<turn actor="Agent" part="l_shoulder">
            <dir value="front"/>
            <speed value="slow"/>
</turn>
```

2.3.3 Time Reference

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

```
<turn actor="Agent" part="l_shoulder">
            <dir value="front"/>
  <time unit="second" value="2"/>
</turn>
```

This kind of explicit specification of duration in scripting actions does not satisfy the parametrization principle. STEP introduces 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. In XSTEP, the elements of the time reference are defined in DTD as follows:

```
<!ENTITY % Speed        "(fast|slow|intermedia|very_fast|very_slow)">

<!ELEMENT time EMPTY>
<!ATTLIST time
value CDATA #REQUIRED
unit (second|minute|beat) #REQUIRED>
```

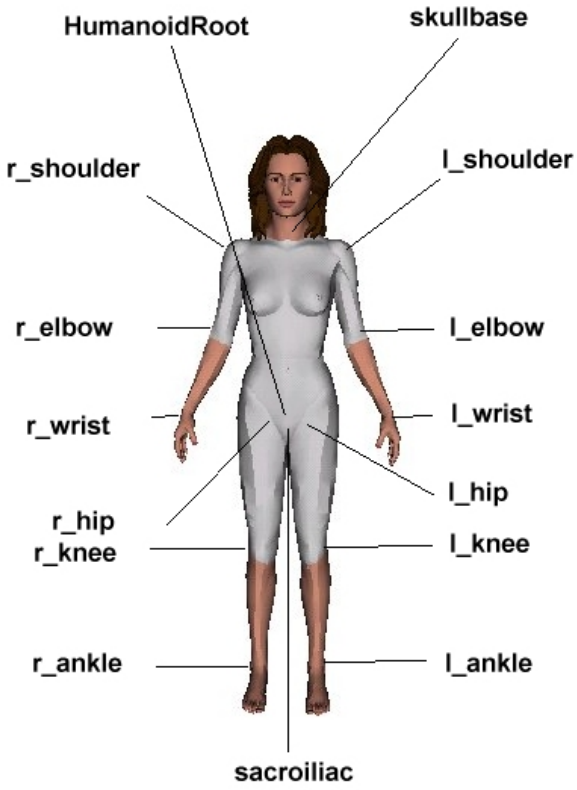

Figure 3: Typical Joints for Humanoid

```
<!ELEMENT speed EMPTY>
<!ATTLIST speed
value %Speed; #REQUIRED>
```

3 XSTEP:XML-encoded STEP

3.1 Actions Operators

Turn and move are 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. In XSTEP, the elements of the primitive actions are defined in DTD as follows:

```
<!ELEMENT turn ((dir|rotation),(speed|time))>
<!ATTLIST turn
actor CDATA #REQUIRED
  part  %BodyPart; #REQUIRED>

<!ELEMENT move ((dir|increment|position),(speed|time))>
<!ATTLIST move
actor CDATA #REQUIRED
part  %BodyPart;  #REQUIRED>

<!ELEMENT position EMPTY>
<!ATTLIST position
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED>


<!ELEMENT increment EMPTY>
<!ATTLIST increment
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED>
```

Similar with SMIL, XSTEP has the same timing operators/tugs: sequence action 'seq' and parallel action 'par'. Based on the semantics of dynamic logics, XSTEPhas the following action operators:

- non-deterministic choice operator 'choice': the action <choice> $Action_1, ..., Action_n$ </choice> denotes a composite action in which one of the $Action_1$, ...,and $Action_n$ is executed.
- repeat operator 'repeat': the action <repeat action="Action" times="T"/> denotes a composite action in which the *Action* is repeated T times.

3.2 High-level Interaction Operators

When using high-level interaction operators, XSTEP 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:

- execution: <do state="ϕ"/>, make the state ϕ true, i.e. execute ϕ in the meta language.

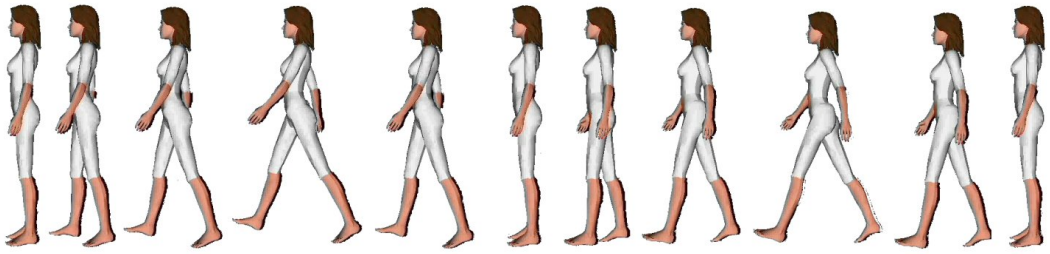

Figure 4: Walk

- conditional: <if_then_else cond="ϕ" then="$action_1$" else="$action_2$"/>: if ϕ holds, then execute $action_1$ else execute $action_2$.

3.3 Example: 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 XSTEP as follows:

```
<action name="walk(Agent)">
      <seq>
        <par>
          <turn actor="Agent" part="r_shoulder">
            <dir value="back_down2"/>
            <speed value="fast"/>
          </turn>
          <turn actor="Agent" part="r_hip">
            <dir value="front_down2"/>
            <speed value="fast"/>
          </turn>
          <turn actor="Agent" part="l_shoulder">
            <speed value="fast"/>
            <dir value="front_down2"/>
          </turn>
          <turn actor="Agent" part="l_hip">
            <dir value="back_down2"/>
            <speed value="fast"/>
          </turn>
        </par>
        <par>
          <turn actor="Agent" part="l_shoulder">
            <dir value="back_down2"/>
            <speed value="fast"/>
          </turn>
          <turn actor="Agent" part="l_hip">
            <dir value="front_down2"/>
            <speed value="fast"/>
          </turn>
          <turn actor="Agent" part="r_shoulder">
            <dir value="front_down2"/>
            <speed value="fast"/>
```

```
          </turn>
          <turn actor="Agent" part="r_hip">
            <dir value="back_down2"/>
            <speed value="fast"/>
          </turn>
        </par>
      </seq>
    </action>
```

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:

```
<action name="walk_forward_step(Agent)">
       <par><script_action name=walk_pose(Agent)/>
            <move actor=Agent part="humanoidRoot">
<dir value="front"/><speed value="fast"/>
</move>
</par>
</action>
```

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

```
<action name="walk_forward_step07(Agent)">
        <par><script_actionn name="walk_pose(Agent)"/>
             <move actor="Agent" part="humanoidRoot">
<increment x=0 y=0 z=0.7/><speed value="fast"/>
</move>
</par>
</action>
```

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

```
<action name="walk_forward_step0(Agent,StepLength)">
          <par><script_action name="walk_pose(Agent)">
               <move actor="Agent" part="humanoidRoot">
<increment x="0" y="0" z="StepLength"/><speed value="fast"/>
</move>
</par>
</action>
```

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

```
<action name="walk_forward(Agent,StepLength,N)">
        <repeat action="walk_forward_step0(Agent,StepLength)" times="N"/>
</action>
```

The animations of the walk based on those definitions are just simplified and approximated ones. As analysed in [8], a realistic animation of the walk motions of human figure involve 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 XSTEP 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 STEP. That is discussed in details in [16].

4 XSTEP: Components and Implementation

4.1 Components of XSTEP

A complete XSTEP code consists of these three components: library, head and embedded_code.

- **library**. XSTEP is mainly used to construct gesture/action libraries. These definitions of scripting actions are located in the XSTEP library component. Namely, they are located inside the tag $<$ *library* $>$ with or without a name of the library. The scripting actions in the libraries are usually formatted as general rules with variables according to the requirement on the prototypability. They can be re-usable by the calling from other internal/external actions. So-called *internal actions* are ones located at the same XSTEP files, whereas the *external actions* are ones located at other files.

- **head**. The head component in XSTEP consist of the following elements:

  1. world: states the url of the virtual world/avatar, or whether avatar codes are embedded, so that XSTEP can load the virtual world into the web browser;
  2. starting action: states an instantiated action so that XSTEP can start the action for the presentation;
  3. meta-language statement: states the meta-language for the high-level interact operators. The meta-language is considered to be the DLP as default;
  4. ontology claim: states the inline url, or indicates the embedded specification, for the ontology of XSTEP codes. The default ontology claim is considered to the recommended reference system based on H-anim specification in this paper.

- **embedded_code**. Embedded codes consists of other XML-based codes, like SVG/X3D codes which specify virtual-worlds/avatars, or XML-based ontological specification, which will be further studied in [17].

4.2 Implementation of XSTEP

We have implemented a STEP kernel by using the Distributed logic programming language DLP[9][15]. The scripting actions in STEP can be embedded into DLP codes of embodied agents with the STEP kernel. These actions can be called by the interfacing predicates of the STEP kernel for the purpose of the presentation of the embodied agents.

A STEPtestbed with respect to the default ontology, i.e., the recommended reference systems, has been implemented. Users can use the STEP testbed on web browsers to construct their own STEP scripting actions and test them online without any knowledge of DLP and VRML.

We have also implemented an XSTEPeditor based on IBM's XML editor Xeena[10]. This XSTEP editor would help the author to edit XSTEP codes and translate them into STEP scripts so that they can be run from the STEP tool, testbed, or DLP codes. A screenshot of the XSTEP editor on Xeena is shown in Figure 5.

We are now working on the development of the tool so that the STEP and XSTEP codes can be run from a standalone file.

5 Conclusions

In this paper we have proposed the markup language XSTEP for embodied agents. Moreover, we have discussed the requirements on the markup language design for embodied agents and several aspects of the implementation and application of the markup language XSTEP. In the following, we would like to make a comparison on XSTEP with other XML-based markup languages for humanoids, and discusses the further work.

[9] http://wasp.cs.vu.nl/step
[10] http://www.alphaworks.ibm.com/tech/xeena

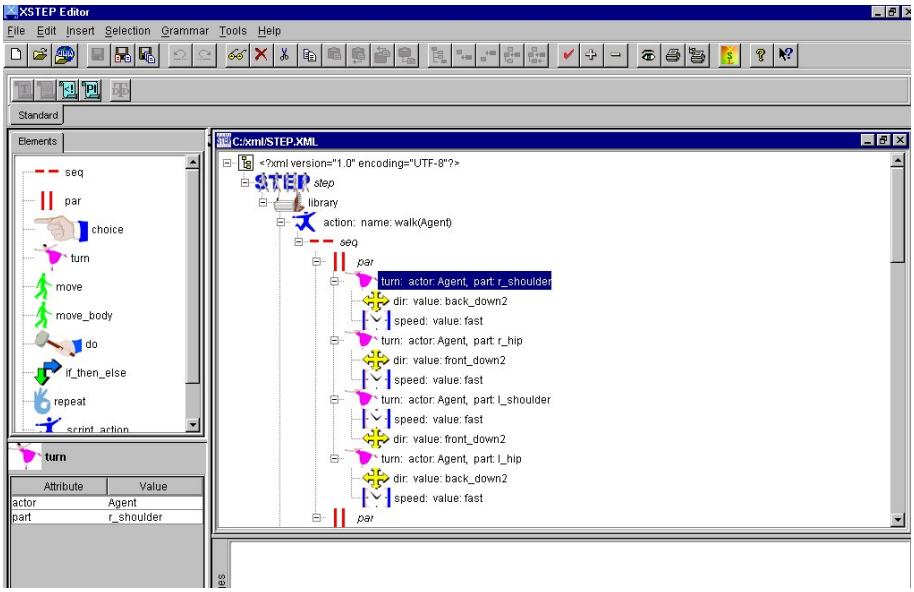

Figure 5: Screenhot of XSTEP editor on Xeena

5.1 Comparison

XSTEP can be considered to be one of VHML (Virtual Human Markup Language)-like languages[20]. The language VHML is designed to accommodate the various aspects of human-computer interaction, including facial animation, body animation, speech, emotional representation, and multimedia. XSTEP and VHML share a lot of the common goals.

One of the difference between XSTEP and the existing vhml is: XSTEP is developed based on the formal semantics of dynamic logic, so that it has a solid semantic foundation, in spite of a rich variants of the compositional operators and the interaction facilities on the worlds. Secondly, Prolog-like parametrization in XSTEP would make it more suitable for the interaction with intelligent embodied agents.

An interesting examples for the animated humanoid avatars is provided by *Signing Avatar*.[11] The scripting language for *Signing Avatar* is based on the H-anim specification and allows for a precise definition of a complex repertoire of gestures, as examplified by the sign language for the deaf. Nevertheless, this scripting language is of a proprietary nature and does not allow for high-order abstractions of semantically meaningful behavior.

More comparison and related work discussion with STEP and XSTEP can be found in the papers [7, 15].

5.2 Further work

- **architecture of gesture/action libraries**. In the existing implementation of STEP/XSTEP, we consider only the unique action/gesture library which is embedded in the applications. A flexible and powerful approach is to allow inline libraries and hierarchical architecture of the libraries. A further investigation on the architectures of gesture libraries is needed so that the libraries can be reusable more efficiently and can be shared by different authorship.

- **ontology of human markup languages**. More human markup languages are expected to be proposed in coming years. These languages may use completely different terminology and semantics models. An ontological investigation for human markup language is needed so that the presentations and their libraries can be interoperable[17].

- **facial expression and emotion models in XSTEP**. We are going to extend XSTEP with facial expressions. These facial expression can be marked as the tags 'anger', 'happy' and 'sad', like those

[11] http://www.signingavatar.com

are suggested in VHML. The terminology can be formalized based on emotion models and further specified by the corresponding ontological claim which is based on the survey in [17].

- **speech and other multimedia modes in XSTEP**. We are also planning to extend XSTEP with speech/voice and other multimedia modes, so that we can enrich embodied agents with the functionality needed to create convincing embodied agents in a meaningful context.

References

[1] Alice web site: `http://www.alice.org`.

[2] Bell, J., A Planning Theory of Practical Rationality. Proc. AAAI'95 Fall Symposium on *Rational Agency: Concepts, Theories, Models and Applications*, 1-4.

[3] DLP web site: `http://www.cs.vu.nl/~eliens/ projects/logic/index.html`.

[4] R. Earnshaw, N. Magnenat-Thalmann, D. Terzopoulos, and D. Thalmann, Computer Animation for Virtual Humans, *IEEE Computer Graphics and Applications* 18(5), 1998.

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

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

[7] Anton Eliëns, Zhisheng Huang, and Cees Visser, A platform for Embodied Conversational Agents based on Distributed Logic Programming, Proceedings of AAMAS 2002 WORKSHOP: Embodied conversational agents - let's specify and evaluate them, 2002.

[8] Franois Faure, Gilles Debunne, Marie-Paule Cani-Gascuel, Franck Multon, Dynamic analysis of human walking, Proceedings of the 8th Eurographics Workshop on Computer Animation and Simulation, Budapest, September 1997.

[9] Humanoid animation working group: `http://h-anim.org/Specifications/H-Anim1.1/`, 2001.

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

[11] Zhisheng Huang, Anton Eliëns, Alex van Ballegooij, Paul 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.

[12] Zhisheng Huang, Anton Eliëns, and Paul de Bra, An Architecture for Web Agents, *Proceedings of the Conference EUROMEDIA'2001*, SCS, 2001.

[13] Zhisheng Huang, Anton Eliëns, and Cees Visser, Programmability of Intelligent Agent Avatars, *Proceedings of the Autonomous Agents'01 Workshop on Embodied Agents*, 2001.

[14] Zhisheng Huang, Anton Eliëns, and Cees Visser, *3D Agent-based Virtual Communities*, *Proceedings of the 2002 Web 3D Conference*, ACM Press, 2002.

[15] Zhisheng Huang, Anton Eliëns, and Cees Visser, STEP: a Scripting Language for Embodied Agents, submit to the workshop of lifelike animated agents, Tokyo, 2002.

[16] Zhisheng Huang, Anton Eliëns, and Cees Visser, STEP: a Scripting Language for Embodied Agents (full version), WASP Research Report, Vrije University Amsterdam, 2002. `http://wasp.cs.vu.nl/step/paper/script.pdf`.

[17] Zhisheng Huang, Anton Eliëns, and Cees Visser, An ontological investigation on human markup languages, in preparation, 2002.

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

[19] Stuart C. Shapiro, Eyal Amir, Henrik Grosskreutz, David Randell, and Mikhail Soutchanski, Commonsense and Embodied Agents: A Panel Discussion, Common Sense 2001: The Fifth International Symposium on Logical Formalizations of Commonsense Reasoning, Courant Institute of Mathematical Sciences, New York University, New York, NY, May 20–22, 2001. http://www.cs.buffalo.edu/~shapiro/Papers/commonsense-panel.pdf

[20] VHML web site: http://www.vhml.org.

[21] WASP project home page: http://wasp.cs.vu.nl/wasp.

6 Appendix A: XSTEP DTD

```
<!-- XSTEP DTD, version 0.35-->


<!ENTITY % BodyPart        "(l_shoulder|r_shoulder|l_hip|r_hip|l_elbow|r_elbow|l_knee|r_knee|
                             l_wrist|r_wrist|l_ankle|r_ankle|humanoidRoot|skullbase|sacroiliac)">
<!ENTITY % Direction       "(front|back|left|right|up|down|back_down2|back_down|front_down2|
                             front_down|front_down1|back_up|front_up|turn_90|turn_45|turn_180|
                             turn_270|turn_360|turn_n90|turn_n45|turn_n180|turn_n270|turn_n360|
                             front_right|front_right2|front_right1|back_down_turn_90|
                             left_back_down_turn_90|right_back_down_turn_n90|left_front_down_turn_90
                             left_front_down|right_front_down|down_from_front|up_via_center|
                             up_via_right|up_via_side|up_via_left|side_up|side|left_up|side_down|
                             left_down|side2_down|left2_down|side1_down|left1_down|center_up)">
<!ENTITY % Speed        "(fast|slow|intermedia|very_fast|very_slow)">
<!ENTITY % Operator        "(do|if_then_else|repeat)">
<!ENTITY % Action        "(seq | par | choice | turn | move |move_body |do | if_then_else |
                             repeat | script_action|tempo)">
<!ENTITY % MoveElement1 "((dir|increment|position),(speed|time))">
<!ENTITY % MoveElement2 "((speed|time),(dir|increment|position))">
<!ENTITY % MoveElement "(%MoveElement1; | %MoveElement2;)">

<!ENTITY % TurnElement1 "((dir|rotation),(speed|time))">
<!ENTITY % TurnElement2 "((speed|time),(dir|rotation))">
<!ENTITY % TurnElement "(%TurnElement1; | %TurnElement2;)">


<!ELEMENT step (head?, library+, embedded_code?)>

<!ELEMENT head (world?,start?,meta_language?,ontology_claim?)>

<!ELEMENT world EMPTY>
<!ATTLIST world
url CDATA #IMPLIED
embedded (yes|no) #IMPLIED
>

<!ELEMENT start EMPTY>
<!ATTLIST start
action CDATA #REQUIRED
library CDATA #IMPLIED
>

<!ELEMENT meta_language EMPTY>
```

```
<!ATTLIST meta_language
value (dlp|java|prolog) #REQUIRED
>

<!ELEMENT ontology_claim EMPTY>
<!ATTLIST ontology_claim
name CDATA #REQUIRED
url CDATA #IMPLIED
embedded (yes|no) #IMPLIED
>


<!ELEMENT embedded_code (avatar?,ontology?)>

<!ELEMENT avatar ((svg|x3d)?)>
<!ATTLIST avatar
name CDATA #IMPLIED
>

<!ELEMENT ontology CDATA>

<!ELEMENT tempo EMPTY>
<!ATTLIST tempo
value CDATA #REQUIRED
>


<!ELEMENT library (action)* >
<!ATTLIST library
name CDATA #IMPLIED
>


<!ELEMENT action %Action;>
<!ATTLIST action
name CDATA #REQUIRED
>



<!ELEMENT turn %TurnElement;>
<!ATTLIST turn
actor CDATA #REQUIRED
  part  %BodyPart;  'l_shoulder'>

<!ELEMENT move %MoveElement; >
<!ATTLIST move
actor CDATA #REQUIRED
part  %BodyPart;  'l_shoulder'>

<!ELEMENT move_body %MoveElement;>
<!ATTLIST move_body
actor CDATA #REQUIRED>
```

```
<!ELEMENT time EMPTY>
<!ATTLIST time
value CDATA #REQUIRED
unit (second|minute|beat) #REQUIRED>


<!ELEMENT speed EMPTY>
<!ATTLIST speed
value %Speed; #REQUIRED>


<!ELEMENT dir EMPTY>
<!ATTLIST dir
value %Direction; #REQUIRED>

<!ELEMENT position EMPTY>
<!ATTLIST position
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED >

<!ELEMENT rotation EMPTY>
<!ATTLIST rotation
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED
r CDATA #REQUIRED>


<!ELEMENT increment EMPTY>
<!ATTLIST increment
x CDATA #REQUIRED
y CDATA #REQUIRED
z CDATA #REQUIRED >


<!ELEMENT script_action EMPTY >
<!ATTLIST script_action
name CDATA #REQUIRED>

<!ELEMENT do EMPTY >
<!ATTLIST do
state CDATA #REQUIRED>

<!ELEMENT if_then_else EMPTY >
<!ATTLIST if_then_else
condition CDATA #REQUIRED
then CDATA #REQUIRED
else CDATA #REQUIRED>

<!ELEMENT repeat EMPTY >
<!ATTLIST repeat
action CDATA #REQUIRED
times CDATA #REQUIRED>
```

```
<!ELEMENT par (%Action;)+ >

<!ELEMENT seq (%Action;)+  >

<!ELEMENT choice (%Action;)+  >
```

7 Appendix B: XSTEP XSL: translation of XSTEP into STEP

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text"/>

<xsl:template match="/step">
%% STEP script
%% Created by Xstep.xsl
<xsl:apply-templates/>
%% End of STEP script
</xsl:template>

<xsl:template match="head">
%% created from the head of xstep
<xsl:apply-templates/>
%%end of the head
</xsl:template>

<xsl:template match="library">
%% start of the action library <xsl:value-of select="@name"/>
<xsl:apply-templates/>
%% end of the action library <xsl:value-of select="@name"/>
</xsl:template>

<xsl:template match="action">
script(<xsl:value-of select="@name"/>,Action):-
   Action =<xsl:apply-templates/>,
!.
</xsl:template>

<xsl:template match="start">
start_action(<xsl:value-of select="@action"/>).
</xsl:template>

<xsl:template match="world">
worldURL(<xsl:value-of select="@url"/>).
</xsl:template>

<xsl:template match="meta_language">
meta_language(<xsl:value-of select="@value"/>).
</xsl:template>


<xsl:template match="seq">
```

```
seq([<xsl:apply-templates/>
])<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>

<xsl:template match="par">
   par([<xsl:apply-templates/>
   ])<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>

<xsl:template match="choice">
  choice([<xsl:apply-templates/>
])<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>

<xsl:template match="dir"><xsl:value-of select="@value"/></xsl:template>

<xsl:template match="speed"><xsl:value-of select="@value"/></xsl:template>

<xsl:template match="increment">increment(<xsl:value-of select="@x"/>,
<xsl:value-of select="@y"/>,<xsl:value-of select="@z"/>)</xsl:template>

<xsl:template match="position">position(<xsl:value-of select="@x"/>,
<xsl:value-of select="@y"/>,<xsl:value-of select="@z"/>)</xsl:template>

<xsl:template match="rotation">rotation(<xsl:value-of select="@x"/>,
<xsl:value-of select="@y"/>,<xsl:value-of select="@z"/>,
<xsl:value-of select="@r"/>)</xsl:template>

<xsl:template match="time">time(<xsl:value-of select="@value"/>,
<xsl:value-of select="@unit"/>)</xsl:template>



<xsl:template match="turn">
turn(<xsl:value-of select="@actor"/>, <xsl:value-of select="@part"/>,
<xsl:apply-templates select="(dir|rotation)"/>,<xsl:apply-templates select="(speed|time)"/>)
<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>


<xsl:template match="move">
move(<xsl:value-of select="@actor"/>, <xsl:value-of select="@part"/>,
<xsl:apply-templates select="(dir|position|increment)"/>,
<xsl:apply-templates select="(speed|time)"/>)
<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>

<xsl:template match="move_body">
move_body(<xsl:value-of select="@actor"/>,
<xsl:apply-templates select="(dir|position|increment)"/>,
<xsl:apply-templates select="(speed|time)"/>)
<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>

<xsl:template match="script_action">
script_action(<xsl:value-of select="@name"/>)
<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>

<xsl:template match="if_then_else">
if_then_else(<xsl:value-of select="@condition"/>,
```

```
<xsl:value-of select="@then"/>, <xsl:value-of select="@else"/>)
<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>

<xsl:template match="do">
do(<xsl:value-of select="@state"/>)
<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>


<xsl:template match="repeat">
repeat(<xsl:value-of select="@action"/>,
<xsl:value-of select="@times"/>)
<xsl:if test="not(position()=last())">,</xsl:if></xsl:template>



<!-- ignore all not matched -->
<!-- xsl:template match="*" priority="-1"/ -->

</xsl:stylesheet>
```