ACTS Project N. AC040

D 4.4 - Integration of the interaction techniques into the applications

Project Number:	ACTS Project N. AC040
Project Title:	COVEN - Collaborative virtual environments
Deliverable Type:	P

CEC Deliverable Number:	A040-THO-LCR-DS-P-044.b1
Contractual Date of Delivery to the CEC:	31st December 1997
Actual Date of Delivery to the CEC:	19th January 1997
Title of Deliverable:	Integration of the interaction techniques into the applications
Workpackage contributing to the Deliverable:	WP4
Nature of the Deliverable:	R**
Author(s):	Véronique Normand (Thomson-CSF), Nadia Magnenat Thalmann (Univ. of Geneva), Daniel Thalmann (EPFL), Christian Babski (EPFL), Jean-Benoît Bonne (Thomson-CSF), Stéphane Carion (Univ. of Geneva), Alexis Dollé (Thomson-CSF), David Roussel (Thomson-CSF).

Abstract:
This report describes the main results of the COVEN Activity 4.4, which aimed at integrating interaction techniques into the COVEN applications. Two distinct sets of interaction techniques were integrated: virtual human representation and animation techniques, as well as multimodal, spoken interaction techniques. This document presents the integration of both sets of techniques through two separate parts.
Integration of the virtual human modelling and animation techniques required to convert existing body representation formats to formats supported by the COVEN Platform. A bridge was then developed between dVS/Dive worlds and our body control libraries to animate the converted body hierarchy.
Integration of the multimodal interaction techniques first required work at a conceptual level so as to refine our existing approach to referent computation, and to extend our existing mono-user functional frame to address the requirements of the multi-participant perspective. Integration work specifically focused on the implementation of a process architecture within the dVS frame. Finally, a large component of work addressed the development of application-specific natural language resources (grammars) and command interpretation mechanisms ; this was done within the context of the WP2 Interior Arrangement scenario.
Keyword list: dVS, Dive, virtual humans, multimodal interaction, referent computation.

Executive summary

This report describes the main results of the COVEN Activity 4.4, which aimed at integrating interaction techniques into the COVEN Demonstrator applications. Two distinct sets of interaction techniques were integrated: virtual human representation and animation techniques, as well as multimodal, spoken interaction techniques. This document presents the integration of both sets of techniques through two separate chapters.

Integration of the virtual human modelling and animation techniques required to convert existing body representation formats to formats supported by the COVEN Platform; we applied exactly the same method in both dVS and Dive worlds: conversion of body surfaces from internal SM format to dVS .bgf format and Dive .vr format; conversion of our body hierarchy to dVS .bod format and Dive .vr format specialised for body definition and user representation. A bridge was then developed between the dVS/Dive worlds and our body control libraries to animate the converted body hierarchy.

Integration of the multimodal interaction techniques first required work at a conceptual level so as to refine our existing approach to referent computation, and to extend our existing mono-user functional frame to address the requirements of the multi-participant perspective; new concepts were defined, and a new functional architecture was derived and implemented. Integration work specifically focused on the implementation of a process architecture within the dVS frame. Finally, a large component of work addressed the development of application-specific natural language resources (grammars) and command interpretation mechanisms ; this was done within the context of the WP2 Interior Arrangement scenario.

Table of contents

1. Introduction

COVEN is addressing the requirements of collaborative interaction in CVEs, with a specific focus on two main issues:

- embodiment, that is providing users with appropriate virtual bodies so as to convey key communication information on presence, location, identity, but also activity, including actionpoints (i.e. object zones where they are manipulating).

- spoken interaction, that is providing interaction modes complementary to 3D direct manipulation all the while respecting VE immersion requirements; and exploring the impact of the collaborative dimension onto this form of interaction.

Embodiment

The COVEN developments regarding embodiment have focused on realistic human representations. While realism is clearly not the only answer to the embodiment issue (see e.g. Benford et al., 1997), this choice was motivated by the requirements of some of our applications (in particular, the Travel Rehearsal simulation-type scenario and the citizen-oriented Virtual Travel Agency). Our work mainly consisted in extending the EPFL & University of Geneva libraries of dynamic models of humans (cf. D4.1), and integrating them into the COVEN platform.

Integration of the virtual human modelling and animation techniques required to convert existing body representation formats to formats supported by the COVEN Platform; we applied exactly the same method in both dVS and Dive worlds: conversion of body surfaces from internal SM format to dVS .bgf format and Dive .vr format; conversion of our body hierarchy to dVS .bod format and Dive .vr format specialised for body definition and user representation. A bridge was then developed between the dVS/Dive worlds and our body control libraries to animate the converted body hierarchy.

Multimodal, spoken interaction

Classically, interaction within VEs is the direct manipulation of 3D representations. While the interest of the direct manipulation paradigm of interaction cannot be questioned, the limits of this paradigm are also clear, making it unpractical to refer to abstract concepts, to objects that do not exist yet (typically to create them), to specific properties of objects, and to large and evolving sets of objects. One option is to have recourse to ‘traditional’ interaction modes such as textual commands, 2D or 3D menus, with the risk to prevent full immersion within the environment, or to threaten the premise of VR interface transparency. In COVEN, we are exploring the alternative solution of spoken natural language interaction (Normand et al., 1997) (Billinghurst & Savage, 1996) (Karlgren et al., 1995); a prototype system was developed and applied within the context of the Interior Arrangement application scenario, featuring speech-based multimodal interaction capabilities.

Integration of the multimodal interaction techniques first required work at a conceptual level so as to refine our existing approach to referent computation, and to extend our existing mono-user functional frame (cf. D4.2) to address the requirements of the multi-participant perspective; new concepts were defined, and a new functional architecture was derived and implemented. Integration work specifically focused on the implementation of a process architecture within the dVS frame. Finally, a large component of work addressed the development of application-specific natural language resources (grammars) and command interpretation mechanisms ; this was done within the context of the WP2 Interior Arrangement scenario.

Structure of the document

This document is structured into two distinct parts:

- Part I presents the main features of the integration of the virtual human modelling and animation techniques.

- Part II presents the main aspects of the developments involved in the integration of the multimodal, spoken interaction techniques.

Part I

Integrating
the virtual human modelling and animation techniques

1. Introduction

Main task of this activity was to import bodies from our system to COVEN platform (DIVE and dVS) to be used as users graphical representation. This work implied to retrieve bodies geometry and hierarchy in both applications, but also to be able to perform animation in connection with user interaction.

2. Humanoid Integration in COVEN platform

The full integration inside COVEN platform was performed in two steps. First step consisted in exporting bodies graphical representation to specific formats of the COVEN platform in order to be usable as users representation inside shared 3D worlds (avatar). Second step permit to animate those bodies according to interactions performed by users within an on-going session.

2.1 Static Level

This work was performed for each application dVS and DIVE. Obtaining body surfaces was close to perform a classical conversion from one graphical 3D format to an other, except that we had to write this converter.

We wrote two converters which permit to export from our in-house file format to specific geometric formats of DIVE and dVS.

The second part of this static level was to retrieve the body hierarchy by defining a body file. It makes possible to retrieve our classical hierarchy in terms of joints and degrees of freedom (Fig. I.1) [boul1] within COVEN platform.

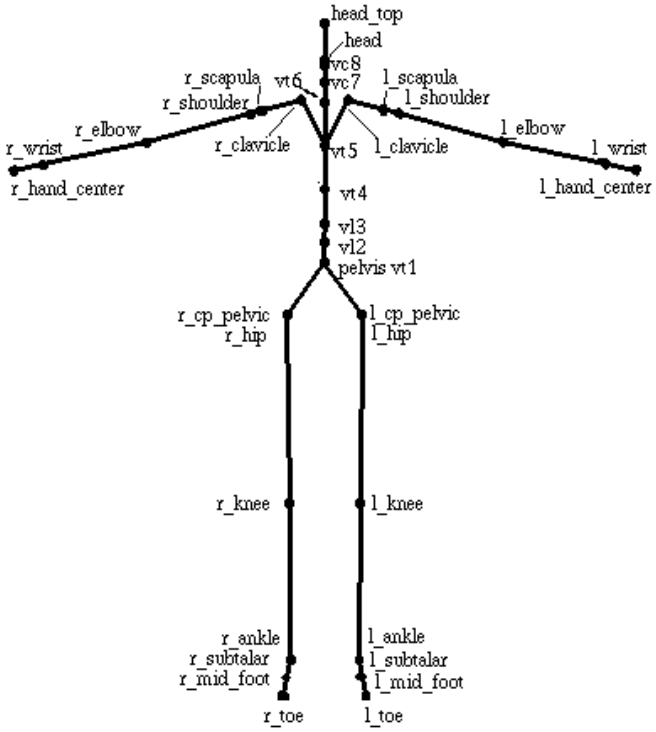

Figure I.1 : Body hierarchy - joints list

This hierarchy is needed to be able to animate the user's avatar in a realistic way. We were obliged to export our skeleton in two formats as long as both applications, DIVE and dVS, have their own built-in avatar definition file format.

Dive avatar definition file :

```
/*-----------------------------------------------------
# ---          Body Hierarchy Converter v1.0         ---
# ---                   LIG/EPFL                    ---
# ---           From BodyLib to DIVE Format         ---
# --- Conversion Time :
#     Wed Oct  1 09:40:41 1997
# -----------------------------------------------------*/

#include "dive.vh"
actor "JoBill" {
   "top" id 500
   "body" id 501
   "right_eye" id 502
   "left_eye" id 503
}

object { /* --- Body Top Hierarchy Object */
   id 500
   name "top"
   realobj on
   object {
      id 501
      name "body"
      fixedxyz v 0 3.14 0
      object {
         id 504
         name "stdman_position"
         translation v 0.000000 0.600000 0.000000
      object {
         id 502
         name "right_eye"
         fixedxyz v 0.0 -3.14 0.0
         translation v  -0.100000 0.400000 0.0
      }
      object {
         id 503
         name "left_eye"
         fixedxyz v 0.0 -3.14 0.0
         translation v 0.100000 0.400000 0.0
      }
         object { /* stdman_lower_torso */
            id 1
            name "stdman_lower_torso"
            translation v 0.000000 -0.300000 0.0
            /* include Buffman_lower_torso ---------------------- */
#include "body/bodyparts/lod0/Buffman_lower_torso.vr"
            /* end of Buffman_lower_torso ---------------------- */
            object { /* stdman_upper_torso */
               id 2
               name "stdman_upper_torso"
               translation v 0.000000 0.185870 0.0
               /* include Buffman_upper_torso ---------------------- */
#include "body/bodyparts/lod0/Buffman_upper_torso.vr"
               /* end of Buffman_upper_torso ---------------------- */
               object { /* stdman_head */
                  id 4
                  name "stdman_head"
                  translation v 0.000000 0.309783 0.0
                  /* include Buffman_head ---------------------- */
#include "body/bodyparts/lod0/Buffman_head.vr"
                  /* end of Buffman_head ---------------------- */
              } /* ----- End of stdman_head */
               object { /* stdman_left_upper_arm */
                  id 14
                  name "stdman_left_upper_arm"
                  translation v -0.204644 0.184322 0.0
                  /* include Buffman_left_upper_arm ---------------------- */
```

```
#include "body/bodyparts/lod0/Buffman_left_upper_arm.vr"
                  /* end of Buffman_left_upper_arm ---------------------- */
                  object { /* stdman_left_lower_arm */
                     id 15
                     name "stdman_left_lower_arm"
                     translation v -0.002134 -0.266947 0.0
                     /* include Buffman_left_lower_arm -------------------- */
#include "body/bodyparts/lod0/Buffman_left_lower_arm.vr"
                     /* end of Buffman_left_lower_arm ---------------------- */
                     object { /* stdman_left_hand */
                        id 16
                        name "stdman_left_hand"
                        translation v -0.011441 -0.259565 0.0
                        /* include Buffman_left_hand ---------------------- */
#include "body/bodyparts/lod0/Buffman_left_hand.vr"
                        /* end of Buffman_left_hand ---------------------- */
                      } /* ----- End of stdman_left_hand */
                   } /* ----- End of stdman_left_lower_arm */
                } /* ----- End of stdman_left_upper_arm */
                 object { /* stdman_right_pick_upper_arm */
                    id 11
                    name "stdman_right_pick_upper_arm"
                    translation v 0.204663 0.184397 0.0
                    /* include Buffman_right_pick_upper_arm ----------------- */
#include "body/bodyparts/lod0/Buffman_right_pick_upper_arm.vr"
                    /* end of Buffman_right_pick_upper_arm ------------------ */
                    object { /* stdman_right_pick_lower_arm */
                       id 12
                       name "stdman_right_pick_lower_arm"
                       translation v 0.002269 -0.266950 0.0
                       /* include Buffman_right_pick_lower_arm -------------- */
#include "body/bodyparts/lod0/Buffman_right_pick_lower_arm.vr"
                       /* end of Buffman_right_pick_lower_arm --------------- */
                       object { /* stdman_right_hand */
                          id 13
                          name "stdman_right_hand"
                          translation v 0.011492 -0.259563 0.0
                          /* include Buffman_right_hand ---------------------- */
#include "body/bodyparts/lod0/Buffman_right_hand.vr"
                          /* end of Buffman_right_hand ---------------------- */
                        } /* ----- End of stdman_right_hand */
                     } /* ----- End of stdman_right_pick_lower_arm */
                  } /* ----- End of stdman_right_pick_upper_arm */
               } /* ----- End of stdman_upper_torso */
            } /* ----- End of stdman_lower_torso */
             object { /* stdman_hip */
                id 0
                name "stdman_hip"
                translation v 0.000000 -0.300000 0.0
                /* include Buffman_hip ---------------------- */
#include "body/bodyparts/lod0/Buffman_hip.vr"
                /* end of Buffman_hip ---------------------- */
                object { /* stdman_left_thigh */
                   id 8
                   name "stdman_left_thigh"
                   translation v -0.094239 -0.127174 0.0
                   /* include Buffman_left_thigh ---------------------- */
#include "body/bodyparts/lod0/Buffman_left_thigh.vr"
                   /* end of Buffman_left_thigh ---------------------- */
                   object { /* stdman_left_leg */
                      id 9
                      name "stdman_left_leg"
                      translation v 0.004842 -0.463846 0.0
                      /* include Buffman_left_leg ---------------------- */
#include "body/bodyparts/lod0/Buffman_left_leg.vr"
                      /* end of Buffman_left_leg ---------------------- */
                      object { /* stdman_left_foot */
                         id 10
                         name "stdman_left_foot"
                         translation v 0.003921 -0.382547 0.0
                         /* include Buffman_left_foot ---------------------- */
#include "body/bodyparts/lod0/Buffman_left_foot.vr"
                         /* end of Buffman_left_foot ---------------------- */
                       } /* ----- End of stdman_left_foot */
                   } /* ----- End of stdman_left_leg */
                } /* ----- End of stdman_left_thigh */
                 object { /* stdman_right_thigh */
                    id 5
```

```
                name "stdman_right_thigh"
                translation v 0.094239 -0.127174 0.0
                /* include Buffman_right_thigh ---------------------- */
#include "body/bodyparts/lod0/Buffman_right_thigh.vr"
                /* end of Buffman_right_thigh ---------------------- */
                object { /* stdman_right_leg */
                   id 6
                   name "stdman_right_leg"
                   translation v -0.004557 -0.463862 0.0
                   /* include Buffman_right_leg ---------------------- */
#include "body/bodyparts/lod0/Buffman_right_leg.vr"
                   /* end of Buffman_right_leg ---------------------- */
                   object { /* stdman_right_foot */
                      id 7
                      name "stdman_right_foot"
                      translation v -0.003831 -0.382559 0.0
                      /* include Buffman_right_foot ---------------------- */
#include "body/bodyparts/lod0/Buffman_right_foot.vr"
                      /* end of Buffman_right_foot ---------------------- */
                   } /* ----- End of stdman_right_foot */
                } /* ----- End of stdman_right_leg */
             } /* ----- End of stdman_right_thigh */
          } /* ----- End of stdman_hip */
       } /* ----- End stdman_position */
       object {
          name "left_foot"
          translation v 0.050000 -0.800000 0.000000
       }
       object {
          name "right_foot"
          translation v -0.050000 -0.800000 0.000000
       }
    } /* ----- End Body */
} /* ----- End Top */
```

dVS avatar definition file :

```
# ------------------------------------------------------
# ---          Body Hierarchy Converter v1.0         ---
# ---                   LIG/EPFL                     ---
# ---            From BodyLib to dVS Format          ---
# --- Conversion Time :
#     Wed Mar  5 14:44:19 1997
# ------------------------------------------------------

stdman_position { # To put camera at the same position of the head
                  # and not of the torso which is now the top level
                  # of the dvs body hierarchy
  %ifdef LIG_OFFSET
        POSITION (0.000000, 0.300000, -1.500000)
        ORIENTATION (0,0,0)
  %else
        ALIGN_PARENT
  %endif
  stdman_lower_torso {
     POSITION (0.000000,-0.500000,0.000000)
     VISUAL "body/lod0/Buffman_lower_torso" {
        VECTORINTERSECT OFF
    }
     stdman_upper_torso {
        POSITION (0.000000, 0.185870, 0.000000)
        VISUAL "body/lod0/Buffman_upper_torso" {
           VECTORINTERSECT OFF
       }
        stdman_head {
           POSITION (0.000000, 0.309783, 0.000000)
           VISUAL "body/lod0/Buffman_head" {
              VECTORINTERSECT OFF
           }
        } # ----- End of stdman_head
        stdman_left_upper_arm {
           POSITION (-0.204644, 0.184322, 0.000000)
           VISUAL "body/lod0/Buffman_left_upper_arm" {
              VECTORINTERSECT OFF
           }
```

```
            stdman_left_lower_arm {
               POSITION (-0.002134, -0.266947, 0.000000)
               VISUAL "body/lod0/Buffman_left_lower_arm" {
                  VECTORINTERSECT OFF
                }
                stdman_left_hand {
                   POSITION (-0.011441, -0.259565, 0.000000)
                   VISUAL "body/lod0/Buffman_left_hand" {
                      VECTORINTERSECT OFF
                    }
                } # ----- End of stdman_left_hand
            } # ----- End of stdman_left_lower_arm
         } # ----- End of stdman_left_upper_arm
%ifdef LIG_HAND
         stdman_right_pick_upper_arm {
            POSITION (0.204663, 0.184397, 0.000000)
            ORIENTATION(-90,0,0)
            VISUAL "body/lod0/Buffman_right_pick_upper_arm" {
               VECTORINTERSECT OFF
           }
            stdman_right_pick_lower_arm {
               POSITION (0.002269, 0.000000, -0.266950)
               VISUAL "body/lod0/Buffman_right_pick_lower_arm" {
                  VECTORINTERSECT OFF
                }
#                    stdman_right_pick_hand {
#                       POSITION (0.011492, 0.000000, -0.259563)
#                       VISUAL "body/lod0/Buffman_right_pick_hand" {
#                          VECTORINTERSECT OFF
#                        }
#                    } # ----- End of stdman_right_pick_hand
%else
         stdman_right_pick_upper_arm {
            POSITION (0.204663, 0.184397, 0.000000)
            VISUAL "body/lod0/Buffman_right_upper_arm" {
               VECTORINTERSECT OFF
           }
            stdman_right_pick_lower_arm {
               POSITION (0.002269, -0.266950, 0.000000)
               VISUAL "body/lod0/Buffman_right_lower_arm" {
                  VECTORINTERSECT OFF
                }
                    stdman_right_pick_hand {
                       POSITION (0.011492, -0.259563, 0.000000)
                       VISUAL "body/lod0/Buffman_right_hand" {
                          VECTORINTERSECT OFF
                        }
                    } # ----- End of stdman_right_pick_hand
%endif

            } # ----- End of stdman_right_pick_lower_arm
         } # ----- End of stdman_right_pick_upper_arm
      } # ----- End of stdman_upper_torso
   } # ----- End of stdman_lower_torso
   stdman_hip {
      POSITION (0.000000,-0.500000,0.000000)
      VISUAL "body/lod0/Buffman_hip" {
         VECTORINTERSECT OFF
     }
      stdman_left_thigh {
         POSITION (-0.094239, -0.127174, 0.000000)
         VISUAL "body/lod0/Buffman_left_thigh" {
            VECTORINTERSECT OFF
        }
         stdman_left_leg {
            POSITION (0.004842, -0.463846, 0.000000)
            VISUAL "body/lod0/Buffman_left_leg" {
               VECTORINTERSECT OFF
            }
            stdman_left_foot {
               POSITION (0.003921, -0.382547, 0.000000)
               VISUAL "body/lod0/Buffman_left_foot" {
                  VECTORINTERSECT OFF
                }
            } # ----- End of stdman_left_foot
         } # ----- End of stdman_left_leg
      } # ----- End of stdman_left_thigh
      stdman_right_thigh {
```

```
            POSITION (0.094239, -0.127174, 0.000000)
            VISUAL "body/lod0/Buffman_right_thigh" {
               VECTORINTERSECT OFF
            }
            stdman_right_leg {
               POSITION (-0.004557, -0.463862, 0.000000)
               VISUAL "body/lod0/Buffman_right_leg" {
                  VECTORINTERSECT OFF
               }
               stdman_right_foot {
                  POSITION (-0.003831, -0.382559, 0.000000)
                  VISUAL "body/lod0/Buffman_right_foot" {
                     VECTORINTERSECT OFF
                  }
               } # ----- End of stdman_right_foot
            } # ----- End of stdman_right_leg
         } # ----- End of stdman_right_thigh
      } # ----- End of stdman_hip

         # ----- To identify somebody by changing texture in texture/world.rgb
         cube {
            POSITION (0.0,0.35,0.0)
            SCALE (1.5,0.7,0.5)
            VISUAL "body/m_cube" {
                 VECTORINTERSECT OFF
            }
         } # ----- End of cube

   # ----- if OFFSET is on, we shouldn't attach
   # ----- the hand to the point of view but to
   # ----- the real body head.
   #%ifndef LIG_OFFSET
     } # End of Top Level Body Part (our one)
   #%endif
```

2.2 Dynamic Level

This level permit to animate user's avatar in real time, within COVEN platform. To avoid to have to make the same work twice, we succeed in writing a generic body controller which is able to connect to any application DIVE or dVS without any change. This connection is based on a shared memory segment allocated by the main application. The external body controller can connect to this shared memory segment in order to drive the user's avatar according to inputs it receives from the main application.

The external body controller is based on libraries developed in both labs LIG/EPFL and University of Geneva. These libraries basically permit to perform several kinds of animation like walking motion (based on a parametrically walking motor), keyframe animation or real-time recorded animation (by using a set of magnetic sensors fixed on the body of a real user). In a more complex way, it is also able to mix several animation and applying a notion of weight to animation [boul2]. A walking body can then play an animation with his right arm and continuing to walk. Any COVEN platform developers can write their own body controller process, based on different libraries (to be able to make a direct interface between a specific device and the body to animate for example). Just by following the shared memory interface specification (c.f. section on this topic), it is very easy to control the body defined in the main application.

As long as each application has different functionalities, performances from one platform to an other are different (mainly in network configuration). But, in terms of body animation, capabilities are the same whatever the platform we are using. In the following sections, we will describe each system based on their own external application interface.

- DIVE

In the following graph (Fig. I.2), we have an overview of how the system is working within DIVE. Each DIVE clients specified in the graph is an enhanced version (made from the default one) which is able to allocate shared memory. This new application is based on DIVE external application interface. It includes capabilities of the default version plus two sets of functionalities directly linked to body animation. The first set of functionalities will permit to detect and to retrieve a body structure in the database of DIVE. The second set will use network data distribution functions of the DIVE external application interface to perform animation and send it to connected clients.

According to the kind of avatar used by the user, the DIVE application will allocate or not shared memory (the enhanced version can be used with any type of body). Once the shared memory is allocated (user's avatar is conform to the description made in the previous section), the external body controller process can be launched. When the connection with the main program is established, user's avatar will be animated according to what he is doing in the shared 3D world.

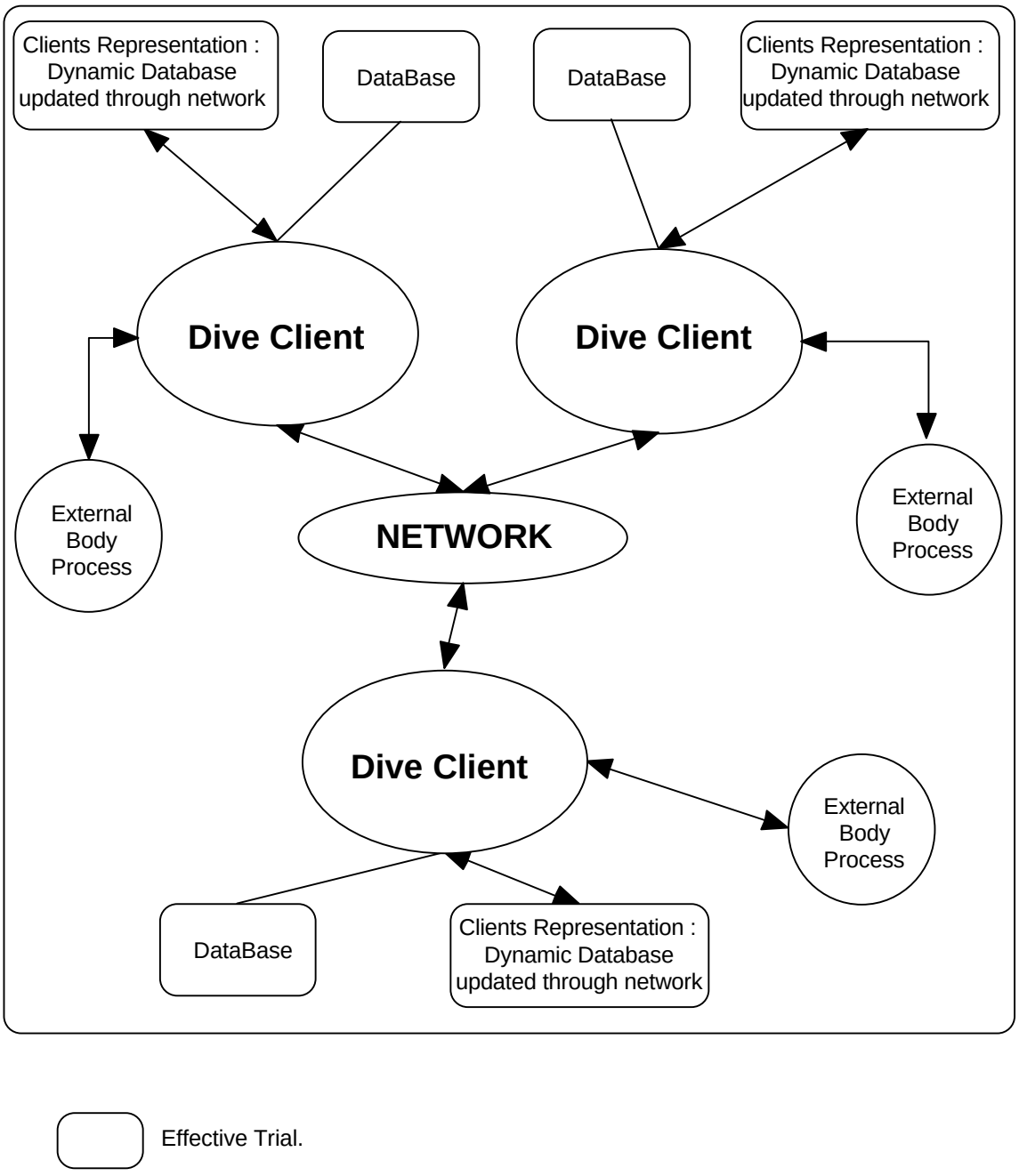

Figure I.2 : Dive System

The shared memory segment is used to transmit data in both ways from and to the DIVE application. Data flow coming out from the main application to the body controller is composed of interactions performed by the user : outputs like speed change, direction change or object to pick are sent to the external process to adapt body animation to the kind of action which is about to be displayed and sent to the network. This way is also useful at the beginning of the connection to inform the external body controller about the structure of the shared 3D world. As long as the body controller is not integrated at the level of the main application, it is not possible for this process to access to the world database to locate objects to avoid for example. This means that a set of information has to be sent to initialise the external process concerning the surrounding 3D world. The incoming data flow, from the external process to the main application, concerned only the body animation and is only composed by a set of sixteen (number of different parts in the animated body) axis angle values. These values are computed according to interactions received by the body process from the main application : resulting values can be a mix between a walking movement and a specific arm animation asked by the user through keyboard.

All clients who want to participate to an on-going session does not have to use the specific DIVE program. Default version of the DIVE browser can also participate to the trial. This user will be able to see body animation from other clients but his own graphical representation will not be animated (or just by using default animation capabilities of DIVE).

DIVE platform is able to sent through network data related to an incoming client. Thus, the graphical representation of a new user will be sent to others participants at the beginning of the connection. This system permit to any client to join an on-going session at any time, without any specific preparation. With the same system, any user can add objects to the shared 3D scene, objects that will increase the global database of available 3D objects in the scene. These imported objects will be manipulated as any objects of the initial scene. This permit, for example, to import private crowds which will follow the user (c.f. WorkPackage 8.1).

- dVS/dVISE

As it was exposed in the introduction of this section, the external body controller process used with dVS is the same as the one used with DIVE. Differences of performance between both platforms are mainly due to two reasons :

- Different network approach
- Different global architecture.

dVS is working by launching a set of processes which all have a specific task to do (network communication, devices management...). Due to the way dVS is working (client-server solution), it was not possible to write a new version of the main application which will be launched by any clients. This is mainly due to the fact that the main program is only launch on the server side but not by all connected clients. We were obliged to directly change the process in charge of the client graphical representation called *vcbody*, which manages, locally, each user's avatar.

The communication is established between the external body controller and the *vcbody* process. The protocol used for communication is exactly the same as the one described in the DIVE paragraph and used the same shared memory interface (see following section).

Any client communication pass through the server which dispatch information to other clients. All the needed sets of files which define the shared 3D world have to be locally present at each client side. No geometry is sent through the network, only references to local files. This is also applicable to the graphical representation of each user. With such a system, if clients want to have their own personalised avatar, all the needed information, which include textures, geometries, materials, have to be transferred to all clients and to the server before the beginning of the session. If a client wants to join an on-going session, he has to use a default graphical representation or he has to use the avatar of an already connected client.

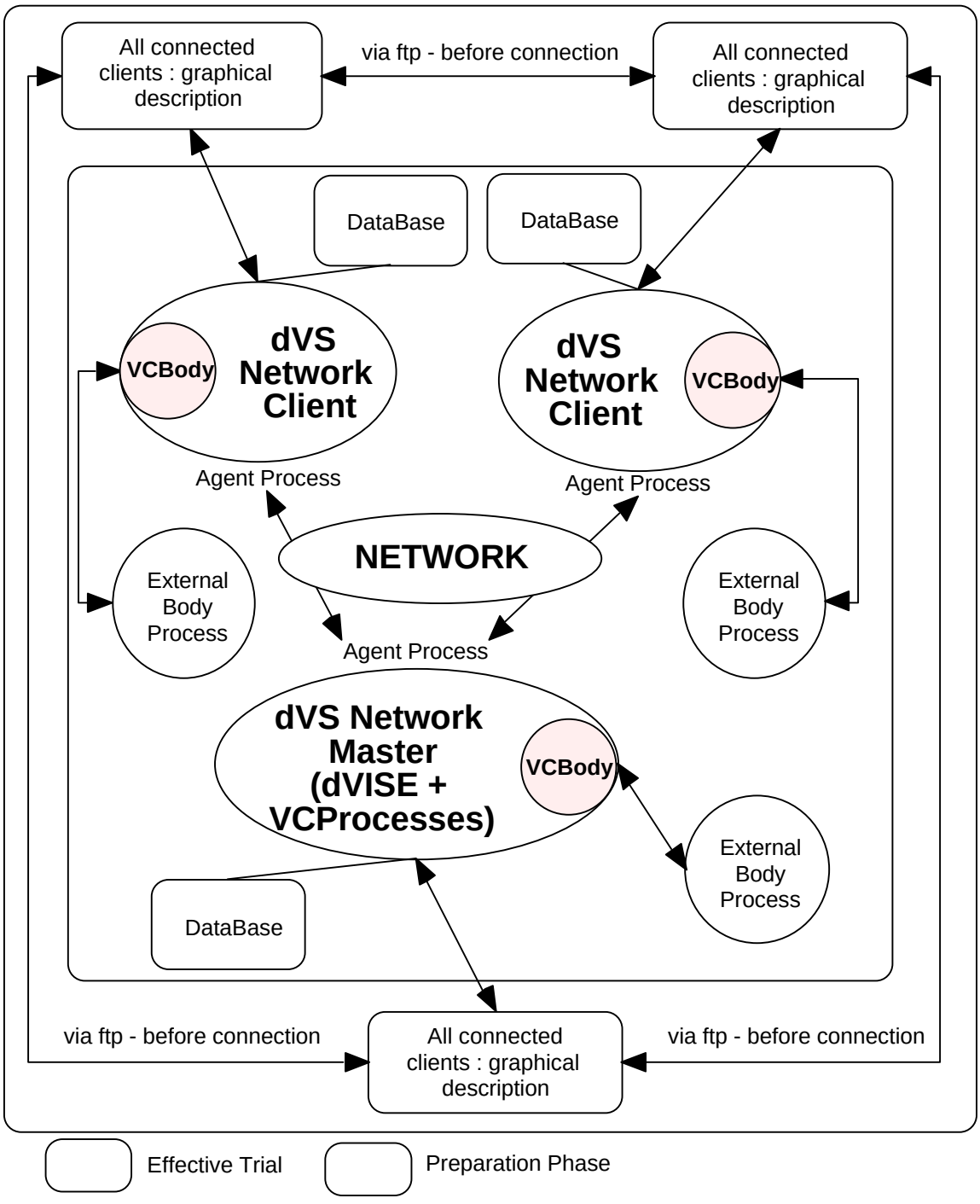

Figure I.3 dVS System

3. Shared Memory Functions

All functionalities of the shared memory interface will be described in details in the annex section. They are separated in 2 sets :

- a set of functions used by the main application
- a set of functions used by the external body controller.

The following graph (Fig I.4) describe the way unix shared memory is working. By using a key as an identification of a data structure in the shared memory segment, it is even possible to use several shared memory segment with the same application (which is the case when you are using a animated body and crowds (c.f. Workpackage 8.1) simultaneously).

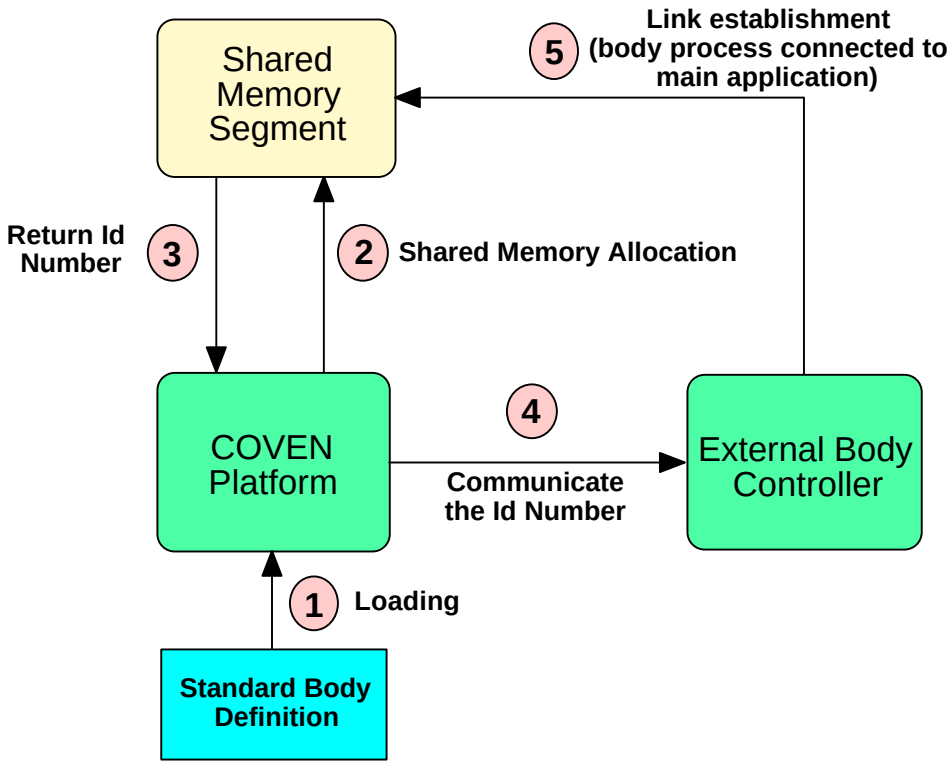

Figure I.4 Shared Memory System

Communication of the identification number from the COVEN application to the body controller can be done through several different ways :

- COVEN platform can launch itself the external process
- User can launch manually the external process (which let him the possibility to not activate body animation - actual solution)

- COVEN platform and external body controller can use classical socket system (or unix signal utilities) to first communicate automatically with each other, to send the initial information which is the identification number.

The shared memory segment is organised in a structure which group all data needed for the communication of virtual humanoid's animation. The communication system is based on a system of flags to « wake up » the concerned program when data have to be retrieved and updated.

4. Conclusion

The final integration of the whole system inside applications developed inside COVEN project like the *Business Application*, was simplified by the fact that we changed directly a specific process instead of writing an enhanced version of the main program (which was made with DIVE). It mainly avoid to have to mix what was develop and integrated in the main application by other partners with our system. To be able to use animated body for user graphical representation inside *Business Application* for example, partners just have to replace the default *vcbody* process by the new one without any side effect on the main application. In the other hand, we accessed source code which are not supposed to be modified, which could be a problem if some COVEN application developers want to enhance or rewrite a new body controller with some new functionalities.

5. References

[boul1] R. Boulic, T. Capin, Z. Huang, L. Moccozet, T. Molet, P. Kalra, B. Lintermann, N. Magnenat-Thalmann, I. Pandzic, K. Saar, A. Schmitt, J. Shen, D. Thalmann, *The HUMANOID Environment for Interactive Animation of Multiple Deformable HumanCharacters*, Proc. Eurographics `95, Maastricht, August 1995, pp.337-348.

[boul2] R. Boulic, P. Becheiraz, L. Emering, and D. Thalmann, *Integration of Motion Control Techniques for Virtual Human and Avatar Real-Time Animation*, Proc. VRST '97, ACM Press, 1997, pp.111-118.

6. COVEN Application Developers Annex

6.1 Main Program Functions

int vrh_get_free_key()

Return the first free key number usable for a shared memory segment. This function should be called first to use **vrh** interface.

int vrh_exit(int key)

Input Parameter :
Key : result of a call to ***vrh_get_free_key()***
Exiting function : send a signal to the external body controller (to terminate it automatically).
Return TRUE if function works, else FALSE.

int vrh_create_shmem(int key)

Input Parameter :
Key : result of a call to ***vrh_get_free_key()***
Allocation of the shared memory segment.
Return TRUE if function works, else FALSE.

int vrh_get_axis_angle(int key, int actor_id, float **axis_angle_table)

Input Parameters :
Key : same as the one used for ***vrh_create_shmem***
Actor_id : must be 0 (user's avatar)
Output Parameter :
Axis_angle_table : array of [HUMAN_NB_BODY_PARTS][4] floats. Axis angle is x,y,z and angle value arround this axis.
Retrieve a new set of values to update body position. To be used after a TRUE return of ***vrh_check_active*** function (see below).
Return TRUE if function works, else FALSE.

int vrh_check_active(int key, int actor_id)

Input Parameters :
Key : same as the one used for ***vrh_create_shmem***
Actor_id : must be 0 (user's avatar)
Return TRUE if a new set of values have to be retrieved to update body position. Values are retrieved by using ***vrh_get_axis_angle*** function.

int vrh_check_grasped(int key, int actor_id)

Input Parameters :
Key : same as the one used for ***vrh_create_shmem***
Actor_id : must be 0 (user's avatar)
Return TRUE if grasp animation (perform by using inverse kinematic motor on the right arm of the avatar) is finished, else FALSE. Prevent to deal with several requests to grasp several different objects.

int vrh_remove_shmem(int key)

Input Parameter:

Key : same as the one used for ***vrh_create_shmem***

Remove shared memory segment (to avoid to do it manually by using **ipcs**/**ipcrm** unix commands). The exiting signal must be sent to the external body controller by using ***vrh_exit*** function before removing the shared memory segment.

Return TRUE if function works, else FALSE.

int vrh_set_velocity(int key, int actor_id, float velocity)

Input Parameters :

Key : same as the one used for ***vrh_create_shmem***

Actor_id : must be 0 (user's avatar)

Velocity : new speed in mm/s.

Change walking velocity (walking motor) of the body. This velocity has to be in mm/s.

Return TRUE if function works, else FALSE.

int vrh_set_animation(int key, int actor_id, char animation)

Input Parameters :

Key : same as the one used for ***vrh_create_shmem***

Actor_id : must be 0 (user's avatar)

Animation : key pressed by user.

User pressed a key (event catched by COVEN platform EAI) which could be link to an animation. This function sends the request to the body controller which will check the pressed key ***animation*** (a-z and A-Z only) with information it got from the animation file loaded at the beginning.

Return TRUE if function works, else FALSE.

int vrh_set_pick(int key, int actor_id, float *body_pos,
float *obj_center_pos, float *pos)

Input Parameters :

Key : same as the one used for ***vrh_create_shmem***

Actor_id : must be 0 (user's avatar)

Body_pos : array of [3] floats to specified the actual 3D position of the user (in mm).

Obj_center_pos : array of [3] floats to specified the actual object's center position(in mm).

Pos : array of [3] floats to specified the 3D position where the user clicked on the desired object (in mm).

The user tried to pick an object (by clicking on it). Last body position, center position of the object and the 3D position of the clicked point (all in mm) are sent to the body controller to animate the right arm by using inverse kinematic in order to reach the object.

Return TRUE if function works, else FALSE.

6.2 **External Body Controller Functions**

int vrh_init(int key)

Input Parameter:

Key : retreived from the main program where effective allocation is performed

Establish the connection with the shared memory segment allocated by the main application (see previous section). This is the first function to call in the body controller program.

Return TRUE if function works, else FALSE.

int vrh_set_axis_angle(int key, int actor_id, float **axis_angle_table)

Input Parameters :

Key : same as the one used for ***vrh_init***

Actor_id : must be 0 (user's avatar)
Axis_angle_table : array of [HUMAN_NB_BODY_PARTS][4] floats. Axis angle is x,y,z and angle value arround this axis.
Send to the main application program a new set of values (result of all on-going actions : walking, picking, playing animation) to update body position.
Return TRUE if function works, else FALSE.

int vrh_check_velocity(int key, int actor_id)

Input Parameters :
Key : same as the one used for ***vrh_init***
Actor_id : must be 0 (user's avatar)
Check if a new velocity was sent by the main application with ***vrh_set_velocity*** function.
Return TRUE if yes, else FALSE.

int vrh_check_animation(int key, int actor_id)

Input Parameters :
Key : same as the one used for ***vrh_init***
Actor_id : must be 0 (user's avatar)
Check if a new animation request was sent by the main application with ***vrh_set_animation*** function.
Return TRUE if yes, else FALSE.

int vrh_check_pick(int key, int actor_id)

Input Parameters :
Key : same as the one used for ***vrh_init***
Actor_id : must be 0 (user's avatar)
Check if a pick request was sent by the main application with ***vrh_set_pick*** function.
Return TRUE if yes, else FALSE.

int vrh_get_velocity(int key, int actor_id, float *output_velocity)

Input Parameters :
Key : same as the one used for ***vrh_init***
Actor_id : must be 0 (user's avatar)
Output Parameters :
Output_velocity : new velocity for walking motor (in mm/s)
Retrieve new velocity (in mm/s) to apply to the walking motor from the main application.
Return TRUE if function works, else FALSE.

int vrh_get_animation(int key, int actor_id, char *output_animation)

Input Parameters :
Key : same as the one used for ***vrh_init***
Actor_id : must be 0 (user's avatar)
Output Parameters :
Output_animation : new animation request to check
Retrieve new animation (character [a-z] [A-Z]) to check from the main application. This function will check if an animation is associated with the recieved character (according to animation file - defined by user). If yes, the animation will be played.

Example of user's defined animation file (loaded at the beginning of the body process) :

```
# Body Animation file version 0.1

MiraLigAnimation hi {
       file hi2.TRK
       weight 100
```

```
        playmode SWING
        multiplay 1
        playkey a
}

MiraLigAnimation welcome {
        file welcome.TRK
        weight 100
        playmode FORWARDS
        multiplay 1
        playkey b
}
```

Return TRUE if function works, else FALSE.

int vrh_get_pick_pos(int key, int actor_id, float *body_pos,

float *obj_center_pos, float *pos)

Input Parameters :

Key : same as the one used for ***vrh_init***

Actor_id : must be 0 (user's avatar)

Output Parameters :

Body_pos : array of [3] floats to specified the actual 3D position of the user (in mm).

Obj_center_pos : array of [3] floats to specified the actual object's center position(in mm).

Pos : array of [3] floats to specified the 3D position where the user clicked on the desired object (in mm).

Retrieve information for inverse kinematic motor (in order to pick an object) and launch the right arm animation.

Return TRUE if function works, else FALSE.

int vrh_grasped(int key, int actor_id)

Input Parameters :

Key : same as the one used for ***vrh_init***

Actor_id : must be 0 (user's avatar)

Specified to main program that inverse kinematic animation (for picking an objet with the right arm) is finished. It avoids to mix several requests for picking different objects at the same time.

Return TRUE if function works, else FALSE.

int vrh_get_exit(int key)

Input Parameter :

Key : same as the one used for ***vrh_init***

Return TRUE if the body controller received an exit signal from the main application.

Part II

Integrating the multimodal, spoken interaction techniques

1. Introduction

This chapter describes the work that was undertaken so as to further develop and integrate the spoken natural language techniques within the COVEN Demonstrator.

A first phase of activity was dedicated to the refinement of our conceptual approach and the revision of our functional architecture. This was required so as to improve our support to referent computation within virtual environments (VEs), and to extend our existing mono-user functional frame to address the requirements of the multi-participant dimension of shared VEs. New concepts were defined, and a new functional architecture was derived and implemented.

A second phase of work was more specifically dedicated to the actual integration of our revised functional modules within the COVEN Platform environment and Demonstrator. A process architecture was implemented within the dVS frame. A large component of work addressed the development of application-specific natural language resources (grammars) and command interpretation mechanisms ; this was done within the context of the WP2 Interior Arrangement Demonstrator scenario.

Structure of this chapter

The body of this chapter is structured into five sections:

- Section 2 recalls the general (mono-user) functional architecture previously described in D4.2. This architecture will be re-considered, further discussed and adjusted in the course of this chapter so as to match the constraints of COVEN application integration.

- Sections 3 and 4 present the results of our first phase of work, at conceptual and functional architecture level: section 3 presents our refined and consolidated approach to referent computation, in a mono-user frame for clarity sake; section 4 describes the adjustments that were performed on our functional architecture in order to address the requirements of the multi-user dimension.

- Section 5 presents the results of our second phase of work, at platform and Demonstrator level: the integration of the functional architecture with the dVS platform, and the application-specific developments that were performed so as to provide the COVEN Interior Arrangement Demonstrator with spoken natural language interaction features.

- Section 6 summarises the achievements and assesses the status of these results.

An Annex part is also provided, which offers elements of a multimodal interaction scenario as well as a detailed (although partial) view of the linguistic resources that were developed for the Interior Arrangement Demonstrator.

Related documents

COVEN deliverable D2.5, Report on the on-line version of the business application, (January 98) contains a description of the Interior Arrangement application within which natural language spoken interaction features were developed. In particular, this report describes the UI design choices that were performed for this application, and the way the classical direct manipulation interaction mode and the spoken interaction mode were integrated.

COVEN deliverable D4.2, Initial report on the multimodal interaction techniques, (January 97) provides a general overview of our initial developments.

2. Overview of the (mono-user) functional architecture

The general functional architecture previously described in D4.2 is recalled in this section. This architecture will be re-considered, further discussed and adjusted in the course of this chapter so as to match the constraints of COVEN application integration.

For simplicity sake, a mono-user perspective is adopted so as to allow to focus exclusively on multimodal interaction within VEs. The multi-participant dimension will be fully addressed in section 3.

Figure II-1 depicts the abstract functional architecture of the system in a mono-user context. Our choice is to distinguish four main sub-systems:

- a domain functions subsystem, which holds the data and functions of the application domain.
- a VE subsystem, which holds the data and functions in charge with the representation of the virtual scene and its inhabitants and the support of basic actions within the virtual world; this is also were distribution over the network is handled.
- a Natural Language Processing (NLP) subsystem, which holds the data and functions specific to the handling of natural language interaction.
- an interaction subsystem in charge with the general co-ordination of the overall interaction.

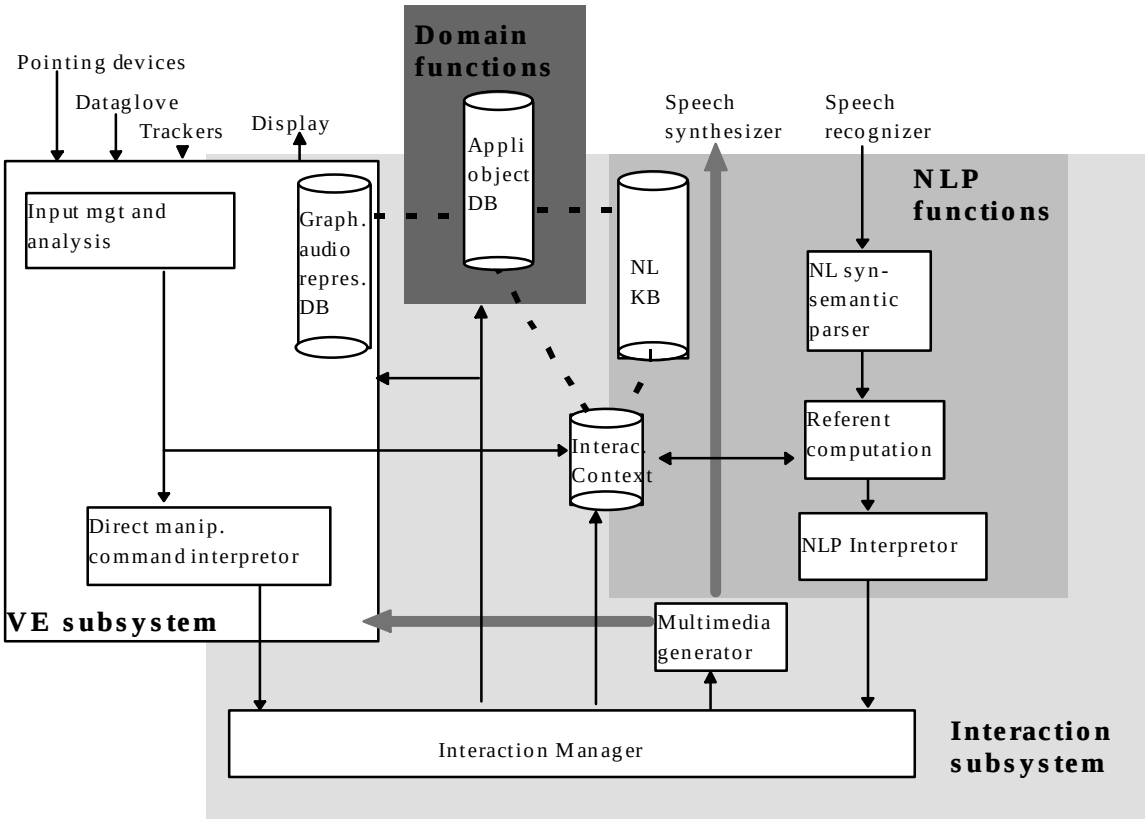

Figure II-1: Functional architecture of the multimodal VE system - mono-user view

2.1 The domain subsystem

The domain subsystem holds the data and functions implementing the concepts of the application domain. An object-oriented design and implementation approach is adopted, in particular with the management of object types or classes, a simple class inheritance hierarchy (used to represent the hierarchy of concepts), and the notion of object property (attribute - value couple) and method. An object repository was implemented; it stores object classes, relationships between classes, object instances and object instance properties. It supports queries based on object classes names and object property lists: retrieval of objects satisfying conditions of category and/or property values; validity check of an object (does it still exist). Participant-dependent access rights on objects and methods are managed.

In the Interior Arrangement Demonstrator scenario (cf. D2.5), an inheritance tree of classes: furniture, work-surface, desk, table, seat, chair, shelves, decoration, plant, computer, telephone, etc. is managed. Domain operations are: the creation, deletion of objects; change to their property values (colour, material, style, position); change to their access rights.

2.2 The VE subsystem

The VE subsystem is in charge with the visualisation of and the basic interaction with the virtual world, its objects and inhabitants. Basic components of this system classically are a database that holds the representations of the world entities, a visualiser module, a collision manager in charge with detecting the collisions between the VE entities, and an event manager, which receives and handles user interaction events (coming from pointing/tracking devices) as well as collision events and possibly time- or frame-triggered events. Support for distribution over the network is the responsibility of this subsystem. In the COVEN context, we are using the dVS/dVISE part of the COVEN platform.

The basic interaction that takes place strictly within the VE subsystem is of direct manipulation type; actual interpretation of actions, and execution of the specified commands takes place through the co-ordination of the Interaction Manager in the Interaction subsystem.

In the general context of VE interaction, and specifically multimodal interaction, direct actions initiated and supported within the VE subsystem are:

- viewpoint navigation interaction (dVS/dVISE interaction mode and mechanisms).
- selection and de-selection of objects, pointer-based (see the notion of ‘selected set’ managed in the Interaction subsystem).
- designation of locations within the space, pointer-based (support to the notion of passive designation gestures managed in the Interaction subsystem).

More specifically in the Interior Arrangement scenario, VE interaction also involves:

- interaction with the Interior Arrangement menu system, which allows to specify creation, deletion, colour and material change operations on objects of the selected set.
- direct object positioning (managed by dVISE).

2.3 The Interaction subsystem

The Interaction subsystem is in charge with the global management of the multimodal interaction within the system. This involves:

1. co-ordinated feedback at all stages of a command specification, whatever the specification mode (pointer-based, speech-based or multimodal).
2. checking of command specification completion and correction, and error feedback in case of bad specification.

3. checking of user rights to execute the current command, and appropriate error feedback (for direct-manipulation commands, this is done at the time of the specification; for speech-based commands, this is done after the specification).
4. execution of the command through the launching of possibly several application methods (case of complex commands on several objects).
5. feedback on the result of the execution.
6. support to the implementation of the global Interaction Context for referent computation (see section 3).
7. management of the current *selected set* of objects. The selected set or selection is a classical concept in user interfaces, traditionally used to support the designation by the user of the object(s) on which her next command should apply (object + command paradigm).
8. management of *designation gestures*, that is pointer-based designations of objects or of positions performed by the user within the VE. These designations are different from object selections: they are passive (from VE viewpoint) gestures denoting locations within the 3D space, or objects in a zone around the designated location (see section 3.2). They are exploited by the NLP subsystem at referent computation time.

It is important to note that actions 1 to 5, which take place along the specification and the execution of user commands, are common to all commands, whatever their specification mode (pointer-based, speech-based or multimodal).

Some limited feedback mechanisms were implemented as generic feedback common to any application scenario: highlight of objects identified as the referents in a NL utterance; adjustment of the selection to the command objects at the end of the command execution (empty selection for the ‘delete’ command). In the Interior Arrangement application scenario, additional graphical, textual and audio interaction status feedback were designed, as functions of the multimedia generator - this will be evoked when presenting the multi-participant system in section 4; more details may be found in D2.5, Interior Arrangement section.

2.4 The Natural Language Processing subsystem

The NLP subsystem supplies a literal meaning to user utterances through speech recognition followed by syntactico-semantic parsing. Referent computation then allows to relate the noun phrases of the parsed utterances to objects of the environments; the completed parsed structures are then translated into command specifications ready for interpretation and execution by the Interaction subsystem.

2.4.1 Speech recognition

The speech recognition system outputs an un-interpreted linguistic sequence corresponding to the operator's command. As described in D4.2, we are using a connected-speech, speaker-independent recognition system, with a relaxed Context-Free Grammar constrained syntax - Nuance™ from Nuance Communication. Preliminary versions of the system are using a speaker-dependent speech recognition system, Datavox from Vecsys.

Static data involved: a Context-Free Grammar description of the target application utterances.

2.4.2 From NL utterance parsing to multimodal command interpretation

Natural language parsing is dedicated to the extraction of the literal meaning of the text strings delivered by the speech recognition system. Our approach uses the traditional steps of syntactic and semantic analysis, with an emphasis on the integrated processing of syntax and semantics (cf. D4.2). The syntactic parser recovers sentence syntagmatic structure and is the starting point for natural language understanding: its role is to identify meaning components as well as functional relations between parts of a sentence. We are using a parsing approach based the formalism of a Tree-Adjoining Grammar (TAG), which is a lexicalised grammar that combines ease of description with powerful combinatorial properties (Joshi et al., 1975). This approach is called Tree-Furcation Grammar (TFG), and is evoked in (Halber & Roussel, 1997).

The output of the parser is a TFG tree corresponding to the analysis of the recognised utterance. Within this tree, nodes corresponding to noun phrases requiring referent computation have an empty :ref seme. The referent computation function fills in these semes with the object identifier lists corresponding to their reference value. Referent computation is the core issue in supporting multimodal interaction - section 3 below details our approach and the main features of the referent computation function.

Finally, the completed TFG tree is translated into a command request description, along a formalism expected by the Interaction Manager.

3. Refining our approach to referent computation

The objective of this section is to clarify our approach to speech-based multimodal interaction support, with the refinement of our approach to the central issue of natural language referent computation.

Reference concerns the relation of symbols to things outside the system of symbols. The role of referent computation in Natural Language Processing is to match the noun phrases of utterances with the objects of the environment (here, objects of the VE application).

As an example, consider the following interaction scenario:

1. User: « Add two chairs »
   These referents need to be created, they do not pre-exist.
2. User: « Change *this* chair to wooden » + designation gesture
   The reference is a combination of anaphoric co-reference (refers to the previous 2 chairs) and deictic information (the gesture).
3. User: « Set *this one* to blue »
   Is this a reference to the chair of utterance 2, or is this a direct reference to some new salient chair in the changed perception field of the user?
4. User: « Move *it* there » + designation gesture
   A simple co-reference to object of the previous utterance.
5. User: « Move *the blue desk* to the meeting room »
   Is there only one blue desk, or is this a direct reference to some particularly salient blue desk in the changed perception field of the user, or in the task context of the user?
6. User: « Remove *the* chairs »
   Is this a co-reference to the chairs of utterance 1, is this a direct reference to some particularly salient group of chairs in the perception or task field of the user, or is this a direct reference to all the chairs of the environment?

Both linguistic phenomena (anaphoric co-references), task-related and perception-related phenomena need to be taken into account when solving the references.

Referent computation is a central issue for speech-based interaction, and more specifically for multimodal interaction. This section presents and discusses our concepts and approach. General referent computation principles and assumptions are first introduced; the concepts of referential context and referential gestures are then described before presenting the main features of our referent computation algorithm. A test scenario illustrating the main aspects in the behaviour of our referent computation algorithm is provided in the Annex part of this Chapter.

3.1 NL referent computation principles

3.1.1 General approach; notion of a referential viewpoint

Important progress has been made by the AI community in the past twenty years regarding models for the identification of referents. Most of the approaches have focused on models providing ways to limit the search spaces that contain possible referents; these models were motivated not by theories of the fundamental nature of reference (which seem to be more or less insufficient), but rather by theories of the structure of human discourse (Grosz, 1977), (Sidner, 1986), (Grosz et al. 1995). Work within the multimodal interaction community has focused on applying these models for implementing working referent processing subsystems within speech-based human-computer interaction systems; this requires the definition of practical interpretations of these models, the derivation of heuristics from these (incomplete) theories as a way to drive the referent processing algorithm.

The approach based on human discourse structure theories raises a number of critics as many aspects in human-computer interaction differ from human-human interaction situations. In particular:

- influence of the task: HCI systems are used within the context of a task, which may be more or less structured, open or closed depending on the type of system. Whatever the characteristics of the task at hand, it can be foreseen that the task context (position of the user in the task tree at a given moment, result of the current action, status of completion of the pending tasks) influences the interaction context, and, in particular, the salience of the referents.

- influence of the perception: referents here are not purely intellectual discourse objects as in most discourse studies but application objects that are for the main part represented in visual or sound mode by the system. It must be noted that the perceptual nature of some referent identification processes was already highlighted in some discourse structure cognitive studies, especially those related to indexical references (Moulton & Roberts, 1994). The influence of perception leads to moving away from the denotational view of reference to a view where contrast of the referent to its background is an important element.

Our approach to referent computation is an evolution of (Pouteau, 1995), and uses the notions of referential contexts and axiology developed in (Gaiffe, 1992).

3.1.2 Referential contexts

A *referential context* holds a structured list or set of objects that were distinguished in a particular way, constituting possible candidate objects for referents. A multimodal dialogue builds up a number of different possible referential contexts: contexts built up by the discourse (Sidner, 1986), subject to such linguistic phenomena as anaphoric co-references; contexts built up from the activity of the user (in particular, task contexts), and from the perception the user gets of his environment, subject to « direct » (non linguistic) references. Within COVEN, only contexts related to discourse structure and perception are managed; heuristics related to the task structure could not be developed within our time and resource frame.

Referent search takes place within these different restricted sets of candidate objects, a given set (or referential context) corresponding to a particular referential viewpoint; the search goes on until a referential viewpoint is found, which allows to unambiguously identify the referent(s) searched for; or fails when all referential viewpoints have unsuccessfully been considered. The referent search is performed in different ordered contexts along a set of heuristic rules based on the characteristics of the noun phrase (in particular, its determiner value).

3.1.3 Axiologies

Within a given context, referent computation is constrained by the application of contrasting *axiologies*: each property p denoted by the noun phrase needs to allow the uncovering of a contrast between subsets of objects through the construction of two non empty sets P and (non P) so that:

- $\forall x \in P : p(x) = 1$
- $\forall x \in (\text{non } P) : (\text{non } p(x)) = 1$

This notion of axiology, based on a gricean economy principle from cognitive cost viewpoint, allows to eliminate contexts where it cannot be applied. For the noun phrase ‘the blue chair’, a valid search context has to contain both chairs and objects that are not chairs, and blue chairs and chairs that are not blue; if not, the gricean assumption is that the user would not have taken the pain to mention properties of colour and category.

Adjustments to the general axiology approach:

A first adjustment that was performed concerns category properties. In our opinion, it can be questioned whether the economy principle is always valid, especially regarding the category property: from a cognitive cost viewpoint, it seems difficult to argue that qualifying an object through its category ('the chair') would be more costly than to identify it through generic qualifiers such as 'the object' (even if all objects of the context are chairs) - no added word. It is thus tempting to relax the axiology constraint in the case of category qualifiers, and to restrict the respect of this constraint to the case of properties leading to additional qualifying adjectives.

Now, application of the axiology principle cannot be performed without the availability of dialogue capabilities in the system allowing to handle axiology-related failures in a satisfying way. Indeed, it is not acceptable to simply come up with an empty referent set and refuse execution of the user's command because s/he said « the red chairs » instead of « the chairs », and there are only red chairs in the environment. Our simple approach within COVEN is to relax the axiology constraint when a total axiology-related failure is encountered (that is, no referential context could be found that contains candidates verifying the axiology principle); the referents computed are then presented to the user (through highlighting), and confirmation of the command is requested.

As a final remark, it should be mentioned that we are only using the axiology principle locally to a referent computation: the sets built up through the application of the axiology principle do not have a persistence over several dialogue utterances. Memorisation of these sets could usefully be envisaged as a support to the understanding of phenomena in chained sequences of utterances.

3.1.4 General assumptions

Our approach in COVEN is based on three main assumptions:

1. A fully multimodal dialogue is assumed, with the construction of a discourse structure encompassing both linguistic elements and elements of gestures / direct manipulations. Anaphora are thus supported across media: an utterance may refer to objects (previously or concurrently) distinguished through direct action; a direct action may refer to objects (previously only) distinguished through language[1].
2. Pre-eminence of the discourse memory over perception: when processing a referent, the hypothesis for an anaphoric referent is handled preferentially, while the hypothesis of a direct reference to an object of the environment is considered in a second time only. In other words, the memory of previous elements of the multimodal discourse is considered to be more present in the user's mind than the visual/audio perception of the environment.
3. Pre-eminence of the speech channel over the gesture channel, when used concurrently: a gesture is used to discriminate objects within a set identified beforehand by the utterance.

These assumptions are certainly not generalisable assertions; they are simply our work hypothesis from which a number of design choices were made. Assumption 1 is questionable from a theoretical viewpoint, although no theory exists that clearly settles this issue. In our specific context, this seems a valid assumption since the UI itself favours cross-media references through the implementation of the 'selected set' as the constant repository of the preferential topic (see below), embodying in a perceptible way the first referential context. Assumption 2 is frequent in multimodal systems; in some specific situations, it can be foreseen that perception is stronger than discourse memory - heuristics still need to be developed to characterise these situations.

---

1 This is made possible by our usage of the 'selection' notion, as explained hereafter.

3.2 Referential gestures: selection gestures and designation gestures

Support for multimodal interaction is one objective for our system; more specifically, we are addressing the combination of referential gestures and natural language utterances. Our analysis is that these referential gestures may involve two types of gestures: passive so-called designation gestures and active selection gestures. Both types of gestures will be exploited at referent computation time.

3.2.1 Classical pointing gestures - active selection gestures

In classical graphical environments, pointing gestures generally have the immediate effect of triggering an action, from the simple selection of the object (placed in the common concept of 'selected set') to the launching of an application command (e.g. case of the push buttons). Among these gestures, *selection gestures* are specifically of interest when considering multimodal interaction since they may be interpreted as the designation of objects to be used as the referents of utterances.

Worth of notice is the fact that, in standard direct manipulation interactions, the referent of a gesture is never ambiguous; it is immediately identified from the position of the pointer at the time of the designation (in 3D environments, a projection may be performed so as to identify the object).

3.2.2 'Designation' gestures

In natural communication, deictic gestures are passive[2] gestures that can be vague or ambiguous; possible ambiguities are raised by the analysis of their accompanying utterances. The possibility for raising the ambiguity in a designation gesture thanks to spoken information is an interesting feature of multimodal interaction: it allows gestures to refer to composite objects, to objects partially hidden by other objects, or to very small items, which would not be possible with simple classical pointing gestures; it also supports a level of imprecision.

Two types of designation gestures are addressed in the current prototype, corresponding to the designation of an object and of a location within the 3D scene.

3.2.3 Cohabitation of passive designation and active selection gestures - practical considerations

An issue then is: how to allow the smooth cohabitation of both passive deictic gestures and active selection gestures (especially when objects are possibly subject to both types of gestures), and of both vague deictic gestures and unambiguous manipulative gestures?

In the COVEN context, a strict constraint was not to question the form and effect of the standard active pointing gestures within a VE. Standard direct selection and manipulation gestures need to remain valid in the multimodal interaction context; even more, standard selection gestures (object selection + command selection paradigm of interaction) designate referents which should seamlessly be usable in a forthcoming direct manipulation or natural language utterance. The current implementation therefore imposes a different form for passive, purely referential gestures: while active gestures are based on simple mouse clicks, designation gestures are performed through a combination of mouse click + keyboard modifier key (in an immersive context, a different mouse button could be used, or active gestures could be performed e.g. through double clicks).

---

2 Passive with regards to the VE: they are not interpreted within the strict VE scope.

A target solution, rather than to leave it to the user to explicitly indicate the nature of the gesture s/he is performing as in the current implementation, is to automatically categorise the gesture depending on its synchronisation with the possible speech utterance, in a way related to (Streit, 1997)[3].

3.3 The Interaction Context component of the system

The Interaction Context component of the system is the central repository of multimodal interaction data; it holds and maintains the different referential contexts managed by the system, as well as the referential gestures performed by the user.

3.3.1 Referential contexts managed by the system

The system was designed to manage five different referential contexts:

1. *topic*: referents computed for the preceding command; at the head of the topic, the '*preferential topic'* contains the objects on which the command was applied[4].
2. *dialogue history*: list of the previous topics. A window of eight topics is maintained (simple management of discourse memory attenuation). The topic structure is retained within the history, allowing to maintain a memory of the object sets that were constructed by the discourse; this structure is in particular exploited when handling plurals, as explained later on in this section.
3. *VE-salient objects*: objects of the VE scene with particular salience characteristics at the origin of contrasts within the scene, without any relationship to the interaction history. Numerous salience criteria may be envisaged, in relation to:
    - the recent actions performed in the scene, e.g. a new object in the field of view or audition for the user, an object that was newly created or highlighted by someone else, etc.
    - the intrinsic perceivable properties of the object (e.g. colour, size), at the source of possible contrasts.
    - the spatial cohesion of the scene objects (groups of objects, isolated objects, forefront and background, etc.).
    - etc.
4. *perceptible objects*: the general set of the objects perceptible (visually or audio) by the participant at a given time.
5. *the environment*: the whole set of the VE application objects.

Contexts 1 and 2 are related to the interaction history (anaphoric contexts) while contexts 3 and 4 are related to the perception the user has of the objects in the VE; context 5 is the global set of application objects, whether represented in the interface or not.

---

3 One solution is to check the synchronisation between the gesture and the deictic expression in the utterance; one problem here is that 1) this requires precise word-level time stamps from the speech recognition system, which is not obvious in the current commercial systems; 2) in the case of active gestures, their interpretation will be delayed until the utterance analysis is complete, which may lead to lags in the interaction. A simpler solution is to check only the occurrence of the gesture within the time frame of an utterance, and to consider that active gestures cannot be performed along with an utterance and therefore can only be passive; this is a drastic solution, which may be OK in the COVEN context.

4 Actually whether the command was indeed executed or not (e.g. if cancelled later on) has no importance here.

It is important to note that, in line with our assumption 1) above, the anaphoric contexts are fully multimodal: the objects of a given topic are taken seamlessly both from pure NL referent computation results and from direct manipulation or designation operations within the VE.

3.3.2 Referential gestures

Along with the management of these contexts, the Interaction Context component of the system is the repository of the pointer-based designations of objects or, mainly, of positions performed by the user within the VE.

The Interaction Context also manages the selection, or selected set, which is another source of referential gestures as described in 3.2. Our approach is to give the selection a central role in the overall multimodal interaction, having it contain the 'preferential topic' at a given moment, i.e. the objects in the immediate interaction focus. At the end of the execution of a command, whatever the interaction mode used to specify the command elements, the command objects are automatically used to update the selected set; the selected set thus always contains the elements currently in the immediate task focus of the user, whether these come from direct selection operations performed by the user (to prepare for a new command) or from the results of the previous command of the user (assumption of task continuity).

The content of the selection is used as preferred focus by the NLP functions at referent computation time; it is also used as the objects of menu commands and dVISE direct manipulations, thus unifying speech-based and direct-manipulation commands. This is further developed in D2.5, where we present a view of the selected set as a *cross-mode representation of the task focus*.

3.3.3 Practical considerations

The Interaction Context clearly is a global structure at the interface between the VE and the NLP subsystems; in the current implementation, contexts 1 and 2 are maintained within the NLP subsystem, with the updates taking into account partial elements provided by the VE subsystem through the Interaction subsystem regarding the user's direct actions on the environment: selections and direct manipulation operations. Contexts 3 to 5, as well as designations are maintained by the Interaction subsystem, with the support of the VE subsystem.

3.4 Multimodal referent search - general algorithm

Note that a test scenario illustrating the main aspects in the behaviour of our referent computation algorithm is provided in the Annex part of this Chapter.

3.4.1 Completing the referent description

Before actually starting the referent search, the referent computation function attempts at completing the referent description that it has received as input. In case of an anaphoric noun phrase, the class of the referent is searched for in the topic if needed (e.g., « this one »). Inference of properties is also attempted, depending on the global utterance, along a list of predefined rules. Some of these inferences are purely mechanical ways for exploiting information already present in the syntacto-semantic tree; this concerns information of relative position (e.g. 'the table near the plant') and ownership (e.g. 'john's table'). Other inferences are real deductions of assumed properties depending on the predicate of the utterance: typically, the utterance 'set the chair to blue' may lead to infer that the chair referred to is not blue.

3.4.2 Search algorithm

Once the referent description is completed, the referent computation first attempts at exploiting the *designation gestures* of the user, if any (see section 3.2) for a distinction between designation gestures and selection gestures). If referents still remain to be computed, the system enters the stage of referential context search.

a Exploitation of designation gestures (support of multimodal interaction)

Two types of designation gestures may meaningfully be interpreted in the current prototype, corresponding to the designation of an object and of a location within the 3D scene. These referential gestures are retrieved from the Interaction Manager; the matching of gestures and deictic referents is attempted starting from the last deictic referent in the sentence, and the last gesture performed by the user. A deictic location reference is directly identified from the co-ordinates of its corresponding designation gesture; an object reference is computed through searching the VE spatial zone around the 3D co-ordinates of the pointer (support for ambiguous gestures), taking into account the object category, properties and cardinality attached to the related noun phrase.

Extraneous designation gestures are simply skipped in the current prototype. A more sophisticated approach could naturally be implemented so as to improve the matching of gestures and referents, taking into account the synchronisation of gestures and noun phrases.

b Searching referential contexts

If unsolved referents remain, the referent computation algorithm enters the stage of search within the different contexts, using the axiology principle.

The characteristics (category and properties) of the VE objects contained in the referential contexts are compared with the characteristics of the references that appear in the sentence. The search starts in the anaphoric contexts - the main assumption is that references will preferentially be anaphoric: a reference preferentially relates to the previous interactions rather than to the visual or audio perception - this is our assumption 2 of section 3.1. The search depends on the characteristics of the noun phrase (definite or not, deictics, pronouns) as follows:

Indefinite referents (e.g. case of the noun phrase ‘a blue chair’) are not handled: they correspond to creation commands only.

Demonstrative syntagms (e.g. case of the noun phrase ‘this blue chair’) are searched for successively in the topic, the VE-salient and the perceptible objects contexts.

Pronoun references are searched for in the preferential topic only.

Definite referents (e.g. ‘the blue chair’) are searched for successively in all contexts: the anaphoric contexts (topic and history), the VE-salient context, the perceptible objects and the whole DB of objects. Same thing for the possessive referents.

Co-ordinated noun phrases are handled separately; the final referent set is the union of the results of the separate searches. Related noun phrases (e.g. ‘the chair near the desk’) are computed one after the other, in the appropriate order.

The referent search stops as soon as it is successful within a given context, and the axiology principle is satisfied. In case of failure in all contexts, a description of the failure is forwarded to the Interaction Manager, which is in charge with providing the proper feedback to the user[5].

c Particular case of the selection gestures

While the designation gestures are handled separately at the beginning of the algorithm, selection gestures are handled transparently within the contextual search part of the algorithm. Indeed new user selection gestures are regularly retrieved from the Interaction Manager and used to update the topic referential context, as part of the contexts update mechanisms outlined hereafter.

3.4.3 Update of the referential contexts

Perception-related contexts are not actively maintained but are computed 'on the fly', when needed by the referent computation mechanisms. This is naturally not the case for anaphoric contexts.

As previously mentioned, the anaphoric referential contexts are fully multimodal contexts: they are built and maintained out of the elements of the multimodal discourse of the user. Update of these contexts takes as input both linguistic structures and VE-actions elements.

Upon the reception of an utterance for handling, the NLP subsystem 'synchronises' with VE subsystem in that it collects from the Interaction Manager a description of the direct actions that were performed by the user within the VE since the last utterance (menu actions, direct manipulations for object positioning and 'manual' updates of the selection). In the current prototype, only the referents of these actions are of interest; the Interaction Manager thus provides the NLP subsystem with a list of sets of referents corresponding to the sets of objects that were manipulated in the VE (possibly several objects for a given action); the contents of the user-modified selection set, if not empty, is transmitted as the head of this list. These elements are used to update the history as well as the topic referential contexts; in particular, if not empty, the contents of the selection set will be placed within the topic context.

After the handling of an utterance, the corresponding linguistic structures (including the list of computed referents) are used to update the topic context, which leads to also update the history context. While the linguistic structures are currently directly stored in the contexts, only their referent identifiers are used later on by the referent search algorithms.

In the current prototype, information of category, property, and status (in particular, destruction of objects) are not stored within the anaphoric contexts but are retrieved from the Domain subsystem when needed by the referent computation mechanisms (actually, only the information of category could be stored as the other elements are highly subject to changes during the session). Heterogeneity of the multimodal context data (linguistic structures as well as simple object identifier lists) prevents from considering more sophisticated linguistic computations, which however does not appear as a real limitation for our current applications so far[6]. Of more interest would be the storing and exploitation of action descriptions, which should be considered in extensions to our current work.

---

5 Unfortunately real dialogue capabilities could not be implemented in the first integrated prototype, due to lack of resources. Only contextual feedback messages are provided.

6 In particular, our English dialogues do not require the handling of issues of gender, as French dialogues would.

3.4.4 Case of the plurals

Noun phrases referring to more than one objects are handled along the same principles as the singular referents - the referent search algorithm looks for groups of objects (of defined or undefined cardinality, depending on the noun phrase) in the different contexts as indicated above[7].

A number of questions arise when handling plurals:

- can the referent group be composed over several contexts?
- within the dialogue history context, can the referent group be composed over several topics? When should the search stop in the case of a group of undefined cardinality (e.g. 'the chairs')?

Our referential viewpoint approach leads us to consider that referent groups cannot be composed over several contexts, but need to be retrieved within the boundaries of a unique context.

The second question is an open question: in some scenarios, the topics frontiers does not seem fully relevant (e.g. interaction 1: 'add a blue chair'; interaction 2: 'put the yellow chair here'; interaction 3: 'remove these chairs'). The question then becomes: when do we stop collecting referents? How to quantify the exact scope of the discourse memory attenuation phenomenon? We believe some heuristics may be developed to answer these questions - for example, one heuristic would consider the discourse continuity as a factor for group identification. In our current implementation, we deliberately made the (simplistic) choice to keep within the limits of the topics structure.

In the anaphoric contexts, a group of objects is looked for within two object sets: the preferential topic set and the overall topic set ; a group is never composed over several topics in the dialogue history - topics strictly delimit the possible object groups. In the three VE-related contexts, the search for groups is performed freely, depending on the cardinality.

Further to refining the approach described above, an issue of particular interest is the search for object groups within the VE contexts. It is foreseen that visual perception plays a major role in the identification of groups of objects, independently of co-references. Spatial cohesion of a set of objects constitutes a primary contrasting feature for distinguishing a group; contrasts based on the sharing of a perceptible property (e.g. colour) also need to be taken into account. This calls for models and algorithms allowing to detect groups of objects in a 3D scene, which should be the focus of further explorations.

---

7 Sets with the « all » quantifier (« all the chairs ») are handled differently in the current prototype, with the search taking place in the 5th context only (the whole DB).

4. Addressing the requirements of the multi-participant dimension

COVEN is addressing situations where several participants may interact with the application in parallel, possibly (but not always) in a co-operative/synergistic way. Ideally, in a full collaborative interaction context, it should be possible to:

- individually interact with a system so as to perform some local operations, without necessarily taking into account what's going on around in the shared environment.
- work in some form of co-operation with the other participants, with varying degrees in the connections and dependencies.

Co-operation modes

We propose to define two different modes of co-operative work:

1. '*loose'* co-operation mode with the other participants: one is aware of the presence and activity of the others; collaborative work is performed through inter-personal negotiations and adjustments (typically, using the audio communication channel); however interaction with the system remains personal.
2. '*tight'* co-operation mode with one or several other participants: interaction with the system is collective, shared with the other participants; selection of objects is collective (no personal selected set)[8]; specification of a given command may be performed by several different users; commands (including spoken utterances) may refer to objects in commands issued by other participants.

It is clearly not our postulate that these two modes fully match the characteristics of co-operative activities in general. These were rather defined as two extreme modes that should allow to cover different types of co-operative activities, with the possibility to switch between modes during a session, depending on the task requirements and individual preferences. Our objective in the support of these two modes is to investigate their requirements, build an understanding of the issues that are raised with regards to natural language interaction in particular, and allow for experiments to take place so as to better assess their validity and limits.

Toward the revision of the functional architecture

The general requirements with regards to interaction management are that the system needs to be able to manage parallel streams of interaction, involving both personal and multi-user, collaborative multimodal Interaction Contexts. This requires the management of collective referential contexts, and mechanisms allowing to switch between individual and collective viewpoints.

This section presents the new concepts that were defined so as to answer these requirements, as well as the corresponding adjustments that were performed on our initial functional architecture (recalled in section 2).

---

8 This constraint was defined for simplicity sake, as the coexistence of personal and collective selections seemed difficult to manage from the user point of view.

Our approach relies on the concept of 'interaction agent' attached to each participant, and on the sharing of interaction agents in collaborative interaction situations. An interaction agent manages a particular interaction stream for a given participant; an interaction agent shared by several participants manages a collective interaction stream. Our concepts of collective multimodal Interaction Context and interaction agent are described and discussed hereafter; the functional architecture of a multi-participant multimodal interaction system is then presented.

4.2 Collective referential contexts; collaborative Interaction Context

As far as we can tell, referent computation in a n-logue context has not specifically been addressed in the literature. We were not able to find any piece of reported work on multi-user multimodal interaction addressing the referent computation issue; as to theoretical work, human discourse structure theories usually focus on monologue situations, although they often claim that their findings do apply to multi-party discourses. Significantly, (Grosz et al., 95) present their 'centering' theory of local coherence of discourse with the mention that: « [the examples] in this paper are single speaker texts. However, centering also applies to dialogue and multi-party conversations. Issues of the interaction between turn-taking and changes in centering status remain to be investigated. »

In the absence of specific grounding work, considering our lack of resources for theoretical investigations, our approach is to extend our mono-participant system through the simplistic hypothesis that a collective referential context will be built up and exploited in a way analogous to simple referential contexts - this is in line with the claims of e.g. (Grosz et al., 95). This first approach will clearly need to be revised later on as experimentations provide us with deeper insight into the specificities of multi-user references.

Collective Interaction Context

In our approach, a collective Interaction Context, as for a simple Interaction Context, consists in five different referential contexts. Adjustments to the collective dimension are the following:

1. topic: (referents computed for the preceding command; at the head of the topic, the 'preferential topic' contains the objects on which the command was applied).
   Collective dimension: the topic is built up from the last command uttered by a participant.
2. dialogue history: (list of the previous topics. A window of eight topics is maintained).
   Collective dimension: the dialogue history is built up from the ordered concatenation of the topics issued by the different participants, in the chronological order of their different utterances and/or actions.
3. VE-salient objects: (objects of the VE scene with a particular salience characteristic, without any relationship to the interaction history: e.g. a new object in the field of view or audition for the user, an object that was newly created or highlighted by someone else, etc.)
   The *collective dimension* does not fundamentally impact the VE-salient objects context, which mainly depends on the perception of the current user. However, specific salience criteria may be defined in relation to the specific sub-group of participants currently interacting together. This remains to be investigated.
4. perceptible objects: (the general set of the objects perceptible (visually or audio) by the participant at a given time).
   The *collective dimension*: does not impact the perceptible objects context, which only depends on the perception of the current user.
5. the environment: (the whole set of the VE application objects).
   No impact of the collective dimension.

A collective Interaction Context thus contains a data set gradually built up from the combination of the chronological interactions of the different participants in the considered group. Synchronous interactions are not a problem in COVEN: they are prevented by the process architecture of the dVISE system, which centralises interaction handling, automatically queues and orders interactions (see section 5). A collaborative Interaction Context however retains a participant-specific dimension since contexts 3 and 4 are dependent on the personal perception of this participant.

Collective referential gestures

In the current design, our choice was to attach a collective selection to a collective Interaction Context. This means that the selected set in a co-operative context is built and used by the group of participant; parallel individual selections are not managed. This choice seems coherent with the assumption of tight group collaboration; if individual selections are needed within a group collaboration, this will require negotiation within the group, and sequencing of the interaction of individuals.

In the same way, designation gestures are used collectively: a designation gesture may be performed by a different user from the one that utters the command.

4.3 Interaction agents

In a multi-participant context, the system needs to be able to manage several parallel interaction streams at a given moment, involving personal and/or group Interaction Contexts depending on the presence of participants working on their own or in sub-groups. Moreover, the system needs to support the dynamic switching of interaction focus from personal to collective, and vice-versa: a participant needs to be able to switch from a solitary work (however within a group) to a collaborative work within a given and chosen sub-group of 2 or more persons.

4.3.1 Concept of Interaction Agent

Our approach is to define the notion of *Interaction Agent* as a system component in charge with the support of a particular stream of multimodal interaction. An Interaction Agent monitors and manages the interaction of one participant or of a group of participants (an *interaction group*), whether speech-based or direct-manipulation-based. An Interaction Agent should indeed be able to seamlessly manage interactions independently of the interaction resources of the user:

- fully speech-based multimodal interaction, for users with spoken input facilities (microphone and speech recogniser system) available in addition to mouse and keyboard devices (or joystick in a non-desktop context).
- multimodal interaction with natural language support through keyboard input of NL commands, corresponding to users who do not have spoken input facilities but still wish to use natural language, in a mouse and keyboard desktop environment.
- non NL interaction, corresponding to users with basic input facilities only.

An Interaction Agent is attached an Interaction Context, collective or individual depending on the number of participants in its current stream of interaction.

4.3.2 Registering with and un-registering from an Interaction Agent

At initialisation time, a participant is provided with a personal Interaction Agent. Later on, if this participant decides to closely work with another participant, he will explicitly ask to share the Interaction Agent of this other participant, temporarily letting go of his personal agent.

Explicit registering as described above relies on a negotiation between the two participants; the owner of the Interaction Agent should be able to reject the registration of other participants if he is not ready to engage into close group work, or if he decides to quit group work and go back to personal activities. Another solution would be to instanciate specific 'group Interaction Agents', and to request that all participants to the group work explicitly register to the group Interaction Agent. This solution has the advantage that personal Interaction Agents are not drafted by other participants, and that the decision of a participant to leave the group does not threaten the existence of the interaction group. The disadvantage of this solution is that there is a risk for a proliferation of Interaction Agents. For the moment, the first solution is adopted; a migration to the second solution may be envisaged later on depending on confirmed usability issues.

A solution based on the implicit registering with an Interaction Agent depending on e.g. proximity would require the strong assumption that collective work prevails upon personal work, and that two persons close together will tend to work together. This does not seem realistic, especially in the current application context, which is why an explicit approach was preferred.

4.3.3 Interaction agent states

In the course of the management of interaction streams, an Interaction Agent may go through seven main states:

1. Idle state.
2. On-going user request specification: a user utterance is being received by the (speech recognition) system; or the user has begun a direct manipulation or menu interaction sequence.
3. Processing of the request is on-going: the end of the utterance was detected, and the system is processing the recognised utterance for extracting its meaning; or the system is collecting the different parameters specified by the user through menus.
4. Operation being performed: meaning was successfully extracted, and the corresponding operation is being performed.
5. Failure encountered: meaning could not successfully be extracted, and the system has abandoned the processing of the utterance; or partial meaning could be extracted, but the objects referred to in the utterance could not be identified. In the current prototype, this state is specific to the handling of natural language interaction as there is no possibility for such failures with direct manipulation or menus in the Interior Arrangement application.
6. Ambiguity encountered: meaning was extracted, but an ambiguity remains in the objects referred to by the utterance. A confirmation or negation is requested from the user (clarification sub-dialogue); this is typically the case where the axiology constraint was relaxed, as evoked in 3.1.3. This state is specific to the handling of natural language interaction.
7. End of execution: execution of the operation is over now, and the system is ready to handle new requests.

4.3.4 Representation issues

The issue of the representation of Interaction Agents within the VE needs to be considered when designing the application User Interface. There are several requirements for a form of representation, in particular: the need to provide feedback as to the type (individual or collective) and status of the interaction at a given moment; the need to indicate the existence or absence of collective interaction streams at a given moment, and to allow registering requests.

Depending on the addressed requirement, several types of representations may be envisaged, using the different complementary output communication channels: graphical (static or dynamic images), textual and audio. Within the Interior Arrangement scenario, our design makes use of iconic representations (representing the system global status), contextual text messages describing states and issues, audio messages and graphical indications on the scene objects (highlight of the computed referents in particular). This is fully described in D2.5.

4.4 Multi-user functional architecture - principles

The considerations and choices evoked in the previous sections lead to significantly modify our initial functional architecture of Figure II-1 (see section 2) since multiple Interaction Contexts and interaction managers need to be introduced. Multiple Interaction Contexts are needed so as to allow for the co-existence of individual and collective interaction focuses; multiple interaction managers are needed so as to provide for the possibly parallel interaction streams in the different interaction groups (including mono-user groups) - these interaction managers are now called Interaction Agents in our current terminology in an attempt to better convey the multiplicity dimension, together with the complete behaviour (from perception to action and expression) that they feature. Figure II-2 is a schema of the functional blocks involved in a multi-participant situation where two users (A and B) are engaged in group interaction while user C is working on his own.

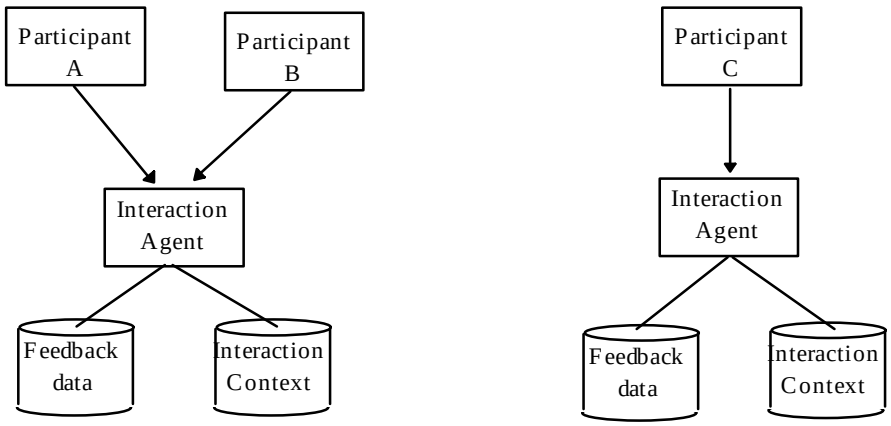

Figure II-2: multiple Interaction Contexts and agents

The impact of the multi-participant dimension on the system functional architecture is hereafter assessed and discussed. A general view of the new architecture with its main functional blocks is provided, which is further detailed in the following sections.

4.4.1 General view of the main functional components

Figure II-3 attempts at providing a general view of the main components in our functional architecture of a multi-participant multimodal system. The main blocks of the mono-user architecture of Figure II-1 are retained, while new and possibly multiple components are introduced so as to account for the multiple participants, Interaction Contexts and Agents.

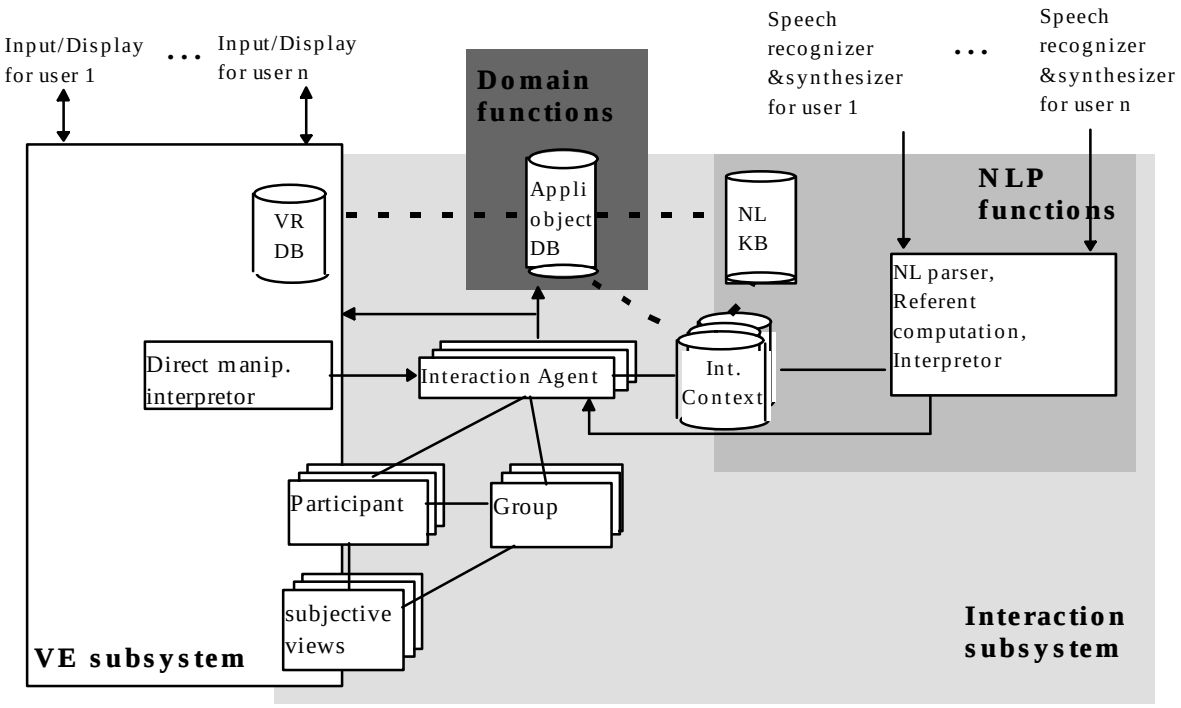

Figure II-3: Functional architecture of the multimodal CVE system

The domain subsystem is unchanged, except that ownership and access rights onto the application objects are now managed for each participant. All other subsystems are adjusted as detailed hereafter.

4.4.2 The VE subsystem

The multi-participant VE subsystem for the main part simply makes use of the multi-user functions of its underlying platform (dVS/dVISE), which provide for parallel input handling, DB update propagations and display update synchronisation.

In addition to these basic functions, the main requirements at VE level relate to the management of several participants, and of subjective views for each participant. Indeed the system must be able to receive and process input (direct manipulation, selection, designation gestures) from each separate participant, forwarding this interpreted input together with the identification of the participant to the Interaction subsystem.

The VE must also be able to process and render graphical, textual and audio output that may be different for each participant, typically when handling interaction feedback. This requires the implementation of subjective viewing mechanisms at VE system level.

4.4.3 The Interaction subsystem

As discussed earlier, this subsystem is profoundly impacted by the multi-participant dimension, with the introduction of the concept of Interaction Agent (see section 4.3) and the management of multiple Interaction Contexts. The concepts of participant and group of participant need also to be represented together with their management functions; finally the VE system subjective viewing mechanisms need application-level encapsulation functions to allow their specification and access in terms of application objects, participant and group.

4.4.4 The NLP subsystem

The core algorithms of the NLP subsystem remain unchanged. However, in order to take into account the multiplicity of participants and the possible sharing of Interaction Contexts, some adjustments are needed within the constraints of the system process architecture. This is evoked hereafter in section 5.

5. Implementation within the dVS environment

5.1 Organisation of the code modules

The functional architecture described in the previous sections was implemented as:

- A set of C++ classes within separate frameworks (dynamic libraries) for the Domain and Interaction subsystems, as well as the specific developments performed in the frame of the VE subsystem. The choice of C++ was motivated by the fact that the API of dVS/dVISE is in C or C++, and we favour an object-oriented design.

- A number of lisp modules implementing the NLP subsystem functions. The choice of lisp (acl) was motivated by the constraint we had to be able to re-use our existing parsing technology, which was developed in lisp.

Figures II-4 to II-6 below provide an overview of the different modules that were developed.

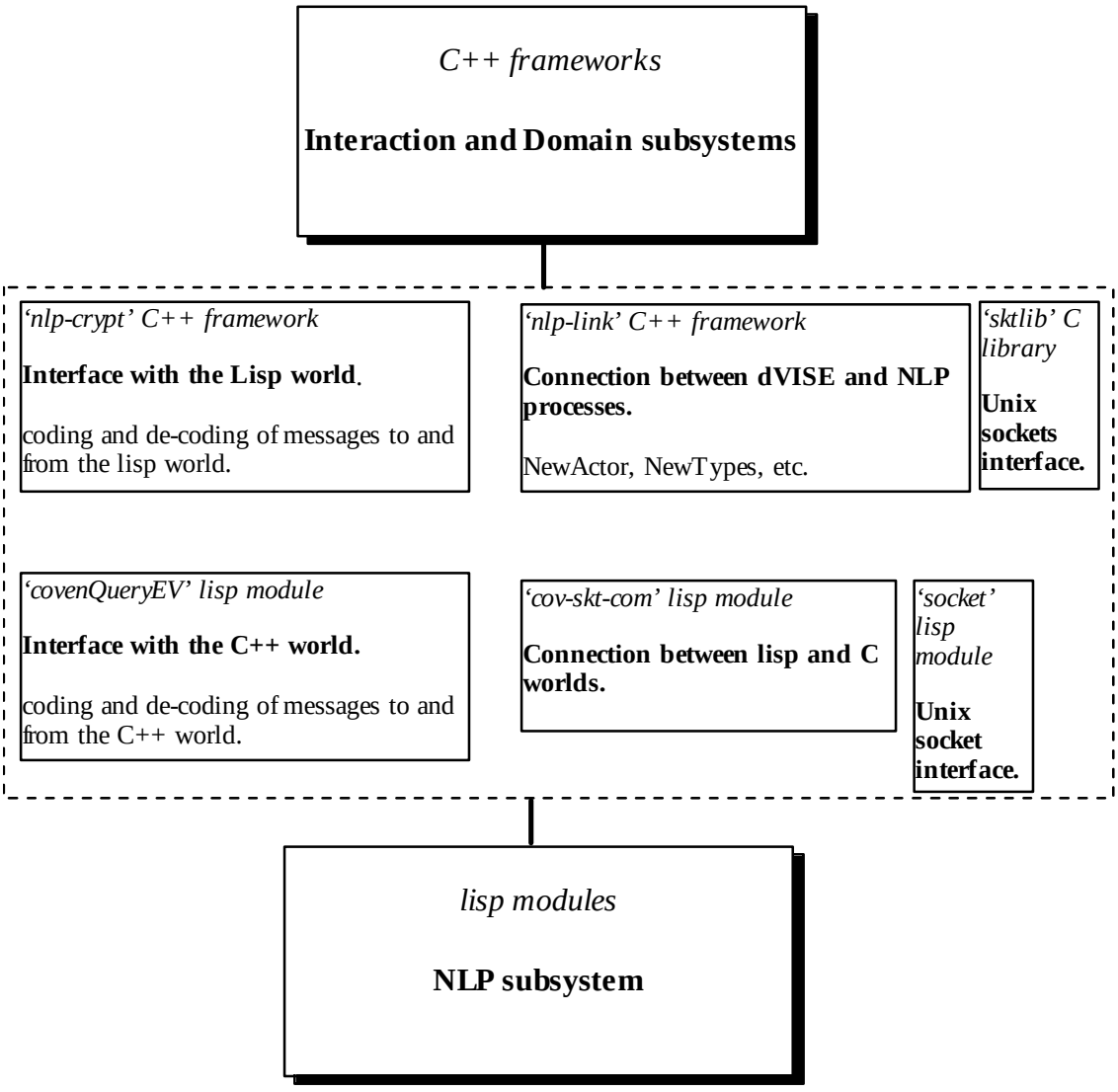

Figure II-4: General view of the module architecture

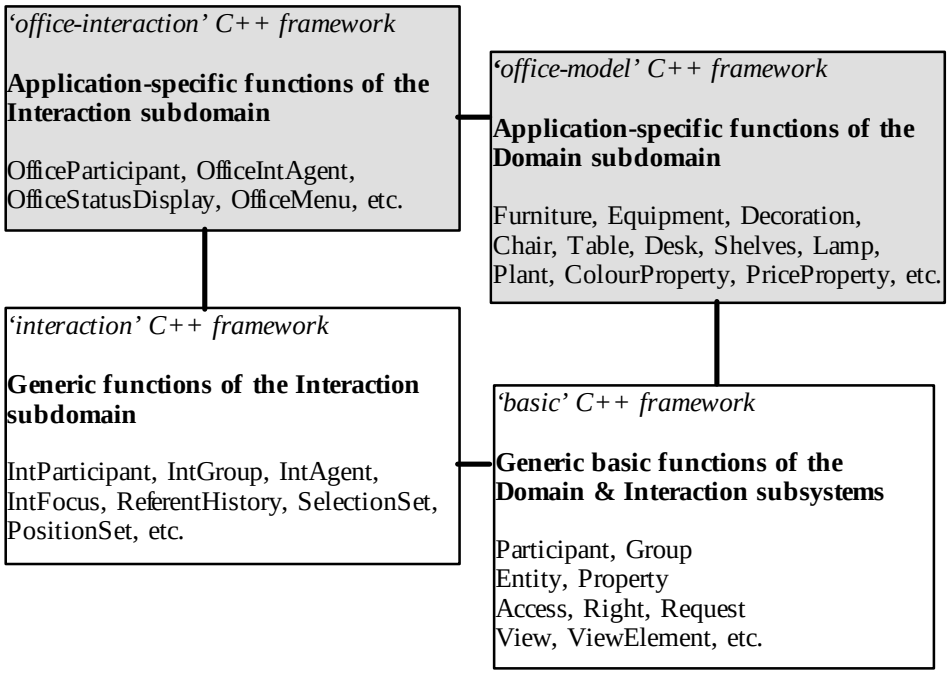

Figure II-5: Modules of the Interaction and Domain subsystems.
Generic modules appear as white boxes;
(Interior Arrangement) application-specific modules appear as grey boxes.

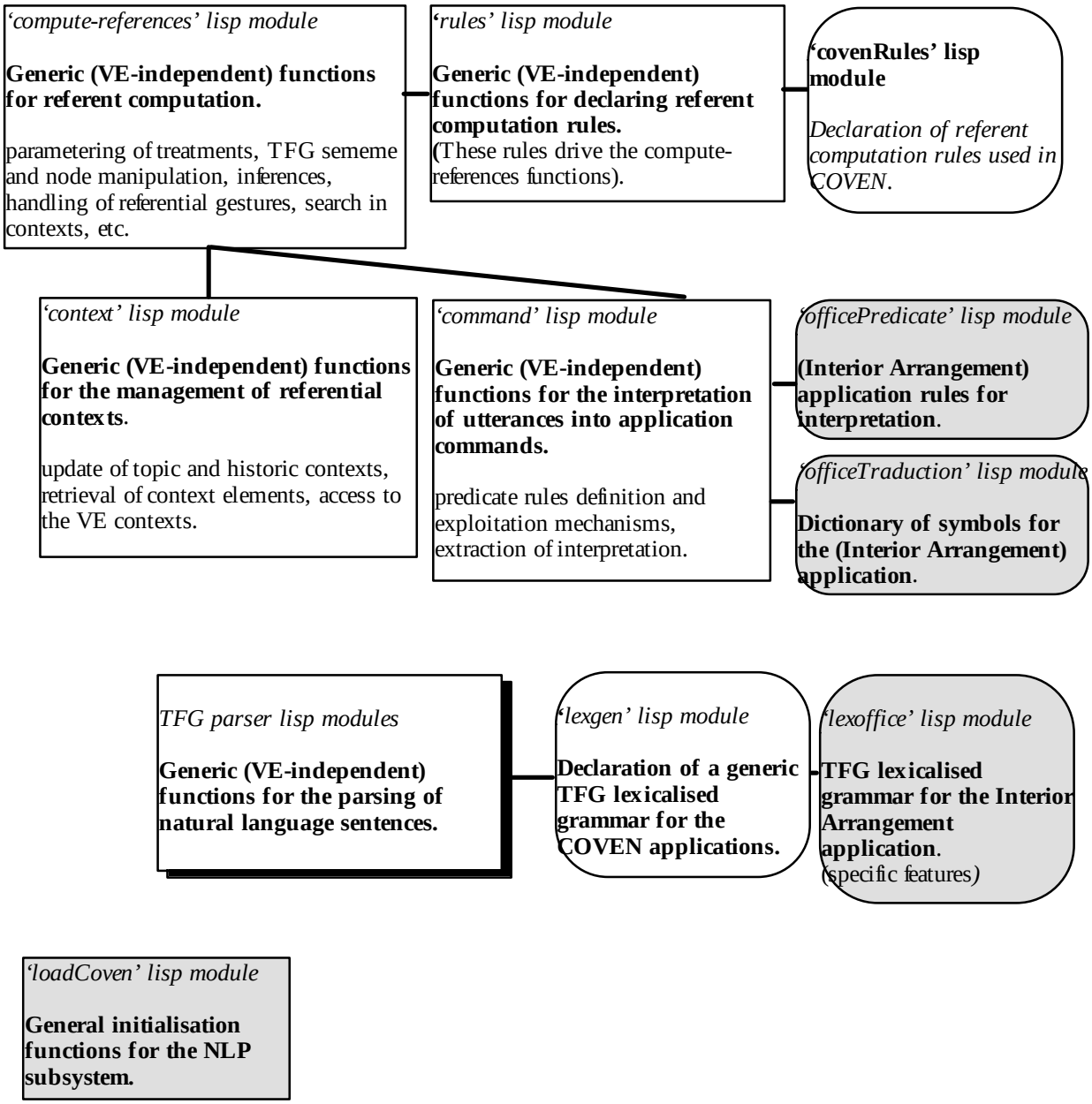

Figure II-6: Modules of the NLP subsystem.
Generic modules appear as white boxes;
(Interior Arrangement) application-specific modules appear as grey boxes.
Rules and data declarations appear in rounded boxes.

5.2 Application-specific developments

As can be seen from the module architecture presented above, a significant part of the developments addressed functions and data required for exploiting our multimodal interaction facilities within the COVEN Interior Arrangement Demonstrator.

At Domain subsystem level, the application-specific developments address the implementation of the interaction-independent actual functions and data of the specific application domain. At Interaction subsystem level, the application-specific developments address the implementation of the application User Interface (menu system and contextual interaction feedback in particular). A general description of these realisations is provided in the WP2 deliverable D2.5.

At NLP subsystem level, the application-specific developments involve the definition of sets of rules as well as linguistic resources specific to the Interior Arrangement sub-language interface:

- rules describing the application commands (actions and parameters), and the way to translate the semantic structures resulting from the parsing and referent computation into command structures interpretable by the Interaction subsystem.

- a lexicalised grammar describing the application interface sub-language from syntactic as well as semantic viewpoint. This required to augment our existing generic grammar with structures specifically required by the application (typically, application specific objects, adjectives and actions).

- a CFG grammar of the application sub-language, with maximum constraints. This grammar is needed as input to the speech recognition system used in the current prototype.

Elements of the CFG grammar and of the lexicalised grammar are provided in the Annex part of this Chapter.

5.3 General process architecture

The design of our process architecture was driven by a set of constraints as well as a search for flexibility and performance. Our constraints are that speech recognition and synthesis require dedicated processes, and that the NLP functions are written in lisp while the dVS world consists in C or C++ processes. Flexibility requirements are that it should be easy to add or remove a speech-enabled user in the process architecture. Performance requirements are that it should be possible to distribute the CPU-demanding NLP processes over the network.

Our multi-participant process architecture within the dVS/dVISE context is depicted in Figure II-7 below. The central dVISE application process holds the functions and data of the Domain and Interaction subsystems, with the dVS platform mechanisms providing the infrastructure for the distribution of and network access to these data and functions. For each speech-enabled participant, a set of related processes are instantiated:

- speech recognition process: this process is running the functions of a chosen commercial speech recognition system.
- the NLP process: this process holds the functions of the NLP subsystem. This is a lisp process as the reuse of NLP techniques previously developed in Thomson required to integrate lisp libraries.
- the NLP actor is a new C++ actor in the basic dVS runtime architecture. This actor is in charge with the communication between the NLP process and the dVISE application process (interaction functions).

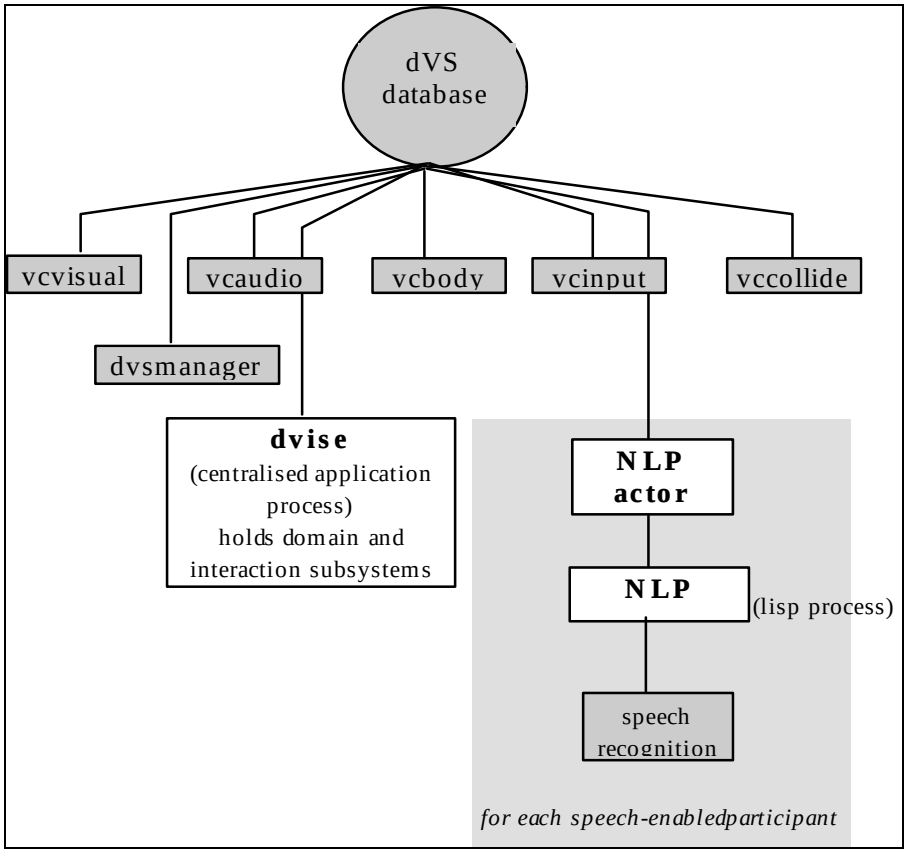

Figure II-7: the process architecture on the server site

(The new or touched processes are indicated as white boxes; grey boxes represent existing processes of the dVS architecture and the speech recognition system).

N.B.: a participant without spoken input facilities but willing to use natural language commands through keyboard input will use a configuration consisting in an adjusted NLP actor and a NLP process. A participant using dVISE (mainly, pointing) interaction exclusively will not launch any of these processes.

Communication between the speech recognition, NLP process and NLP actor processes directly uses Unix sockets.

The motivation for introducing the NLP actor[9] for connecting the NLP processes to the dVS world is that this allows to benefit from the flexibility of the dVS actor architecture: automatic launching from a preferences file; possible distribution over the network; shared-memory based communication with the dVISE actor (no explicit socket-based communication to program and manage).

9 N.B.: a solution integrating the NLP lisp process directly as a dVS actor will possibly be investigated depending on the performances of the current approach, and the availability of resources. This solution would suppress a process and a socket connection, but would require some work for implementing the interface between lisp and the C dVS-based functions; with respect to this, the current solution is considered as less costly in terms of adaptation work, but potentially less performing.

a Issues in the distribution of the NLP data

As evoked above, the Domain, Interaction and VE data are stored within the dVISE process which automatically ensures their distributed access over the network. This is however not the case of the specific NLP data, which raises a particular issue in the management of collective Interaction Contexts.

As evoked in section 3, for practical reasons the anaphoric contexts of the Interaction Context (topic and dialogue history) are maintained within the NLP module, with updates taking into account data provided by the Interaction subsystem. Due to the lisp implementation constraints (the NLP functions and data are lisp structures in a lisp process), distribution of the anaphoric contexts is thus not directly available. This is not a problem as long as personal Interaction Contexts are managed, with the NLP data in an NLP process remaining personal to one user only. Collective Interaction Contexts however require that several users share the same anaphoric contexts.

Rather than programming mechanisms to distribute collective anaphoric contexts, which would be inefficient, our approach is to centralise the NLP computations regarding a given collective Interaction Context onto the site of this collective Interaction Context. This means that the utterance issued by a given user (more precisely, the text string output by the speech recogniser on the site of this user) will be routed through the Interaction subsystem onto the NLP process in charge with the relevant collective Interaction Context. This approach is foreseen to be more efficient since only a short text string will be transferred over the network; the second advantage of this approach is that conflicts as to the update of the collective anaphoric contexts are avoided.

b Scope of parallelism

The current architecture design allows a degree of parallelism in the specification and analysis of commands, while actual command execution is strictly ordered and centralised in the dVISE process. Figure II-8 below develops an example scenario with two users issuing a multimodal speech-based command more or less at the same time; the actual limits of the parallelism are highlighted, with the central dVISE process strictly sequencing the computations related to direct manipulation, selection and designation (items 1 in the diagram), Interaction Context (items 2), interaction management and command execution (items 3).

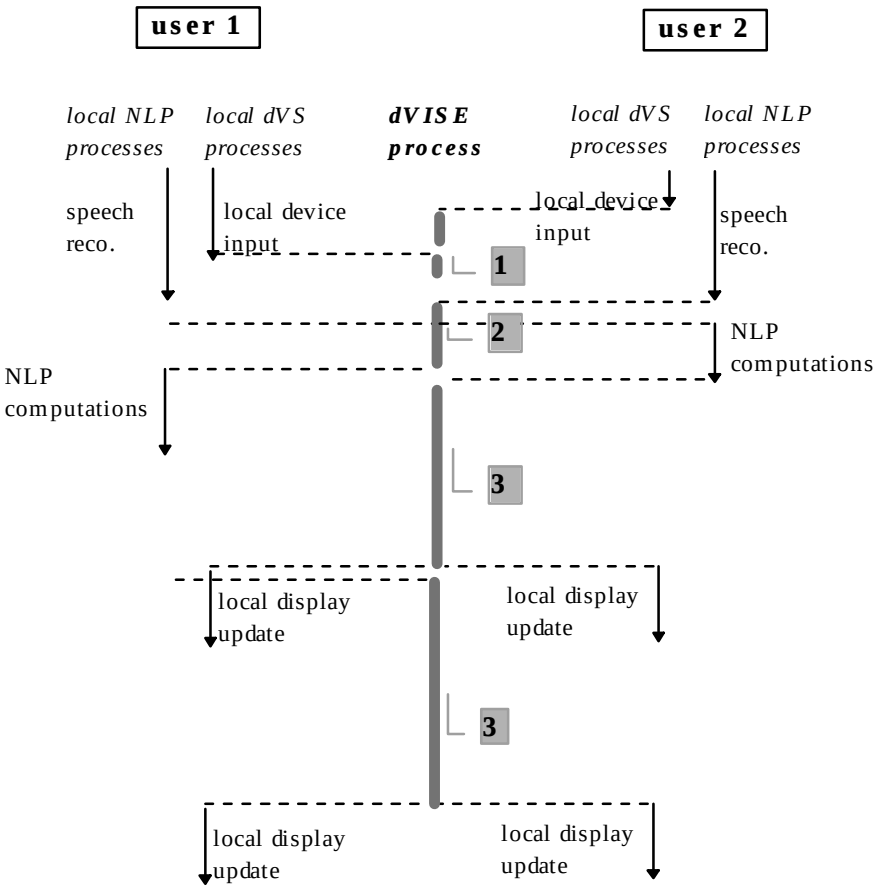

Figure II-8: example parallelism scenario

(N.B.: only the main exchanges between the local processes and the dVISE process are shown).

stage 1: handling of VE direct manipulation, designation gestures and menu interaction
stage 2: computation of and search within VE-related referential contexts
stage 3: actual execution of the command and interaction feedback

At a given point in a multi-participant session, stages 1, 2 and 3 of commands will possibly be interleaved at dVISE level, with commands at different degrees of completion being managed by the functions of the dVISE process, in different contexts (Interaction Agents and Interaction Contexts).

6. Summary and status

This chapter has presented the work that was undertaken so as to further develop and integrate our spoken natural language techniques within the COVEN Demonstrator.

We presented the results of our in-depth revision of the functional architecture. Our approach to referent computation was refined, with the consolidation of our general approach, the definition of referential contexts, of referential gestures, and the development of a referent computation algorithm. New concepts were designed to address the requirements of the multi-user dimension: co-operation modes were defined, and we introduced the concepts of collective referential contexts, interaction group and interaction agent.

The practical implementation of our architecture within the dVS environment was then presented, with an overview of the module organisation and the process architecture, and a general presentation of the application-specific developments.

At the time of writing, most of the elements described in this chapter are implemented, and are being further consolidated and tested. A restriction needs to be done on the available co-operation modes: only the 'loose' co-operation mode is currently operational and tested; the 'tight' co-operation mode should be completed and tested in the coming weeks. A second restriction concerns the 'VE salient objects' referential context, which could not be implemented in our resources constraints.

Further extensions to this work thus involve:

- the development of the 'VE salient objects' referential context, involving models and functions to exploit the spatial properties of objects within the 3D scene, and to detect the contrasts evoked in section 3.2.

- the completion and experimentation of the 'tight' co-operation mode, so as to assess the interests and limits of our approach (notions of interaction group and collective referential contexts in particular).

7. References

Joshi, A.K., Levy, L.S. and Takahashi, M. (1975). Tree adjunct grammars, *Journal of Computer and System Science*, 10, 1.

Gaiffe, B. (1992). *Référence et dialogue homme-machine: vers un modèle adapté au multimodal*. Ph.D. Thesis in Computer Science. Université Henri Poincaré, Nancy.

Grosz, B. (1977). *The representation and use of focus in dialogue understanding*. Technical Report 151, Artificial Intelligence Center, SRI International.

Grosz, B., Joshi, A. K., and Weinstein, S. (1995). *Centering: a framework for modelling the local coherence of discourse*. Report 95-01, The Institute for Research in Cognitive Science, University of Pennsylvania. January.

Halber, A. & Roussel, D. (1997). Parsing strategies for spoken language interfaces based on Lexicalised Tree Grammar. In *Proceedings of Eurospeech'97*, Rhodes, Greece, September.

Moulton, J. and Roberts, L. D. An AI module for reference based on perception. *Proceedings of the AAAI workshop on Integration of Natural Language and Vision Processing*. Ed. McKevitt, P. Seattle.

Normand, V., Pernel, D. and Bacconnet, B. Speech-based Multimodal Interaction in Virtual Environments: Research at the Thomson-CSF Corporate Research Laboratories. Submitted as a 'lab review' to *PRESENCE: Teleoperators and Virtual Environments.*

Pouteau, X. (1995). *Dialogue homme-machine multimodal: une communication 'naturelle' pour l'opérateur?* Ph.D. Thesis in Computer Science. Université Henri Poincaré, Nancy.

Sidner, C. (1986). Focusing in the comprehension of definite anaphora. In *Readings in Natural Language Processing*, pp. 363-394, Morgan-Kauffman.

Streit, M. (1997). Active and passive gestures - problems with the resolution of deictic and elliptic expressions in a multimodal system. Proceedings of Referring Phenomena in a Multimedia Context and their Computational Treatment, workshop of the ACL SIG on Multimedia Language Processing. Madrid. July. pp-44-51.

Trost, H., Heinz, W., Matiasek, J. and Buchberger, E., (1992). Datenbank-DIALOG and the Relevance of Habitability. *Proceedings of the Third Conference on Applied Natural Language Processing, Trento, Italy.*

8. Annex

8.1 Test scenario for referent computation

Anaphoric references
1. U: <speech> change all chairs to brown 2. U: <dir. manip.> change view point Check that all chairs of the environment were changed. 3. U: <speech> add three chairs. 4. U: <pointer selection> select a chair. 5. U: <speech + loc. desig.> take it there. Check that the selected chair was moved onto the designated location. 6. U: <speech> change the chair-s to blue. Change view point, and check that only the recently manipulated chairs were changed to blue. 7. U: <pointer selection> select three chair-s out of the two sets and move them. 8. U: <speech> change those chair-s to red. Change view point, and check that only the selected chairs were changed.
Search for contrasts (axiologies).
1. U: <pointer selection> select 3 chair-s, one red and two blue. 2. U: <dir. manip.> move the 3 chair-s. 3. U: <speech + pointer> place the red chair there. S: the red chair in the recent set is moved, axiology helped disambiguation. *The proper red chair could be identified as constrasting sets of chairs could be built within the interaction history.* *4.* U: <speech + pointer> take the blue chair there. S: message indicating ambiguity. *The blue chair could not be identified as constrasting sets of chairs could not be built in any of the managed referential contexts.* 5. U: move view point 6. U: <speech> remove the grey desk. S: message asking for confirmation - economy principle not respected. *The grey desk was identified in the general referential context of the global environment, as there is only one desk in the current scene, and it is grey. However, since the economy principle was not respected, the system asked to user to confirm her command.* 7. U: <speech> yes.
Perception-based, direct references
1. U: <dir. manip.> change view point so as to visualise ONE table and mel-s-room. 2. U: <speech> move the table to mel-s-room Check that one non ambiguous table was moved, i.e. the one in the field of perception. 3. U: <dir. manip.> change view point so as to visualise the two chairs. 4. U: <speech> suppress these chair. Check that only the two chair-s were suppressed. *Both the table and the chairs could be identified based on perception criteria, after a search in the*

anaphoric referential contexts resulted in a failure.

8.2 A small sub-set of the lexicalised grammar of the Interior Arrangement application

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;    ENTITY    OBJECTS      ;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(lexie 'furniture
      (make-lexie :arbre '(N (furniture ))
                  :fonctions '((N . :object))
                  :ancres '(furniture )
                  :gram '((:num . :sing))
                 :reco '(piece-of-furniture)
                  :sememe '((:det)(:material)(:colour)
                      (:entity . :object)(tr-plural)
                      (:object . :furniture))
       ))

(lexie 'chair
      (make-lexie :arbre '(N (chair))
                  :fonctions '((N . :object))
                  :ancres '(chair)
                  :gram '((:num . :sing))
                 :reco '(seat)
                  :sememe '((:entity . :object)(tr-plural)
                      (:object . :chair)(:type)
                      (:obj-property)(:det))
       ))

(lexie 'table
      (make-lexie :arbre '(N (table))
                  :fonctions '((N . :object))
                  :ancres '(table)
                  :gram '((:num . :sing))
                  :sememe '((:entity . :object)(tr-plural)
                      (:object . :table)(:obj-property)(:det))
       ))

(lexie 'desk
      (make-lexie :arbre '(N (desk))
                  :fonctions '((N . :object))
                  :ancres '(desk)
                  :gram '((:num . :sing))
                  :sememe '((:entity . :object)(tr-plural)(:obj-property)(:det)
                      (:object . :desk))
       ))

(lexie 'lamp
      (make-lexie :arbre '(N (lamp))
                  :fonctions '((N . :object))
                  :ancres '(lamp)
                  :gram '((:num . :sing))
                  :sememe '((:entity . :object)(tr-plural)
                      (:object . :lamp)(:obj-property) )
       ))

(lexie 'shelvesN
      (make-lexie :arbre '(N (shelves))
                  :fonctions '((N . :object))
                  :ancres '(shelves)
                  :gram '((:num . :pl))
                  :sememe '(((:det . :collective)(:det))
                      (:obj-property)(:collective)(:number . :one)
                      (:object . :shelves))
       ))

...
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;; MATERIAL values ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(lexie 'plastic
      (make-lexie :arbre '(N* (N (plastic)))
                  :ancres '(plastic)
                  :sememe '((:property . :obj-property)(:material . :plastic)
                          (:obj-property . :material))
       ))

;
(lexie 'fabric
      (make-lexie :arbre '(N* (N (fabric)))
                  :ancres '(fabric)
                  :sememe '((:property . :obj-property)(:material . :fabric)
                          (:obj-property . :material))
       ))

(lexie 'wood
      (make-lexie :arbre '(N (wood))
                  :ancres '(wood)
                  :sememe '((:property . :obj-property)
                      (:material . :wood)   (tr-en)
                          (:obj-property . :material))
       ))

(lexie 'metal
      (make-lexie :arbre '(N* (N (metal)))
                  :ancres '(metal)
                  :sememe '((:property . :obj-property)(:material . :metal)
                          (:obj-property . :material))
       ))

(lexie 'marble
      (make-lexie :arbre '(N* (N (marble)))
                  :ancres '(marble)
                  :sememe '((:property . :obj-property)(:material . :marble)
                             (:obj-property . :material))
       ))

;;;;;;;;;;;;;;;;;;;;;;;;;;; STYLE values ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(lexie 'swivel
      (make-lexie :arbre '(N* (A (swivel)))
                  :ancres '(swivel)
                  :sememe '((:property . :obj-property)(:type . :swivel)
                          (:obj-property . :type))
       ))

(lexie 'simple
      (make-lexie :arbre '(N* (A (simple)))
                  :ancres '(simple)
                  :sememe '(((:property . :obj-property)
                          (:property . :repr-property))
                          (:type . (:simple))
                          ((:repr-property . :type)
                          (:obj-property . :type)))
       ))

(lexie 'fixed
      (make-lexie :arbre '(N* (A (fixed)))
                  :ancres '(fixed)
                  :sememe '((:property . :obj-property)(:type . :simple)
                          (:obj-property . :type))
       ))
```

8.3 Elements of the CFG grammar of the Interior Arrangement application

8.3.1 View of the ‘high level’ derivations of the CFG

Ce graphisme de format Encapsulated Postcript (EPS) ne comprend pas d'aperçu
L'impression sera bonne sur une imprimante Postscript.
Nom du fichiSCREEN1.DMP
Titre :
Créateur : xgrabsc
Date de créatio Thu Jan 15 17:38:54 1998

Ce graphisme de format Encapsulated Postcript (EPS) ne comprend pas d'aperçu.
L'impression sera bonne sur une imprimante Postscript.
Nom du fichiSCREEN2.DMP
Titre :
Créateur : xgrabsc
Date de créatio Thu Jan 15 17:40:41 1998

Ce graphisme de format Encapsulated Postcript (EPS) ne comprend pa
L'impression sera bonne sur une imprimante Postscript.
Nom du fichiSCREEN3.DMP
Titre :
Créateur : xgrabsc
Date de créatio Thu Jan 15 17:41:13 1998

Ce graphisme de format Encapsulated Postcript (EPS) ne comprend p
L'impression sera bonne sur une imprimante Postscript.
Nom du fichiSCREEN4.DMP
Titre :
Créateur : xgrabsc
Date de créatio Thu Jan 15 17:41:55 1998

Ce graphisme de format Encapsulated Postcript (EPS) ne comprend pas d'aperçu.
L'impression sera bonne sur une imprimante Postscript.
Nom du fichiscreen5.dmp
Titre :
Créateur : xgrabsc
Date de créatio Thu Jan 15 17:43:05 1998

8.3.2 Full CFG, native format

```
S
S -> # . #_1_0
#_1_0 -> phrase . phrase_2_0
phrase_2_0 -> #

phrase
phrase -> ACTION
phrase -> CONFIRM
phrase -> UI-ACTION

ACTION
ACTION -> I-WANT-TO . I-WANT-TO_1_4
I-WANT-TO_1_4 -> ACT
ACTION -> ACT

I-WANT-TO
I-WANT-TO -> let . let_1_6
let_1_6 -> me
I-WANT-TO -> I . I_1_7
I_1_7 -> d . d_2_7
d_2_7 -> like . like_3_7
like_3_7 -> to
I_1_7 -> would . would_2_8
would_2_8 -> like . like_3_8
like_3_8 -> to
I_1_7 -> want . want_2_9
want_2_9 -> to

ACT
ACT -> CREATE-ACT
ACT -> CHANGE-ACT
ACT -> MOVE-ACT
ACT -> DELETE-ACT
ACT -> BODY-ACT
ACT -> COMM-ACT

CREATE-ACT
CREATE-ACT -> V-CREATE . V-CREATE_1_16
V-CREATE_1_16 -> UN-N-OBJ . UN-N-OBJ_2_16
UN-N-OBJ_2_16 -> AT-LOC
UN-N-OBJ_2_16 ->

V-CREATE
V-CREATE -> add
V-CREATE -> create

CHANGE-ACT
CHANGE-ACT -> V-CHANGE . V-CHANGE_1_20
V-CHANGE_1_20 -> the . the_2_20
the_2_20 -> N-COLOR . N-COLOR_3_20
N-COLOR_3_20 -> of . of_4_20
of_4_20 -> N-ITEM0 . N-ITEM0_5_20
N-ITEM0_5_20 -> to . to_6_20
to_6_20 -> A-COLOR
the_2_20 -> N-MATERIAL . N-MATERIAL_3_21
N-MATERIAL_3_21 -> of . of_4_21
of_4_21 -> N-ITEM0 . N-ITEM0_5_21
N-ITEM0_5_21 -> to . to_6_21
to_6_21 -> A-MATERIAL
V-CHANGE_1_20 -> N-ITEM . N-ITEM_2_22
N-ITEM_2_22 -> to . to_3_22
to_3_22 -> A-PROP-NTY
V-CHANGE_1_20 -> N-ITEM-TY . N-ITEM-TY_2_23
N-ITEM-TY_2_23 -> to . to_3_23
to_3_23 -> A-TYPE
CHANGE-ACT -> make . make_1_24
make_1_24 -> N-ITEM . N-ITEM_2_24
N-ITEM_2_24 -> A-PROP-NTY
make_1_24 -> N-ITEM-TY . N-ITEM-TY_2_25
N-ITEM-TY_2_25 -> A-TYPE
CHANGE-ACT -> paint . paint_1_26
paint_1_26 -> N-ITEM . N-ITEM_2_26
N-ITEM_2_26 -> to . to_3_26
to_3_26 -> A-COLOR
N-ITEM_2_26 -> A-COLOR

V-CHANGE
V-CHANGE -> change
V-CHANGE -> set

N-MATERIAL
N-MATERIAL -> material

N-COLOR
N-COLOR -> colour

MOVE-ACT
MOVE-ACT -> V-MOVE . V-MOVE_1_32
V-MOVE_1_32 -> N-ENTITY . N-ENTITY_2_32
N-ENTITY_2_32 -> TO-LOC

V-MOVE
V-MOVE -> put
V-MOVE -> transport
V-MOVE -> move
V-MOVE -> take
V-MOVE -> fly

DELETE-ACT
DELETE-ACT -> V-DELETE . V-DELETE_1_38
V-DELETE_1_38 -> N-ITEM

V-DELETE
V-DELETE -> remove
V-DELETE -> suppress
V-DELETE -> delete

BODY-ACT
BODY-ACT -> V-GESTURE . V-GESTURE_1_42
V-GESTURE_1_42 -> to . to_2_42
to_2_42 -> N-PERSON
V-GESTURE_1_42 ->
BODY-ACT -> V-GO . V-GO_1_44
V-GO_1_44 -> TO-LOC
V-GO_1_44 -> to . to_2_45
to_2_45 -> N-PERSON

V-GESTURE
V-GESTURE -> bow
V-GESTURE -> wave

V-GO
V-GO -> fly
V-GO -> go

COMM-ACT
COMM-ACT -> V-TALK1 . V-TALK1_1_50
V-TALK1_1_50 -> with . with_2_50
with_2_50 -> N-PERSON
V-TALK1_1_50 ->
COMM-ACT -> V-TALK2 . V-TALK2_1_52
V-TALK2_1_52 -> to . to_2_52
to_2_52 -> N-PERSON
COMM-ACT -> join . join_1_53
join_1_53 -> N-PERSON
join_1_53 -> in

V-TALK1
V-TALK1 -> talk
V-TALK1 -> discuss
V-TALK1 -> speak

V-TALK2
V-TALK2 -> talk
V-TALK2 -> speak

CONFIRM
CONFIRM -> yes
CONFIRM -> ok
CONFIRM -> agreed
CONFIRM -> right
CONFIRM -> no
CONFIRM -> don-t
CONFIRM -> cancel . cancel_1_66
```

```
cancel_1_66 -> N-COMMAND
CONFIRM -> I-WANT-TO . I-WANT-TO_1_67
I-WANT-TO_1_67 -> cancel . cancel_2_67
cancel_2_67 -> N-COMMAND

N-COMMAND
N-COMMAND -> command
N-COMMAND -> the . the_1_69
the_1_69 -> command
N-COMMAND -> my . my_1_70
my_1_70 -> command

UI-ACTION
UI-ACTION -> I-WANT-TO . I-WANT-TO_1_71
I-WANT-TO_1_71 -> UI-ACT
UI-ACTION -> UI-ACT

UI-ACT
UI-ACT -> SELECT-ACT
UI-ACT -> VIEW-ACT

SELECT-ACT
SELECT-ACT -> V-SELECT . V-SELECT_1_75
V-SELECT_1_75 -> N-ENTITY

V-SELECT
V-SELECT -> select
V-SELECT -> pick

VIEW-ACT
VIEW-ACT -> ADJUST-ACT
VIEW-ACT -> SET-V-ACT

ADJUST-ACT
ADJUST-ACT -> V-ADJUST . V-ADJUST_1_80
V-ADJUST_1_80 -> N-ENTITY-V
ADJUST-ACT -> set . set_1_81
set_1_81 -> N-ENTITY-V . N-ENTITY-V_2_81
N-ENTITY-V_2_81 -> back . back_3_81
back_3_81 -> to . to_4_81
to_4_81 -> A-STATE
to_4_81 -> A-STATE1 . A-STATE1_5_82
A-STATE1_5_82 -> N-VIEWING
N-ENTITY-V_2_81 -> to . to_3_83
to_3_83 -> A-STATE1 . A-STATE1_4_83
A-STATE1_4_83 -> N-VIEWING

V-ADJUST
V-ADJUST -> hide
V-ADJUST -> dim
V-ADJUST -> highlight
V-ADJUST -> emphasise
V-ADJUST -> display
V-ADJUST -> show

N-VIEWING
N-VIEWING -> view
N-VIEWING -> viewing
N-VIEWING -> representation
N-VIEWING -> display
N-VIEWING -> visualise-ation

SET-V-ACT
SET-V-ACT -> reset . reset_1_95
reset_1_95 -> N-VIEW

N-VIEW
N-VIEW -> DET-VIEW . DET-VIEW_1_96
DET-VIEW_1_96 -> A-V-TYPE . A-V-TYPE_2_96
A-V-TYPE_2_96 -> N-VIEWING
DET-VIEW_1_96 -> N-VIEWING

DET-VIEW
DET-VIEW -> the
DET-VIEW -> this
DET-VIEW -> my
DET-VIEW ->

A-V-TYPE
A-V-TYPE -> default
A-V-TYPE -> current
A-V-TYPE -> main
A-V-TYPE -> normal

N-ENTITY
N-ENTITY -> N-PERSON
N-ENTITY -> N-ITEM

N-ENTITY-L
N-ENTITY-L -> N-PERSON
N-ENTITY-L -> N-PLACE
N-ENTITY-L -> N-ITEM-L

N-ENTITY-V
N-ENTITY-V -> N-ITEM
N-ENTITY-V -> N-PERSON-V

N-PERSON
N-PERSON -> NAME-N-PERS
N-PERSON -> PRON-N-PERS

N-PERSON-V
N-PERSON-V -> NAME-N-PERS

NAME-N-PERS
NAME-N-PERS -> nicolas
NAME-N-PERS -> vero
NAME-N-PERS -> jean-benoit

PRON-N-PERS
PRON-N-PERS -> me

UN-N-OBJ
UN-N-OBJ -> UN-DET . UN-DET_1_120
UN-DET_1_120   ->   A-PRS1-NTY   .   A-PRS1-
NTY_2_120
A-PRS1-NTY_2_120 -> FURN-OBJ
UN-DET_1_120 -> FURN-OBJ
UN-DET_1_120 -> A-PRS1-TY . A-PRS1-TY_2_122
A-PRS1-TY_2_122 -> TY-OBJ

N-ITEM
N-ITEM -> it
N-ITEM -> them
N-ITEM -> N-ITEM0

N-ITEM-TY
N-ITEM-TY -> it
N-ITEM-TY -> them
N-ITEM-TY -> N-ITEM1
N-ITEM-TY -> N-IT2-TY

N-ITEM-L
N-ITEM-L -> it
N-ITEM-L -> them
N-ITEM-L -> N-ITEM0-L

N-ITEM-ON
N-ITEM-ON -> it
N-ITEM-ON -> N-ITEM0-ON

N-ITEM0
N-ITEM0 -> N-ITEM1
N-ITEM0 -> N-ITEM2

N-ITEM0-L
N-ITEM0-L -> N-ITEM1
N-ITEM0-L -> N-ITEM2-L

N-ITEM0-ON
N-ITEM0-ON -> N-ITEM1-ON
N-ITEM0-ON -> N-ITEM2-ON

N-ITEM1
N-ITEM1 -> this . this_1_141
this_1_141 -> one
```

```
this_1_141 ->
N-ITEM1 -> that . that_1_143
that_1_143 -> one
that_1_143 ->
N-ITEM1 -> these . these_1_145
these_1_145 -> one-s
these_1_145 ->
N-ITEM1 -> those . those_1_147
those_1_147 -> one-s
those_1_147 ->

N-ITEM1-ON
N-ITEM1-ON -> this . this_1_149
this_1_149 -> one
this_1_149 ->
N-ITEM1-ON -> that . that_1_151
that_1_151 -> one
that_1_151 ->

N-ITEM2
N-ITEM2 -> DEF-DET1 . DEF-DET1_1_153
DEF-DET1_1_153 -> PRS-ALL-OBJ
DEF-DET1_1_153 -> ALL-OBJ

N-ITEM2-L
N-ITEM2-L -> DEF-DET1-L . DEF-DET1-L_1_155
DEF-DET1-L_1_155 -> PRS-ALL-OBJ
DEF-DET1-L_1_155 -> ALL-OBJ

N-ITEM2-ON
N-ITEM2-ON -> DEF-DET-L . DEF-DET-L_1_157
DEF-DET-L_1_157 -> ALL-OBJ-ON
DEF-DET-L_1_157 -> PRS-ALL-OBJ-ON

N-IT2-TY
N-IT2-TY -> DEF-DET1 . DEF-DET1_1_159
DEF-DET1_1_159 -> PRS-ALL-OBJ-TY
DEF-DET1_1_159 -> ALL-OBJ-TY

ALL-OBJ
ALL-OBJ -> FURN-OBJ
ALL-OBJ -> GEN-OBJ

ALL-OBJ-ON
ALL-OBJ-ON -> FURN-OBJ-ON
ALL-OBJ-ON -> GEN-OBJ

ALL-OBJ-TY
ALL-OBJ-TY -> TY-OBJ
ALL-OBJ-TY -> GEN-OBJ

PRS-ALL-OBJ
PRS-ALL-OBJ -> A-PRS . A-PRS_1_167
A-PRS_1_167 -> ALL-OBJ
PRS-ALL-OBJ -> A-PRS-TY . A-PRS-TY_1_168
A-PRS-TY_1_168 -> ALL-OBJ-TY

PRS-ALL-OBJ-TY
PRS-ALL-OBJ-TY -> A-PRS . A-PRS_1_169
A-PRS_1_169 -> ALL-OBJ-TY
PRS-ALL-OBJ-TY -> A-PRS-TY . A-PRS-TY_1_170
A-PRS-TY_1_170 -> ALL-OBJ-TY

PRS-ALL-OBJ-ON
PRS-ALL-OBJ-ON -> A-PRS . A-PRS_1_171
A-PRS_1_171 -> ALL-OBJ-ON

TY-OBJ
TY-OBJ -> chair
TY-OBJ -> seat

FURN-OBJ
FURN-OBJ -> FURN-OBJ2
FURN-OBJ -> TY-OBJ

FURN-OBJ2
FURN-OBJ2 -> table
FURN-OBJ2 -> desk
FURN-OBJ2 -> lamp
FURN-OBJ2 -> file . file_1_179
file_1_179 -> cabinet
FURN-OBJ2 -> plant
FURN-OBJ2 -> phone
FURN-OBJ2 -> telephone
FURN-OBJ2 -> computer
FURN-OBJ2 -> pc
FURN-OBJ2 -> workstation
FURN-OBJ2 -> shelves
FURN-OBJ2 -> book . book_1_187
book_1_187 -> shelves

FURN-OBJ-ON
FURN-OBJ-ON -> table
FURN-OBJ-ON -> desk
FURN-OBJ-ON -> file . file_1_190
file_1_190 -> cabinet
FURN-OBJ-ON -> shelves
FURN-OBJ-ON -> book . book_1_192
book_1_192 -> shelves

GEN-OBJ
GEN-OBJ -> item
GEN-OBJ -> object

DEF-DET1
DEF-DET1 -> DEF-DET
DEF-DET1 -> DEF-DET-P

DEF-DET1-L
DEF-DET1-L -> DEF-DET-L
DEF-DET1-L -> DEF-DET-P-L

DEF-DET
DEF-DET -> the
DEF-DET -> this
DEF-DET -> that
DEF-DET -> every
DEF-DET -> each
DEF-DET -> my

DEF-DET-L
DEF-DET-L -> the
DEF-DET-L -> this
DEF-DET-L -> that
DEF-DET-L -> my

DEF-DET-P
DEF-DET-P -> DEF-DET-P2 . DEF-DET-P2_1_209
DEF-DET-P2_1_209 -> NUMBER-P
DEF-DET-P2_1_209 ->
DEF-DET-P -> DEF-DET-P3

DEF-DET-P-L
DEF-DET-P-L   ->   DEF-DET-P2   .   DEF-DET-
P2_1_212
DEF-DET-P2_1_212 -> NUMBER-P
DEF-DET-P2_1_212 ->

DEF-DET-P3
DEF-DET-P3 -> all . all_1_214
all_1_214 -> the
all_1_214 ->

DEF-DET-P2
DEF-DET-P2 -> these
DEF-DET-P2 -> those

UN-DET
UN-DET -> NUMBER
UN-DET -> a
UN-DET -> an

NUMBER
NUMBER -> one
NUMBER -> NUMBER-P
```

```
NUMBER-P
NUMBER-P -> two
NUMBER-P -> three
NUMBER-P -> four
NUMBER-P -> five

A-PRS
A-PRS -> A-COLOR . A-COLOR_1_227
A-COLOR_1_227 -> A-MATERIAL
A-COLOR_1_227 ->
A-PRS -> A-MATERIAL
A-PRS -> A-STATE1 . A-STATE1_1_230
A-STATE1_1_230 -> A-COLOR
A-STATE1_1_230 -> A-MATERIAL
A-STATE1_1_230 ->

A-PRS-TY
A-PRS-TY -> A-TYPE
A-PRS-TY -> A-COLOR . A-COLOR_1_234
A-COLOR_1_234 -> A-TYPE
A-PRS-TY -> A-MATERIAL . A-MATERIAL_1_235
A-MATERIAL_1_235 -> A-TYPE
A-PRS-TY -> A-STATE1 . A-STATE1_1_236
A-STATE1_1_236 -> A-TYPE

A-PRS1-NTY
A-PRS1-NTY -> A-MATERIAL
A-PRS1-NTY -> A-COLOR . A-COLOR_1_238
A-COLOR_1_238 -> A-MATERIAL
A-COLOR_1_238 ->

A-PRS1-TY
A-PRS1-TY -> A-TYPE
A-PRS1-TY -> A-COLOR . A-COLOR_1_241
A-COLOR_1_241 -> A-TYPE
A-PRS1-TY -> A-MATERIAL . A-MATERIAL_1_242
A-MATERIAL_1_242 -> A-TYPE

A-PROP-NTY
A-PROP-NTY -> A-COLOR
A-PROP-NTY -> A-MATERIAL
A-PROP-NTY -> A-STATE1

A-COLOR
A-COLOR -> blue
A-COLOR -> red
A-COLOR -> yellow
A-COLOR -> grey
A-COLOR -> brown
A-COLOR -> black

A-MATERIAL
A-MATERIAL -> wood-en
A-MATERIAL -> plastic
A-MATERIAL -> metal
A-MATERIAL -> fabric

A-TYPE
A-TYPE -> simple
A-TYPE -> fixed
A-TYPE -> swivel

A-STATE
A-STATE -> normal
A-STATE -> default
A-STATE -> A-S-STATE

A-S-STATE
A-S-STATE -> hidden
A-S-STATE -> highlight
A-S-STATE -> highlighted
A-S-STATE -> dimmed
A-S-STATE -> transparent
A-S-STATE -> bright

A-STATE1
A-STATE1 -> hidden
A-STATE1 -> highlighted
A-STATE1 -> dimmed
A-STATE1 -> transparent

AT-LOC
AT-LOC -> ADV-PLACE
AT-LOC -> in . in_1_273
in_1_273 -> N-PLACE
AT-LOC -> REL-PLACE

TO-LOC
TO-LOC -> ADV-PLACE
TO-LOC -> to . to_1_276
to_1_276 -> N-PLACE
TO-LOC -> in . in_1_277
in_1_277 -> N-PLACE
TO-LOC -> REL-PLACE

ADV-PLACE
ADV-PLACE -> there
ADV-PLACE -> here
ADV-PLACE -> over . over_1_281
over_1_281 -> there

N-PLACE
N-PLACE -> the . the_1_282
the_1_282 -> PRED-PLACE
N-PLACE -> mel-s-room

PRED-PLACE
PRED-PLACE -> meeting-room
PRED-PLACE -> kitchen
PRED-PLACE -> library
PRED-PLACE -> main . main_1_287
main_1_287 -> door
PRED-PLACE -> main-room
PRED-PLACE -> stair-s

REL-PLACE
REL-PLACE -> on . on_1_290
on_1_290 -> N-ITEM-ON
REL-PLACE -> ADV-NEAR . ADV-NEAR_1_291
ADV-NEAR_1_291 -> N-ENTITY-L

ADV-NEAR
ADV-NEAR -> beside
ADV-NEAR -> near
ADV-NEAR -> close . close_1_294
close_1_294 -> to
ADV-NEAR -> around
```