Programmability of Intelligent Agent Avatars

Zhisheng Huang
Vrije University of Amsterdam
De Boelelaan 1081
1081 HV Amsterdam
The Netherlands
huang@cs.vu.nl

Anton Eliëns
Vrije University of Amsterdam
De Boelelaan 1081
1081 HV Amsterdam
The Netherlands
eliens@cs.vu.nl

Cees Vissser
Vrije University of Amsterdam
De Boelelaan 1081
1081 HV Amsterdam
The Netherlands
ctv@cs.vu.nl

ABSTRACT

In this paper, we propose an approach to the programmability of intelligent agent avatars, supported by the distributed logic programming language DLP. Intelligent agent avatars can be considered as one of the applications of web agents. As one of the testbeds of 3D web agents, we are developing and implementing soccer playing avatars. We discuss how the language DLP can be used to support soccer playing avatars using rules to guide their behaviors in networked virtual environments.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence; H.4.m [**Information Systems**]: Miscellaneous

General Terms

Intelligent Agent

Keywords

avatar, intelligent agent, distributed logic programming, networked virtual environment

1. INTRODUCTION

In recent years, more and more applications of networked virtual environments, in particular 3D virtual communities, have been developed. Popular 3D virtual community servers are Active World [1] and Blaxxun Interactive [2]. However, most of them use a relatively simple and rather static representation of their users by avatars. Avatars represented by intelligent agents significantly improve the interface and the capabilities of networked virtual environments [3, 16].

The Blaxxun community server does provide support for agents. Agents in the Blaxxun community server may be programmed to have particular attributes and to react to events in a particular way. As a remark, originally the Blaxxun agents were called bots. In our opinion the functionality of Blaxxun agents does not surpass that of simple bots and we consider the term agent to be a misnomer. Despite the large number of built-in events and the rich repertoire of built-in actions, the Blaxxun agent platform in itself is rather limited in functionality, because event-action patterns are not powerful enough to program complex behavior that requires maintaining information over a period of time.

In this paper, we propose an approach to the programmability of intelligent agent avatars supported by the distributed logic programming language DLP [5]. Our research is part of a Dutch research project WASP, for Web Agent Support Program [15]. As part of this research project we are developing PAMELA, a Personal Assistant for Maintaining Electronic Archives. PAMELA has a general architecture of web agents [8], which can be either 3D avatar-embodied agents, or 2D text-based interface agents. In this paper, we will focus on the approach to 3D avatar-embodies agents. We are developing and implementing soccer playing avatars on the Web as one of the benchmark examples for our 3D web agent framework.

2. WEB AGENTS AND ARCHITECTURE

2.1 A taxonomy of web agents

Many types of web agents have been proposed in recent years, which range from domain-dependent agents, like e-commerce agents, information gathering agents, expertise seeking agents, travel assistant agents, virtual community agents, to function-dependent agents, like negotiation agents, co-operating agents, and problem solving agents. The natural questions related to that phenomenon are: what are the relations among so many different types of web agents? Are they redundant or overlapped? Is there a taxonomy to classify them?

In [7], we propose a taxonomy of web agents, which encompasses agents that provide a text-based interface to, for example, information retrieval services, as well as avatar-embodied guides that help visitors to navigate in virtual environments. This section is a brief introduction to the taxonomy of web agents.

We consider the following three major dimensions of web agents.

- **2D versus 3D**.
  A 2D web agent is one which is aware of http, file, and ftp protocols, whereas a 3D web agent is one which is not only aware of these protocols, but also virtual reality specific protocols. Typical 2D web agents provide a text-based interface, for example, information retrieval services. Typical 3D web agents are avatar-embodied guides that help visitors to navigate in virtual environments.

- **Client versus server**.
  As the names imply, a client web agent is on the client side, whereas a server web agent is on the server side. A typical client web agent can serve as a personal information assistant. A typical server web agent can serve as the front-end of web servers to offer information more intelligently.

- **Singularity versus multiplicity**.
  A single web agent doesn't cooperate with other web agents, whereas frameworks of multiple web agents exhibit agent collaboration patterns in order to achieve a common goal.

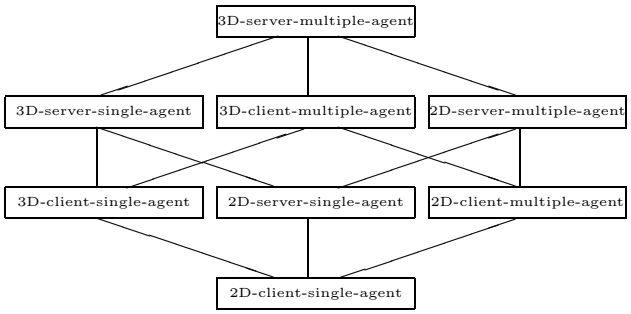


Figure 1: Lattice of Web Agents

There are different types of web agents based on these three dimensions, which consists of a complexity lattice of web agent types as outlined in Figure 1. 3D-server-multiple-agents are at the top of the complexity hierarchy, whereas 2D-client-single-agents are at the bottom. These dimensions are not exhaustive. In particular, we do not consider the dimension "stationariness versus mobility". However, the combinations based on the dimension of "stationariness versus mobility" agents can be generalized accordingly, based on the current taxonomy. All the dimensions we consider for the taxonomy are directly web-dependent.

The dimension "2D-3D" specifies the internet protocols agents have to be aware of, the dimension "client-server" is concerned with internet service modes agents have to offer, and the dimension "singularity-multiplicity" determines the agent communication language. We do not consider the functional dimension which is not directly relevant for the Web, like cooperating agents, problem solving agents, and negotiation agents. We also do not discuss the dimensions which are domain dependent, like expertise seeking agents and e-commerce agents. However, our taxonomy does cover most types of these agents. They are either a 2D agent or 3D agent, either a client agent or server agent, either single agent or multiplicity agent or some of their combinations. Different types of web agents suggest different types of interaction modes with users and web servers. See [7] for details.

2.2 3D Web Agents

In this paper, we focus on the discussion of 3D web agents. The following examples describe typical 3D web agents.

- **3D-Server-Multiple Web Agent**.
  The configuration of 3D-server-multiple-agent on the Web is shown in Figure 2. The agents are part of virtual reality servers, more exactly, virtual community servers, for they interact with multiple users and agents. Users communicate with virtual reality servers via web browsers. The agents interact with each other in the virtual reality server and communicate with web users. As agents in virtual reality, they are usually embodied by their avatars.

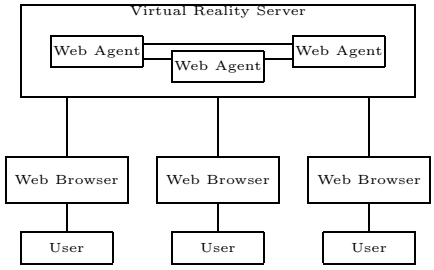


Figure 2: 3D-server-multiple web agent

These kinds of web agents have a great deal of application potentials, which range from avatar-embodied guides that help visitors in information retrieval tasks and navigation in virtual environments, e-commerce shopping assistant agents in 3D-virtual shops, to intelligent agents in multiple player computer games.

- **3D-server-single-agents**.
  3D-server-single agents and their relationship with servers and users are similar to their counterparts of multiple agents, where the single agent serves as the front-end of the servers. Users do not directly communicate with servers, but with an intelligent web agent which is located at the server side. In other words, the agent may be regarded as a portal through which the server is accessed.

  This kind of web agent can intelligently offer or create personalized virtual environments for users, based on users' queries or requests.

- **3D-Client-Multiple-Agent**.
  The relation between 3D-client-multiple-agent, virtual reality server, and web users is shown in Figure 3. The agents are located at the client side, usually serve as users' personal information assistants. Virtual reality

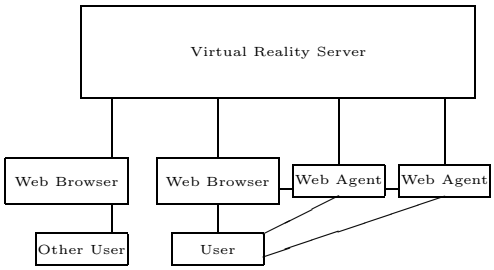


Figure 3: 3D-client-multiple-agent

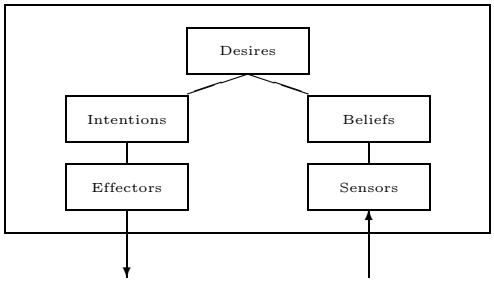


Figure 4: Extended BDI architecture of web agents

servers may have their own web agents. Web users communicate with servers or other users (or agents) either via web browsers or directly via web agents. Other users may also have their own web agents (which are omitted in the figure for reasons of simplicity). This type of web agent is particularly useful in the 3D chat arena, in which the agents can serve as an intelligent personal assistant to help users with information gathering, and interface with other intelligent web agents.

2.3 Architecture of 3D Web Agents

A lot of intelligent agent architectures have been proposed in the agent communities. The most popular architecture is the so-called BDI architecture [13], which consists of three mental attitude components: Beliefs, Desires, and Intentions. In [8], we propose an extended BDI architecture for web agents, namely, a BDI architecture extended with a set of sensors and effectors, which is shown in Figure 4.

The architecture is general, so it covers 2D web agents, that provide a text-based interface to for example information retrieval services, as well as 3D web agents like avatar-embodied guides that help visitors to navigate in virtual environments. In this paper, we focus on the architecture of 3D web agents, in particular, the components of the interfaces with 3D virtual worlds, namely the functionalities of the sensors and effectors of web agents.

3D web agents under the extended BDI architecture are supposed to interface with 3D virtual worlds which are built in the Virtual Reality Modeling Language VRML [11]. VRML is the most popular language in networked virtual environments.

In this paper, we do not describe a complete set of sensors and effectors, for more and more sensors and effectors are added to the architecture, with the development of the 3D web agents. The core primitives of sensor predicates for 3D web agents are :

- $getViewpointPosition(Agent, X, Y, Z)$: get the agent's viewpoint position;
- $getViewpointOrientation(Agent, X, Y, Z, R)$: get the agent's viewpoint orientation;
- $getPosition(Object, X, Y, Z)$: get the object's position;
- $getRotation(Object, X, Y, Z, R)$: get the object's rotation.

The core effector primitives for 3D web agents are:

- $loadURL(URL)$; load a 3D VRML world;
- $setViewpointPosition(Agent, X, Y, Z)$: set agent's viewpoint position;
- $setViewpointOrientation(Agent, X, Y, Z, R)$: set agent's viewpoint orientation;
- $setPosition(Object, X, Y, Z)$: set object's position;
- $setRotation(Object, X, Y, Z, R)$: set object's rotation.

3. DISTRIBUTED LOGIC PROGRAMMING (DLP)

Distributed logic programming [5] combines logic programming, object oriented programming and parallelism. The distinguishing feature of DLP with respect to other proposals is the support for distributed backtracking over the results of a rendezvous between objects. The use of DLP as a language for the implementation of intelligent web agents, is motivated by the following language characteristics:

- **Functionality**. DLP accepts the syntax and semantics of logic programming languages. It is a high-level declarative language, suitable for the construction of software architectures in the domain of artificial intelligence. In particular, it's a flexible language for reasoning and manipulation of knowledge and beliefs. Moreover, DLP incorporates object oriented programming concepts, which make it a useful tool for programming. The object oriented features in DLP are shown in the figure 5.
- **Distribution**. DLP is also a distributed programming language. DLP programs can be executed at different computers in a distributed architecture. Moreover, DLP allows for multiple threads of control in a single program, which makes it a convenient tool for the implementation of reactive agents. Furthermore, DLP

declaration of objects $: -object\ name.$ $: -object\ name : [base].$ $: -object\ name : [base1, base2, ...].$ $: -end\_object\ name.$
declaration non-logical variables (nlv): $var\ i = 0, j = [1, 2, 3], k = f(a, b, c)$
destructive assignment $nlv := Term$
simplification/evaluation: $nlv := Expression$ other nlv occurrences are replaced by their current value.
object creation: $ObjectRef := new(ObjectNameOrConstructor)$
method invocation: $ObjectRef <- method(...)$
synchronuous communication : $accept(AcceptExpression1, AcceptExpression2, ...)$
accept expression: $method(...) <== [Guard] ==> Body$ $method(...) <== [Guard]$ $method(...) ==> Body$ $method(...)$

Figure 5: Features of DLP

programs are compiled to Java classes, which makes it as a powerful tool for the implementation of mobile web agents.

- **Extensibility**. DLP is an extensible language. Special-purpose requirements for particular application domains can easily be integrated in the existing object-oriented language framework. As an illustration, we've extended the run-time system with a 3D VRML interface, which includes all the sensor and effector primitives mentioned above, but also their VRML EAI (external authoring interface) counterparts [12], including operators for manipulating agent and object related attributes. For instance, the predicate

  $$getSFVec3f(Object, Field, X, Y, Z)$$

  gets a value (which consists of three float numbers X, Y, and Z) of the $Field$ of the $Object$, and

  $$setSFVec3f(Object, Field, X, Y, Z)$$

  assigns the $SFVec3f$ value X, Y, and Z to the $Field$ of the $Object$.

4. BENCHMARK EXAMPLE: SOCCER PLAYING

We have selected the soccer game as one of the benchmark examples to test the implementation of 3D web agents and explore the scenarios in which several avatars are playing the soccer game in VRML 3D virtual worlds. These avatars can be supported either by ordinary users, like human beings, or by intelligent agents, which are controlled by active DLP objects. Moreover, we also use DLP programs to control the whole scenario of the game and the formulation of the physics and dynamics of the soccer game, for instance, the behavior of the soccer ball and the score of the game.

A demonstration version of WASP soccer game is now available for download from the WASP web site [15]. The program runs in a Netscape web browser for which a Blaxxun VRML browser has been installed. A screenshot of the soccer playing game is shown in the following figure:

The intelligent agent avatars in the WASP soccer game have the following functions:

- the cooperation of players by reasoning about players' positions and roles;
- cognitive model of the soccer game;
- formulation of the physics of soccer ball;

In this section, we discuss the following issues:

4.1 Soccer agents and their roles

Based on their controlling modes, soccer player avatars are classified in two types: *soccer player* and *soccer player user*. Soccer players are controlled by a software agent, whereas soccer player users are controlled by a human user. Each avatar may play one of the following four roles: *goal keeper*, *defender*, *mid-fielder*, and *forward*. Each role has its own active area in the soccer field.

Each non-human player has the following cognitive loop: sensing–thinking–acting, which is shown in figure 6. By sensing, avatars use their sensors to retrieve the necessary information about the current situation. The main information sources are: agent position, soccer ball position, and the goal gate position. After sensing, the perceived information becomes part of the agent's belief. In the stage of thinking, avatars have to reason about other players' positions and roles, and decide how to react, based on their preferences, which come from the desire component, and the information about the current situation, which comes from the belief component. Thinking results in a set of intentions, more exactly, a set of intended actions. By acting, avatars use their effectors to take the intended actions.

4.2 Cognitive model of the soccer game

In the current version of the soccer game, we do not require that agents know all the rules of the soccer game, like penalty kick, free kick, corner kick, etc [6]. In the simplified soccer game, soccer players have only the following kick actions [14]: kick, pass, interception, and 1-2 pass.

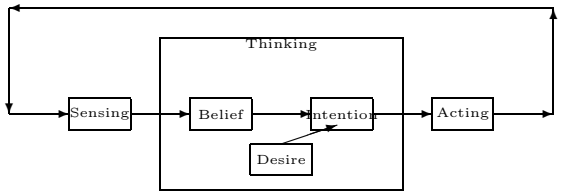
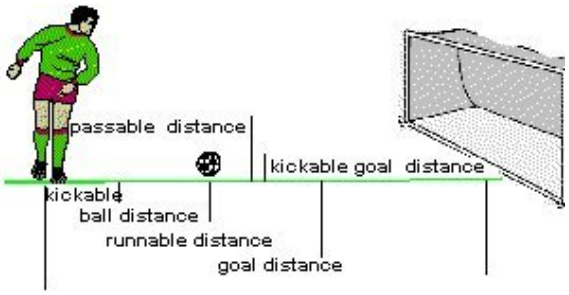


Figure 6: cognitive loop of intelligent avatars

The agents in the WASP soccer game use a simple cognitive model of soccer game, in which the agents consider the information about several critical distances, then make a decision to kick. The considered critical distances are:

ball distance: a distance between ball and the player;
goal distance: a distance between player and the goal gate;
kickable ball distance: a distance the agent can directly kick the ball;
kickable goal distance: a distance the agent can kick the ball to the goal;
runnable distance: a distance the agent can run to the ball;
passable distance: a distance the agent can pass the ball to a team-mate.



In the *desire* component, players have the desire to gain more scores, namely, kick more goals by themselves, or by team-mates. Furthermore, each player prefers shooting by themself to passing the ball to a team-mate. Despite this simplified "cognitive" soccer game model, each player shows a remarkably, intelligent behavior.

4.3 Formalizing physics of soccer ball

In virtual worlds, we consider a three dimensional coordinate system, in which each point is represented by a vector, like $\langle x, y, z\rangle$. Suppose that a soccer ball is kicked from an initial point $\langle x_0, y_0, z_0\rangle$ with initial velocity $v = \langle v_x, v_y, v_z\rangle$ meter/second.

The acceleration due to gravity is in the negative y-direction and there is no acceleration in the x-direction and z-direction.

```
kickedwithStaticStart(Ball, X0, Y0, Z0, Vx, Vy, Vz,
    UpdateDelay) : −
        T_total is Vy/4.9,
        steps := 1,
        repeat,
            delay(UpdateDelay),
            T' is steps ∗ UpdateDelay,
            X' is X0 + Vx ∗ T',
            Y' is Y0 + Vy ∗ T' − 4.9 ∗ T' ∗ T',
            Z' is Z0 + Vz ∗ T',
            setPosition(Ball, X', Y', Z'),
            ++ steps,
        steps > T_total//UpdateDelay,
        !.
```

Figure 7: Soccer ball behavior with static start

```
: −object soccer_game : [bcilib].

var url =' soccer5.wrl'.

var timelimit = 3000.

main : −
        loadURL(url),
        Ball := new(ball('ball', ballposition,
                timelimit)),
        GoalKeeper1 := new(goalKeeper(goalKeeper1,
                timelimit)),
        GoalKeeper2 := new(goalKeeper(goalKeeper2,
                timelimit)),
        UserMe := new(soccerPlayerUser(me_red10,
                timelimit)),
        Blue9 := new(soccerPlayer(blue9,
                timelimit)),
        Blue8 := new(soccerPlayer(blue8,
                timelimit)),


: −end_object soccer_game.
```

Figure 8: Starting soccer game in DLP

We have the following equations:

$$(1)\quad x = x_0 + v_x * t$$
$$(2)\quad y = y_0 + v_y * t - 1/2 * g * t^2$$
$$(3)\quad z = z_0 + v_z * t$$

where t is the time parameter, and g is the acceleration of a body dropped near the surface of the Earth. We take $g = 9.8$. Suppose that the soccer ball is kicked from the ground with a static start. The total time T_{total} of the soccer ball taken from being kicked to falling back on the ground can be calculated from the equation (2), by letting $y = 0$ and $y_0 = 0$. Thus, $T_{total} = v_y/4.9$. The behavior of the soccer ball kicked with static start can be expressed in DLP as shown in Figure 7.

4.4 Multiple threads of control

DLP allows to create multiple threads to control the games, which makes it relatively easy to model the behavior of multiple players. The starting scenario, which is described in

```
look_at_ball(Player, Ball) : −
        getSFVec3f(Player, position, X, _, Z),
        getPosition(Ball, X1, _, Z1),
        X ≠ X1,
        !,
        Xdif is X − X1,
        Zdif is Z1 − Z,
        R is atan(Zdif/Xdif) − sign(X − X1) ∗ 1.57,
        setRotation(Player, 0.0, 1.0, 0.0, R).
```

Figure 9: Looking at the ball in DLP

```
eyebrow_up(Object, Range) : −
        getPosition(Object, X, Y, Z),
        Y1 is Y + Range,
        setPosition(Object, X, Y1, Z).

eyebrowUp_and_lipMove(Times, Interval, Range) : −
        getPosition(r_eyebrow, X1, Y1, Z1),
        getPosition(l_eyebrow, X2, Y2, Z2),
        eyebrow_up(r_eyebrow, 0.01),
        eyebrow_up(l_eyebrow, 0.01),
        lip_move(lower_lip, Times, Interval, Range),
        setPosition(r_eyebrow, X1, Y1, Z1),
        setPosition(l_eyebrow, X2, Y2, Z2).
```

Figure 10: Facial Animation in DLP

DLP, is shown in Figure 8.

4.5 Extensions of sensors/effectors

DLP has been extended with a set of core sensor/effector primitives for 3D web agents. In addition, these sensor/effector primitives can be extended to higher sensor/effector abstraction levels. For example, if we want soccer player avatars to look at the soccer ball, we can define the operator 'look-at-ball' in terms of the core sensor/effector primitives and other available operators in DLP, as shown in Figure 9.

The intelligent behaviors of game playing avatars heavily depend on the situation knowledge of the agent avatars. The most useful information for determining playing strategies is to reason about the position of other players (team-mates, or opponent players) and the position of the soccer ball. Logic programming languages offer a convenient way for knowledge representation and reasoning; they have shown to be an effective high-level tool for the programmability of intelligent agent avatars.

4.6 Gestures and Facial Animation

WASP soccer avatars have several built-in gestures, like ball-kicking and ball-holding. These gestures are designed with VRML's animation controlling machinery, the ROUTE semantics[11]. Avatars can be controlled by DLP programs to decide when these gestures are made. Moreover, DLP programs can also be used to create gestures and animations on avatars. The avatars are HANIM 1.1 compliant humanoids [9]. Figure 10 is an example of a simplified DLP program which controls the facial animation of embodied agents for non-verbal communicative acts, like looking certain with eyebrow up, as described in [10].

5. CONCLUSIONS

We have proposed an approach for the programmability of intelligent agent avatars, which are supported by the distributed logic programming language DLP. We selected soccer player avatars as one of the benchmark examples of 3D web agents. Although the WASP soccer game is still under development, our experiments have shown that the technologies of intelligent web agents and distributed logic programming are convenient and powerful tools for the implementation of distributed intelligent agent avatars.

6. REFERENCES

[1] ActiveWorlds, `http://www.activeworlds.com`.

[2] Blaxxun Interactive Inc. `http://www.blaxxun.com`.

[3] Wolfgang Broll, Eckhard Meier, Thomas Schardt, *Symbolic Avatars Acting in Shared Virtual Environments*, `http://orgwis.gmd.de/projects/VR`, 2000.

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

[5] Anton Eliëns, *DLP, A language for distributed logic programming*, Wiley, 1992.

[6] FIFA, Laws of soccer games, `http://www.fifa.com`, 2001.

[7] 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, pp. 765–769, 2000.

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

[9] Humanoid Animation Working Group, `http://www.hanim.org/`, 2001.

[10] I. Poggi, C. Pelachaud, and F. De Rosis, Eye communication in a conversational 3D synthetic agent, *AI Communications*, IOS Press, 2000.

[11] ISO, *VRML97: The Virtual Reality Modeling Language, Part 1: Functional specification and UTF-8 encoding*, ISO/IEC 14772-1, 1997.

[12] ISO, *VRML97: The Virtual Reality Modeling Language, Part 2: External authoring interface*, ISO/IEC 14772-2, 1997.

[13] A. Rao, and M. Georgeff, Modeling Rational Agents within a BDI-Architecture, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, 473-484, Morgan Kaufmann Publishers, 1991.

[14] Tomoichi Takahashi, *LogMonitor: from player's action analysis to collaboration analysis and advice on formation*, Robocup 99, 1999.

[15] WASP project home page: `http://www.cs.vu.nl/~huang/wasp`.

[16] Watson, M., *AI Agents in Virtual Reality Worlds – programming intelligent VR in C++*, Wiley, 1996.