Ymir: A Generative Model of Psychosocial Dialogue Skills

7.

In the past chapters we have looked at the complex issues involved in human face-to-face interaction. Some of those have been addressed individually in the literature, while some have not. The biggest piece missing though is a general way to put these items together to create a full model of face-to-face communication. The argument made here is the following: We need to look at the full loop of perception-action of an agent to come up with a correct model, because actions in dialogue are a mixture of closed-loop (guided with perceptual feedback) and open-loop (ballistic), and dialogue is interactive, with real-time planning happening on many levels. To do this we need a foundation where components that have already been developed can be accommodated, and new developments in the theory of multimodal dialogue can be "plugged in", tested, redesigned and retested.

Why "Ymir"?
Nordic religion, as preserved in Icelandic Sagas [Sturluson 1300~1325], tells about Ymir—a giant who lived in times before the heaven and earth. Ymir was killed by the Nordic gods Ó›inn, Vili and Vé, who turned Ymir's "blood into the seas, his bones into the mountains, his teeth and broken bones into rocks and gravel, his head into the heaven and his flesh into the earth." The earth then became a source of many new imaginative humanoid life-forms.

Ymir is pronounced *e*-mir, with the accent on the first syllable.

In this chapter I propose a new generative model of human psychosocial dialogue skill. Instead of dealing with a single issue, or a few small elements of face-to-face, multimodal dialogue, this model is intended to be a bridge, addressing all issues necessary to fill the gaps which in the past have prevented us from creating artificial characters that can engage in such dialogue.

The model is Ymir. Ymir does essentially what Fehling et al. [1988] call resource-bounded problem solving. The problem is dialogue; the resources are time, information and computational power. On the practical side, the general thought is that Ymir be used for creating softbots (and robots) whose purpose in life is to receive commands from humans, ask questions when appropriate, but otherwise do the job as best their knowledge allows them to. In the following discussion we can therefore envision building up to a humanoid robot who receives commands and turns them into executable actions in its domain of expertise. On the theoretical side, Ymir could be used to test theories about human discourse, because it provides the possibility to turn cer-

tain dialogue actions on and off at will—something that was impossible to do before, even with a skilled actor.

The approach taken to dialogue expertise can be likened to that taken to an expert system: we want a system that is expert at multimodal, face-to-face communication. The justification comes from the fact that unlike expert systems that tackle a niche area for limited purposes, multimodal dialogue is a general communication method used by all, and therefore we need only build this system once.

7.1 Overview of Architectural Characteristics

Following the model of face-to-face dialogue introduced in Chapter 5. (“Multimodal Dialogue as Layered Feedback Loops” on page 77), Ymir is a layered system. It employs one or more topic knowledge bases, and it uses a special action scheduler module for composing and scheduling motor actions. Motor actions are expected to be carried out by an animation system that either has addressable absolute positions for each “muscle” (analogue—e.g. a servo system, or digital—e.g. computer graphics or stepper motors), that lies below the system itself.

Sensory input is expected to be multimodal, and although the system *works* with a single mode input, no advantages would be taken of multimodal synergistic effects in that case. Ymir can accommodate any number of sub-modules, running in parallel or serial, that work in concert to interpret and respond to the dialogue as it unfolds. By being modular, Ymir offers researchers opportunities to experiment with various computational schemes for handling specific sub-tasks of multimodal interaction, such as natural language parsing, natural language generation and arbitration of multimodal motor responses. At the highest level, Ymir makes no specifications about the content of particular agent behaviors or interpretive processes and is therefore culture-independent.[1]

Ymir addresses all the features of dialogue presented in Chapter 5. (page 65). A summary of these manifests itself as the following list of requirements, all of which Ymir fulfills:

1. Co-existence of *reactive* and *reflective* behaviors,
2. *incremental interpretation* co-exists with *real-time response* generation, providing *seamlessness*, and
3. it handles *multiple data types* (spatial, boolean, symbolic, analogic).

1. Just like a telephone asks no questions about the language spoken on it, Ymir is not limited to the conversational rules of any single culture.

Features of three AI approaches have been adopted in Ymir: *Blackboard systems* [Adler 1992, Nii 1989, Engelmore & Morgan 1988, Selfridge 1959], *Schema Theory* [Arbib 1992] and *behavior-based* systems [Maes 1990a, 1989]. In the broadest sense, Ymir uses multiple knowledge sources that cooperate to provide a solution to a problem—in this case to interpret user actions and generate appropriate responses. Like Schema Theory, Ymir is highly distributed and contributes therefore to research in distributed artificial intelligence (DAI) [cf. Bond & Gasser 1988, Huberman 1988, Huhns 1987]. In contrast to the many approaches proposed various problem domains in the AI literature, the main novel and distinguishing features of Ymir are:

1. A distributed, modular approach to perception, decision and action.
2. A layered combination of reactive and reflective behaviors.
3. Dialogue-related interpretation is separated from topic interpretation.
4. Dialogue management is viewed as having complete process control (*when* something happens as opposed to *what* happens) of overt and covert actions.
5. Motor actions are split into two phases; a decision (or intentional) phase and a composition/execution phase.
6. Intentions to act vary in their specificity: the more specific an intention is (e.g. blinking) the fewer morphologies (ways to do it) exist; the less specific it is (e.g. looking confused) the more options there are in the way it will eventually be realized.
7. The final morphology of an intention is chosen at run-time.

Following Nii [1989] we can describe a computational system at any of three levels: The *model* is the least specific, showing the ideology behind the approach, the *framework* is more specific, detailing the pieces of the system and their interconnections, and the *specification* being the most detailed one, showing how to implement the particular system. In this chapter we will focus on the model and framework perspectives. We will turn to a more in-depth look at the implementation in the next chapter (page 111). Now let's get an overview of Ymir's main elements.

7.2 The 6 Main Elements of Ymir

The six main types of elements in Ymir are:

1. A set of semi-independent processing layers, Γ.
2. A set of blackboards, Φ.
3. A set of perceptual modules, ρ.
4. A set of decision modules, Π.

5. A set of behaviors, β, and behavior morphologies, βm (specific motor programs).
6. A set of knowledge bases, κ.

Starting with the layers, we will now take a closer look at these six elements. Then we will go into the Blackboards, followed by a discussion on virtual sensors, multimodal descriptors and decision modules (starting on page 97), and the behaviors and behavior morphologies.

7.2.1 Layers

There are four layers in Ymir:

1. *Reactive Layer (RL).*
2. *Process Control Layer (PCL).*
3. *Content Layer (CL).*
4. *Action Scheduler (AS).*

Each of these layers contains particular element types:

$$\Gamma_{(RL)} = \{\rho, \Pi\}$$

$$\Gamma_{(PCL)} = \{\rho, \Pi\}$$

$$\Gamma_{(AS)} = \{\beta, \beta m\}$$

$$\Gamma_{(CL)} = \{\kappa\}$$

FIGURE 7-1. The Reactive Layer contains mainly two types of processes, {1} perceptual (sphere and prism), and {2} decision modules (cube).

Each layer contains processes with similar time-specificity and functionality. The PCL is the main control element, with partial control over the other three layers. Processes in the RL can exert limited process control. Processes in the he Reactive and Content Layers perform functional[2] and content analysis of input. The AS produces specific motor morphologies, while knowledge bases in the CL interpret input about a specific domain and produces actions that are applicable in response to the content of that input. Different kinds of delays and delay constants exist at each of its four levels. Time stamping and the use of synchronized clocks is a general way to deal with temporal constraints: Using time stamping, delays are logged and treated like any other data in the system. We will now take a closer look at each of the layers.

2. See "Functional Analysis: A Precursor to Content Interpretation and (sometimes) Feedback Generation" on page 71.

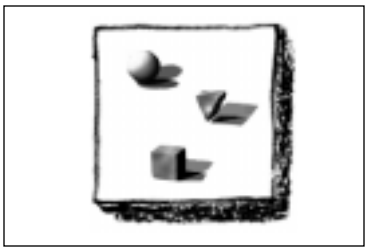

FIGURE 7-3. The Process Control Layer contains two kinds of processing modules (in multiples), {1} perceptual modules (sphere and prism) and {2} decision modules (cube).

Reactive Layer (RL)

The role of processes in the Reactive Layer is to compute timely information on user actions and make these available to decision modules in the same layer. Sensory data from each mode is processed with modules called *virtual sensors*, and mode-specific data is combined in processes called *multimodal descriptors*. Reactive Decision Modules use this data to issue actions with a relatively high speed/accuracy trade-off. I.e. as long as the results are quick, and correct more than 50% of the time—above chance performance[3]—it doesn't matter that they are less than 100% accurate, because *a* response is more important than it being correct.

Computation at this lowest level is assumed to be "immediate", i.e. without delay. This simplification can be made by using computing mechanisms that are significantly faster than the fastest response needed in human interaction, ideally a fraction of 100 msec. In spite of actions being immediate in this layer, every event at the RL level is nonetheless time stamped for the benefit of computations in other layers.

ACTION: smile
CREATOR: PCL
EL: 600 ms
STAMP: 35243

FIGURE 7-2. Example of a behavior request message sent from the Process Control Layer to the Action Scheduler.

Process Control Layer (PCL)

The role of processes PCL is to control global aspects of dialogue: to turn the correct kinds of internal processes on and off, recognize the global context of dialogue and manage communicative behavior of the agent. It deals with issues such as *when* a question should be answered; what to do when information is missing, when to greet; etc. It also controls internal actions related to the dialogue such as starting to listen, making predictions about the knowledge needed in a particular interaction; making predictions about what to expect next; managing multi-turn information exchange, etc. Modules in the PCL can control (turn on and off; change thresholds in) the processes in the Reactive Layer, as well as its own. This feature is essential since many of the lower level processes don't have global enough information to decide when they should be active and when not.

3. This is not to say, of course, that striving for as high a recognition accuracy as possible should not be tried.

In addition to the perceptual and decision modules, the PCL also has a few processes specifically related to "book keeping". These will be discussed in Chapter 8.

Action Scheduler (AS)

The AS can be thought of as a kind of "cerebellum". Its role is to receive *behavior reques*ts (βr) from the Reactive, Process Control and Content Layers, prioritize these and choose a specific morphology for them (see "Morphological and Functional Substitutability" on page 76). There are many ways to realize behavior requests, which can be thought of as the "intention" to perform a specific act: one example is given in Figure 7-2. A behavior request is thus a decision to do a specific action, independent of its form. Behavior requests can be specified at various levels of detail. Specific behavior morphologies (βm) are chosen from a library of alternatives[4]. Increasing generality in the specification of a behavior, e.g. "pull left corner of mouth up halfway" vs. "smile", means more options in its morphology (see page 101).

To chose between βm options, the AS uses a trade-off algorithm. To take an example of how the algorithm works, if the AS receives a request for the behavior **acknowledge**, it can use dialogue state and the amount of load on the various degrees of freedom of the agent's motor system to choose a way to express this. The usual method could be to say "yes", but if the user is speaking, perhaps a nod would be more appropriate; however, if the agent's head is moving, it might choose to give verbal back channel feedback anyway. There are many ways to realize such a scheduling algorithm; we will see one particular method in the next chapter.

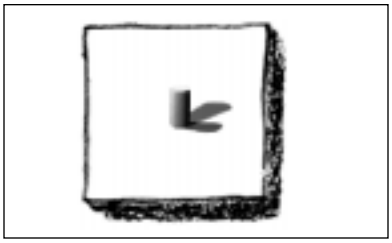

FIGURE 7-4. The Action Scheduler Layer contains {1} a lexicon of behaviors and behavior morphologies (cylinder), as well as {2} scheduling mechanisms for requests to perform these (not shown).

4. Complex behaviors that span long stretches of time need to be interruptible, as well as computed incrementally to allow relevant, unexpected events to be taken into consideration.

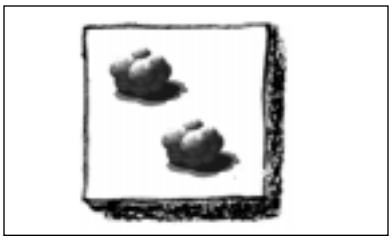

FIGURE 7-5. The Content Layer contains several knowledge bases (shown as cloud-like blobs).

Content Layer (CL)

The role of the CL is to host the processes that make sense of the *content* of the input and generate acceptable responses based on this. For example, given the multimodal input "Delete [gesture] those", the CL should be able to combine the verbal and gestural actions, and come up with a correct action in the topic domain (i.e. remove a set of objects).

The Content Layer contains one or more Topic Knowledge Bases (TKBs), which contain information about how to interpret a person's multimodal acts, how to generate responses to those acts, and how to communicate its status to other parts of the system, particularly the Process Control Layer. The CL also contains a Dialogue Knowledge Base (DKB), which stores meta-knowledge about all other knowledge bases in the layer. A meta-knowledge DKB allows the system to select the most relevant TKB at any point in time.[5] To take an example, if the agent knows the two topics of music and computer graphics, and hears the utterance "Turn the blue box sideways" the DKB will recognize that this utterance probably refers to the computer graphics topic, and will notify the TKB containing the knowledge necessary to interpret computer graphics-related utterances, which will in turn interpret the input and generate some actions that will make the user's wish come true.[6]

Once a Topic Knowledge Base has generated usable output, it will notify the DKB, which has the necessary knowledge to know *when* to execute this action in the dialogue.

5. Alternatively, we can run a DKB in parallel with any one (or more) TKBs which is considered relevant at any point in time. Output from the KB that produces the best interpretation will be selected. This has two consequences: {1} The KBs have to give a measure of their success and{2} equally good interpretations from different KBs have to arbitrated. This can be done in many ways, including asking the speaker a question, and because the conflict happens at the meta-level, that question would be composed in the DKB.
6. If the DKB is uncertain which KB to pipe the input to, it might pipe it to more than one TKB and choose the outcome that is rated as a "good recognition" by the TKB.

In Ymir, the DKB is considered a central part of the system's psychosocial skills. In fact, any knowledge that has to do with dialogue, such as knowledge about participants, their body parts, instruments used in interaction (mouths, hands, eyes, etc.), greetings, good-byes, etc., rightfully belongs in the DKB and are considered a topic (albeit a meta-topic) in and of itself. The argument behind this view is the same as behind the push to embody the computer: knowledge about interaction is intrinsic to the interaction and necessary to conduct it correctly. While history about the task—be it excavation or moon landings—is stored in the relevant TKB, history about the interaction, and references to the interaction ("Go back to when I told you ..."), are stored and treated in the DKB. The big win in this modular approach is that it allows for the one-time creation of dialogue knowledge ("Look over here", "Listen to me"), with domain-dependent knowledge being "plugged in" by the agent's designer (and by the system at run-time) in a modular fashion.[7]

Summary of Layers

Placing these four parts of Ymir within Coordination Theory (see Figure 5-2 on page 69) [Malone & Crowston 1991, Crowston et al. 1988, Malone et al. 1988], the Reactive Layer and the Process Control Layer are product-oriented hierarchies: they contain a complete set of heterogeneous processes to produce a product—the product being external and internal actions. The Content Layer is also a product-oriented hierarchy: its job is to produce descriptions from the user's commands that can be executed in the agent's world. (For example, upon hearing the words "Delete the blue box", the system should be able to locate the I.D. of the blue box, find the correct command to remove items, and apply that command to the I.D. of the requested object.) The Action Scheduler, however, repetitively carries out the same kind of process, and is thus what Coordination Theory calls a functionally-oriented hierarchy.

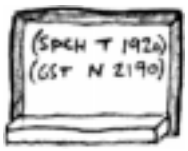

FIGURE 7-6. Blackboards contain information accessible to a certain set of modules.

7.2.2 Blackboards

How do all these processes talk to each other? Psychological research has shown that in perceptual processing, different information becomes available at different times: for example, low-frequency visual information and motion becomes available sooner than higher-frequency information [Card et al. 1983]. A person can select how long to wait before reacting to a particular stimulus, depending on the current trade-off between cost of delay and cost of errors. This points to a process where information that has already been computed is made available to the rest

7. I would like to thank Richard A. Bolt for pointing me to the issue of modularity; see also Walker & Whittaker [1990].

of the system. In A.I. a blackboard is a metaphor for a global information exchange [Selfridge 1959]. Any process with access to a blackboard can look for information relevant to its own processing. In Ymir we use the idea of more than one blackboards, each with limited access. There are three main blackboards. The first one is for information exchange primarily between the Reactive Layer and the Process Control Layer. This blackboard is called the *Functional Sketchboard* (FS). It stores intermediate and final results of low-level (high-speed) perceptual processes such as motion, whether the agent hears something or not, and first-pass multimodal descriptions.

The second is the *Content Blackboard* (CB), servicing communication between the PCL and the Content Layer. This blackboard is the key to the separation of process control and content analysis in Ymir. Here results are posted that are less time-critical than those on the Functional Sketchboard.

The third blackboard in Ymir is the *Motor Feedback Blackboard* (MFB), where the Action Scheduler posts the progress of behaviors currently morphing and executing. In the MFB the PCL and CL can read status of formerly initiated behaviors and replace those that are cancelled or have failed for some reason.

7.2.3 Perceptual Modules

This section explains input processes of an agent, and input data representation. There are currently two kinds of perceptual processes in Ymir, *virtual sensors* and *multimodal descriptors*. But before describing these, let's look at the philosophy behind the approach.

Background

The organization of low-level perceptual processes in Ymir follows the so called purposive, qualitative, or animate perception approaches [Aloimonos 1993] in that it is purpose-directed and ego-centric. It is based on the general idea that a situation is an important factor in selecting the perceptuo-motor skills of an animal, to maximize attentional and mental faculties for the particular tasks that situation calls for. This is closely related Agre & Chapman's [1990, 1987] *indexical-functional* representation in their *Pengi* system, where objects and dependencies are represented in terms of the effect they have on the agent's goals, in ego-centered terms, e.g. instead of representing a flying bee with the symbol `BEE-23` it uses labels like `the-bee-i-am-now-chasing`, defining the bee in direct relationship to the perceiver [Lyons & Hendriks 1992].

At higher levels the system's perception becomes more complex, slower and more general. Here we can expect the agent to be recognizing objects, faces, body parts, and relating them to its lexical and relational knowledge bases, to generate verbal output for instance. These kinds of processes are not specified in Ymir (this is quite a research topic), but general ideas about how these could be handled in a dialogue system will become apparent as we continue.

The perceptual abilities of an agent are assumed to be grounded in knowledge about the *interaction*, such as knowledge about participants, body parts, turn taking, etc., by *situational indexing*. Thus, an agent created in this architecture is not expected to be trying to avoid obstacles, prevent itself from getting killed, etc., while it engages in interaction with humans, and therefore does not require any non-communication based perception. For instance, upon hearing a voice in a given (perceived) location, an orienting response toward that voice could be triggered. The architecture could of course be expanded to include perceptions of other things than only those relating to communication. How broad the ego-centered approach can be made is an open question. Now, let's take a look at the two perceptual elements in Ymir, Virtual Sensors and Multimodal Descriptors.

Virtual Sensors

The simplest processes in Ymir's perceptual system are the virtual sensors, which process simple features of the dialogue and output Boolean values. These sensors are considered to lie at least one level above the energy transducer layer, such as a retina or cochlea, or, in the case of the Gandalf system (Chapter 9.), the space sensing cubes, eye tracker and microphone. A virtual sensor is usually associated with a single mode: an example would be a vocalization sensor that turns on or off depending on whether the user is making sounds with his or her throat.

The virtual sensors in the Ymir system fall into the following categories:

1. *Prosodic*
2. *Speech*
3. *Positional*
4. *Directional*

Prosodic sensors track the intonation of the speech, pauses and volume of vocalization; *speech* sensors are a general class of processes that look at the speech content. They could for example be sensitive to certain words that play a role in dialogue orchestration such as cue phrases; they would also track the global functional aspects of speech, such as determining whether an utterance is syntactically correct, whether it makes sense pragmatically, what the topic is, etc., as much as these can be gleaned from just looking at the speech (more extensive analysis of

these is done at higher levels, albeit at a slower pace). Examples of each of these classes will be given in the chapter on Gandalf. *Positional* sensors track the absolute position of objects (position in real-space) or the relative position of two or more objects, e.g. displacement of the eyebrows from a resting position, and *directional* sensors track the direction of objects (e.g. gaze, trunk or head). Two fundamentally different kinds of virtual sensors are postulated:

1. *Static sensors*, which report a current static state to be true or false, and
2. *dynamic sensors*, which track the change of a certain feature of a single mode over time and report on the conditions over a given duration of time.

For example, a static sensor could report whether a person was looking at the agent's face or not. A dynamic sensor could report whether the person glanced away quickly and then back. The theory behind this is that short patterns of activity may have significance for the interaction [cf. Argyle, Lefebvre & Cook 1974].

Multimodal Descriptors

Processing the output provided by the virtual sensors is a net of what I call *Multimodal Descriptors*. The descriptors aggregate information from the Virtual Sensors to compute intermediate, multimodal "functional sketches" of the user's behavior. An example is a descriptor that tries to determine whether the user is giving the turn. Another might combine information from a spatial sensor and a speech sensor to provide a {SOUND, LOCATION} pair that can be used by the agent for orienting itself toward a person. Two kinds of descriptors are proposed,

1. *static descriptors* and
2. *dynamic descriptors*.

As with Virtual Sensors, *static* descriptors simply respond to a static situation, whereas *dynamic* descriptors detect patterns over time intervals—as reported by the virtual sensors—such as a specific combination of arm and eye movements for a given interval. For example, bringing up your hand and uttering something ("ahhh") while the agent is talking may constitute a wish to interrupt. This could be detected by a static descriptor sensitive to the conditions of either hand in gesture space and vocalization present. A head nod and a particular vocalization ("aha") combines into a single "back channel feedback" report, detected with a single dynamic descriptor.

Summary of Virtual Sensors and Multimodal Descriptors

We can now summarize the perceptual system. The virtual sensors receive data from the sensing equipment and do initial computations to prepare it. Multimodal descriptors monitor the status of the virtual sensors through a blackboard and change states. In the implementation of Ymir, this is based on first-order logic combinations of these, as well as the states of other descriptors. Later versions of Ymir might use Fuzzy Logic [c.f. Kacprzyk 1992] for this purpose, or other methods, provided they are fast and flexible enough.

7.2.4 Decision Modules

Decision modules look at the state of the agent's knowledge, which includes a representation of the outside world as well as the state of its own processing, and make decisions about what to do from moment to moment. These decision can affect both the outward behavior of the agent or the internal processing inside the agent's "mind", and thus fall broadly into two categories:

1. *External Decision Modules*—those that initiate overt actions, and
2. *Internal Decision Modules*—those that only change the internal state.

Granularity of the modules varies according to their task. Each decision module contains knowledge about where to look for data (which blackboard), what to do with it and how to communicate its status to other modules by posting information to the blackboards.

Decision modules in the Reactive Layer search for specific conditions in the Reactive Layer's Functional Sketchboard; the Process Control Layer's decision modules can look for conditions in both the Functional Sketchboard and the Content Blackboard (see "Blackboards", above).

7.2.5 Representation of Behaviors

We are now ready to look in detail at the fourth and last layer in Ymir, the Action Scheduler.

Background

Research on errors in human and animal locomotion have supported a model in which distinct levels of representation are at work for any motor act [Rosenbaum et al. 1992]. For example, levels activated earlier provide information spanning longer stretches of time, e.g. the global act of moving your arm/hand/finger to enter the expression "27 + 9 + 3" into a calculator. Levels actuated later provide smaller and smaller constituents for that behavior, e.g. individual key presses. This model

would indicate that *information* needed to execute an act like that would also need to be represented at multiple levels: locating the calculator in real-space is more coarse than locating its individual buttons. Rosenbaum et al. [1992] have proposed what they call the Knowledge Model: Motor control is performed by autonomously functioning modules that compete for execution, and that these modules carry information about postures. This model, and similar ones [Rosenbaum et al. 1991, Albus et al. 1987] are aimed at explaining complex motions like those of the arm moving the hand to press a button in an elevator, all the way down to the feedback provided from the muscles.

Behaviors & Behavior Morphologies

The approach taken here is in some ways similar to Rosenbaum et al.'s [1992]. The idea of stored postures is used in the Action Scheduler, as is the idea of hierarchical storage of increasingly smaller units. However, choosing between alternative actions is done by a monolithic algorithm, not competing individual modules. In Ymir, action is split into two phases: an action request (or intentional, decision) phase and a composition/execution phase. As discussed in the section on the Reactive and Process Control layers, the first phase is based on a collection of decision modules that can request specific actions, specified at varying levels of detail. The second phase happens here, in the AS. This method for representing behavior leads to a database where functional and morphological definitions co-exist in the same space, with no distinct division lines between the two classes. An example of what such a database could look like is given in the chapter on Gandalf (Section 9.7.1 on page 153).

Behaviors are indexed at various levels of detail: `smile`, `pull-corners-of-mouth-up`, `move-motor-x-to-position-y`. Obviously, there must be hundreds of ways a person could smile, fewer ways in which one could pull the corners of mouth up, and perhaps only one way to move a muscle to a particular location. We can make a tree, where particular morphologies are given as the tree's leafs (Figure 7-7). As we travel up the tree, the flexibility for various implementations of a particular act, like smile, or show-taking-turn, increases. Of course, given more options, it takes longer to choose the best one. Decision modules in the Reactive Layer related to external behavior generally request highly specific actions; those in the Process Control Layer usually make a more general specification.

Action requests issued by the RL are generally specified at a lower level than those in the PCL, since these are under tighter time constraints (with the effect that the AS—see below—doesn't have to spend valuable time composing a set of motor commands for the action involved, but simply looks up the default motor schema and sends it to the anima-

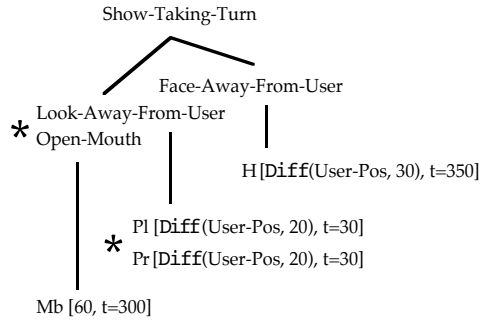

FIGURE 7-7. The hypothetical behavior *Show-Taking-Turn* has two possible instantiations, *Turn-Away-From-User* and the parallel pair {*Look-Away-From-User, Open-Mouth*}. Each of these point to low-level motor commands with degrees and time in milliseconds. The function Diff returns a setting that is guaranteed to not include the user's position in the agent's line of sight. Parallel actions are marked with a star.

H = agent's head, P = pupil (left and right), t = time in milliseconds, other numbers represent degrees (and relative position in the case of motor Mb), Mb = bottom mouth motor (see Figure 8-11 on page 123).

tion module). Examples of actions issued by the RL are `show-taking-turn` and `look-at-person`. Examples of the more extensive actions included in the PCL are `indicate-response-delay` and `express-confusion`. To determine which layer a potential action should belong to, one can use time-specificity (i.e. those that typically have expectied lifetimes under 1 second are likely to belong in the RL than the PCL; see "Temporal Constraints" on page 69), and/or the time the actions spans (RL if less than 1 second). Another criteria is the kind of perceptual data the behavior needs to be executed reliably; those requiring complex data are less likely to belong to the RL. Guidelines for determining which layer a particular behavior belongs to are likely to emerge out of further research on this topic.

Generating Manual Gesture

The dialogue management system, by itself, can support the generation of four classes of dialogue-related manual gesture [Rimé & Schiaratura 1991], independent of the topic knowledge base(s) used[8]. These are {1} *emblem gestures* related to the dialogue (e.g. holding up a hand to signal "Stop speaking!"), {2} *deictic gestures* (involving objects in the dialogue knowledge base), {3} *beats* and {4} *butterworths* (see Figure 3-2

8. Since iconic, pantomimic and deictic gestures related to the topic of discussion cannot be generated without reference to knowledge of the topic, and the knowledge residing in the dialogue system contains no topic knowledge, these would be generated in the corresponding knowledge base.

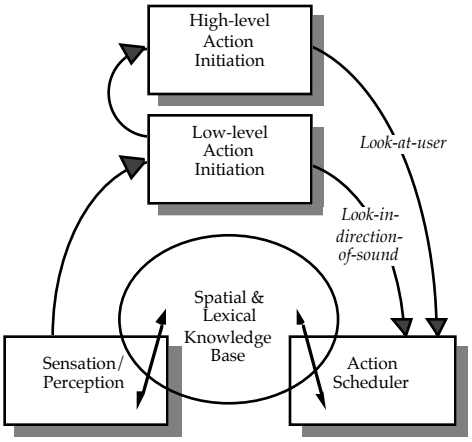

FIGURE 7-8. The Action Scheduler has access to a spatial knowledge base that is kept updated by the sensory and perceptual mechanisms. Examples of messages sent to the AS from the Reactive and Process Control layers are shown in italic letters.

on page 44). These are all requested from the RL and PCL by calling the appropriate type of gesture with the optional parameters (such as a 3-D vector for deictic gestures) and treated in the same way as other actions in the Action Scheduler.

Any gesture related to the topic should be generated in the corresponding Topic Knowledge Base.

Spatio-Motor Skills

To allow an agent to move in relation to surrounding objects such as a person or a task area, the AS needs access to a spatial knowledge base. Examples of such actions would be `Look-at-User` and `Turn-to-Area-[X]`[9]. I propose that this should be done with access to a common spatial knowledge base that is fed with information from the sensors[10] (Figure 7-8).

9. This behavior, unlike the other examples, contains a variable. There is nothing in Ymir that excludes such modules—in fact, for complex behaviors they will prove essential.

10. A simplified version of this approach has been implemented in Gandalf (Chapter 9., page 129).

When do we go Ballistic?

When, in the process from intention to execution, does an action, or part of an action, become impossible to cancel? This question about where to go ballistic is an important one. As we established in Chapter 5. the incrementality and reactive nature of dialogue allows participants to interrupt each other at a moment's notice. In Ymir, once the AS has sent the commands out, the agent's behavior is ballistic, i.e. there is no way to cancel their effect after this point. This last part of the path should therefore be kept very short, typically less than a second. For actions longer than a second, one would expect them to be composed—or at least *executed* [Kosslyn & Koenig 1992]—incrementally so that they can be cancelled at any time.

Creating Behavior Classes with Cascaded Decision Modules

The idea behind cascaded decision modules is this: Suppose you're walking along a narrow trail in the woods and you've decided to put your food down somewhere, when suddenly you realize it looks like a puddle and you don't want to get wet. So you cancel your original motion goal and replace it with a new one. The new goal results in a motion that moves your foot to a different location along a different path. The new goal in this example is one decision module of a group of cascaded decision modules, all aimed at placing your foot in various ways on the ground. By cascading a number of decision modules, corresponding to a number of behavior morphologies, each triggered in the case of another's cancellation, inappropriateness or failure, whole classes of behaviors can be built up.

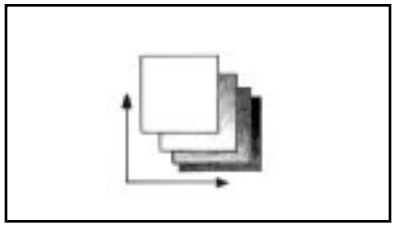

FIGURE 7-9. Cascading decision modules with slightly different triggering timing (abscissa) and conditions (mantissa) allows us to cover classes of behaviors.

For example, if the agent performs the action `Show-Taking-Turn` but the user continues to speak, the `Show-Taking-Turn` behavior must have failed somehow, either in execution or in indicating to the user what it was intended to show—or perhaps the user just decided to ignore it. The outcome in any case is the same: the user simply continues to speak, as measured by the virtual sensors. But how should the agent respond? It has already decided to take the turn, and may have stopped listening, yet has been unsuccessful in showing this to be the case. And its `Show-Taking-Turn` module has already fired (and won't fire again unless it is reset). The solution to this apparent deadlock lies in designing a second module, called `Show-Take-Turn-2`, that activates if the two states of user-speaking and `Agent-Has-Turn` occur simultaneously for a longer-than-normal period, e.g. 500 ms. The outward behavior resulting from this second decision might be more exaggerated than that of the first, including perhaps a manual gesture to try to show the user that the agent really wants the turn.

Another example of using cascaded behaviors is the situation where you know have been asked a question, and you have already shown that you're taking the turn, but you haven't come up with the answer yet. To deal with this delay, people engage in various verbal and non-verbal activities; they look up, say "ahhh" or fill in with more extensive elaborations like "I know this...hang on a second." With cascaded behaviors, delays as these can be taken care of automatically depending on the duration from the time you took the turn. The decision module/behavior `Produce-Filler` could produce an initial "ahh", `Produce-Filler-2` might be designed to take care of an additional delay. Yet other decision modules could take care of pauses that hadn't been filled but should have been—we can imagine a collection of 10-20 such modules, each with slightly differing conditions for triggering. This puts the creation of the behaviors in the hand of the agent designer, but the selection of each option in the hands of the run-time system.

Using cascaded decision modules (Figure 7-9), whole classes of condition-action situations can be addressed. How this is done for a particular agent—and this applies to all modules in the system— is largely an empirical task that depends on a number of factors including the potential users, their diversity, and cultural background.

7.2.6 Knowledge Bases: Content Interpretation & Response Generation

A knowledge base provides a system's ability to "understand" language, gesture, facial expressions and gaze. The topic of an interaction could revolve around car mechanics, architecture, plumbing, or the solar system. The knowledge base will provide the ability to key together actions such as pointing at a wall and the utterance "remove that wall" so that the multimodal act can be understood and an appropriate reaction to the command generated.

Ymir contributes to our understanding of interpretation in that it provides a framework for better specifying the *limitations* that interpretive processes have to work under—regardless of how they are exactly implemented. The major requirements the TKBs used in a real-time dialogue system have to fulfill:

1. Processes have to be *incremental*.
2. Processes have to be *reportable*.

Incrementality means that interpretation happens at a fine enough granularity to allow meaningful communication between knowledge bases and the PCL to happen during a user's utterance. *Reportability* refers to the system's ability to report on its own status and progress. These principles are the basis for allowing a modular approach to knowledge

representation. Other benefits are also expected. If these principles are adhered to, the system's ongoing topic interpretation of a user's multimodal act can be constrained by the information posted to the functional sketchboard. For example, if the Sketchboard has an indication that a communicative gesture was made during the user's turn, gestural analysis of the segment can be started up in the TKB even before the user has stopped speaking. If the interpretation process encounters problems, an ordered list of progressively less likely functional roles for the multimodal act in question can be used to provide a next viable candidate. The Process Control Layer will take care of any delays that such a disruption may introduce into the interaction.

For each TKB, and the DKB, non-real time blackboards could be used for more exact functional analysis[11], using a high accuracy/speed tradeoff, but still working along the same lines as the sensor/descriptor system. An important requirement of this design is that the sketches provided by the fast analysis match more than 50% (= chance) of the analysis performed at the higher levels, because otherwise any real-time feedback based on these sketches is not correlated with the higher-level functioning of the system. To be sure, people often make mistakes in their functional analysis (e.g. mistaking a deictic nod for an acknowledgment) which they effectively mend in the process of the dialogue. But such mistakes are disruptive and hence should be avoided in a computer agent. We will come back to this issue in Chapter 9.

If achieving good correlation between low-level analysis and high-level interpretation proves to be a difficult task, there is at least one reason to be optimistic about this being a problem: If the user is following Grice's maxim [Grice 1989] about mutual cooperation in dialogue, over time, she is likely to modify her own behavior to make the two consistent in the agent. This modification would probably take only a few hours—certainly less than days—of interaction. So, if the agent consistently displays the same inconsistency between the real-time feedback and higher-level analysis—i.e. as long as there exists a morphology of user behavior that results in the same analysis on both levels—we may expect such problems to disappear over time.[12]

11. The exact implementation of the KBs is not specified in our architecture; this is one of the issues that should be left to later research.

12. In fact, data in the evaluation experiment of the Gandalf system (Chapter 10.) seemed to support this claim.

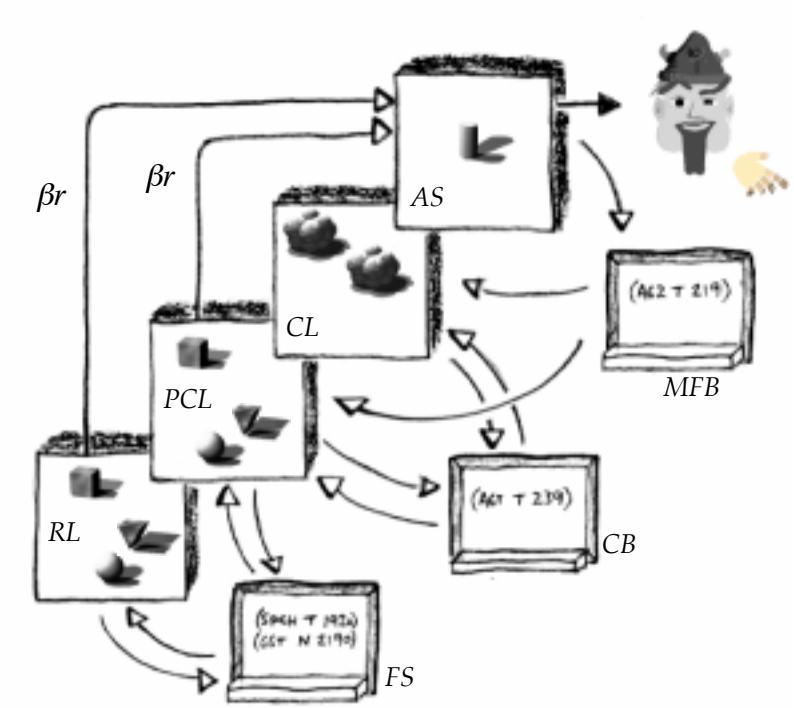

FIGURE 7-10. Overview of the Ymir architecture, showing the four layers, types of processes in each layer (square, sphere, prism, blobs, cylinder), communication links (hollow arrows) and motor control (black arrow), as well as the blackboards used for communication between the layers (βr = behavior requests, RL = Reactive Layer; PCL = Process Control Layer; CL = Content Layer; AS = Action Scheduler; FS = Functional Sketchboard; CB = Content Blackboard; MFB = Motor Feedback Blackboard). Not shown are the direct control links from the decision modules in the PCL to the Multimodal Descriptors in the RL. Multimodal information is assumed to be streaming in to each of the RL, PCL and CL as needed.

7.3 Ymir: Summary of all Elements

Figure 7-10 summarizes the layers in Ymir and provides an overview of their interconnections. Hollow arrowheads show the flow of information; the black arrowhead indicates absolute commands. Notice that flow between layers is not deterministic; each layer decides what to do with the data received from elsewhere according to time-constraints and the type of data. Layers further down the page are generally slower layers, with the exception of the Action Scheduler, which uses time as a variable to determine its processing. Not shown is the ability of State Decision Modules to turn the activity of perceptual modules in the PCL and RL on and off.

7.4 A Notation System for Face-to-Face Dialogue Events

To facilitate the discussion about the model presented above, and the way it treats dialogue, it would help to have some kind of simplified, clear way of presenting the complexities of face-to-face events.

We start by defining the following concepts:

- **A.** *Dialogue Condition* (C_d)
- **B.** *Action* (A)
- **C.** *Action Trigger* (A_t)
- **D.** *Expected Lifetime of an Action* (A_{el})
- **E.** *Action Execution Time* (A_{et})
- **F.** *Dialogue Participant* (P_d)

Any dialogue condition (C_d) could be a situation appropriate to respond to. It consists of a collection of necessary states of the dialogue participants' modes (hands, face, gaze, etc.), dialogue (e.g. who is speaking), mental state, etc. A C_d of significance to a particular dialogue participant, P_d, is indexed in this participant with an Action Trigger (A_t). A response A to a condition C_d gets initiated upon the A_t event, if the condition is detected.

$$\textbf{Detect}\,(C_d) \Leftrightarrow A_t \tag{7.1}$$

The action's execution time (A_{et}) determines how long after the A_t the action starts to take effect. The expected lifetime of an action (A_{el}) is used to determine the likelihood that an action is outdated when it comes time to execute it (move the "muscles").

$$\textbf{Execute}\,(A) \text{ IF } A_{et} < A_{el} \tag{7.2}$$

For example, a ball comes flying in your direction ($EVENT_1$, or E_1). You decide to [catch the ball]-(A'), but at a critical moment you [see that you won't be able to]-(E_2), and you decide to [give up]- (E_3) on the action, before it's been fully executed. In this example, $A_t = E_1$. Action A' gets initiated at **time**[E_1]. The A_{et} of A was too long, and the action's A_{el} allowed you to cancel the action before it was over. In this example, A_{el} is computed *during* execution of the action: you compare the progress of the action (distance between hand and ball) to its goal (ball in hand); before the goal is reached you decide, based on your prediction, to cancel it. For very fast, reactive responses, A_{el} has to be precomputed, because the perception necessary to assess the progress of such short actions would take too long. For an intention spanning a long period, A_{el} can be computed during execution. We will see an imple-

mentation of these mechanisms for reactive behaviors in the next chapter.

7.5 Summary

We presented Ymir—a generative model of psychosocial dialogue skill. The model supports the necessary framework for the creation of a computer controlled character capable of real-time orchestration of seamless, multimodal input and output. The model proposes the distinction between dialogue management and topic expertise; emphasis is on the former rather than the latter. As a result, the model underspecifies characteristics of topic knowledge but makes instead some specifications for the interaction protocol between the administrative tasks of multimodal dialogue on the one hand and topic knowledge on the other.

The model is multi-layered, with each layer providing a specific set of processes. These processes provide certain computational services, with the ability to communicate their results to other modules via blackboards. Actions in the system offer various degrees of "reactiveness", from very reflex-like to highly "intelligent". The morphology of actions is not fixed for any but the lowest level actions: the form of behaviors can be modelled at various levels of detail, with various combinatorial options. Execution of actions is prioritized according to where the "intention" to perform them originated, in the Reactive Layer, the Process Control Layer, or the Reflective Layer. The model as described here is not fixed; it is presented with enough flexibility as to be able to meet various demands on the implementation of its parts. Hopefully, this will allow it to become a guide to various approaches within the topic of face-to-face dialogue and real-time interaction. In the next chapter we will se one example of implementation of Ymir.