1

a platform for intelligent multimedia

We have developed a platform for *intelligent multimedia*, based on distributed logic programming (DLP) and X3D/VRML. See [Platform]. Now, before giving a more detailed description of the platform, let's try to provide a tentative definition of *intelligent multimedia*.

intelligent multimedia

> ... intelligent multimedia *provides a merge between technology from AI, in particular agent-technology, and multimedia ...*

However shallow this definition might be, it does indicate that we are in a multidisciplinary field of research that investigates how we may approach multimedia in a novel manner, using knowledge technology developed in Artificial Intelligence. More pragmatically, *intelligent multimedia* characterizes a programmatic approach to multimedia making use of high-level declarative languages, in opposition to low-level third generation and scripting languages, to reduce the programming effort involved in developing (intelligent) multimedia systems. Does this make the application themselves more intelligent? Not necessarily. In effect, nothing can be done that could not have been done using the available programmatic interfaces. However, we may argue that the availability of a suitable programming model makes the task (somewhat or significantly) easier.

In our Multimedia Authoring II course, students become familiar with our *intelligent multimedia* technology.

Multimedia Authoring II – virtual environments

- *intelligent services in virtual environments*

Knowledge of Web3D/VRML, as taught in Multimedia Authoring I, is a prerequisite. The course gives a brief introduction to logic programming in Prolog and DLP and then continues with building virtual environments using agent-technology to control the dynamic aspects of these environments.

distributed logic programming

The language DLP has a respectable history. It was developed at the end of the 1980s, [DLP], and was implemented on top of Java at the end of the 1990s. In retrospect, the language turned out to be an agent-programming language *avant la lettre*. What does it offer? In summary:

DLP

- *extension of Prolog*
- *(distributed) objects*
- *non-logical instance variables*
- *multiple inheritance*
- *multi-threaded objects*
- *communication by rendez-vous*
- *(synchronization) accept statements*
- *distributed backtracking*

Basically, the language is a distributed object-oriented extension of Prolog. It supports multiple inheritance, non-logical instance variables and multi-threaded objects (to allow for distributed backtracking). Object methods are collections of clauses. Method invocation is dealt with as communication by rendez-vous, for which synchronization conditions may be specified in so-called *accept* statements. As indicated above, the current implementation of DLP is built on top of Java. See [OO], appendix E for more details.

DLP+X3D platform

Our platform is the result of merging VRML with the distributed logic programming language DLP, using the VRML External Authoring Interface. This approach allows for a clear separation of concerns, modeling 3D content on the one hand and determining the dynamic behavior on the other hand. As a remark, recently we have adopted X3D as our 3D format. The VRML profile of X3D is an XML encoding of VRML97.

To effect an interaction between the 3D content and the behavioral component written in DLP, we need to deal with two issues:

- control points: *get/set* – position, rotation, viewpoint
- *event-handling* – asynchronous accept

We will explain each of these issues separately below. In addition, we will indicate how multi-user environments may be realized with our technology.

control points The control points are actually nodes in the VRML scenegraph that act as handles which may be used to manipulate the scenegraph. In effect, these handles are exactly the nodes that may act as the source or target of event-routing in the 3D scene. As an example, look at the code fragment below, which gives a DLP rule to determine whether a soccer player must shoot:

```
findHowToReact(Agent,Ball,Goal,shooting) :-
   get(Agent,position,sfvec3f(X,Y,Z)),
   get(Ball,position,sfvec3f(Xb,Yb,Zb)),
   get(Goal,position,sfvec3f(Xg,Yg,Zg)),
   distance(sfvec3f(X,Y,Z),sfvec3f(Xb,Yb,Zb),DistB),
   distance(sfvec3f(X,Y,Z),sfvec3f(Xg,Yg,Zg),DistG),
   DistB =< kickableDistance,
   DistG =< kickableGoalDistance.
```

This rule will only succeed when the actual distance of the player to the goal and to the ball satisfies particular conditions, see section **??**. In addition to observing the state of the 3D scene using the *get* predicate, changes to the scene may be effected using the *set* predicate.

event handling Our approach also allows for changes in the scene that are not a direct result of setting attributes from the logic component. Therefore we need some way to intercept events. In the example below, we have specified an observer object that has knowledge of, that is inherits from, an object that contains particular actions.

```
:- object observer : [actions].
var slide = anonymous, level = 0, projector = nil.

observer(X) :-
   projector := X,
   repeat,
     accept( id, level, update, touched),
   fail.

id(V) :-  slide := V.
level(V) :- level := V.
touched(V) :- projector←touched(V).
update(V) :- act(V,slide,level).
:- end_object observer.
```

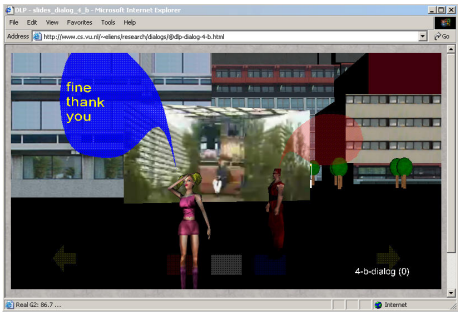
The constructor sets the non-logical variable *projector* and enters a repeat loop to accept any of the incoming events for respectively *id*, *level*, *update* and *touched*. Each event has a value, that is available as a parameter when the corresponding method is called on the acceptance of the event. To receive events, the *observer* object must be installed as the listener for these particular events.

The events come from the 3D scene. For example, the *touched* event results from mouse clicks on a particular object in the scene. On accepting an event, the corresponding method or clause is activated, resulting in either changing the value of a non-logical instance variable, invoking a method, or delegating the call to another object.

An observer of this kind is used in the system described below, to start a comment (dialog) on the occurrence of a particular slide.

case studies

To illustrate the potential of our DLP+X3D platform, we will briefly sketch two additional case studies deploying embodied agents, respectively the use of dialogs in VR presentations (fig. a), and a scripting language for specifying gestures and movements for humanoids (fig. b).



(a) *dialog in context*

dialogs in virtual environments

Desktop VR is an excellent medium for presenting information, for example in class, in particular when rich media or 3D content is involved. At VU, I have been using *presentational VR* for quite some time, and recently I have included dialogs using balloons (and possibly avatars) to display the text commenting on a particular presentation. See figure (b) for an example displaying a virtual environment of the VU, a propaganda movie for attracting students, and two avatars commenting on the scene. The avatars and their text are programmed as annotations to a particular scene as described below.

Each presentation is organized as a sequence of slides, and dependent on the slides (or level within the slide) a dialog may be selected and displayed. See the *observer* fragment presented above.

Our annotation for dialog text in slides looks as follows:

```
<phrase right="how~are~you">
<phrase left="fine~thank~you"/>
<phrase right="what do~you think~of studying ..."/>
...
<phrase left="So,~what~are you?"/>
<phrase right="an ~agent" style="[a(e)=1]"/>
<phrase left="I always~wanted to be~an agent" style="[a(e)=1]"/>
```

In figure (b), you see the left avatar (named *cutie*) step forward and deliver her phrase. This dialog continues until *cutie* remarks that she *always wanted to be an agent*. The dialog is a somewhat ironic comment on the contents of the movie displayed, which is meant to introduce the VU to potential students.[1]

Furthermore, there are a number of style parameters to be dealt with to decide for example whether the avatars or persona are visible, where to place the dialogs balloons on the display, as well as the color and transparancy of the balloons. To this end, we have included a *style* attribute in the *phrase* tag, to allow for setting any of the style parameters.

Apart from phrases, we also allow for gestures, taken from the built-in repertoire of the avatars. Below we discuss how to extend the repertoire of gestures, using a gesture specification language.

Both phrases and gestures are compiled into DLP code and loaded when the annotated version of the presentation VR is started.

STEP – a scripting language for embodied agents

Given the use of humanoid avatars to comment on the contents of a presentation, we may wish to enrich the repertoire of gestures and movements to be able, for example, to include gestural comments or even instructions by gestures.

Recently, we have started working on a scripting language for humanoids based on dynamic logic. The STEP scripting language consists of basic actions, composite operators and interaction operators (to deal with the environment in which the movements and actions take place).

The basic actions of STEP consist of:

- *move* – `move(Agent,BodyPart,Direction,Duration)`
- *turn* – `turn(Agent,BodyPart,Direction,Duration)`

These basic actions are translated into operations on the control points as specified by the H-Anim 1.1 standard.

As composite operators we provide sequential and parallel composition, as well as *choice* and *repeat*. These composite operators take both basic actions and user-defined actions as parameters.

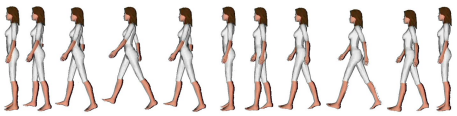
Each action is defined using the *script*, by specifying an action list containing the (possibly compound) actions of which that particular action consists. As an example, look at the definition of *walking* below.

```
script(walk(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),
```

[1] Clearly, our approach is reminiscent to the notorious *Agneta & Frida* characters developed in the Persona project. See the *research directions* of section **??**.

```
            turn(Agent,r_shoulder,front_down2,fast),
            turn(Agent,r_hip,back_down2,fast)])
    ], !.
```

Notice that the *Agent* that is to perform the movement is given as a parameter. (Identifiers starting with a capital act as a logical parameter or variable in Prolog and DLP.)



(b) *walking humanoid*

Interaction operators are needed to conditionally perform actions or to effect changes within the environment by executing some command. Our interaction operators include: *test*, *execution*, *conditional* and *until*.

Potentially, an action may result in many parallel activities. To control the number of threads used for an action, we have created a scheduler that assigns activities to a thread from a thread pool consisting of a fixed number of threads.

As a demonstrator for STEP, we have created an instructional VR for *Tai Chi*, the Chinese art of movement.

XML encoding Since we do not wish to force the average user to learn DLP to be able to define scripts in STEP, we are also developing XSTEP, an XML encoding for STEP. We use *seq* and *par* tags as found in SMIL, as well as *gesture* tags with appropriate attributes for speed, direction and body parts involved. As an example, look at the XSTEP specification of the *walk* action.

```
<action type="walk(Agent)">
<seq>
 <par speed="fast">
 <gesture type="turn" actor="Agent" part="r_shoulder" dir="back_down2"/>
  ...
 </par>
 <par speed="fast">
  ...
 <gesture type="turn" actor="Agent" part="r_hip" dir="back_down2"/>
 </par>
</seq>
</action>
```

Similar as with the specification of dialog phrases, such a specification is translated into the corresponding DLP code, which is loaded with the scene it belongs to. For XSTEP we have developed an XSLT stylesheet, using the Saxon package, that transforms an XSTEP specification into DLP. We plan to incorporate XML-processing capabilities in DLP, so that such specifications can be loaded dynamically.

related work

There is an enormous amount of research dealing with virtual environments that are in one way or another inhabited by embodied agents. By way of comparison, we will discuss a limited number of related research projects.

As systems that have a comparable scope we may mention [Environments] and DIVE, that both have a client-server architecture for realizing virtual environments. Our DLP+X3D platform distinguishes itself from these by providing a uniform programmatic interface, uniform in the sense of being based on DLP throughout.

The Parlevink group at the Dutch University of Twente has done active research in applications of virtual environments with agents. Their focus is, however, more on language processing, whereas our focus may be characterized as providing innovative technology.

Both [Jinni] and [Scripts] deal with incorporating logic programming within VRML-based scenes, the former using the External Authoring Interface, and the latter inline logic scripts. Whereas our platform is based on distributed objects, Jinni deploys a distributed blackboard to effect multi-user synchronisation.

Our scripting language may be compared to the scripting facilities offered by Alice, which are built on top of Python. Also, *Signing Avatar* has a powerful scripting language. However, we wish to state that our scripting language is based on dynamic logic, and has powerful abstraction capabilities and support for parallelism.

Finally, we seem to share a number of interests with the VHML community, which is developing a suite of markup languages for expressing humanoid behavior. 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.

future research

In summary, we may state that our DLP+X3D platform is a powerful, flexible and high-level platform for developing VR applications with embodied agents. It offers a clean separation of modeling and programming concerns. On the negative side, we should mention that this separation may also make development more complex and, of course, that there is a (small) performance penalty due to the overhead incurred by using the External Authoring Interface.

Where our system is currently lacking, clearly, is adequate computational models underlying humanoid behavior, including gestures, speech and emotive characteristics. The VHML effort seems to have a rich offering that we need to digest in order to improve our system in this respect.

Our choice to adopt open standards, such as XML-based X3D, seems to be benificial, in that it allows us to profit from the work that is being done in other communities, so that we can enrich our platform with the functionality needed to create convincing embodied agents in a meaningful context.