Gandalf: Humanoid One

9.

Using Ymir as a foundation, a character can be built by specifying perceptual, decision and behavior modules, plus the specific procedures needed for data and internal computations. In this chapter we will present the first character designed in Ymir, *Gandalf*. Gandalf (lower right corner, this page) is a “straw-cartoon”—it has been given the minimal set of modules necessary for face-to-face interaction.[1] It shows that Ymir is a sufficient and appropriate platform for developing communicative characters. It also points to issues that need to be addressed further in future research. These will be discussed in the conclusion chapter, page 185.

The computer ... can give you the exact mathematical design, but what’s missing is the eyebrows.

—Frank Zappa

Why “Gandalf”?

As told in the Icelandic Sagas [Sturluson 1300~1325], after the gods Ó›inn, Vili and Vé had killed the giant Ymir (pronounced e-mir), and used his carcas to make the heaven and the earth, worms sprung to life in the newly created soil. The gods subsequently changed these worms to dwarfs, 63 of them to be exact. Gandalf—or *Gandalfr*—was one of these worms-turned-to-dwarf creatures (predating J.R.R. Tolkien’s [1937] character of the same name by about a millenium :-)

9.1 The Gandalf Prototype: Overview

Gandalf is a collection of control rules implemented in Ymir Alpha, in Lisp (Chapter 8.), plus the necessary hardware and support software for sensing, acting and embodiment.

9.1.1 Prototype Setup

Gandalf appears to users on its own monitor (Figure 9-1). A model of the solar system (Figure 9-2) appears on a large screen in front of the user. Gandalf is an expert in the solar system and can tell users facts about the planets. It can also travel to the planets, zoom in and out, and start and stop the planets’ moons in their orbits. The speech recognition used is a speaker-independent, continuous speech recognizer from BBN called HARK [BBN 1993]. This recognizer is grammar based and has

1. See also Chapter 10.3.2, page 162 about analysis and action schemes for responding to deictic gestures.

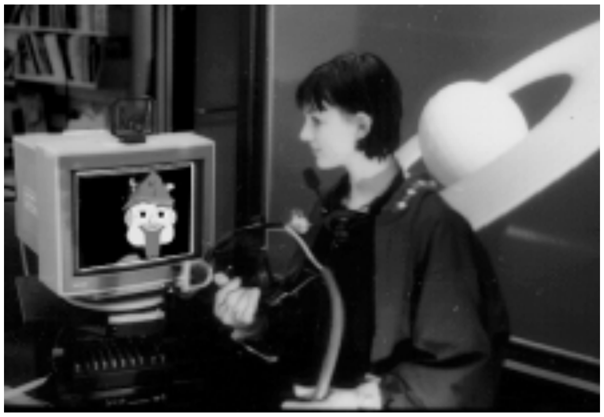


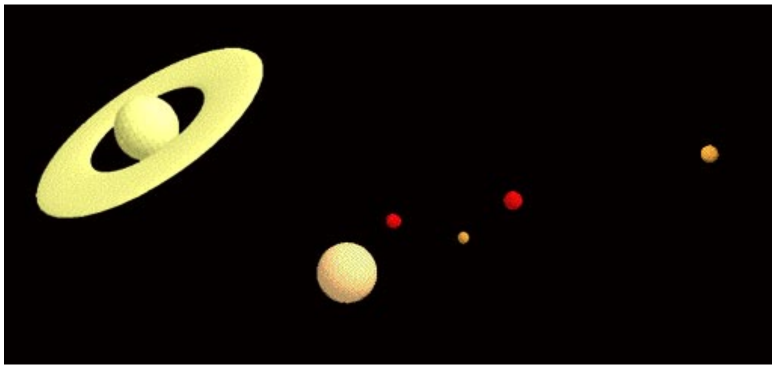
FIGURE 9-1. A user gets ready for interacting with Gandalf.

been programmed to recognize utterances like "Take me to Jupiter" and "Tell me about Saturn". Intonation analysis is performed by a custom-built analyzer. Gandalf "sees" the user via a body-tracking suit that uses magnetic space-sensing cubes, and an eye tracker that the user wears [Bers 1995, 1996]. By mapping out Gandalf's monitor in real-space he knows his own position in real space. Mapping the big-screen display allows him to know the real-space position of the planets.

Eight computers are used to run Gandalf's software:

1. A Digital Equipment Corporation Alpha 3000/300 workstation runs most of Ymir: Reactive Layer, Process Control Layer and Knowledge Base.
2. A Digital Equipment Corporation 5000/240 workstation runs the Action Scheduler.

FIGURE 9-2. Gandalf knows general facts about our solar system. It commands a graphical model of the planets with logarithmically scaled distances and moon sizes.



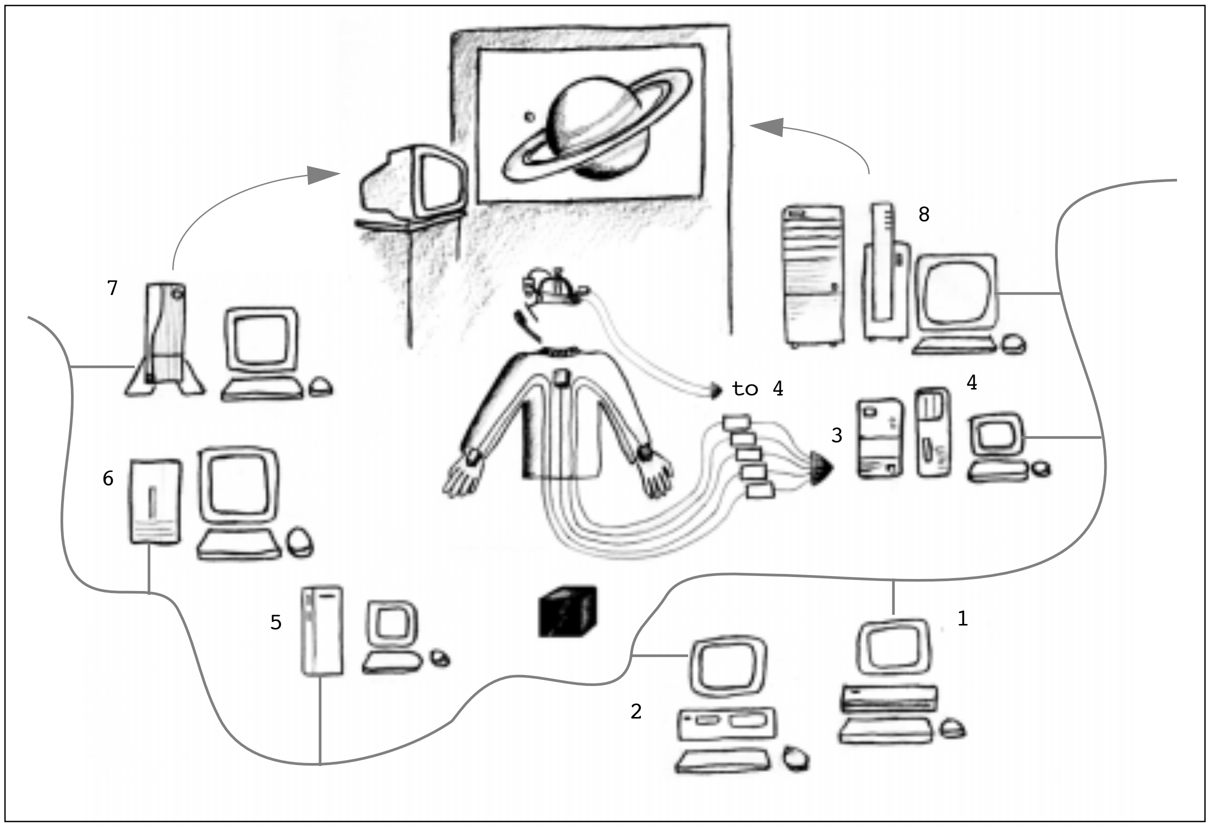


FIGURE 9-3. Gandalf system layout. Grey arrows show display connections; grey line is Ethernet. Eye tracker is connected to computer 4 via a serial port and the data is fed to computer 3 via another serial connection; output from the jakcet is connected to computer 3 via serial connections (dark arrowheads). Black cube (connected to computer 3) generates the magnetic field for sensing the posture of user's upper body. Microphone signal is split to two computers (6 & 7) via an audio mixer (connections not shown).

Legend
1. DEC 5000/240
2. DEC Alpha 3000/300
3. PC (Intel 486)
4. PC (Intel 386)
5. Macintosh Quadra 950
6. SGI Iris
7. SGI Indigo
8. HP Apollo 9000/750

3. A PC is used to collect data from a body tracking suit.
4. A PC is used to collect data from an eye tracker.
5. A Macintosh Quadra 950, with a Roland CP-40 pitch-to-MIDI converter is used to track and analyze intonation.
6. A Silicon Graphics Iris is used for speech recognition.
7. A Silicon Graphics Indigo2 is used to animate the character.
8. A Hewlett-Packard Apollo 9000/750 animates a virtual solar system.

The configuration is presented graphically in Figure 9-3. An Ethernet connection provides data flow between all computers (a serial line connects computers 3 & 4). Further detail on the hardware and software can be found in Appendix A2 on page 203.

9.2 Gandalf: Technical Description

Gandalf consists of a collection of virtual sensors, decision modules and a behavior lexicon. Its knowledge base, whose mechanisms were mentioned in the last chapter, is also presented here in full detail, along with each module and communication primitives. Most of Gandalf's modules aimed toward generating appropriate turn taking and maintaining the flow of the interaction: very little has been done in terms of providing it with sophisticated perceptual mechanisms or extensive topic knowledge. Needless to say, Ymir provides the necessary hooks to integrate such mechanisms into the control of the agent's behavior. To give the reader a complete picture of what it takes to design a (minimal) agent in Ymir, in this chapter we will look at all the modules needed to get Gandalf working.

9.2.1 Where do Gandalf's Control Rules Come From?

Gandalf's modules are designed with the single purpose of supporting intelligent dialogue behavior on behalf of the agent: enabling it to look where you're pointing[1], glance at you when you're done speaking, turn its head back to you when it's finished doing what you asked it to do, etc. For the most part, this was done by data-mining the psychological literature (Chapter 3., page 37). Most of its Decision and Behavior modules, therefore, trace their origin to one or more papers reviewed in that chapter. Some fine-tuning and modification of modules was also performed after user testing ("Human Subjects Experiment" on page 147).

9.2.2 Virtual Sensors

Gandalf has a total of 16 virtual sensors (Table 9-1), of which 3 are prosody sensors, 9 are body sensors and 3 are speech sensors. One sensor is specialized for monitoring body data input, and is called vision-sensor (sensor number 10, Table 9-1). It allows Gandalf to know whether its vision is working properly. It POSTs FALSE if any of the data from the body tracking stops updating normally.

The prosody sensors are added for homogeneity; the features they represent are computed on a separate machine (see above) and sent over a

1. See Table 10-1 on page 172 for perception and action modules designed for deictic gestures.

NAME: facing-work-screen **1** TYPE: body-sensor-fix-ref DATA-1: nil DATA-2: work-screen INDEX-1: `get-head-direction` INDEX-2: nil FUNC: `facing?`	NAME: looking-at-me **2** TYPE: body-sensor-fix-ref DATA-1: nil DATA-2: agent-screen INDEX-1: `get-gaze-direction` INDEX-2: nil FUNC: `u-looking-at-me?`
NAME: facing-me **3** TYPE:body-sensor-fix-ref DATA-1: nil DATA-2: agent-screen INDEX-1: `get-head-direction` INDEX-2: nil FUNC: `facing?`	NAME: r-hand-in-gest-space **4** TYPE: body-sensor-var-ref DATA-1: nil DATA-2: nil INDEX-1: `get-r-wrist-position` INDEX-2: `get-trunk-direction` FUNC: `hand-in-gest-space?`
NAME: turned-to-me **5** TYPE: body-sensor-fix-ref DATA-1: nil DATA-2: agent-screen INDEX-1: `get-trunk-direction` INDEX-2: nil FUNC: `turned-to?`	NAME: l-hand-in-gest-space **6** TYPE: body-sensor-var-ref DATA-1: nil DATA-2: nil INDEX-1: `get-l-wrist-position` INDEX-2: `get-trunk-direction` FUNC: `hand-in-gest-space?`
NAME: complete-synt **7** TYPE: speech-sensor DATA-1: nil INDEX-1: index-1 FUNC: `syntax-complete?`	NAME: speaking **8** TYPE: prosody-sensor DATA-1: nil INDEX-1: speech-on-idx FUNC: `u-speaking?`
NAME: looking-at-l-hand **9** TYPE: body-sensor-var-ref DATA-1: nil DATA-2: nil INDEX-1: `get-gaze-direction` INDEX-2: `get-l-wrist-position` FUNC: `u-looking-at-hand?`	NAME: see-user **10** TYPE: vision-sensor DATA-1: *socket-object1* DATA-2: nil INDEX-1: nil INDEX-2: nil FUNC: `body-socket-monitor`
NAME: facing-domain **11** TYPE: body-sensor-fix-ref DATA-1: nil DATA-2: work-screen INDEX-1: `get-head-direction` INDEX-2: nil FUNC: `facing?`	NAME: looking-at-r-hand **12** TYPE: body-sensor-var-ref DATA-1: nil DATA-2: nil INDEX-1: `get-gaze-direction` INDEX-2: `get-r-wrist-position` FUNC: `u-looking-at-hand?`
NAME: complete-gram **13** TYPE: speech-sensor DATA-1: nil INDEX-1: index-2 FUNC: `grammar-complete?`	NAME: complete-pragm **14** TYPE: speech-sensor DATA-1: nil INDEX-1: index-3 FUNC: `pragmatics-complete?`
NAME: intonation-up **15** TYPE: prosody-sensor DATA-1: nil INDEX-1: nil FUNC: `inton-direction?`	NAME: intonation-down **16** TYPE: prosody-sensor DATA-1: nil INDEX-1: nil FUNC: `inton-direction?`

TABLE 9-1. Virtual sensors used in the Gandalf prototype. The "agent-screen" variable holds the plane of the Gandalf's monitor (substituting for the plane of its face).

socket. These sensors provide a simple way to receive the intonation data and `POST` it to the Sketchboard.

Sensor `compl-pragm` is currently always true—there is no checking for pragmatic correctness yet in Gandalf. In addition to sensors 13 and 14, a Multimodal Descriptor called `complete-utter` `POST` TRUE immediately after the user stops speaking, unless either 13 or 14 are false.

9.2.3 Multimodal Descriptors

Gandalf has a total of 10 Multimodal Descriptors (Table 9-3, page 136). Module 7, `is-active`, is an example of a descriptor that takes the state of another Descriptor as well as that of a Sensor into account. Module 9, `complete-utter`, works in a very primitive way: if a template in either the Dialogue Knowledge Base or the Topic Knowledge Base has been filled completely, it `POST`s as TRUE. The full implementation of this scheme is currently precluded by the inability of the speech recognition to recognize words incrementally. In an ideal system, this module would allow a character to respond with appropriate facial expression or body language—even verbally—before actually generating a complete response to the content of an utterance.

9.2.4 Decision Modules

Gandalf has a total of 35 Decision Modules. Of these, 16 belong to the Reactive Layer and 19 to the Process Control Layer. The three types of Decision Modules are: State Decision Modules, Internal Decision Modules and External Decision Modules.

State Decision Modules

Six Decision Modules are used to keep states (Figure 9-7 on page 144). These are based on a generalized finite state machine approach, where one or more State Decision Modules are associated with a single state. Each state module has an associated list of perceptual modules (Multimodal Descriptors) that should be active while that module is true. When two or more modules are activated, all the perceptual modules associated with the old state are turned off, and the perceptual modules in the newly entered state modules are turned on (meaning they are called on every update in the main loop). This way various internal processes can be turned on and off depending on the particular conditions of a collection of state modules, which in turn are only active during their associated state.

NAME: giving-turn POS-CONDS: (looking-at-me 0.3)(facing-me 0.3) NEG-CONDS: (gesturing 0.3)(speaking 0.4) THRESH: 1.0	**1**
NAME: taking-turn POS-CONDS: (gesturing 1.0)(speaking 1.0) NEG-CONDS: (looking-at-me 0.5) THRESH: 1.0	**2**
NAME: wanting-turn POS-CONDS: (speaking 0.5)(hand-in-gest-space 0.6) NEG-CONDS: nil THRESH: 1.0	**3**
NAME: want-back-ch-feedb POS-CONDS: (looking-at-me 0.5)(speaking 0.5) NEG-CONDS: nil THRESH: 1.0	**4**
NAME: looking-at-hands POS-CONDS: (u-looking-at-r-hand 0.5)(u-looking-at-l-hand 0.5) NEG-CONDS: nil THRESH: 1.0	**5**
NAME: hand-in-gest-space POS-CONDS: (l-hand-in-gest-space 0.5)(r-hand-in-gest-space 0.5) NEG-CONDS: nil THRESH: 1.0	**6**
NAME: is-active POS-CONDS: (hand-in-gest-space 1.0)(speaking 1.0) NEG-CONDS: nil THRESH: 1.0	**7**
NAME: gesturing POS-CONDS: (hand-in-gest-space 0.5)(speaking 0.6) NEG-CONDS: nil THRESH: 1.0	**8**
NAME: complete-utter POS-CONDS: (complete-pragm 0.5)(complete-syntax 0.5) NEG-CONDS: nil THRESH: 1.0	**9**
NAME: addressing-me POS-CONDS: (turned-to-me 1.0)(facing-me 1.0)(facing-domain 1.0) NEG-CONDS: nil THRESH: 1.0	**10**

TABLE 9-2. Multimodal Descriptors used in the Gandalf prototype. In descriptor 10, any one of its conditions will trigger an `addr-me` message to get `POST`ed to the Functional Scketchboard.

NAME: take-turn-1 TYPE: RL-State-Dec-Mod MSGS: take-turn NEXT-STATES: (give-turn dial-on) POS-CONDS: (addressing-me wanting-turn) NEG-CONDS: (KB-Exe-Act Spch-Data-Avail) ACTIVE-DESCR: (taking-turn wanting-turn gesturing addressing-me saying-goodbye concluding hand-in-gest-space is-active looking-at-hands)	**1**
NAME: take-turn-2 TYPE: RL-State-Dec-Mod MSGS: take-turn NEXT-STATES: (give-turn dial-on) POS-CONDS: (addressing-me wanting-turn) NEG-CONDS: (KB-Exe-Act) ACTIVE-DESCR: (taking-turn wanting-turn gesturing addressing-me saying-goodbye concluding hand-in-gest-space is-active looking-at-hands)	**2**
NAME: give-turn-1 TYPE: RL-State-Dec-Mod MSGS: give-turn NEXT-STATES: (take-turn dial-on) POS-CONDS: ((FS-time-since 'speaking 50) giving-turn) NEG-CONDS: (taking-turn) ACTIVE-DESCR: (taking-turn giving-turn gesturing addressing-me saying-goodbye concluding is-active looking-at-hands want-back-ch-feedb complete-utter)	**3**
NAME: dial-on-1 TYPE: PCL-State-Dec-Mod MSGS: dial-on NEXT-STATES: (dial-off) POS-CONDS: (saying-goodbye) NEG-CONDS: (dial-off) ACTIVE-DESCR: (saying-goodbye)	**4**
NAME: dial-off-1 TYPE: PCL-State-Dec-Mod MSGS: dial-off NEXT-STATES: (give-turn dial-on) POS-CONDS: (saying-my-name) NEG-CONDS: (dial-off) ACTIVE-DESCR: (addressing-me gesturing)	**5**
NAME: dial-off-2 TYPE: PCL-State-Dec-Mod MSGS: dial-off NEXT-STATES: (give-turn dial-on) POS-CONDS: (addressing-me) NEG-CONDS: (dial-off) ACTIVE-DESCR: (addressing-me)	**6**

TABLE 9-3. State Decision Modules used in the Gandalf prototype. These are part of the Reactive Layer. The ACTIVE-DESCR slot contains the names of all Multimodal Descriptors that should be operative during the state. They are the descriptors that are essential to determine transitions to the next state, as specified in the NEXT-STATES slot. When a state node gets posted its MSGS value is put on the Functional Sketchboard.

NAME: exe-DKB-act TYPE: PCL-Dec-Mod EL: 2000 MSGS: (post-DKB-act) POS-CONDS: (dial-on TKB-act-avail take-turn) NEG-CONDS: nil POS-RESTR-CONDS: (CL-act-avail) NEG-RESTR-CONDS: nil	**1**
NAME: exe-TKB-act TYPE: PCL-Dec-Mod EL: 2000 MSGS: (post-TKB-act) POS-CONDS: (dial-on TKB-act-avail take-turn) NEG-CONDS: nil POS-RESTR-CONDS: (CL-act-avail) NEG-RESTR-CONDS: nil	**2**

TABLE 9-5. External Decision Modules in the *Process Control Layer* involved in deciding the timing of content delivery.

Internal Decision Modules

Five internal Decision Modules are used to deal with speech data (Table 9-4), four of which are PCL modules, one a reactive. Currently the only internal processes have to do with "cleaning"—such as deleting old speech from the parse buffer, etc.

External Decision Modules

Of the twenty-six external Decision Modules in Gandalf, fourteen belong to the PCL and have to do with content delivery and process control (Table 9-8 & Table 9-7 on page 140); twelve are dedicated to reactive behaviors (Table 9-6, page 139 & Table 9-8, page 141). Two of the PCL decision modules decide execution of content-related material (Table 9-8). These POST available acts—that is, acts that were successfully generated in response to a user's multimodal input—to the *BEHAVIOR-REQUESTS* queue (see "Behavior Requests" on page 118). Two different periodic decision modules were made in the Reactive Layer to produce eye blinks (Table 9-8, page 141); one is active during interaction, the other before and after the dialogue.

"...gesture is not simply a way to display meaning but an activity with distinctive temporal, spatial, and social properties that participants not only recognize but actively use in the organization of their interaction."

—Charles Goodwin (1986, p. 47)

9.3 Spatial Data Handling

The body of a user is represented geometrically. This is not a requirement of Ymir, and is obviously not the only way to represent a user's body in a multimodal system. Certain computations, however, are simple to deal with in geometric terms, such as gaze direction and deictic

(continued on page 141)

NAME: parse-speech **1**
TYPE: PCL-Int-Dec-Mod
EL: 200
MSGS: (parse-speech)
POS-CONDS: (giving-turn spch-data-avail)
NEG-CONDS: nil
POS-RESTR-CONDS: (give-turn)
NEG-RESTR-CONDS: nil

NAME: rem-spch-addr-to-others **2**
TYPE: PCL-Int-Dec-Mod
EL: 200
MSGS: (remove-speech-to-others)
POS-CONDS: ((`FS-time-since` 'speaking 50))
NEG-CONDS: (addressing-me KB-exe-act)
POS-RESTR-CONDS: (speaking)
NEG-RESTR-CONDS: nil

NAME: remove-partial-parses **3**
TYPE: PCL-Int-Dec-Mod
EL: 100
MSGS: (rem-part-parses)
POS-CONDS: (KB-succ-parse KB-exe-act)
NEG-CONDS: (addressing-me)
POS-RESTR-CONDS: nil
NEG-RESTR-CONDS: (KB-exe-act)

NAME: report-no-words **4**
TYPE: PCL-Int-Dec-Mod
EL: 50
MSGS: (compose-DKB-trouble-report 'no-words)
POS-CONDS: ((`CB-time-since` 'rcv-spch 350) addressing-me giving-turn)
NEG-CONDS: (CL-act-avail KB-succ-parse KB-exe-act speaking)
POS-RESTR-CONDS: (give-turn)
NEG-RESTR-CONDS: nil

NAME: clear-spch-buffers **5**
TYPE: RL-Int-Dec-Mod
EL: 10000
MSGS: (clear-spch-buffs)
POS-CONDS: (give-turn)
NEG-CONDS: nil
POS-RESTR-CONDS: (take-turn)
NEG-RESTR-CONDS: nil

TABLE 9-4. Internal Decision Modules used in Gandalf's *Reactive* and *Process Control* layers. The function `CB-time-since` returns true if the time since the passed message (e.g. `rcv-spch`) was posted to the **C**ontent **B**lackboard exceeds the passed time (e.g. 350 centiseconds). EL = Expected Lifetime; FS = Functional Sketchboard.

NAME: show-take-turn **1** TYPE: RL-Ext-Dec-Mod EL: 500 MSGS: show-take-turn POS-CONDS: (take-turn) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (take-turn)	NAME: show-give-turn-1 **2** TYPE: RL-Ext-Dec-Mod EL: 200 MSGS: show-give-turn POS-CONDS: (give-turn) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (give-turn)
NAME: show-addr-me-1 **3** TYPE: RL-Ext-Dec-Mod EL: 20 MSGS: smile POS-CONDS: (TKB-exe-speech-act) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (turned-to-me)	NAME: show-give-turn-2 **4** TYPE: RL-Ext-Dec-Mod EL: 200 MSGS: show-give-turn POS-CONDS: (give-turn) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (give-turn)
NAME: show-addr-me-2 **5** TYPE: RL-Ext-Dec-Mod EL: 20 MSGS: eyebrow-greet POS-CONDS: (saying-my-name turned-to-me facing-me) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (turned-to-me)	NAME: show-listen **6** TYPE: RL-Ext-Dec-Mod EL: 20 MSGS: brows-in-pensive-shape POS-CONDS: (saying-my-name turned-to-me facing-me) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (turned-to-me)
NAME: initialize **7** TYPE: RL-Ext-Dec-Mod EL: 20 MSGS: face-neutral POS-CONDS: (dial-off) NEG-CONDS: nil POS-RESTR-CONDS: (dial-on) NEG-RESTR-CONDS: nil	NAME: show-not-addr-me **8** TYPE: RL-Spatial-Dec-Mod EL: 100 MSGS: (turn-to ’work-space) POS-CONDS: (dial-off) NEG-CONDS: (turned-to-me) POS-RESTR-CONDS: (taking-turn) NEG-RESTR-CONDS: nil
NAME: look-puzzled **9** TYPE: RL-Ext-Dec-Mod EL: 100 MSGS: look-puzzled POS-CONDS: (turned-to-me facing-me (FS-time-since ’facing-me 400)) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (turned-to-me)	NAME: look-aloof **10** TYPE: RL-Ext-Dec-Mod EL: 100 MSGS: look-aloof POS-CONDS: (turned-to-me facing-me (FS-time-since ’facing-me 800) dial-off) NEG-CONDS: (speaking KB-parsing) POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (turned-to-me)

TABLE 9-6. External Decision Modules used in Gandalf’s *Reactive Layer*. Expected Lifetime (EL) values are in centiseconds. These modules control Gandalf’s reactive behavior.

NAME: hesitate-1 **1** TYPE: PCL-Ext-Dec-Mod EL: 100 MSGS: hesitate POS-CONDS: (dial-on take-turn spch-data-avail (FS-time-since 'speaking 70)) NEG-CONDS: (CL-act-avail speaking) POS-RESTR-CONDS: (give-turn) NEG-RESTR-CONDS: nil	NAME: show-done-exe-1 **2** TYPE: PCL-Ext-Spatial-Dec-Mod EL: 2000 MSGS: (Look-At 'user) POS-CONDS: nil NEG-CONDS: (TKB-exe-world-act) POS-RESTR-CONDS: (TKB-exe-world-act) NEG-RESTR-CONDS: nil
NAME: turn-to-user **3** TYPE: PCL-Ext-Dec-Mod EL: 2000 MSGS: (Turn-To 'user) POS-CONDS: (TKB-exe-speech-act) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (TKB-exe-speech-act)	NAME: show-done-exe-2 **4** TYPE: PCL-Ext-Spatial-Dec-Mod EL: 2000 MSGS: (Turn-To 'user) POS-CONDS: nil NEG-CONDS: (TKB-exe-world-act) POS-RESTR-CONDS: (TKB-exe-world-act) NEG-RESTR-CONDS: nil
NAME: show-content-delivery **5** TYPE: PCL-Ext-Spatial-Dec-Mod EL: 2000 MSGS: (Look-At 'user) POS-CONDS: (DKB-exe-act looking-at-me) NEG-CONDS: nil POS-RESTR-CONDS: (take-turn) NEG-RESTR-CONDS: nil	NAME: show-know-addressing **6** TYPE: PCL-Ext-Spatial-Dec-Mod EL: 2000 MSGS: (Turn-To 'user) POS-CONDS: (TKB-exe-world-act looking-at-me facing-me speaking) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (TKB-exe-world-act)
NAME: pay-attention-to-act **7** TYPE: PCL-Ext-Spatial-Dec-Mod EL: 2000 MSGS: (Turn-To 'work-space) POS-CONDS: (TKB-exe-world-act) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (TKB-exe-world-act)	NAME: show-idle **8** TYPE: PCL-Ext-Dec-Mod EL: 2000 MSGS: restless POS-CONDS: nil NEG-CONDS: (facing-me) POS-RESTR-CONDS: (facing-me) NEG-RESTR-CONDS: nil
NAME: show-listening-1 **9** TYPE: PCL-Ext-Dec-Mod EL: 2000 MSGS: (Turn-To 'user) POS-CONDS: (speaking addressing-me) NEG-CONDS: nil POS-RESTR-CONDS: (take-turn) NEG-RESTR-CONDS: nil	NAME: look-at-domain **10** TYPE: PCL-Ext-Spatial-Dec-Mod EL: 2000 MSGS: (Turn-To 'work-space) POS-CONDS: (facing-domain take-turn) NEG-CONDS: (speaking) POS-RESTR-CONDS: (taking-turn) NEG-RESTR-CONDS: nil
NAME: look-relaxed **11** TYPE: PCL-Ext-Dec-Mod EL: 2000 MSGS: face-neutral POS-CONDS: (TKB-exe-world-act) NEG-CONDS: nil POS-RESTR-CONDS: nil NEG-RESTR-CONDS: (TKB-exe-world-act)	NAME: show-listening-2 **12** TYPE: PCL-Ext-Spatial-Dec-Mod EL: 2000 MSGS: (Look-At 'user) POS-CONDS: (speaking addressing-me) NEG-CONDS: nil POS-RESTR-CONDS: (take-turn) NEG-RESTR-CONDS: nil

TABLE 9-7. External Decision Modules used in Gandalf's *Process Control Layer*. Expected Lifetime (EL) values are in centiseconds.

(continued from page 137)

gestures. All spatial features in the system are computed from data supplied by a body model server [Bers 1996, 1995a] which provides a geometric model of a person's upper body. We will now look at the representation of geometric elements derived from this model.

9.3.1 Spaces & Positional Elements

Space is divided into *volumes*, *planes* and *points*. The hope is that one can get away with this simplification without sacrificing too much of the agent's perceptuo-motor skills. Three spatial features are of crucial importance to conversants in multimodal dialogue:

1. *Work volume*
2. *Gesture volumes*
3. *Face planes*

These are important because they mark the boundary (albeit in a somewhat fuzzy manner) that events, objects or information can be located in. But knowing the size and shape of these is not enough; one needs to know the positions of these volumes and planes, and, in particular, objects within these. To point your eyes at any particular location in space, you need to be able to define that point relative to yourself. Coarse body movements can be generated on the basis of coarse knowledge—e.g. the boundaries of a volume. This can help you to point your head in the general direction (e.g. if you know someone is standing to your left, you turn your head to the left to get your eyes pointing in the right direction.) But to fixate on an object, its position needs to be defined more precisely. Accurate positional information allows one to apply the necessary force on the muscles of one's eyes to move them into the particular configuration that points them at the spot. In conversation, these positional data are essential to the information exchange:

TABLE 9-8. Periodic Decision Modules used in the Gandalf prototype. These belong to the *Reactive Layer*.

NAME: blink TYPE: RL-Per-Dec-Mod EL: 1000 MSGS: blink PERIOD: 300 POS-CONDS: (dial-on) NEG-CONDS: nil	**1**
NAME: blink-slowly TYPE: RL-Per-Dec-Mod EL: 1000 MSGS: blink-slowly PERIOD: 300 POS-CONDS: (dial-off) NEG-CONDS: nil	**2**

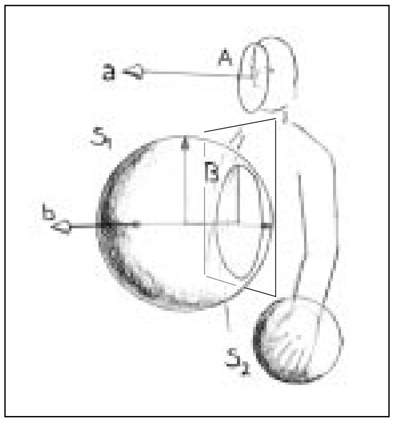


FIGURE 9-4. Geometric definitions of gesture space (S1), face space (A) and hand space (S2), along with normals showing direction of head (a) and trunk (b).

1. Position of *work volume*
2. Position of *gesture volumes*
3. Position of *face planes*
4. Position of *hands* (or *hand volumes*)

Volumes are mapped out as shown in Figure 9-4. Work space and faces are simply three-dimensional planes—a circular one for the face and a square one for the work space display (not shown). Position is given by the planes' centers.

The gesture space's primary role is to be an indicator of intentional gesturing, which mainly happens in the space right in front of the speaker's body [Rimé & Schiaratura 1991]. Self-adjusters [Ekman & Friesen 1969], which seldom have a communicative function, are automatically excluded by lifting the gesture space approximately 10 cm from the body of the user (plane B in Figure 9-4) because these happen close to the body. McNeill's research [1992] has indicated that the type of gesture and its place of articulation may be correlated. If this turns out to be the case, a finer division of gesture space would be useful for determining the function of manual gestures.

The user's hands are surrounded by a 20 cm diameter sphere, in order to give their position a larger margin, making it into a volume. This is especially useful when computing whether the user is looking at his or her hands. The following method is used for checking if a hand is inside gesture volume:

$$inside(S_g, h) = \begin{pmatrix} 0 \mathit{if}\ (C_{Sh} - C_{Sg} > R_{Sg})\ \mathit{or}\ (\operatorname{acos}(|C_{sh} - C_B| \bullet b) > 90°) \\ 1 \mathit{otherwise} \end{pmatrix} \quad \{9.1\}$$

where h is a hand, S_g is gesture space (S_1 in Figure 9-4), C_{Sh} defines the center of hand space (S_2 in Figure 9-4), C_{Sg} defines the center of gesture space, R_{Sg} is the radius of gesture space, C_B is the center of plane B, and • is dot product (refer to Figure 9-4).

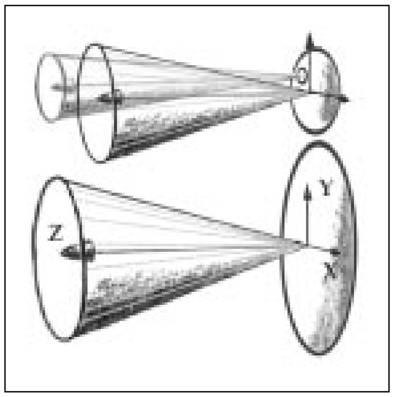


FIGURE 9-5. Directional elements of the user's upper body are treated as cones (gaze not shown) whose overlap on objects in the environment (such as the agent's face and the workspace screen) constitutes "facing" the objects in question.

9.3.2 Directional Elements

When is a person "facing" someone or something? When is a person "turned" in a given direction? These are questions that need to be answered by any multimodal agent, because their answers are required for successful participation in a face-to-face conversation. There are undoubtedly numerous ways to answer it. Here, as before, we use geometric definitions. The directional features extracted in Gandalf are:

1. Direction of *gaze*.
2. Direction of *head*.
3. Direction of *trunk*.

Absolute direction is computed after the position and relative orientation of relevant body parts is known. Figure 9-4 shows the two normals used for head and trunk (gaze direction not shown). The head, gaze and trunk normals are further modified by making them cones (Figure 9-6) so that their interception with other spaces, such as the agent's face space, is broader: interception happens if the inside of the cone overlaps the area or point in question. The angle of the cones is graded such that gaze has the narrowest (a 20° cone), then the head (35°), and lastly the trunk (40°).

The user is facing point *p* if point *p* falls within the boundary of head cone. More generally, to find if plane A is facing a point *p* in space, the following method is used:

$$facing(A, p, T_A) = \begin{pmatrix} 1 if \operatorname{acos}((p - C_A) \bullet n_A) < T_A \\ 0 otherwise \end{pmatrix} \qquad \{9.2\}$$

where C_A defines the center of plane A, n_A is plane A's normal and T_A is the angular threshold of plane A's cone (Figure 9-6).

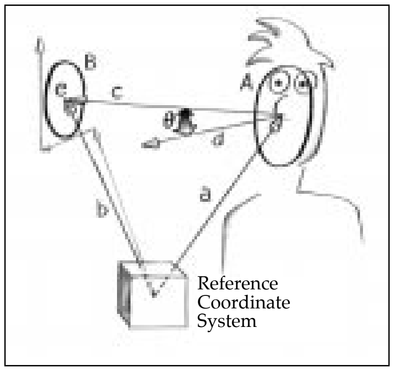


FIGURE 9-6. Geometry defining the "facing" function (see Equation 9.2). Center of Face Plane A is defined by vector a, d is plane A's normal. Center of plane B is defined by vector b. θ defines plane A's cone. By comparing the angle betwen vectors d and c to a threshold, one can determine whether the person on the right is "facing" plane B, which couldfor example represent the agent's face.

9.4 Prosody

Methods have been suggested for automatic analysis of prosody [Wang & Hirschberg 1992], but very few have tried to do analysis in real-time. I use a real-time intonation analyzer that I designed, that detects the following boolean, time-stamped events:

1. Speech on/off.
2. Intonation going up.
3. Intonation going down.

Notice that although detecting a feature like "speech on/off" may seem trivial, this is only true if we use a dedicated microphone which is unlikely to pick up anything besides the dialogue participant's speech. Using a signal processing approach with artificial ears, this may be a significantly more difficult task.

The intonation analysis is performed using a windowing technique, where a window is 300 ms. Each new window starts where the last one ended. The slope of the intonational contour is tracked in each window and checked against a threshold. If over the threshold and different from last window, a time stamped status report is given about intonational direction.

To analyze direction, a time-dependent algorithm is used whose output depends on the prior output, one window back in time. The robustness

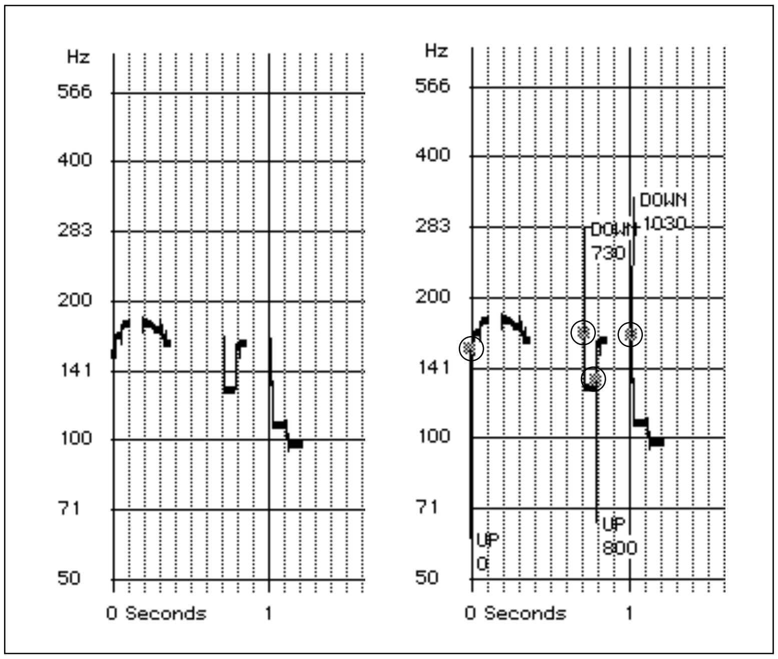


FIGURE 9-7. Example of intonation for the utterance "Take me to Jupiter" plotted to a logrithmic frequency scale. On the right we see the result of the real-time intonation analysis. Segmentation of pitch direction is marked with vertical bars, giving timing (in msec) and direction of the audio stream. Slanting lines from the grey dots on the graph to the markers (e.g. "DOWN / 730") indicate time delay. Where no slant is seen, the analysis took less than 10 ms to compute. The last "DOWN" marked may have taken about 10-15 ms.

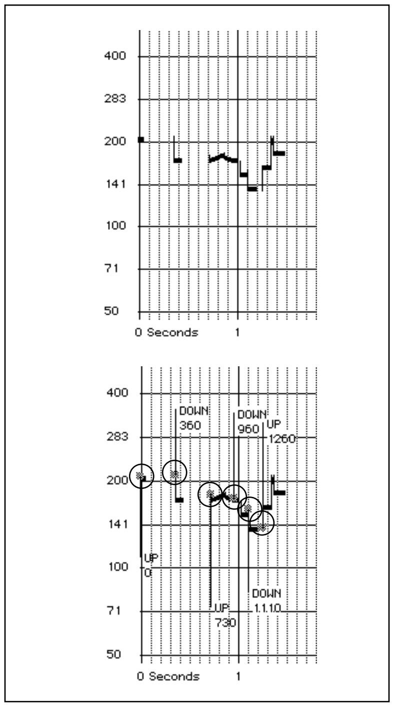


FIGURE 9-9. Results of analysis of the question "What planet is that?".

of intonation analysis is relatively high, considering that this is a real-time system. (We estimate that in normal interaction, about every tenth utterance is impossible to analyze. This reliability is high enough for an interactive system—keep in mind that the agent can always ask the user a question when the data doesn't make "sense".) Other modes of course help correct for occasional failures in the analysis process.

By running this intonation system on a dedicated machine, real-time response can be assumed. To give the reader a feel for the performance of the algorithm, several examples are given (Figures 9-7, 9-8 & 9-9).

9.4.1 Future Additions

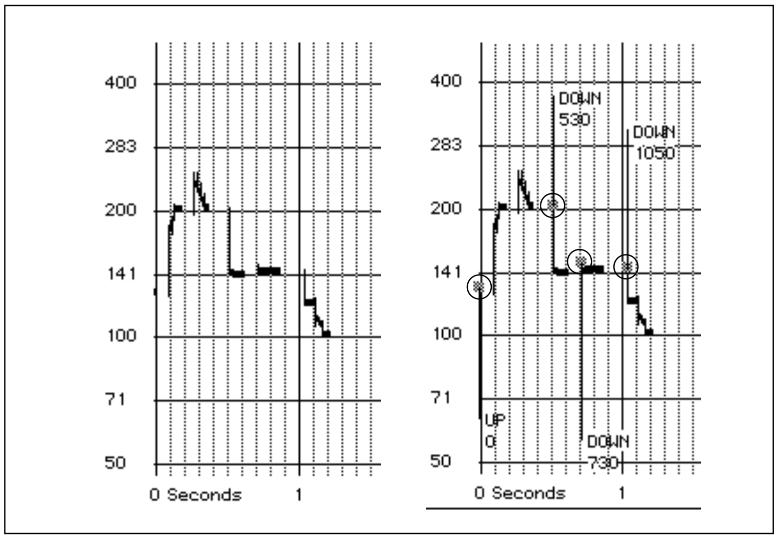
Future work includes using temporal multimodal descriptors to extract information regarding the intonation pattern over a full utterance, for determining whether an utterance could be a filler (relatively short and flat pitch pattern), question (final rise) or command (final fall) [Pierre-

```
(defun compute-direction (datapoints)
 (let* ((beginning (first-half datapoints))
        (ending (last-half datapoints))
        (pitch-a (compute-average-pitch beginning))
        (pitch-b (compute-average-pitch ending)))
        (slope 0)
        (snap-anlge   0) ;degrees
        (up-thresh   20) ;degrees
        (down-tresh -20) ;degrees
        (duration 200))  ;ms
   (setf slope (compute-angle pitch-a pitch-b duration)))
   (cond  ((eq *last-direction* 'UP)
             (setf snap-angle (expt slope 1.5)))
          ((eq *last-direction* 'DOWN)
             (setf snap-angle (expt slope 2.5)))
          (t (setf snap-angle (expt slope 0.8)))))
   (cond  ((> snap-angle up-thresh)
            (setf *last-direction* 'UP))
          ((< snap-angle down-thresh)
            (setf *last-direction* 'DOWN))
          (t (setf *last-direction* 'FLAT)))))
```

ALGORITHM 9-9. This function sets the global variable *last-direction* to the current direction. If *last-direction* is either UP or DOWN, it is trasmitted, along with frequency and a time-stamp, to Ymir.

humbert & Hirschberg 1990]. More extensive analysis would of course be preferable, including volume of speech and absolute high/low point recognition [Pierrehumbert & Hirschberg 1990] instead of relative. With such data one could more easily find pitch accents that could be synchronized with the words provided by the speech recognizer, and thus distinguish between the theme and rheme of an utterance [Clark &

FIGURE 9-8. Another example of pitch contour for the utterance "Take me to Jupiter". On the right is the result of the real-time analysis of the direction of intonation.



Brennan 1990, Prevost 1996]. This is a more difficult problem, but one that should be solvable in the near future. We are testing a method that adds absolute pitch (Hz) to the UP/DOWN markers and are hoping that this representation of the intonational contour will make it relatively straight forward to find pitch accents.

One problem with absolute scales is that pitch range varies between individuals and even highs and lows may vary highly between utterance for the same individual. A third source of difficulty is obtaining accuracy in the pitchtracking itself. These issues will have to be addressed in future work.

9.5 *Topic & Dialogue Knowledge Bases*

9.5.1 Speech Recognition

As mentioned in the beginning of this chapter, the current prototype uses a beta version of the HARK system from BBN [1993]. Time stamping is a necessity in any real-time system where different pieces of the same puzzle are analyzed separately, to piece them back together again. HARK provides time stamps (using additional in-house developed post-processing) for each word. Incremental interpretation is crucial for not disrupting the natural flow of multimodal behavior. Ideally the speech recognition would be continuous, although in reality it doesn't happen until a significant pause (250 msec[1]) is found in the audio stream.

As discussed in the section on interpretive knowledge, and as supported by user testing (page 155), more robust results would be achieved by using multiple speech recognizers: one that spots keywords, one that spots transitional cues (e.g. *cue phrases* [Grosz & Sidner 1986, Cahn 1992]), one that has grammar and vocabulary for a particular topic and one that recognizes fillers. Transitional cues would be used to weight various parts of the topic grammar (or turn them on or off) according to what the system thought the current topic is. Ymir is very well suited for this kind of "distributed" speech recognition scheme: We are currently designing Internal Decision Modules for this task, interfacing with the built-in features of HARK.

1. This number is user-definable.

9.5.2 Natural Language Parsing & Interpretation

The natural language parser/multimodal output generator used in Gandalf is based on a continuous speech recognition model (in spite of the current speech recognition being a "batch processing" recognizer), i.e. if fed with one word at a time, it will try to fit the words together even before all of them have arrived. A selected sample of utterances it recognizes are shown in Figure 9-10. It uses semantic templates to parse the utterances. This works quite well for a small domain, and can be extended to handle multimodal interpretation. The parser is indifferent to word order (i.e. it makes no distinction between "That planet—tell me about it" and "Tell me about that planet"). While it may not be the most sophisticated way to parse natural language, there is little reason at the current state of technology to apply complex grammar rules when parsing in a real-time, face-to-face multimodal system, since word order and actions in such an interaction are much more loosely connected by grammar than for example words written on a page.

Show me X
Take me to X
Tell me about X
What else?
Tell me more
Is that [deictic gesture] X?
What planet is that [deictic gest]?
Zoom [in | out]
Tilt it this way [wrist gesture].
Stop [the] animation
Start [the] animation
Tell me about the moon[s]
Hello [Gandalf]
Goodbye [Gandalf]
Gandalf?

FIGURE 9-10. A sample of the utterances Gandalf recognizes. X stands for the name of any planet in the solar system, and the sun. Brackets show options.

Outline of Parsing Process

For each word received, the parser tries to fit it into a semantic template. If a template has already been activated that can accommodate the word, it puts it there, otherwise it finds all templates that could possibly fit the word and marks them as active. A template gets a score depending how many of its slots have been filled. Whenever this score is higher than the template's pre-set threshold, the knowledge base tries to produce a response based on the content in the template. If it does, it posts this fact to the Content Blackboard (see section 8.3.2, page 98). If it doesn't, time passes and no message is posted to the Content Blackboard. This condition is monitored by decision modules in the PCL, which will then initiate a "problem report" generated by the Dialogue Knowledge Base. The DKB will look at the messages posted by the TKB and generate an appropriate response, e.g. "I'm sorry, I didn't get that". For partially filled templates, the DKB could generate more intelligent responses such as "Which planet did you want to go to?" or "Please repeat the name of the planet". The response itself is kept in a list in the knowledge base, and executed when Decision Modules in the PCL decide to.

When the agent takes the turn, a module in the PCL fetches the response and sends it to the {1} virtual world if it is an action (TKB-world-act) or to the {2} Action Scheduler if it is a speech act (TKB-speech-act or DKB-speech-act). If the TKB-world-act contains complimentary facial expressions, speech or manual gesture, the action is split to each destination.

9.5.3 Multimodal Parsing & Interpretation

The above scheme was intended to be extended to multimodal parsing. Ongoing work involves extending it by adding multimodal *meta*-templates with slots for each mode. A deictic meta-template could for example contain slots for sentences involving use of deictic phrases ("That one", "...these two") and a slot for deictic gestures. Extensive rules about how to fill the template's slots can make this scheme quite flexible. While for example Sparrell's VECIG system [Sparrell 1993] was completely driven by speech, my approach combines top-down with bottom-up processing in each mode, as well as a across modes: The idea is that maximum information be gleaned from a bottom-up approach (e.g. morphology, combinations of mode-dependent information) and that this will guide top-down hypotheses regarding the content of the multimodal acts. An intonation pattern typical for questions can for example activate a speech-parsing template for questions; a gesture that looks very much like an iconic gesture could activate a template for spatial information. Likewise, a speech template could be marked for probability of co-verbal gesture, thus initiating gesture analysis, even if the bottom-up approach fails. This would of course not be useful unless the speech recognition is incremental.

A nice feature of this approach is that a template activated by events in one mode can indicate what kinds of multimodal actions could be expected. Another big advantage is that interpretation is not solely driven by speech content: over the course of a multimodal action, any mode can contribute to the hypothesis-building about its content and meaning. Information posted to the Functional Sketchboard will obviously play a large part in this extension.

9.5.4 Topic: The Solar System

The system's knowledge is based on a lexico-spatial database of planets. Each planet, represented as a CLOS object, is defined as a point in the virtual world, and is accessed through its unique name. Four functions provide the Gandalf with the ability to generate responses to action-related queries, {1} `Go-To`, {2} `Zoom` {3} `Freeze-Anim`, and {4} `Tilt`. The first takes a planet object as an argument, the second takes a direction as an argument (zoom "in" or "out"). `Freeze-Anim` takes a boolean state and stops or starts the clock of the world animation engine. `Tilt` takes a direction, given with a wrist gesture. A number of utterances lead to the same use of these functions, for example "Show me Jupiter" and "Let's go to Jupiter" both result in the `Go-To` function being called with the "Jupiter" object.

9.6 Action Scheduler

9.6.1 Behaviors

Gandalf's Behavior Lexicon contains 83 behaviors, specified at various levels of detail. Below is a full listing of the specification of the lexicon. Notice that only a subset of this lexicon is used so far by the decision modules. A number of these will have to wait for future extensions of Gandalf.

9.6.2 Motor System

Gandalf's motor system, the ToonFace Animator (Appendix A1), runs on a Silicon Graphics Indigo2. Currently the loop time for a complete redraw of the face and hand is 150 ms.

9.6.3 Behavior Lexicon

At the end of the chapter (page 153) is the list used to generate CLOS behavior objects for Gandalf, alias the *Behavior Lexicon*, by calling the function `Make-Behaviors` with the list `*behavior-lexicon*`. The list, albeit minimal, is sufficient for rudimentary dialogue skills.

Behaviors come into two main groups: {1} *Morphological* and {2} *Functional*. Morphological behaviors are named after the way they *look*, for example, the behavior `brows-in-u-shape` specifies a shape for the brows to take. Nothing is said about what circumstances such a behavior should or could be used in, nor what possible meanings such a behavior could carry. On the other hand, the behavior `show-taking-turn` specifies a dialogue *function*. There are many ways for showing that you are going to say something, one being opening the mouth slightly, another is glancing away briefly [Kleinke 1986, Goodwin 1981, Duncan 1972]. Within these classes, various sub-classes of facial and manual gesture have been implemented.

9.7 Examples of System Performance

Now that we have shown how a complete character is built in Ymir, let's look at some run-time data from this prototype to get a better idea of how it performs. All modules shown in these graphs can be found in the tables in this chapter.

Figure 9-11 shows the internal events of Gandalf in its interaction with the actor Alan Alda during a visit from the television series "Scientific

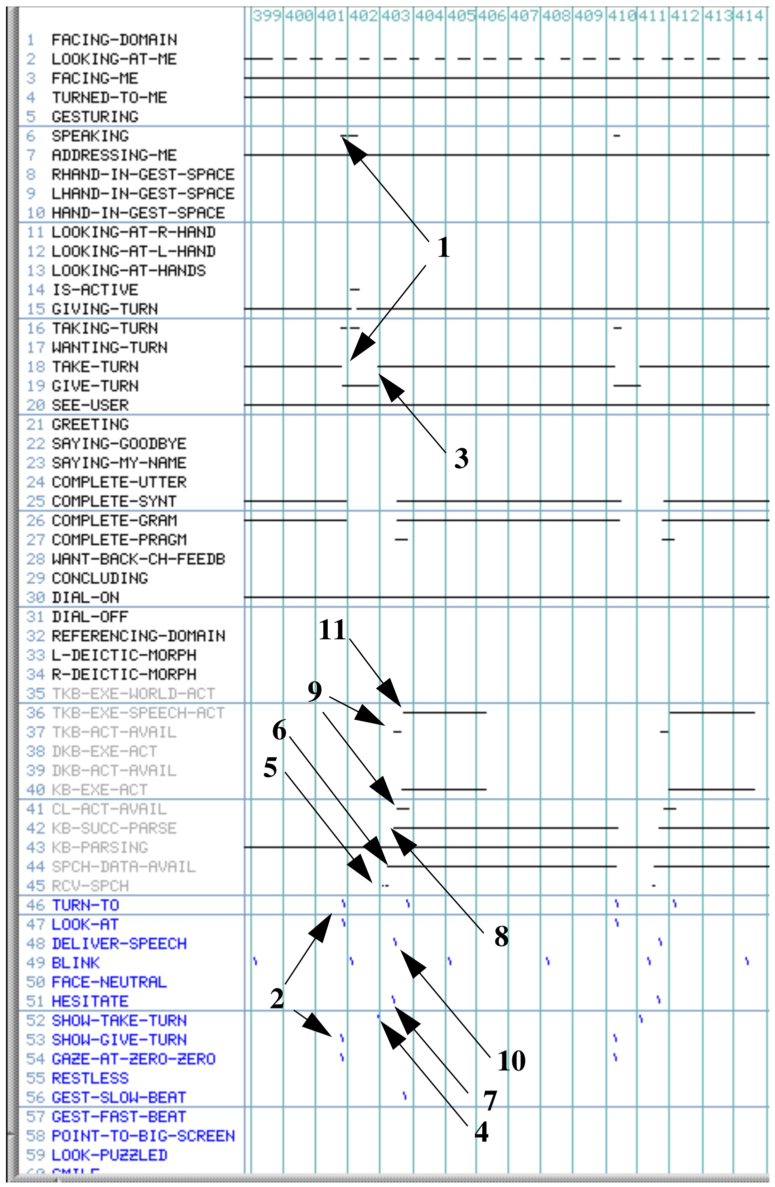


FIGURE 9-11. Graph showing internal states of Gandalf during interaction over a 16 second interval (each vertical line marks a second). When the person starts speaking Gandalf gives turn [1], turns to the user and shows that he is giving turn [2]. When the person falls silent Gandalf takes the turn [3], and shows that it is doing so [4]. At about the same time something is recieved from the speech recognizer [5] and shortly thereafter they are reported as available words [6]. These are then parsed and reported as successfull parse [8] (meanwhile Gandalf hesitates because he has taken the turn but has nothing to say as of yet [7]). When a response is available [9], Gandalf delivers this [10] and this event is posted internally [11]. (The highly rythmic gaze pattern observed at the top of the graph indicates a bad eye calibration.) Notice that modules 1 through 34 all relate to the *user's* behavior.

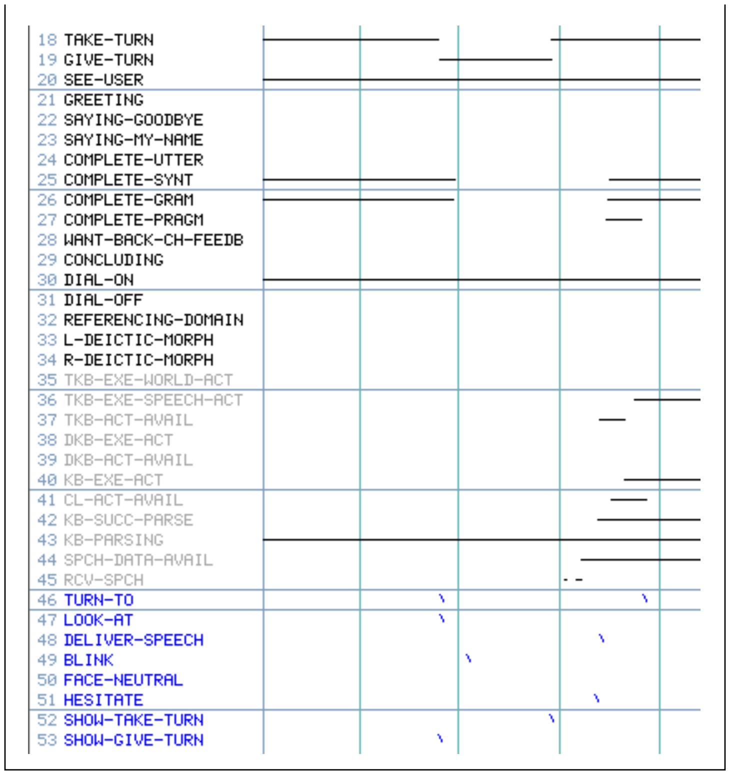


FIGURE 9-12. Interaction example in Figure 9-11 plotted at a high resolution. Each vertical line marks a second.

American Frontiers". Several features of the system are displayed in this example, as explained in the Figure text. The request made was "Tell me more about Mars".

Figure 9-13 shows yet another example of internal events during interaction with a user. This example spans 23 seconds, during which time the user makes three different requests, "Take me to the Sun", "Take me to Jupiter" and "What planet is that". Notice that although everything seems to have worked correctly internally in the first request, Gandalf does not execute the action (there is no line drawn for TKB-EXE-ACT after the request). This is because the Sun was already on the screen at the time of the request, and instead of executing the action, the DKB produces the utterance "This is Sun, dude".

Figure 10-11 on page 170 and Figure 10-13 on page 171 show another example of Gandalf's internal events when interacting with a human.

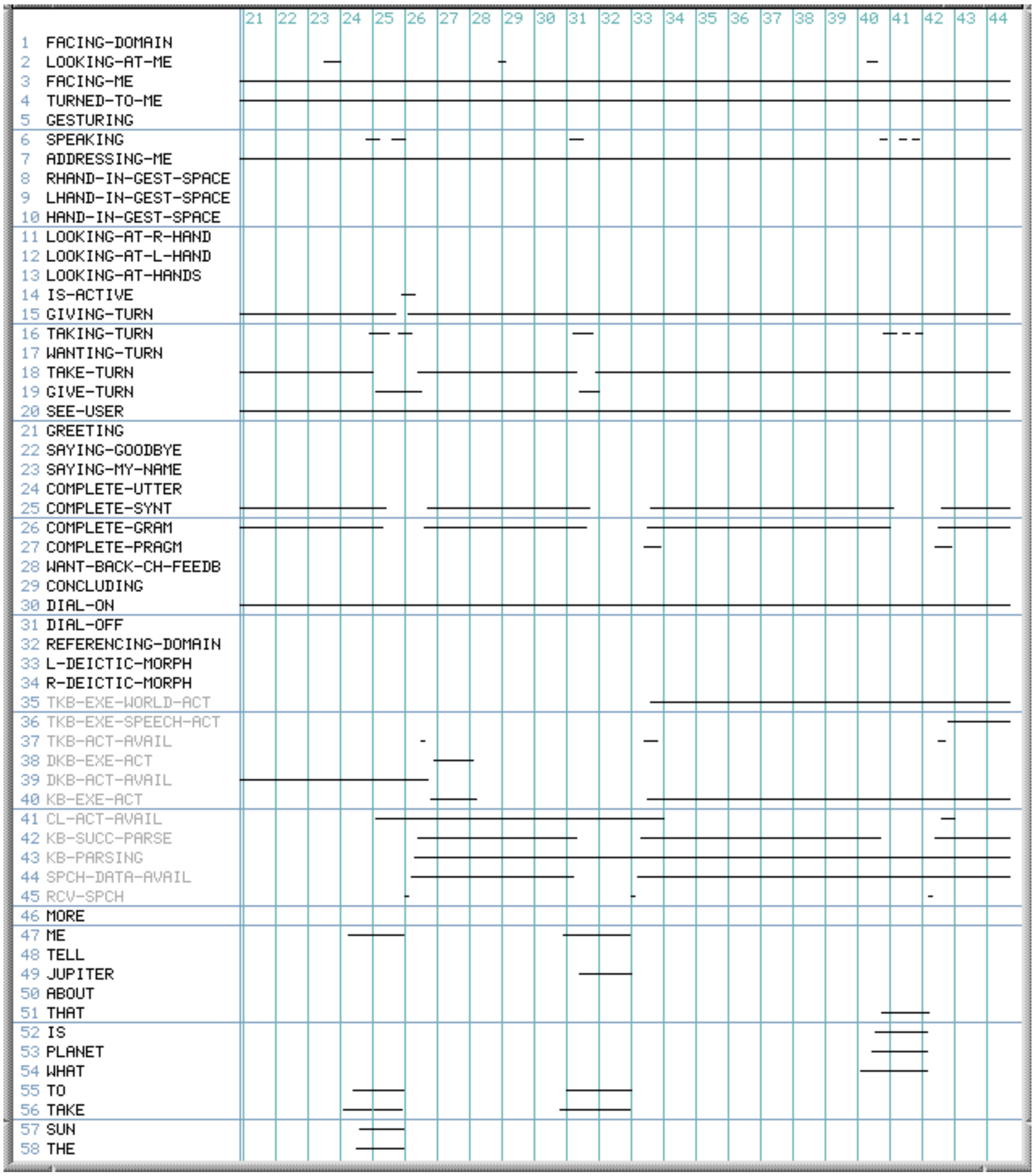


FIGURE 9-13. Example of internal events during an interaction betwen a user and Gandalf. Words spoken by the user are shown in lines 46 through 58 (beginning of line marks the time the word was uttered; end of line maks the time when Gandalf received the word.) See text for more details.

9.7.1 Behavior Lexicon Listing

```
(setf *behavior-lexicon*
  ;GENERAL LAYOUT: (<list-of-acts> (<first-act>(<first-act-element>)(<second-act-element>))
  ;                                (<second-act (<...>))
  ;                                )
  ;ACT TEMPLATE: (name class (((act-name-of-option-1 delay exec-time)(act-name delay exec-time) etc*)
  ;                           (etc*)))
  ;MOTORS:  (motor-name class delay exec-time pos/data)

 '(
  ; MORPHOLOGICAL DEFINITIONS[1]
   ;Features
    ;neutral
 (face-neutral act (((mouth-neutral 100 400)
                    (eyes-neutral 0 300)
                    (brows-neutral 0 500))))
 (brows-neutral act (((left-brow-neutral 0 400)(right-brow-neutral 0 400))))
 (left-brow-neutral mot-lev (((Bll 0 400 30)
                              (Blc 0 400 30)    ;Brow, left, central
                              (Blm 0 400 30)))) ;Brow, left, medial
 (right-brow-neutral mot-lev (((Brm 0 400 30)
                               (Brc 0 400 30)
                               (Brl 0 400 30))))
 (eyes-neutral act (((upper-lids-neutral 0 100)(lower-lids-neutral 0 100))))
 (upper-lids-neutral mot-lev (((Eru 0 100 75)(Elu 0 100 80))))
 (upper-lids-open-wide mot-lev (((Eru 0 100 89)(Elu 0 100 94))))
 (lids-neutral act (((upper-lids-neutral 0 300)(lower-lids-neutral 0 200))))
 (mouth-neutral mot-lev (((Mb  0 200 15) ;Mouth, bottom
                         (Mlv 0 200 60) ;Mouth, left, vertical
                         (Mlh 0 200 40)
                         (Mrv 0 200 60)
                         (Mrh 0 200 40))))
 (mouth-in-n-shape   mot-lev (((Mb 0 200 15)(Mlv 0 40)(Mrv 0 40)(Mlh 0 50)(Mrh 0 50))))
 (head-at-zero-zero  mot-lev (((Hh 0 800 0)(Hv 0 150 0))))
 (head-diag-up-left  mot-lev (((Hh 0 1000 20)(Hv 0 1000 20)))) ;for debugging
 (gaze-at-zero-zero  mot-lev (((Plv 0 50 0)(Plh 0 50 0)(Prv 0 50 0)(Prh 0 50 0))))

 ;actions
 (raise-brows         mot-lev (((Bll 0 400 90)(Blc 0 300 100)(Blm 0 400 90)
                                (Brl 0 400 90)(Brc 0 300 100)(Brm 0 400 90))))
 (lower-brows         mot-lev (((Bll 0 400 5)(Blc 0 400 5)(Blm 0 400 5)
                                (Brl 0 400 5)(Brc 0 400 5)(Brm 0 400 5))))
 (brows-in-v-shape    mot-lev (((Bll 0 400 90)(Blc 0 300 40)(Blm 0 400 10)
                                (Brl 0 400 90)(Brc 0 300 40)(Brm 0 400 10))))
 (brows-in-roof-shape mot-lev (((Bll 0 400 10)(Blc 0 300 50)(Blm 0 400 90)
                                (Brl 0 400 10)(Brc 0 300 50)(Brm 0 400 70))))
 (brows-in-n-shape    mot-lev (((Bll 0 400 50)(Blc 0 300 90)(Blm 0 400 50)
                                (Brl 0 400 50)(Brc 0 300 90)(Brm 0 400 50))))
 (brows-in-pensive-shape mot-lev (((Bll 0 400 95)(Blc 0 300 40)(Blm 0 400 40)
                                   (Brl 0 400 50)(Brc 0 300 10)(Brm 0 400 5))))
 (squint mot-lev (((Elu 0 300 60)(Eru 0 300 60)(Ell 0 300 20)(Erl 0 300 20))))
 (half-closed-eyes mot-lev (((Elu 0 500 50)(Eru 0 500 60))))
 (lower-lids-neutral mot-lev (((Erl 0 300 30)(Ell 0 300 50))))
 (lower-lids-up mot-lev (((Ell 0 300 0)(Erl 0 400 0))))
 (pull-l-mouth-corner mot-lev (((Mlh 0 500 90))))
 (quickly-glance-sideways-and-back act (((gaze-right 0 100)(gaze-at-zero-zero 100 100))))

 ;emblems
 (shake-head act (((turn-head-left      0  50)
                  (turn-head-right    50 100)
                  (turn-head-left    100 100)
                  (head-at-zero-zero 150 50))))
 (nod        mot-lev (((Hv 0 105 -15)(Hv 105 100 0))))
```

1. Morphological behaviors are behaviors that are named after the way they *look*. Contrast with behaviors that are named after what they *do*—i.e. functional definitions.

```
 (wink       mot-lev (((Elu 0 100 0)(Elu 300 100 90))))
 (say-ahh    mot-lev (((Sp 0 250 "[_<,110>aa<550,100>]"))))
 (gaze-up    mot-lev (((Plv 0 100 30)(Plh 0 100 40)(Prv 0 100 30)(Prh 0 100 40))))
 (gaze-away  mot-lev (((Plv 0 100 20)(Plh 0 100 -30)(Prv 0 100 20)(Prh 0 100 -30))))
 (gaze-right mot-lev (((Plv 0 100 20)(Plh 0 100 40)(Prv 0 100 20)(Prh 0 100 40))))

 ;emotional emblems
 (smile mot-lev (((Mlh 0 400 99)(Mrh 0 400 99)(Mlv 0 200 99)(Mrv 0 200 99))))
 (smile-a-little mot-lev (((Mlh 0 400 99)(Mrh 0 400 99)(Mlv 0 200 88)(Mrv 0 200 88))))
 (grin-broadly   mot-lev (((Mrh 0 1000 80)(Mrv 0 500 60)(Mlh 0 1000 80)(Mlv 0 500 60))))
 (grin-a-little  mot-lev (((Mrh 0 1000 75)(Mrv 0 500 55)(Mlh 0 200 40)(Mlv 0 200 50))))

 ;self adjustors
 (blink act (((close-eyes 0 50)(open-eyes 50 50))))
 (blink-slowly act (((close-eyes 0 300)(open-eyes 300 200))))

 ; FUNCTIONAL DEFINITIONS
  ;Back channel feedback / turn control
 (say-aha   mot-lev (((Sp 0 250 "[_<,110>aahxaa<250,130>]"))))
 (look-pensive act (((gaze-away 0 100)(pull-l-mouth-corner 300 500))))
 (look-aloof   act (((gaze-away 0 50)(turn-head-left 200 1000)(raise-brows 800 800))))
 (look-puzzled act (((squint 200 200)(brows-in-roof-shape 0 400))))
 (look-drowsy  act (((half-closed-eyes 0 800)(lower-lids-neutral 0 400))))
 (show-give-turn act (((face-neutral    0  200)(gaze-at-zero-zero 0 100)
                       (head-at-zero-zero 0 600)(raise-brows     0  200))))
 (show-take-turn act (((open-mouth-wide 0  100)
                       (quickly-glance-sideways-and-back 0 300)(blink-slowly 300 400))  ;option 1
                     ((eyebrow-greet   0  500)(quickly-glance-sideways-and-back 0 200)))) ;option 2
 (hesitate act (((say-ahh      0 400))   ;option 1
                ((gaze-up      0 200))   ;option 2
                ((look-pensive 0 600)))) ;option 3

; Notice that show-give-turn is controlled from the DKB, but should be composed completely here.
; Below action for illustrative purposes only - 2/19/96
;(show-give-turn act (((look-at user))))

 (show-listening act (((blink-slowly  0 500))))
 (back-ch-feedb-normal act (((say-aha 0 100)) ;option 1
                            ((nod     0 200)))) ;option 2

 ;other
 (happy act (((raise-brows 0 400)(brows-in-n-shape 400 200)(lower-lids-up 0 300)
                (open-eyes-wide 0 300)(smile 0 200))))
 (greet act (((eyebrow-greet 0 1500))))
 (eyebrow-greet act (((raise-brows 0 200)(upper-lids-open-wide 0 200)
                        (brows-neutral 900 200)(eyes-neutral 1000 300))))
 ;acknowledge
 (ack-normal    act (((say-ok-normal 0 250))))
 (say-ok-normal mot-lev (((Sp 0 250 "[_<,120>ow<,130>kehiy<250,95>]")))) ;speech - sent to DecTalk
 (say-ok-bored  mot-lev (((Sp 0 250 "[ow<,130>k<100,100>ehiy]"))))       ;speech - sent to DecTalk
 (say-all-right-normal mot-lev (((Sp 0 250 "[<,120>ow<,130>lraet<250,95>]"))))

 ;other
 (close-eyes         mot-lev (((Eru 0 300 10)(Elu 0 300 10))))
 (open-eyes          mot-lev (((Eru 0 300 75)(Elu 0 300 80))))
 (open-mouth-wide    mot-lev (((Mb  0 400 60))))
 (close-mouth-tight  mot-lev (((Mb  0 300 15))))
 (open-eyes-wide     mot-lev (((Elu 0 100 99)(Eru 0 100 95)(Ell 0 100 95)(Erl 0 100 95))))

 ; WILDCARD SPEECH
  ;Star is replaced by a value from the TKB [topic knowledge base] or the DKB [dialogue knowledge
base]
 (deliver-speech     mot-lev (((Sp 0 250 *))))



 ; SPATIAL BEHAVIORS
  ;gaze ;Star is replaced by a value from the spatial knowledge base
 (gaze-at-user spatial-mot-lev (((Plh 0 250 *)(Plv 0 250 *)(Prh 0 250 *)(Prv 0 250 *))))
 (look-at spatial-mot-lev (((Plh 0 250 *)(Plv 0 250 *)(Prh 0 250 *)(Prv 0 250 *))))
 (turn-to spatial-mot-lev (((Hh  0 1000 *)(Hv  0 800 *))))
 ;head (mostly for debugging)
(turn-head-toward spatial-mot-lev (((Hh 0 900 *)(Hv 0 900 *))))
(turn-head-left  mot-lev (((Hh 0 150 -20))))
       (turn-head-right mot-lev (((Hh 0 150  20))))
(turn-head-up       mot-lev (((Hv 0 300  20))))
(turn-head-down     mot-lev (((Hv 0 300 -20))))
(head-at-zero-horiz mot-lev (((Hh 0 500   0))))
(head-at-zero-vert  mot-lev (((Hv 0 500   0))))
(turn-head-90-left  mot-lev (((Hh 0 150 -90))))
(turn-head-90-right mot-lev (((Hh 0 150  90))))
(turn-head-45-right mot-lev (((Hh 0 500  45))))
(turn-head-45-left  mot-lev (((Hh 0 500 -45))))

; MANUAL GESTURE
 ;morphological defs
(gest-rest              mot-lev   (((Gr 0 1000 0))))
(hand-raise-palm-fwd    mot-lev   (((Gw 0 600  0))))
(drum-with-fingers      mot-lev   (((Gd 0 600  0))))
(point-to-big-screen    mot-lev   (((Gp 0 1000 0))))

 ;functional defs
(gest-slow-beat         mot-lev   (((Gb 0 1500 0))))
(gest-fast-beat         mot-lev   (((Gb 0 600  0))))
(manual-hold-it-signal  act (((hand-raise-palm-fwd 0 900))))
(gest-greet      act (((hand-raise-palm-fwd 0 600))     ;option 1 - fast
                      ((hand-raise-palm-fwd 0 1000))))  ;option 2 - slower
(restless act (((drum-with-fingers 0 400))))
)) ;End all
```