



DELIVERABLE D7.3

## Virtual Reality presentation demo: Replay of interpreted activity

October 11, 2004

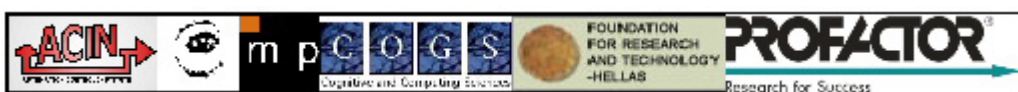
Authors: Vladimír Štěpán, Jiri Zára, Václav Hlaváč  
Czech Technical University, Faculty of Electrical Engineering  
Department of Cybernetics, Center for Machine Perception  
121 35 Prague 2, Karlovo náměstí 13, Czech Republic  
{stepanv,zara,hlavac}@fel.cvut.cz, <http://cmp.felk.cvut.cz>

Project acronym: **ACTIPRET**

Project full title: **Interpreting and Understanding Activities of  
Expert Operators for Teaching and Education**

Action Line IV.2.1: **Real Time Distributed Systems (Cognitive Vision)**

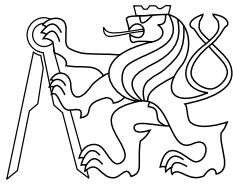
Contract Number: **IST-2001-32184**







CENTER FOR  
MACHINE PERCEPTION



CZECH TECHNICAL  
UNIVERSITY

RESEARCH REPORT

ISSN 1213-2365

# Virtual Reality presentation demo: Replay of interpreted activity

Deliverable 7.3  
of the project IST-2001-32184 ActIPret

Vladimír Štěpán, Jiří Žára, Václav Hlaváč

{stepanv,zara,hlavac}@fel.cvut.cz

CTU-CMP-2004-12

October 11, 2004

Available at  
<ftp://cmp.felk.cvut.cz/pub/cmp/articles/stepan/Stepan-TR-2004-12.pdf>

Authors were supported by the project IST-2001-32184 ActIPret,  
by the project GAČR 102/03/0440, and by the Czech Ministry of  
Education under the project MSM 212300013.

**Research Reports of CMP, Czech Technical University in Prague, No. 12, 2004**

Published by

Center for Machine Perception, Department of Cybernetics  
Faculty of Electrical Engineering, Czech Technical University  
Technická 2, 166 27 Prague 6, Czech Republic  
fax +420 2 2435 7385, phone +420 2 2435 7637, www: <http://cmp.felk.cvut.cz>



## Abstract

The task of ActIPret VR presentation module is to visualize the generalized concepts of the observed human activity. We discuss the problem of visualization at this level of abstraction and suggest two possible approaches. The core of this report is the description of implemented solution including the user instructions.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Visualization problems</b>	<b>2</b>
2.1	The Scene . . . . .	3
2.2	Animation . . . . .	4
2.3	Interaction . . . . .	4
<b>3</b>	<b>Available Options</b>	<b>4</b>
3.1	Sources of Data . . . . .	5
3.2	Possible Approaches . . . . .	5
<b>4</b>	<b>Implementation Issues</b>	<b>7</b>
4.1	Input Data . . . . .	7
4.1.1	Activity Plan . . . . .	7
4.1.2	ORG Log . . . . .	8
4.1.3	SDF . . . . .	9
4.2	Scene Reconstruction . . . . .	12
4.2.1	VRML Objects . . . . .	12
4.2.2	Humanoid . . . . .	13
4.3	Animation And Interaction . . . . .	13
4.3.1	Inverse kinematics . . . . .	14
4.3.2	Event Model File . . . . .	14
4.3.3	Interaction Events . . . . .	16
4.3.4	Hand Animation . . . . .	16
4.4	Two Handed Scenario . . . . .	17
4.5	User Instructions . . . . .	18
4.5.1	Running the VR module . . . . .	19
4.5.2	Scenarios And Directories . . . . .	21
4.5.3	SDF Editor . . . . .	22
<b>5</b>	<b>Conclusions</b>	<b>23</b>

# 1 Introduction

The task of ActIPret VR presentation module is to visualize the results of ActIPret framework in virtual reality (VR). This was not a simple task due to a difference of information useful to VR and to ActIPret framework. Looking at the video sequence of a human operator performing an activity, both VR and ActIPret framework focus on different things. Using ActIPret framework results as an input means visualizing of generalized information, which seems to contradict, as any explanatory visualization serves the purpose of lowering the problem's level of abstraction.

We have suggested two approaches to this problem and we created the ActIPret VR module as the implementation of one of them. These approaches differ in level of abstraction, we call them 'general' and 'instance based' approach. Neither of them is perfect and clear of problems such as collisions. Considering the available resources we have chosen the 'instance based' approach that uses more of the actually observed information.

## Structure of the report

The report is structured as follows: In the Section 2 we discuss the problem of visualization of generalized data, such as ActIPret output. Next, in Section 3, we mention the possible solutions to our particular task. The main part of this report is the Section 4, which brings the description of the solution implemented as ActIPret VR presentation module.

## 2 Visualization problems

The task of the VR presentation module was "to visualize the project's output in VR". The word *visualization* usually means the graphical explanation of information. The project's output is the activity plan. The activity plan describes the semantics of the activity. It is very abstract and basically comprises of the sequence of events. Example of the 'insert CD scenario' activity plan:

```
Hand pressed button openButton
Hand picked up object cd
Hand put down object cd on object Player
Hand pressed button openButton
```

Obviously this does not provide enough information for *visualizing* the activity in VR. We can assume there were some objects in the scene. We

know, there were some interactions among these objects. Let's assume we are an educated human, who knows what is it to press button or pick up something and knows what impact these events have on the objects involved. Then this activity plan can give us pretty good idea what is going on in the scenario.

If the activity plan alone can give us an idea of the activity, it does not allow us to visualize the scenario. The purpose of every visualization is to help our mind in understanding, which mostly mean decreasing the level of abstraction. If we imagine (visualize in our mind) the activity, we create imaginary objects, imaginary operator and we let the object move or be moved between imaginary locations. The locations of these objects are chosen with respect to our experience with handling real objects, for instance all is usually within operator's comfortable reach. We are using lot of knowledge in this visualization process.

The VR presentation module needs this knowledge too, but on the contrary to us it does not have it. The activity plan is too abstract to be visualized reasonably clearly. We need to backtrack the way to the level of abstraction of an activity plan. The activity represented by activity plan needs to be instantiated somehow.

## 2.1 The Scene

There are objects referenced in the activity plan. The instantiation of activity should include localization of these objects. The knowledge of their initial placement allows us to create the scene. This scene would be rather less realistic, as there appears to be objects of two categories in the scene. There are objects relevant to the activity and therefore detected by the system. The other category would cover all the objects that are present, but are not subject of system's attention. Only the object listed in an activity plan and therefore relevant to the activity will be present in the scene.

This can result into a conflict with the explanatory purpose of visualization. People are used to some form of incidence among the objects in surrounding world, that would most likely be missing in such scene. For explanation the CD should not hang in midair, it should be supported by the table instead. Thus the presence of the other kind of objects in the scene might be necessary for the sufficient level of realism.

We should also mention the fact, that the activity plan gives us absolutely no information on the actual appearance of the objects. To visualize an activity plan it would be necessary to use the set of predefined models of all objects playing functional or visual role in the scenario. At this point we create the group of scenario specific data.

## 2.2 Animation

After the scene is ready we can make it move. As we have already pointed out animating the scene is a task nearly equal to the task of animating the virtual human.

The activity plan provides little information usable for animating the virtual human. In the previous section we reduced the level of abstraction of activity plan with known initial object location. Can this information help us animate the scene? We can presume that if the hand interacts with an object, as is suggested by activity plan, it is located somewhere near the object. The expression "somewhere near" is just as vague as the actual information on the location the activity plan gives us. What exactly is this "somewhere near" depends on the type of object and the type of event.

We have to know more about the activity, the position of the hand in the moment when event from activity plan occurred at the least. For more realism and avoidance of possible collisions with objects, the trajectory of the hand during the activity would be useful. Large portion of the animation task, particularly the finger animation, can be performed by connecting gestures associated with the detected events. Since these gestures can be predefined, they belong to the same category as the object models and can be viewed as a scenario specific information.

## 2.3 Interaction

In the real scene the activity is based on interaction of human operator and the objects in the environment. By implementing these interactions in VR we can easily animate the scene just by animating the virtual human. The interactions are the semantics of the scene and as such they are well described by an activity plan.

## 3 Available Options

The VR presentation module has a special position as it is not a part of the project's framework. It is not in the center of project's interest either. Therefore it is quite limited in its possibilities. When designing it, we decided to avoid having a special requirements for input data. The goal was to use the project's output and, if necessary some of the data used by another part of the framework available in some form for use outside of framework. Definitely the VR module has to be scenario independent. As an addition to this, the generality of the input should be as close as possible to the generality of activity plan.

### 3.1 Sources of Data

The sources of data for VR presentation of the activity are very limited within the framework. We have already mentioned the activity plan and demonstrated it on the example (see 2). It describes the semantics of the activity quite well and thus it provides good description of the interactions between human operator and the objects.

One of the lower level modules in the framework, the Object Relation Generator (ORG), generates the log, that basically sums up the results of object detection and recognition and tracking operator's hand. The list of objects along with their positions can be found here for the scene reconstruction task. Also the trajectory of the hand can be obtained here. We can use it to animate our virtual human with some inverse kinematics algorithm.

Neither the activity plan nor ORG log gives us enough information for acceptably realistic VR presentation. We introduce third source of information that we call scenario definition file (SDF). It is the text file that describes the situation in the time when observed activity starts. The SDF is a complete list of all objects in the scene with all the information on them that will be used to accomplish the task of activity reconstruction. This includes relationships between objects and initial positions of objects not detected and listed in ORG log.

As it is suggested in the text above, the VR presentation depends on number of predefined structures. This includes most of the scene with virtual human and some building blocks for hand animation (the gestures).

### 3.2 Possible Approaches

It was agreed there were two possible approaches to VR presentation module, 'general' and 'instance-based'. Obviously they differ in level of abstraction. In practice the difference is in the data used. The 'instance-based' approach works with observed data from the ORG log, most importantly the hand trajectory. On the contrary, 'general' approach should abstract from using the observed data and therefore generates its own hand trajectory.

The '**instance-based**' approach is basically the reconstruction of observed activity in VR. We use detected object positions to place the predefined models into the scene. The models of objects are more than just geometry, as some of them implement certain functionality, for example CD player model for our 'CD scenario' can open and close the tray at the proper event.

The virtual human is animated using the observed hand trajectory. This trajectory has a positive effect as it brings the hand to the appropriate posi-

tion for interaction with the object. Moreover it reduces the risk of collisions with other objects in the scene. This risk is not reduced to zero, because only generic models that might not always match the actual objects in shape, size and proportion are used. The precise scene reconstruction is not the goal of our module, so the models might also be placed with significant error.

The negative effect of using hand trajectory is the need for a time information of the occurrence of interaction events. This information is present in the framework, but does not pass to any output file. The need for this information is in conflict with our rule of avoiding special requirements.

Animating the virtual human with the hand trajectory is a typical task for inverse kinematics algorithm. The fingers can not be animated that way though. The predefined gestures associated with events listed in activity plan can be used. These gestures would be inserted to the animation at the key-frames indicated by the time of event occurrence. Since we have little information on the appearance of particular gestures that occurred in the instance, we can use only the generic gesture for each action. Combination of generic gesture and specific hand trajectory can cause problems, such as collisions between fingers and objects.

The ‘**general**’ approach does not use the observed hand trajectory. It starts with the events listed in activity plan to generate its own trajectory instead. This way we don’t need the time information, because our new animation creates its own timing.

New problems appear though. We don’t know the position of hand relative to the object at the moment of interaction. This position and the posture of the hand depends on the object and event. Therefore we can put together the models of objects with the description of hand postures and positions according to the concept of *smart objects* [1, 2]. Thus the model of object would encapsulate not only the appearance description, but also the hints for virtual human for every possible type of interaction.

The activity plan is the sequence of events. Each event marks the moment when the hand was near some object performing some action. The object and action are indicated in activity plan. With the objects carrying the information about hand’s position and posture, the event can be viewed as a key-frame of the new animation. The posture of the body at the key-frame can be computed by inverse kinematics algorithm and combined with the hand posture. The input for inverse kinematics and the hand posture would be obtained from the object. To utilize all the benefits of this approach, it would be useful to provide also the target orientations of end effector.

Queueing the events as key-frames would not alone yield very realistic output. Each event is associated with action that is typically not sufficiently

described by one key-frame. Let these actions be represented with the sequences of key-frames, much like in the previous (instance based) case. In this case the generic gesture is not combined with the specific trajectory. Thus one source of problems would be avoided.

## 4 Implementation Issues

The ‘**instance-based**’ approach was used to implement the VR presentation module. Although it is not general enough and various problems may occur (collisions), it was possible to implement within our time schedule and the available data.

The ActIPret VR presentation module is Java application connected via External Authoring Interface (EAI) to VRML browser. The browser that has been used is Parallel Graphics Cortona VRML Client (version 4) [7] that also provides the EAI classes. Due to the COM architecture of the Cortona EAI classes, the application is not platform independent and runs on Windows system.

### 4.1 Input Data

The VR module uses three input files to create the VR presentation of observed activity - the *activity plan (AP)*, the *ORG log* and *Scenario Definition File (SDF)*.

As the VR module presents instances of observed activities the three input files are stored in one directory belonging to this instance. The AP and ORG log are generated by framework and thus can be considered instance specific. On the other hand SDF could be one for all instances. We added it to the other two files to allow the user to make changes to make the presentation instance specific too (use of various model sets, etc.).

#### 4.1.1 Activity Plan

Activity plan describes the sequence of events. The file contains two versions - ‘conceptual language version’ and ‘natural language version’. The VR module uses the first to understand the events and create their VRML version and the second to be displayed in text window during the presentation. Example of an Activity Plan for CD scenario would be:

```

=====
Raw Concepts

Line 0: 'PRESSBUTTON, Hand 0, Object ejectButton 3'
Line 1: 'PICKUP, Hand 1, Object cd-intel 1, Location undef'
Line 2: 'PUTDOWN, Hand 1, Object cd-intel 1, Location tray 2'
Line 3: 'PRESSBUTTON, Hand 0, Object ejectButton 3'
=====

Natural Language Version

Using first hand, press the ejectButton.
Using second hand, pick up the CD.
Using second hand, put the CD down on tray.
Using first hand, press the ejectButton.
=====

```

The lines of the "Raw Concepts" are comma separated lists of items that describe the event. First item indicates the type of event, the others are the objects involved in the event (hand, manipulated object and location of event).

Activity Plan as an output of ActIPret framework should be discussed in detail elsewhere.

#### 4.1.2 ORG Log

The ORG log is an output of Object Relation Generator (ORG) module, therefore we will mention it only briefly. It consists of list of following entries.

```

objects: ComponentID#ObjectID#ModelID
appearance event: start-time end-time [pose]

```

Each entry is identified by 3 IDs, first indicates the component that contributed this entry to the ORG, second is the ID of object within that component and third would indicate the recognition model of that object. The names of the VR models are closely related to the ModelID too.

The IDs are followed by one or more 'appearance events' that indicate the position of the object over a time interval.

Example:

```

4#0#cdplayer#
0.000 20.000 [1.29 0.36 0.72 0.00 0.00 0.00]
end_of_object

```

### 4.1.3 SDF

Originally the role of Scenario Definition File (SDF) was to define the assumptions for a scenario and enable rapid move to another scenario. It turned out that such file is vital for VR module and thus we have created our own version specifically for the VR needs after it was decided not to implement the SDF within the framework.

Our SDF is a text file that lists all the objects in the scene. Besides that it has an entry specifying the set of VRML models that should be used to represent these objects in the scene. The object descriptions are encapsulated in tags: `<object>` `</object>`

Between these tags there are object attributes followed by their values. These attributes should provide necessary information that cannot be retrieved from the framework. This includes associating the ActIPret ModelIDs with the VRML model names, moving human parts to H-Anim end-effectors for inverse kinematics (IK) animation, as well as defining relations between objects (subparts) and specifying various model parameters.

List of object attributes (alphabetical order):

- **color** - the value is a color of object written in three float numbers (RGB). In case of objects with models with variable color.
- **end\_effector** - the name of H-Anim Site node within the definition of humanoid - an end-effector for IK animation. The name must be without prefix indicating right or left side.
- **fixed** - no value. The attribute of objects not detected by ActIPret system.
- **ID** - value of this tag is the string that stands for ModelID in ActIPret framework (appears in the ORG log).
- **ik\_base** - the name of H-Anim Joint node within the definition of humanoid - the base of kinematic chain for IK animation. The name must be without prefix indicating right or left side.
- **orientation** - the value is four float numbers - orientation of fixed objects within the scene.
- **part\_of** - if the object is a functional part of another object, this attribute has the same value as ID of this superordinate object.
- **pose** - the value is the filename of VRML file with description of a starting posture of H-Anim humanoid

- **position** - the value is three float numbers - position of fixed objects within the scene.
- **texture** - the filename of the texture file. In case of objects with models with variable texture.
- **VR\_model** - the filename of VRML model of the object (string value).

The set of model is indicated between tags `<model_set>` `</model_set>` and has one attribute **name** that corresponds to the name of directory with the models.

Example SDF for VR presentation of observed CD scenario would be as follows:

```

<model_set>
name profactor
</model_set>

<object>
ID cdplayer
VR_model cdplayer.wrl
</object>

<object>
ID ejectButton
VR_model ejectButton.wrl
part_of cdplayer
</object>

<object>
ID playButton
VR_model playButton.wrl
part_of cdplayer
</object>

<object>
ID stopButton
VR_model stopButton.wrl
part_of cdplayer
</object>

<object>
ID tray
part_of cdplayer

```

</object>

<object>  
ID cd-intel  
VR\_model cd.wrl  
texture intel.gif  
</object>

<object>  
ID Hand  
part\_of human  
end\_effector r\_hand\_tip  
ik\_base r\_acromioclavicular  
</object>

<object>  
ID human  
VR\_model human.wrl  
fixed  
position -1.39 -0.046 1.123  
orientation 0 1 0 3.14159265  
pose starting.wrl  
</object>

<object>  
ID table  
VR\_model table.wrl  
fixed  
position -1.303 0.704 0.458  
</object>

<object>  
ID room  
VR\_model room.wrl  
fixed  
position 0.041 -0.046 2.427  
</object>

<object>  
ID chair  
VR\_model chair.wrl  
fixed  
position -1.415 0.424 1.12

```
orientation 0 1 0 3.14159265
</object>
```

This example shows that objects, that are detected by the system (CD player, buttons, CD) have usually very few attributes. On the other hand the objects not detected (table, chair) must list their positions and orientations.

The hand (ID Hand) might be particularly interesting. It is detected object with no special model (part of human) and it represents single kinematic chain of the human. Each kinematic chain has an end-effector and a base. We will return to this in section 4.4 where we will mention how we would deal with two handed scenarios.

## 4.2 Scene Reconstruction

We use predefined object models. These are inserted into the scene according to the information included in SDF on extracted from ORG log. All the scene reconstruction is based on VRML standard.

The base of this VRML scene is a file that is loaded by the VR module at the start-up. This file contains the script node for connecting animations to the virtual humanoid and the root node of the visual part of the scene. All objects inserted in the scene (including the humanoid) become children of this root node (they are inserted via its ‘addChildren’ event). Besides that the base file contains six light sources and ‘head up display’ construction for displaying textual information (typically sections of activity plan).

### 4.2.1 VRML Objects

As we have already mentioned, models of objects are inserted into the scene. These models are stored in separate files (VRML) that include description of object’s geometry and, in some cases even some functionality. The models are created as VRML prototypes that allows variability of parameters (model of CD can switch between color and texture, each of them is also parameterized).

Another aspect of the prototyping is the possibility to implement special VRML events to trigger the functionality. This is the case of objects that are specified as part of another object (see 4.1.3). The model of CD player can be a good example as it implements VRML input event ‘ejectButton’ to trigger the animation of opening and closing the tray. If the names of these events match the appropriate strings used in activity plan, they can be used to translate the activity plan to VRML description of interaction events.

All the object models implement the fields *translation*, *rotation*, *scale*, *name* and *size*. First three allow the possibility of standard transformations.

Name field can identify the object and size is basically the size of its bounding box. The important difference is that the upper face of this "bounding box" is considered the face of the object that can be used to put something on. It is used when determining the height coordinates of undetected objects lying on top or underneath some known objects.

The only exception are the models for the buttons of the CD player, that only serve the purpose of highlighting the small objects with larger colored spheres.

### 4.2.2 Humanoid

The virtual human that has been used corresponds with the H-Anim 1.1 standard for virtual humanoid [6]. It was downloaded from the VRlab website [8], (namely the personal page of their former researcher Ch. Babski [5]), where it was available for non-commercial use.

The geometry remained unchanged. From the functional point of view we adapted the model to our needs. We will mention in section 4.3.1 the adapted joint constrains. Besides that we added several Site nodes that mark the end-effectors and attachment points for the manipulated object. The end-effectors are in both hands in the metacarpal region (named *r\_hand\_tip* and *l\_hand\_tip*). The attachment points are on the hands too and their location depends on the manipulated object and the type of manipulation (named *r\_hand\_x* and *l\_hand\_x*).

## 4.3 Animation And Interaction

The activity observed by ActIPret system involves human operator performing a manipulation task. Therefore the VR presentation of this activity consists of animation part (motions of the operator) and the interactions (manipulation with the scene). These two parts are closely connected.

The data we can use to show the activity in VR are the trajectory of the hand (from the ORG log) and the activity plan. We can use the hand trajectory to animate the virtual humanoid with the inverse kinematics (IK) algorithm. The point that produces the trajectory is basically a centroid of a skin colored blob detected as hand. The IK algorithm cannot animate the the joints that are below the wrist in the human joint hierarchy.

For finger animation we use the predefined gestures associated with detected actions listed in activity plan. Also the interaction events are associated with these actions and therefore the finger animation is very closely related to resolving the interaction events.

### 4.3.1 Inverse kinematics

The inverse kinematics (IK) algorithm that has been implemented was an iterative numerical method called Cyclic Coordinate Descent (CCD) [4]. The idea of this method is to minimize position and orientation errors by varying one joint variable at a time. Each iteration involves a single traversal of the kinematic chain from the most distal link inward towards the manipulator base. Each joint variable is modified in turn to minimize an objective function.

The CCD is heuristic iterative algorithm with convergence rate that is a little problematic. On the other hand single step of iteration is very simple and the algorithm behaves well in the singular states of kinematic chain. The reason why it was chosen for an ActIPret VR presentation module is, that it tends to use joints close to an end-effector rather than spread the motion evenly among all the joints. For our task of animation operators arm, this can yield more realistic results even without the use of joint constrains.

The only data for IK that we have is the trajectory - points in the space, no orientation. For this reason the error function in our implementation of CCD algorithm is simply the distance between current position of end-effector and its destination.

Each joint of the kinematic chain is constrained with the upper and lower rotation limit and another parameter called "stiffness". Both constrains are introduced by H-Anim standard to increase realism of animation generated by IK algorithms. The values of the limits were adopted with the VRML model of human and in some cases adjusted. The "stiffness" parameter should express the willingness of the joint to move and its value was more problematic. The values we have used in 'CD scenario' appeared as a result of "parameter tuning" process. This joint constrain is currently a subject of our research.

We assume that the operator stays at one place when performing the activity. Since the only source of information on operator's position (position or kinematic chain base) is the SDF, this assumption was necessary to implement the IK animation.

### 4.3.2 Event Model File

Each line of Activity plan stands for an action that occurs in the scene. This action is associated with certain hand gesture and interaction between objects in the scene. To convert the Activity plan line to the hand animation and interaction in the scene, we introduced an Event Model File (EMF) that describes the conversions. For each action that is expected to appear in

the scenario (and in the Activity plan) there must be this Event Model File specified.

The EMF contains three parts that completely describe the event to the VR module. First is the "distance" labelled with `<distance>` tag. It shows how to find event's time of occurrence by finding minimum distance between the two interacting objects. The objects are indicated as indexes of Activity Plan elements. Generally it is the `<distance>` tag followed by the two integer values.

The "hand animation" part in `<animation>` `</animation>` brackets describes the motion of the hand when the event happened. It is the list of relative times and names of files specifying the hang pose.

Finally "VRML event description" between `<event>` `</event>` tags shows how to transform the Activity Plan line to the ActIPret VRML description introduced in [3]. Again the indexes of AP elements are used to indicate the points where that particular appears in VRML description (More in 4.3.3).

An example EMF of the *pick up* event:

```
<distance> 1 2

<animation>
-0.3 neutral
-0.1 reachCD
0 holdCD
</animation>

<event>
target_node 2 parent
event_type MFNode
event_name removeChildren
event_value 2
next_event
target_node human sites hand_x
event_type MFNode
event_name addChildren
event_value 2
</event>
```

The 'conceptual language' version of appropriate Activity Plan line would be:

*PICKUP, Hand 1, Object cd-intel 1, Location undef*

The elements number 1 and 2 of this line are *Hand* and *cd-intel*, objects with these IDs will be used when processing the interaction event.

### 4.3.3 Interaction Events

The VR module, as it is based on VRML standard, uses the VRML prototype to describe an interaction event [3]. The Activity Plan line is simply translated to this form.

The lack of the information on event's occurrence time was the main problem of the interaction events part of the VR module. Each event involves the hand and another object and it most likely occurs when the hand is close to that object. We find the local minima of distance between hand and object as a likely points of event occurrence. We choose those with the smallest distance that maintain the sequence of events.

We can't choose simply the global minimum for each event because there might be two events that involve the same objects (e.g. the *press button* event).

### 4.3.4 Hand Animation

With the VRML interpolator-based animation mechanism, we can easily decompose each gesture into sequence of few postures, that can be used as key-frames.

To describe these postures we have designed a VRML-based data structure.

```
PROTO Pose [  
  exposedField MFString joints []  
  exposedField MFRotation rotations []  
  exposedField SFVec3f position 0 0 0  
  exposedField SFString name ""  
]
```

Field *joints* is a list of joints with non zero rotation, *rotations* a list of rotations that belong to these joints. *Position* indicates the displacement of whole body.

Although this structure has its origin in the need to compose hand animation as a sequence of postures, it has one more use in the VR module. It is used to describe the starting posture of virtual humanoid's body before the IK algorithm is run to create the animation of body.

The Event Model File specifies how the postures are queued. Between the `<animation>` `</animation>` tags, there is a sequence of float-string pairs.

The float value indicates the key-frame of the animation as the portion of the interval between two consecutive event occurrences. The string value is the name of the VRML file with the posture description.

As an example we can show the animation part of Event Model File of the pick-up event:

```
<animation>
-0.3 neutral
-0.1 reachCD
0 holdCD
</animation>
```

If previous event happened at the time 0 and current pick-up event is at the time 1, then at the time 0.7 the hand is in neutral position, at the time 0.9 it reaches for CD (hand is open) and at the time of the pick-up event it holds the CD (fingers grasp it).

The names of posture files should be prefixed with *r\_* or *l\_* for right or left hand performing the action. Thus the EMF entry *neutral* refers to either *r\_neutral.wrl* or *l\_neutral.wrl* file. The application will make the choice based on the knowledge of end-effector that acts in the particular event.

## 4.4 Two Handed Scenario

The VR module can handle two handed scenario under several conditions. The tracked end-effectors must be properly listed in SDF to define the kinematic chains. The virtual humanoid must contain the end-effectors and possibly the manipulation oriented Site nodes. Most importantly no part of one kinematic chains should be influenced by another. They only can have common base.

All joint, end-effector or any other body part names listed in SDF of EMFs are written in "side-independent" form. That means to exclude the *r\_* or *l\_* prefix indicating the right of left side. Each tracked end-effector will be associated with the side by application. It is done by a simple test which one of the two possibilities is closer to the starting point of the trajectory.

After the tracked end-effectors are associated with end-effectors within humanoid model, the IK solver is called for each kinematic chain to animate the corresponding part of virtual humanoid body. At this point we assume that the data will not cause conflict. All effectors should be able to reach their destinations as it happened in the observed reality.

## 4.5 User Instructions

We have described selected issues of VR module's functionality. Let us now explain how to use the VR module.

The program is implemented as Java application. Although it is Java, it is not platform independent. The application is connected to the VRML browser that works only with MS Windows OS (tested on both Windows 2000 and Windows XP). Also the Java classes of the interface between VRML browser and Java application that are provided with the browser are Windows platform dependent.

It is obvious that any machine the VR module might be run on must have appropriate VRML browser installed. This browser is Parallel Graphics Cortona VRML Client version 4 [7].

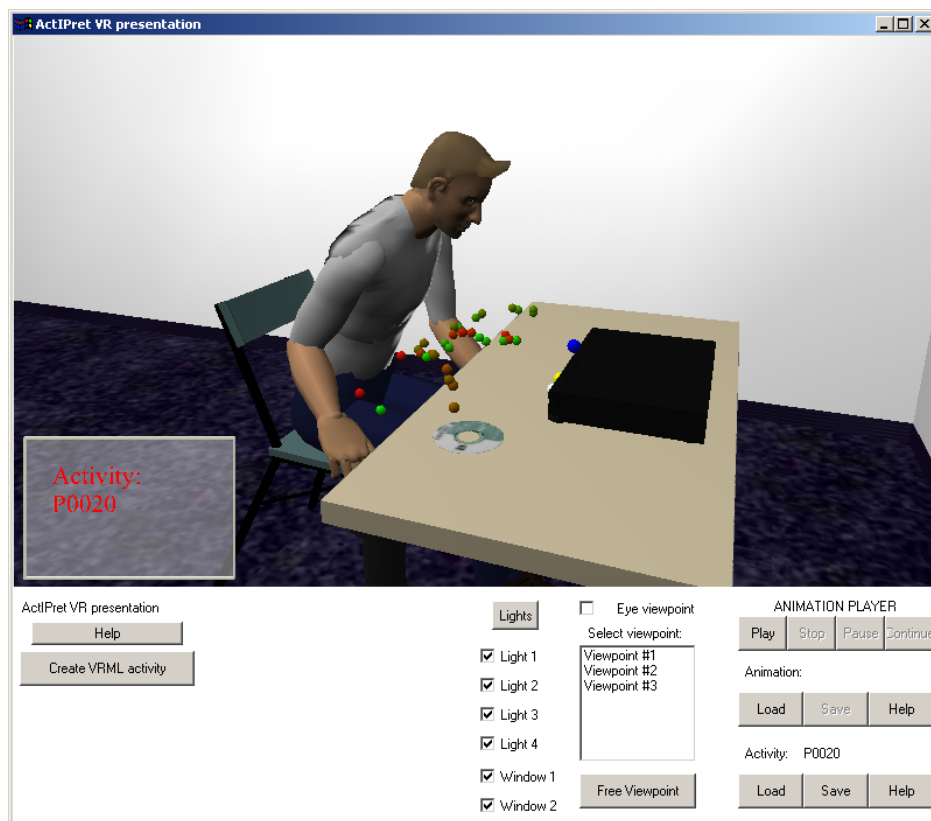


Figure 1: User interface of ActIPret VR presentation module. The VR window already displays reconstructed activity.

### 4.5.1 Running the VR module

To **run** the VR module we use Windows utility `jview.exe` by calling `jview CMain.class` in the VR module root directory. The file **VR\_module\_start.bat** serves this purpose.

A problem with the VRML browser occasionally appears at the start of the program. The application repeatedly fails to retrieve the link to the browser and thus the VR window does not appear and no further work is possible. We recommend to shut the application down and start again. This bug falls on the browser side of the interface.

When we successfully start the program, we can see the user interface shown by Fig. 1. It consists of the large VR window in the upper part and section of control panels below. The VR window contains also the "head-up display" to show textual information (Fig. 2 a).

Control panels are divided into three sections. At the very left part there are controls of the activity reconstruction (Fig. 2 b). These consist of the button to trigger the process and a list box that appears after the button is pushed and offers the selection of scenarios and observed instances of activity. After the scenario and the instance is selected, the reconstruction begins.

The other two sections of control panels belong to the animation player and scene parameters.

The animation player (Fig. 2 c) contains two sets of save/load buttons. That is because this player is intended to be able to view not only special ActIPret activity models, but also simple animations. (The main difference between the two is that the latter does not include any manipulation with objects.) Buttons labelled LOAD open the file with animation / activity models. SAVE buttons enable to write the opened animation / activity model to a VRML file in a format described in [3]. The meaning of all other buttons of the player should be quite clear.

The scene controls (Fig. 2 d) include the switches of all six lights in the scene. Four of them are above the scene and two simulate the windows on one side of the scene. They are inspired by one of the rooms that was modelled for the purposes of the very first ActIPret VR demo.

Besides that scene controls include the viewpoint selection. The *eye-viewpoint* switch provides the view of virtual human eyes, which is not very useful, because we have no information on operator's eye focus and thus we can not animate this viewpoint. Then there is a list of predefined viewpoints that either can be read from the activity VRML description when loading it, or are set as default for the needs of creating new activity model. And finally simple controls of free viewpoint are implemented.

The user of VR module is also supported by simple help (Fig. 3) accessible

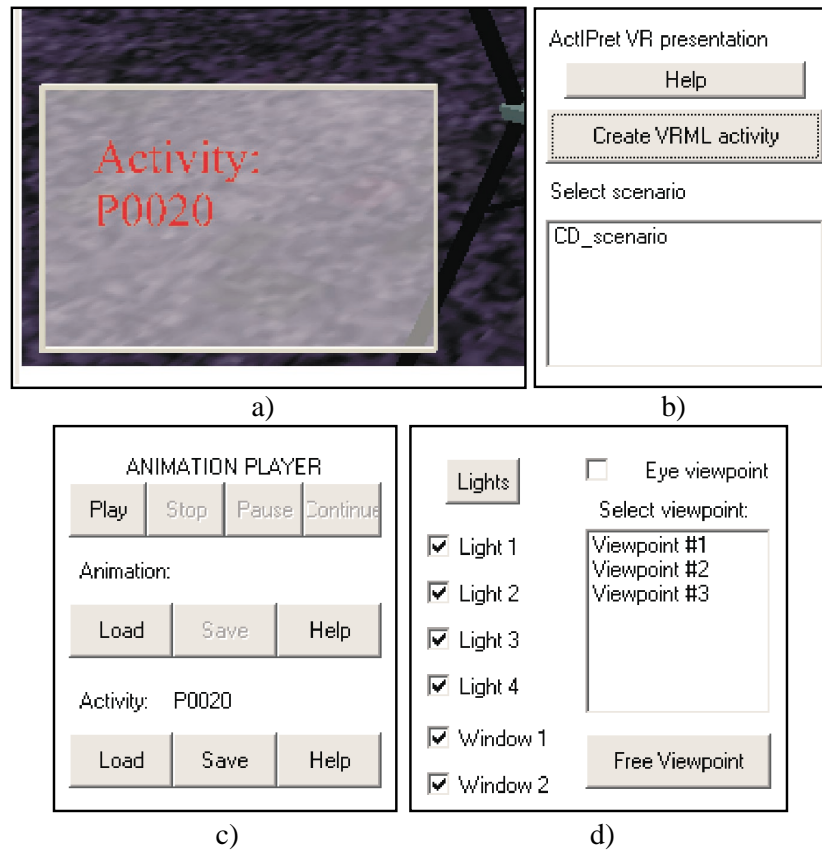


Figure 2: User interface: a) The detail of textual "head-up display". b) Controls of activity reconstruction. c) Controls of animation player. d) Scene lighting and viewpoint controls.

by three buttons, each of them sets different page. Once the help window is opened user can access all three pages though.

At this point we should return to what is depicted on Fig. 1 and describe the reconstructed scene a little closer. The VR window on the image shows the CD scenario scene with the virtual humanoid, the desk, CD player and CD. There are also other objects in the scene, the spheres varying in color with not quite obvious meaning.

Three spheres (blue, white and yellow) on the side of CD player are present as a substitute models to highlight small parts of CD player, the detected buttons (eject, play and stop). The buttons are part of the CD player and therefore modelled only as CD player functions (see 4.2.1) but since it might be useful to make them more visible, they were associated

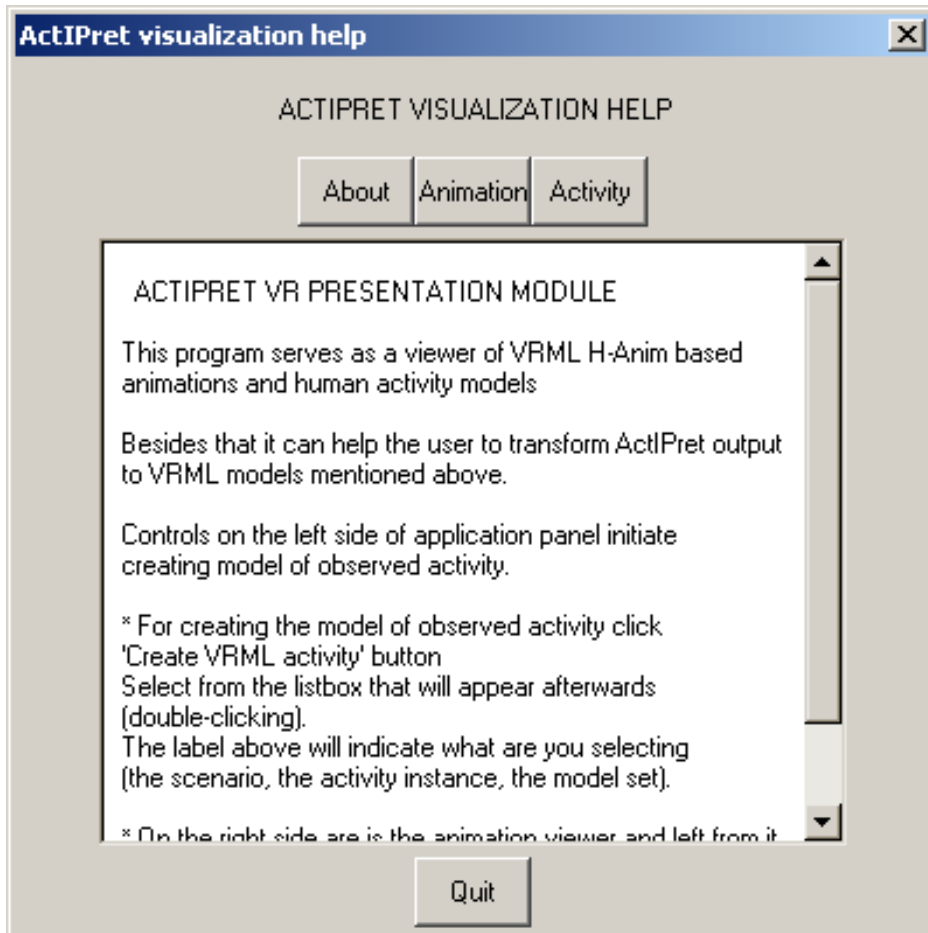


Figure 3: Help window detail.

with these simple spherical constructions in the SDF (see 4.1.3).

When the activity is being reconstructed another set of small spheres is inserted into the scene. These mark the points of the hand trajectory and their color indicates their sequence - red spheres at the beginning fading gradually into green at the end.

#### 4.5.2 Scenarios And Directories

In the root directory of the VR module, there is a directory for each scenario. We have been working with the CD scenario only, but we are fairly confident the VR module would handle any other provided proper data are available. It has not been tested though.

The scenario directory (in our case only *CD\_scenario* exists) contains all

scenario specific data along with all the observed activities. These are stored in four subdirectories: *scene*, *pose*, *events* and *observed*.

The *scene* subdirectory is an envelope for model-set directories with VRML files of object models. We came up with the idea of various model-sets to enable choosing object appearances according to the real setup.

All the VRML files with hand (body) postures are in the *pose* subdirectory. For the CD scenario we use starting pose of body (its use is specified in SDF - see 4.1.3), the neutral, push-button, reach-for-CD and hold-CD hand poses for both right and left hand (use in animation specified in Event Model File - see 4.3.2).

The *events* subdirectory stores the Event Model Files necessary for the scenario. We have these files implemented for push-button, pick-up and put-down events.

Finally the subdirectory named *observed* contains all the data describing the observed activities. Each observed activity subdirectory has a name that can be the name of analyzed sequence. The important thing is that the three files that must be in this directory must have the same name (they differ in postfix). Thus for example for P0020 sequence we have a P0020 directory and P0020.sdf, P0020.org and P0020.apl files with appropriate SDF, ORG log and Activity Plan.

In case we need new objects or hand poses, it is necessary to create them using existing means. We have successfully used the Parallel Graphics VRML Pad editor [7] for any VRML code editing. Since the models must follow certain rules concerning their structure, we recommend to create them by textual editing too. Some appearance related parts might of course be created in some 3D modelling software.

To create hand and body postures we have used Parallel Graphics Internet Character Animator (ICA) [7], a software for hierarchical structure animation that supports H-Anim standard. This program is not very user friendly and easy to control. Its main advantage was the VRML structure input and animation output. The actual pose structure was created simply by copying parts of ICA's results.

### 4.5.3 SDF Editor

To help creating the Scenario Definition Files for the VR presentation we have created a simple utility. This editor enables the user to assemble the scene and specify all the necessary relations (see SDF description in 4.1.3 in the process).

ActIPret system gives us the ORG log that lists all the detected objects. The SDF editor reads this list and creates the scene with these known objects.

At this point it also prompts the user for the information concerning how these objects will be modelled in VR. Then it offers the user to include another objects (not detected, such as table, chair, etc.) from the model set specified in the beginning. These objects can be placed using their (user specified) relations to objects already placed (a table is under the CD-player).

Finally the kinematic chains for the inverse kinematics animation can be selected, described by the end-effector and the base joint. The kinematic chains are associated with the detected object that were previously labelled as part of human body.

The SDF editor solves problem of placing objects into the scene, which is better performed by putting the model on the right place rather than writing down the coordinates. It also helps to choose the end-effectors and base joints as it reads all the candidates and presents them to the user.

It does not specify various parameters such as the texture or color of a CD. On the other hand it is not much of a problem to specify these simply by writing into the generated textual SDF.

## 5 Conclusions

We have implemented one of the two suggested approaches to the VR presentation of generalized concept of observed activity. Besides the general activity plan the chosen approach uses also some of the observed data.

Although a functional VR presentation module was created, our work has another result. We have become familiar with some problems of cooperation in areas that are by their nature quite distant. When seeing the video sequence of human performing an activity, VR and cognitive vision focus on very different things. Thus the ActIPret VR presentation module is an attempt to bridge this quite wide gap.

If we should draw some conclusions concerning the success of this attempt, we would have to say that it was not highly successful. The VR presentation did not get over numerous problems and imperfections. It is not comparable to what a state-of-the-art computer animation can be. On the other hand with very limited input it performs well enough to show the activity.

## References

- [1] L. M. Goncalves, M. Kallmann, and D. Thalmann. Programing behaviors with local perception and smart objects: An approach to solve autonomous agents tasks. In *Proceeding of SIGGRAPH 2001*, 2001.

- [2] M. Kallmann and D. Thalmann. Modeling behaviors of interactive objects for real time virtual environments. *Journal of Visual Languages and Computing*, 13, 2002.
- [3] Vladimír Štěpán, Jiří Žára, and Václav Hlaváč. Virtual reality presentation demo: Human activities in VR. Research report, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, October 2003.
- [4] Ch. Welman. Kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, 1993.
- [5] Homepage of C.Babski, LIG - H-Anim humanoids and animations. <http://ligwww.epfl.ch/~babski/>.
- [6] H-Anim (Human Animation Working Group). <http://www.hanim.org>.
- [7] Parallel Graphics home-page (Cortona VRML browser). <http://www.parallelgraphics.org>.
- [8] Virtual reality lab (former LIG), EPFL Lausanne. <http://vrlab.epfl.ch>.