This number is still far more than I actually implemented. I reduced the number of transitions needed using several techniques. Interrupt transitions do 'double duty' by taking care of the transitions both into and out of a behavior. I cut out many transitions by writing several generic transitions, that could go from any behavior to a particular behavior.

Most importantly, I found in practice that many of the possible transitions did not make sense because of the semantics of the behaviors involved. For the Patient's 8 high-level signifiers there were only 15 maxi-transitions, and for the Patient's 16 low-level signifiers, there were only 25 mini-transitions (this number could have been cut down even more if I had shared mini-transitions between the same low-level signifiers when used in different high-level signifiers). Granted, the Patient is not as complex as it could be; but even in the fully complex unimplemented design of the Patient (shown in Figure C.20), there were 27 maxi-transitions, meaning under half of the possible maxi-transitions actually made sense.

One way to address this problem even further is to use generic transitions for most cases, and specializing them when the generic version is inadequate. For example, the transition out of sigh is always the same, unless sigh is returning to looking around. In this case, going directly from sighing to looking around the world looked odd, since the sigh was very slow and looking tends to consist of quick glances. Therefore, I made a generic sigh transition, then specialized it when going to looking around by adding a slow look. This slow look mitigated between the slowness of sighing and the speed of looking. This is one way to cut down on the complexity of number of transitions; make general ones for everyday use and add small touches for specific cases.

Finally, the separation between the motor system and the action architecture which causes such problems with animation also undermined the agent's ability to physically connect behaviors. When moving from one behavior to another, the agent needs to be able to sense accurately where the body is in order to be able to engage in a proper action sequence leading to the next behavior. The difficulty with the motor system / action architecture split is that you can sense where your body is, but you don't know where it will be when whatever acts that are currently being executed by the motor system are finished. This problem would probably need to be addressed by being able to get more information from the motor system about the position in which the agent can expect the body to be before whatever actions it is currently taking will be scheduled.

Evaluation for Users

Ease of use for the designer only answers some of the questions raised by this thesis. Given that the designer is satisfied with the created agent, that does not yet mean that users will interpret agents in the way the designer intended. Several possible questions still arise:

1. Do users recognize the behaviors the designer is trying to communicate?

2. Do users understand the connections between behaviors that the designer has in mind?

3. Does the addition of narrative sequencing really make the agent seem more intentional?

The detailed analysis of two transitions earlier in this chapter certainly suggests that, with the Expressivator, the user is given more information on which to judge both the agent's behavior and the reasons for the agent's behavioral changes. This is certainly a basis for improved user understanding, but does not necessarily imply actual improvement. In particular, the quality of the animated behavior is not up to snuff, which means users sometimes have trouble interpreting the simple movements of the agent; the animated presentation of the Patient would have to be fine-tuned in order to make the differences in comprehensibility truly striking. Anecdotal evidence from demonstrating the system suggests that the agent appears more intentional or 'alive' with transitions, but the system has hardly been tested under rigorous enough conditions in order to definitively answer these questions.

One reason this testing has not yet been done is because the goal of agent as communication (rather than as a functional tool) problematizes the question of evaluation. A respected technique for testing systems' desired effects on users is to do statistical studies of the impact of the system on various users. One can then conclude that the system is effective if there is a statistically meaningful effect across the pool of users.

But this adequacy across users is not necessarily the best technique to use when the goal is communication. For example, suppose that the agent is in practice incomprehensible to many users. But for a small subgroup of the target population, the agent is not only comprehensible, but makes an enormous and lasting impact on the way in which the users think and lead their lives. For some agent designers, this result may be much more satisfying than an agent which has a marginal impact on many users. Basically, statistical tests may be inadequate for such designers to evaluate the quality of their agents for the same reason that best-seller lists are not necessarily the best technique to judge the quality of a novel: a deep impact on a few people may be much more valuable than a shallow impact on many people. Issues such as this one will have to be explored by researchers delving into this area before we can be confident that the tests we are using are truly meaningful.

But as a first pass, I propose the following technique for evaluating a system like the Expressivator rigorously. Users interact with one of two versions of the system: one with behavior transitions and one without. Users are videotaped using the system, while talking aloud about (1) what they think the agent is doing and (2) why they think the agent is doing it. These protocols can be compared with the designer's intended behavioral communication at each step. Analysis of these videotapes is necessarily subjective (though not arbitrary), since there is no way to determine a meaningful 'quantitative distance' of the user's verbal interpretations from the designer's perhaps not entirely articulated intentions for the system.

If there is a need to get more quantifiable results, users could be surveyed after the video session using statistical techniques similar to those of Scott Neal Reilly [Neal Reilly, 1996] or James Lester [Lester *et al.*, 1997]. They could be asked, for instance, about their perceptions of the agent's personality; presumably, if they understand the agent's behavior and motivations, they will end up with a better understanding of the agent's personality over all. Some open-ended questions on the questionnaire, modeled on Lester's "who does Herman the Bug remind you of?" could round out study of the user's understanding of the agent.

The Expressivator as Narrative Intelligence

The Expressivator is intended as one example of what Narrative Intelligence might look like. The most obvious instantiation of narrative principles in the Expressivator is the use of transitions to form narrative sequences from atomic behaviors. But the narrativity of the Expressivator is more complex, involving not only the technology of the system — signs, signifiers, and transitions — but also such aspects as the philosophy and context of the Expressivator's use that normally do not count as part of the system, technically conceived. This makes sense, since narrative is, in the words of Katherine Hayles, *emergent*: it is a property not of artifacts conceived in isolation, but of those artifacts in the contexts in which they are used and interpreted [Hayles, 1997]. Here, I will review each of the properties of narrative and explain how it is embodied in the use of the Expressivator:

- *Narrative Diachronicity*: Narratives focus on events as they occur over time; similarly, the Expressivator's transitions relate the agent's activities to one another.

- *Particularity*: Narratives are particular; they are not just about abstract concepts, but about particular details. In using the Expressivator, the actual details of animation by which the agent's behaviors are communicated to the audience are similarly essential. Many behavior-based systems leave out this articulation of behavior into its physical presentation, but when a graphical system is intended to communicate, those behaviors must be specified down into the details of movement with a particular body.

- *Intentional State Entailment*: When interpreting a narrative, people focus not so much on what the agent is doing, but on how it feels about what it is doing. Transitions function here to regularly communicate what the agent is thinking about its actions: not just what it does, but *why* it does it.

- *Hermeneutic Composability*: Hermeneutic composability refers to the complicated interrelationships between the interpretation of various pieces of the system. One cannot focus simply on each particular component in isolation, but must look at how they interrelate.

  In the Expressivator, the agent's behaviors do not exist in a vacuum; using transitions, they are brought in relation to one another, both at the level of presentation and at the level of design. The agent's actions become signs and signifiers in ways that depend on the context of interpretation; moving the agent's head could result in a 'nodding' sign in one place, and a 'shaking to get light on' sign in another. These different signs cascade into different signifiers, meaning the agent's understanding of its actions is context-sensitive in ways that approximate those of the hermeneutically reading user.

- *Canonicity and Breach*: Someone's behavior will appear narratively comprehensible when it involves a set of expectations which are set up and then violated; the person must not be entirely predictable. Similarly, the Patient is set up so that there is much variation in behavior; every time the Patient hits its head, for example, it chooses slightly different angles and speeds of attack. The plot itself is an excellent example of story based on canonicity and breach: the Patient is turned off, turned off again, turned off again, until the last time, when instead of the expected turning off, it is killed.

- *Genericness*: The genre expectations of the user, which form the basis for understanding what the system is about, are set up in the context of the system. This means that the proper use of the Expressivator does not limit itself to the construction of behaviors and transitions within the agent. Rather, the Expressivator focuses on the likely user interpretation of the agent, which itself may be influenced by a host of contextual factors. In the Industrial Graveyard, correct interpretation by the user is set up, not only through the Patient's behaviors, but through the design of the user interface (e.g. the graph showing how good or bad the Patient is being), through the informational brochure which users read before they begin to interact with the system, and through the decoration and lighting of the virtual environment. These factors are not external to the system, though they are external to the technology of the Expressivator; they set up the context within which the Patient's behavior will be interpreted.

- *Referentiality*: In a story, the 'facts' are not paramount. Similarly, in the Expressivator, the agent's behaviors are not an absolute, which is then to be communicated as an afterthought to the user. Rather, the agent's behaviors are oriented to and dependent on the interpretation of the user. In this sense, the agent is a narrative, rather than a pre-existing problem-solver.

- *Normativeness*: Narratives depend on the audience's conventional expectation about how people will act. These expectations are here used as a basis for behavioral design. I designed the agent's behavioral sequences on the basis of background knowledge of how the audience would likely interpret the agent's behavior. Nevertheless, the overall experience could have been enhanced by more carefully thinking out the nature of the target audience. My general assumption was that the piece is oriented towards people who think like me.[5] Exploration of ways to explicitly tailor agent presentation towards particular audiences is an essential component of future work.

[5] In that respect, I am no different from many other AI researchers — the only difference is that I explicitly recognize that I am making an inaccurate assumption!

- *Context sensitivity and negotiability*: In Chapter 6, I say that serving up prepackaged narrative without leeway for audience interpretation is throwing away the best properties of narrative. Nevertheless, that is exactly what I do here. I decide on all the behaviors and transitions ahead of time, and then the goal is simply to make sure that those decisions make it across the yawning divide to the user intact.

  In this respect, signs can be taken too literally. If signs are thought of as absolutely everything that must be communicated, one by one, to the user, we end up merely replacing behavioral atomization with signifying atomization. An agent that is so simply and straightforwardly understood is too easy.

  A very different approach that is much more friendly to the value of negotiability is that taken by Simon Penny in his robot, Petit Mal [Penny, 1997a]. The design of Petit Mal explores the extent to which people can attribute meaningful behavior to autonomous robots. Petit Mal is set up, not to elicit any particular behavioral interpretation, but to allow for many possible behavioral interpretations. Far from trying to impose particular interpretations on the user, Penny uses Petit Mal as a blank screen onto which many possible interpretations can be projected. Petit Mal is interpretationally plastic, and never exhausted by the onlooker's musings; this gives its dynamics a degree of liveliness which the Patient lacks.

  The difficulty with this plasticity is that it is relatively low-level. At the internal level, Petit Mal does some simple navigation and obstacle avoidance (which is, of course, regularly interpreted as much more complex behavior). It is not clear how much more complex behaviors can be constructed for Petit Mal without simultaneously greatly constraining the interpretational space. In this sense, Petit Mal and the Patient occupy more or less opposite ends of the spectrum of interpretational negotiability on one end and understandable complexity on the other. If this is so, it might be interesting to now try working for something in the middle.

- *Narrative accrual*: It is not clear how narrative accrual would apply to the work I have done here.

Fundamentally, narrative is more about the *quality* of behavior, rather than its *correctness*. Because this attitude differs from that of the action-selection approach at the heart of behavior-based architectures, a number of changes to the behavior-based framework are necessary. Fundamentally, behaviors should be simple and expressive; intentionality is communicated to the user by clearly displaying the relationships between behaviors. The detailed technical changes the Expressivator makes to Hap are summarized in section A.4 of the Appendix. In general, the changes the Expressivator makes can be summarized as follows:

- Instead of breaking behaviors into physical actions and behaviors, the Expressivator breaks behaviors into signs and signifiers that are communicated to the user. The agent keeps track of the user's likely current interpretation through the sign-management system, which posts signs and signifiers once they have been expressed, allowing the user's likely interpretation of agent activity to influence the agent's behavioral decisions.

- Instead of simply atomizing the agent's activity, the Expressivator includes transitions that express to the user the agent's reasons for changing from one behavior to another, simplifying the user's comprehension of the agent as narrative.

- Instead of having behaviors being basically independent, the Expressivator gives them meta-level controls by which they can coordinate with one another to give the user a coherent picture of the agent's personality and intentions.

The Expressivator combines these systems to try to allow designers to build agents that express their activities and thinking to the user, without giving up many of the advantages that behavior-based architectures can provide. The as yet untested hope is that these agents will appear, not only more understandable, but also more visibly alive.

Chapter 8

Conclusion

In this thesis, I have taken you on a long and circuitous intellectual journey, and now, at the end, it is time to go back to the beginning and see how it all fits together. From a computer science perspective, we tackled problems in integration for behavior-based autonomous agents. We found some inherent limitations in the ability of standard AI methodology to ever fully integrate agents, but discovered ways to mitigate the effect of this underintegration by redefining agents as channels through which agent designers communicate to their audiences. This changes the focus in agent-building from primarily a design of the agent alone, with its communication as an afterthought, to including the agent's comprehensibility in the design and construction of agents from the start. This rethinking of the nature of agents led to the proposal that if agents are to be comprehensible as intentional beings, they should be structured to provide cues for narrative interpretation, the manner in which narrative psychologists have found people come to understand specifically intentional behavior. The Expressivator was developed as one architecture for this 'Narrative Intelligence.' It combines (1) redefinition of behaviors as signifiers and their reorganization in terms of audience interpretation, (2) the use of transitions to structure user-recognized behaviors into narrative sequences, and (3) the use of meta-level controls to strategically undermine over-atomization of the agent's behaviors. Preliminary results are encouraging, but further work, preferably involving the development of support for graphical presentation, will be necessary in order to fully evaluate the implications of and possibilities for the architecture.

From a cultural theory point of view, we started with the identification of a technical problem in computer science with remarkable similarities to some notions of schizophrenia in cultural theory. These similarities are not a coincidence; rather, they can be traced to atomizing methodologies AI inherits from its roots in industrial culture. The disintegration AI researchers can recognize in their agents, like that felt by the assembly line worker and institutionalized mental patient, is at least in part a result of reducing subjective experience to objective atoms, each taken out of context and therefore out of relationship to one another and to the context of research itself. This suggests that the problems of schizophrenia can be mitigated by putting the agent back into its sociocultural context, understanding its behavior as implicated in a cycle of human interpretation, on the part of both its builder and those who interact with and judge it.

Because this is a change in the metaphor at the heart of current systems, the embodiment of this changed perspective in technology has implications beyond some technical tweaks on a pre-existing and unchanging technical base. Instead, it changes at a fundamental level the meaning and usage of many parts of the system, even those that were not intended to be affected, and suggests even in its presentational failures the communicational limitations of a system which sees the essence of an agent purely in terms of the abstractions of its internal code.

This sums up the computer science perspective on the one hand and the cultural theory perspective on the other. But in the introduction, I told you this thesis is not half computer science and half cultural theory; rather, it is a single body of work which can be seen in various ways from either perspective. Now that you have had a chance to see both sides of the coin, I will take some time to step back from the details and discuss the implications of this work from a combined perspective, the one I developed, at times against my will, during the work this thesis represents. First, I will return to the notion of narrative and summarize its relations to schizophrenia and atomization. Then, I will step back to the meta-level to review the role of this thesis as a subjective technology, as one way to synthesize humanistic and engineering perspectives into a knowledge tradition that bridges the enormous chasm splitting contemporary Western intellectual life.

Narrative and Schizophrenia

This section is speculative; it is not intended to represent any kind of final truth. Instead, I will connect up the various strands of the thesis, to form a picture of where they seem to lead as a whole.

In this thesis, we started with schizophrenic agents, and ended up with Narrative Intelligence. Narrative became important for agents when it became clear that default technical approaches to hiding atomization from the user were not helpful in making agents seem intentional. In order to understand intentional behavior, users attempt to construct narrative explanations of what the presumed intentional being is doing; but this approach conflicts with the mechanistic explanations designers themselves need to use in order to identify, structure, and replicate behavior.

This contrast between narrative explanations that explore the meaning of living activity and atomistic explanations that allow for the understanding and construction of mechanical artifacts repeats the criticisms of anti-psychiatry. R.D. Laing and other anti-psychiatrists, after all, complain that the difficulty with institutional psychiatry is that it reduces the patient to a pile of data, thereby making a machine of a living person. Their solution — contextualization — seems at first blush to be a different response than the focus on narrative here. But just as we have seen that science is generally atomizing, we now can see that the methodology of contextualization contrasts with this atomization by being itself, too, a form of narrative. Anti-psychiatry follows the narrative tradition in the following ways: by structuring and relating the ‘data’ of a patient’s life into the semi-coherent story of a meaningful, though painful, existence; by focusing on the patient not as an instance of a disease but as a particular individual and how that person feels about his or her life experience; and by relating the doctor’s narrative to its background conditions and the life context in which it is created and understood. It is only through this process of narrative interpretation that anti-psychiatry feels the psychiatrist can fully respect and understand the patient's subjective experience as a human being.

If atomization involves thinking of human life mechanically, reducing it to a matter of cause-effect, while narrative allows for the full elucidation of meaningful intentional existence, then it seems likely that narrative — and by extension the humanities, for whom narrative is a modus operandi — can address meaningful human life in a way that an atomizing science simply cannot. If humans comprehend intentional behavior by structuring it into narrative, then AI must respect and address that way of knowing in order to create artifacts that stimulate interpretation as meaningful, living beings. This suggests that the schizophrenia we see in autonomous agents is the symptomatology of an overzealous commitment to atomistic science in AI, a commitment which is not necessarily unhelpful (since it forms the foundation for building mechanical artifacts), but needs to be balanced by an equal commitment to narrative as the wellspring of intentionality.

Humanists may recognize the arguments of Gadamer [Gadamer, 1986].

Schizophrenia in Postmodern Culture

But schizophrenia is not simply a difficulty of a contemporary agent-building method; schizophrenic subjectivity is also an important component of contemporary cultural theory. As discussed in the introduction, many cultural theorists identify schizophrenia as a way of thinking about contemporary human experience. This schizophrenia can be understood in a multitude of ways, but one way of understanding it is as a rejection of the idea that people are essentially unified, rational beings, with the suggestion instead that human consciousness is an emergent and somewhat illusionary phenomenon overlaying an actually fractured and distributed existence. While I am far from suggesting that we should go back to the idea that humans should be fundamentally rational, with emotion and meaning being mere distractions from the actual, logical, unified substrate of true humanity, my experience with schizophrenic subjectivity as it manifests itself in AI has led me to the conclusion that there are deep problems with the way schizophrenia is used in cultural theory, as well.

Note to technical readers: the rest of this section may be difficult for you to follow, as it uses the results from analyzing schizophrenia in agents to address a technical debate within cultural theory. I think you may find this different perspective interesting; but if you are feeling unhappy, please feel free to go directly on to the next section.

Specifically, schizophrenia comes about in AI when a living being's activity is reduced to simple atoms with limited interaction. Schizophrenia is in this sense the limit-point of formalization, the point at which important aspects of flowing existence are simply left out of the picture and therefore only appear as gaps or fissures between simply defined atoms. But this suggests that, in some sense, the postmodern (a.k.a. schizophrenic) subject, too, may be simply internalizing and celebrating the atomized view of itself that bureaucracy, industrialization, and modern science and technology have developed.

"The play of significations, their proliferation, their being out of gear with representations, because of the autonomy and arbitrariness of the way the stock of signifiers operates, has contradictory consequences: it opens possibilities for creativity, but it also produces a subject cut off from all direct access to reality, a subject imprisoned in a signifying ghetto." ([Guattari, 1984], 92)

The notion that this postmodern subjectivity is in some sense inherited from the technology we use is gradually becoming commonplace.[1] The idea here is simply an extension to this: that through the structure of the technology which is deeply interfaced with our daily lives, we imbibe the atomistic, objectivizing view of both ourselves and the interactive moment that that technology presupposes. The hyperbole

[1] For elegant descriptions of how this works in practice, see e.g. [Hayles, 1993] [de Mul, 1997].

surrounding hypertext is a case in point; its inheritances from scientistic self-understanding can be seen in its uncritical enthusiasm for technical development and frequent dismissal of criticism of that enthusiasm as neo-Luddism; in the notion of the 'postmodern narrative' as chunks of data with no overarching meaning, and only local structure; in the rhetoric of authorlessness, as though the text sprang from no context and was entirely ahuman; and in the movement of the responsibility for generating narrative understanding squarely onto the shoulders of hapless readers, who are left desperately trying to fabricate a narrative from randomly strewn atoms simply because they are good at it and hypertext technology is not.

If schizophrenia is something we are catching from our technology, then we must simultaneously ask ourselves if that is something we would like to catch. Though schizophrenia has multiple uses and I by no means intend to criticize all of them, I still have deep fears about the sometimes uncritical and whole-hearted postmodern importation of schizophrenia from modern technology as a new — and by extension positive — way of being. This is because the postmodern worldview is dangerously close to making the assumption that the ideas we import from technology come from some shining stratosphere of newness; rather than, as analysis of scientific work frequently makes clear, from a continuous cycling and recycling of metaphors and concepts from broader culture to scientific culture and back again. In the case of schizophrenia, these concepts are recycled from an industrial and institutional culture that most postmoderns would not knowingly choose to embrace, and that in fact only get their alluring aura by coming attached to our new high-tech toys. As Bruno Latour says,

> It is strange to say, but I think much of postmodernism is scientistic. Of course they no longer believe in the promises of science — they leave that to the moderns — but they do something worse: they believe in the ahuman character of science, and still more of technology. For them, technology is completely out of the old humanity; and as for science, it is almost extraterrestrial. Of course, they do not see that state of affairs as bad. They are not indignant at the ahuman dimension of technology — again they leave indignation to the moderns — no, they like it. They relish its completely naked, sleek, ahuman aspect.... I think that this is deeply reactionary because in the end, you push forward the idea that science and technology are something extraordinary, completely foreign to human history and to anthropology. ([Crawford, 1993], 254)

The antidote to this is, again, narrative: putting into context, creating origin stories about, attributing authorship, constructing meaning. This means in particular narrative to connect science and technology to the rest of our cultural life, reminding ourselves once again that science is done by people, that the views, strategies, and goals of those people is shaped, in part, by the culture in which they live. There is no law that says science must be atomizing, and that, by extension, technology must be schizophrenizing. Instead, we can return to the notion of subjective technologies, finding a middle ground between narrative and atomization.

Subjective Technologies

When we began, I set the goal of developing a kind of technology that respects and addresses the complexities of subjective human experience. This is in contrast to much current AI practice — many practitioners of AI, particularly those of the alternativist persuasion, are reluctant to engage in questions of what it *feels* like to be alive in the world. Subjective experience is often felt to be fundamentally illusory and unreliable, something to be replaced at the earliest possible moment with a more objective and testable form of knowledge. Building technology, in this way of thinking, may require a commitment to objectivity, since fuzzy mentalist concepts simply cannot be directly implemented.

In the work presented here, by contrast, subjective experience is essential — that is, the subjective experience of those who build and who come to interact with the agent. The mechanicity of current agents is a subjective experience, which can be fixed not by trying to find ways to make the agent objectively intentional (perhaps a contradiction in terms), but by respecting the subjectivity of that experience in order to enable it to be the best experience possible. The goals the designer has for the agent, independently of its actual effect, are, as well, a subjective factor — probably not completely definable, but nevertheless hopefully achievable through particular design strategies. In this sense, subjective experience and technology are by no means incompatible.

The work I have done here combines technology and subjectivity by seeing an agent as a form of communication, in terms of the intentions of its designer and how it is experienced by the audience. In this light, the major question to be answered is not "how can we objectively and testably reproduce experience?" but "what are the *goals* of the agent-builder in terms of how his or her agent design should be understood, and how can they best be fulfilled?"

The major change this philosophical distinction makes at the technical level is that *comprehensibility is seen as an essential requirement to be engineered in from the start*. Certainly, other AI researchers have been interested in making comprehensibility a goal. But, generally speaking, these attempts have come as an afterthought, at the point where the target user population expresses reluctance to interact with or trust intelligent systems whose behavior they do not understand. The field of expert systems, for example, has had a rash of mostly unsuccessful attempts to modify systems that make correct but obscure conclusions in order to make clear to human users how they came to them. My experience with the Expressivator suggests that it is so difficult to make already-designed systems comprehensible after the fact simply because comprehensibility cannot be adequately addressed through a set of tweaks added at the end. Rather, it requires changes in the way we structure and design agents from the beginning.

Anti-Boxology Re-Visited

This simple fact — that systems designed separately from the context of their human use may not function as well as ones that keep that context in mind from the beginning — brings us back to the postulates of anti-boxology I set forth in the introduction. The anti-boxological perspective sees life as inadequately understood when carved into separate categories; instead, it seeks to relate those categories to each other. When I introduced this concept, I stated rather mysteriously that this thesis would be anti-boxological on several levels: disciplinary, methodological, and technical. We are now ready to go back and look at the thesis as a whole as an instance of anti-boxological thinking.

At the disciplinary level, the engineering approach used here stems from and is continuously informed by a humanistic perspective on agent-building. Engineering and the humanities are not seen as two separate activities with little to say to each other. Instead, they are thought of as two (sometimes vastly) distinct perspectives, which can be profitably put in relationship with one another.

At the methodological level, the development of socially situated AI puts the agent into a sociocultural context that includes the people who build it and the people who observe it. This is reminiscent of the viewpoint of Terry Winograd and Fernando Flores, who argue that rather than thinking about how humans can communicate with computers, we ought to be thinking about how computers can enable better communication between people [Winograd and Flores, 1986]. Here, though, this does not involve the whole-sale rejection of AI, but a change in one of its fundamental metaphors. Instead of seeing agents as autonomous, socially situated AI argues that the agent should often be thought of as a kind of communication. In this agent-as-communication metaphor, the social environment of the agent is, not some unfortunate baggage to be discarded or ignored, but essential to and constitutive of the design of the agent. This change in methodology is directly represented in the technology through the shift in structuring agents from internally-defined behaviors to externally-observable and communicated signifiers.

At the technical level, the parts of the agent are explicitly put in the context of each other and of the agent's overall personality through the use of transitions. Transitions represent for the designer, and express to the user, the relationship between the different pieces of the agent. Meta-level controls provide the technical basis for interrelating behaviors in this way by allowing behaviors to coordinate to present a coherent picture of the agent's overall activity to the user. The details of the agent architecture therefore repeat the themes of the highest level of motivation: we have anti-boxology all the way through.

Lingering Questions

So far, I have discussed the way the thesis works at a high level and in terms of the themes I developed in the introduction. At this point the reader may have followed the argument, understood where we went and how we got there, but still have lingering high-level questions about the thesis. Here, I will try to answer some of the major questions that this work frequently brings up.

Questions from a Technical Perspective

> *As far as I can tell, the Expressivator adds some tweaks to an already-existing architecture in order to let the designer*

> *manipulate the audience's perception of the agent. Your agent doesn't actually become any smarter; the transitions all have to be written by hand. In what sense is this an AI contribution?*

It is true that this thesis follows in the tradition of much of behavior-based AI by designing behaviors — including transition behaviors — by hand. The agent's reasoning is minimal, compared to what some classical AI programs do. Like many other behavior-based systems, the agent makes behavioral decisions based on perception of the environment and memory of its own activities — although unlike these systems, it can also make decisions based on the likely user perception of its activities and based on tokens which represent the reasons for its behavioral changes.

The status of this design-oriented, direct programming approach to agents as a legitimate form of AI has been extensively defended by others (see e.g. [Agre, 1997]), and I will not repeat those arguments here. In addition to these general claims, the Expressivator has its own unique claim to being an AI contribution through its exploration of the changes that must come about in agent structure and design in order to allow agents to be comprehensible. Similar explorations have already occurred, most notably (but not exclusively) by Believable Agents researchers; this thesis adds to them by underlining the importance of *narrative* for human comprehension, and by outlining how this narrativity can be incorporated in AI, both in general in Chapter 6, and as specific technical mechanisms in Chapter 7.

This thesis does not simply provide some randomly-chosen technical hooks for user manipulation, but addresses the question, "what exactly is needed in order to make agents intentionally comprehensible?" It finds the answer in narrative: in order to be intentionally comprehensible, an agent must express not only what it does but also why it does it. The Expressivator then attempts to provide support for precisely this expression, by supporting the design and use of behaviors as communicated signifiers and by expressing the reasons for behavioral change through the use of transitions.

> *The Industrial Graveyard seems effective, but its effectiveness as communication are based on the use of conventions from animation, such as the exaggerated shock reaction to external events. How well do you expect this to map to other domains?*

The same conventions clearly will not work in radically different domains, such as photorealistic rendering. But clear communication is not simply a property of animation; it is also the goal of live-action film, novels, theater, and so forth. At its most fundamental, whatever the domain, the principles of narrativity still hold: the user still needs to be aware of what the agent is doing and why the agent is doing it. The difference between these domains is that *expression* of those activities and the reasons for them will need to be adapted to whatever domain the agent is built for, and however that agent is represented to the user. It seems likely that various kinds of autonomous agents will, over time, develop their own conventions of expressiveness, so that they will not need to be parasitic on more established genres.

> *I could barely wade my way through Chapter 3, but I still understand how the technology works. Couldn't you have built the technology without cultural studies, for example by simply importing suggestions from art and animation as you do in Intermezzo II?*

The short answer is yes, I probably could have — *but I most likely wouldn't have*. Once schizophrenia is identified as a problem, and once it is reframed in terms of agent communication, most of the technical answers I come to are straightforward. *The difficulty is in realizing that the problem needs to be reframed in the first place*.

The most important contribution cultural studies brought to the technical work, independently of any insights that I might have been able to glean purely from art, animation, and psychology, is the level of self-reflexivity that let me step back and realize that I was caught in a double bind: that atomization was both essential to code and the root cause of schizophrenia. Before I had this understanding I had already been trying to tackle the problem of schizophrenia for a number of years. Schizophrenia was at that time for me a gut feeling, not a well-defined concept, a feeling that there was something fundamentally wrong with the way agents were constructed, something that was inhibiting their intentionality. I came up with numerous technical proposals, many half-baked and some more complete, for addressing schizophrenia, each of which seemed upon reflection to repeat the very failures I was trying to address.[2] It wasn't until I realized, by comparing AI methodologies with the practices of assembly line construction and Taylorism, that what I was trying to do was simply and for good reasons not possible, that I realized I needed to rethink what I was trying to do in a deep way.[3]

The second most important contribution from cultural studies for the technical work came then, as I searched for a different way to think about agents that did not involve the same Catch-22: the suggestion on the basis of culturalist perspectives that the difficulty was that the agent is being taken out of context. Once I had the idea that the agent needs to be clearly communicated, much of the rest of the work could follow in a relatively normal technical way, using insights from various fields as they seemed appropriate (and in the manner to which AI researchers are accustomed). Nevertheless, for me the technical work is continuously informed, though perhaps in a less spectacular way, by my cultural studies perspective: from the understanding that interpretation is a complex process quite unlike simple perception to the ferreting out of the implications of changing the metaphors underlying agent architectures, this work is really cultural studies almost all the way through.

Questions from a Critical Perspective

> *Frankly, I find this 'AI Dream' of creatures that are truly alive to be ludicrous, if not downright Frankensteinian. In a world full of social problems, why should this goal matter to a cultural theorist?*

[2]This was a very trying time for my advisor.
[3]This was another trying time for my advisor.

The AI dream of mechanical creatures that are, in some sense, alive, can seem bizarre to those who are new to the idea. It is therefore important to note that this is not an idea that is new in AI, but, as Simon Penny notes, the continuation of a tradition of anthropomorphization that extends back thousands of years [Penny, 1995]. In this sense, the AI dream is similar to the 'writing dream' of characters that ring true, to the 'painting dream' of images that seem to step out of the canvas, to the fantasies of children that their teddy bears are alive, and to many other Pygmalionesque dreams of human creations that begin to lead their own lives.

But there is certainly a sense in which AI brings a new twist to these old traditions. AI as a cultural drive needs to be seen in the context of post-industrial life, in which we are, as described in Chapter 3, constantly surrounded by, interfaced with, and defined through machines. At its worst, AI adds a layer of seductive familiarity to that machinery, sucking us into a mythology of user-friendliness and humanity while the same drives of efficiency, predictability, quantifiability, and control lurk just beneath our perception.

But at its best, AI invokes a hope that is recognizable to humanists — that is invoked, in fact, by Donna Haraway in her "Cyborg Manifesto" [Haraway, 1990a]. This is the hope that, now that we are seemingly inescapably surrounded by technology, this technology can itself become hybridized and develop a human face.[4] This version of the AI dream is not about the mechanistic and optimized reproduction of living creatures, but about the becoming-living of machines. The hope is that rather than forcing humans to interface with machines, those machines may learn to interface with us, to present themselves in such a way that they do not drain us of our humanity, but instead themselves become humanized.

> *AI has a documented history of building military technology and mechanical replacements for human workers. Neither of these goals are ones that many cultural theorists would feel comfortable with. How does your project situate itself within this history?*

It is true that AI has a long and rich history of being used in ways with which cultural theorists generally might not agree. But, like many cultural practices, it cannot be summed up by its dominant uses; AI includes a heterogeneity of viewpoints and purposes. The technical application I work on here is in the subfield termed 'AI, art, and entertainment.' Application domains in this area run the gamut from automated sales representatives to interactive virtual pets to serious attempts at art; compared to the generation of robotic helicopters for the Department of Defense, these applications have, at least until now, been relatively innocuous.

I do believe, however, that AI research cannot proceed without awareness of how the techniques it develops are used in practice, whether or not one personally works on those applications that may be disagreeable. I also believe that this awareness is not particularly well-developed in my work, in any sense other than the relatively common AI strategy: I

[4] This is not to deny that one might want to resist mechanization — it simply bows to the reality that it will probably be a long time before such resistance could bear substantial fruit.

did my best to make my application be one I was willing to stand behind without qualms, and I tried (I think successfully) not to allow my own Department of Defense funding to alter the way in which I did and presented my work.

My own goal with respect to these practices was not to enable or disenable any particular application domain, but to try to develop a strategy for AI research where the application and funding of the research itself can be brought onto the table. Because agents are often seen as existing in a sociocultural vacuum, questions about funding and application are currently seen as ethical questions, to be sure, but ones that come after the fact and do not have a real implication for how research is conducted in the first place. I have tried to replace this model of research with one where the implications of the sociocultural context are made clear as part of the agent design, so that these 'external' questions can be seen for what they really are: at least partially constitutive of the way in which research can be done at all. This is admittedly a first step, but not, I think, a trivial one.

More broadly, I follow Jaron Lanier and J. MacGregor Wise in believing that one of the major dangers inherent in the way we build agents (and indeed, many technical artifacts) today is in the myth of authorlessness that surrounds their construction [Lanier, 1996] [Wise, 1996]. Agents are the creations of human beings, and therefore will always have limitations, some of which can be clearly understood, and some of which are implicit in nontransparent ways in the details of the construction of the technology. The danger of presenting these artifacts as living, independent beings rather than as human products is that the decisions which its human designer made become invisible and therefore unquestionable. The notion of agent-as-autonomous in this sense unintentionally closes off the possibility of critique.

My conviction with respect to this problem is that the users of technology should not be given a technical artifact as a *fait accompli*, but should be able to have a level of critical engagement with the technology. This means the technology and its context should be constructed so that they allow the users to understand how they are being led to interact with the computer and each other in specific ways. This is in fact the rationale behind the user interface design of the Industrial Graveyard: the cartooniness emphasizes that the system was built by a human, and the lack of buttons the user can press reflects the constraints I explicitly put on the user in terms of their interactions. In general, users should be able to realize intuitively on the basis of the software design that any particular technology provides not only possibilities but also constraints, constraints which are often grounded in the culturally-based assumptions of the people constructing the technology. In short, users should be able to understand, too, that technology is not just a set of pre-given tools, but itself social, cultural, and changeable.

The Cultural Studies / AI Hybrid

Now we have come to the end. Before we part ways, I must cash in the promises I made in the introduction when I asked you to consider the most important purpose of this thesis, the synthesis of cultural studies with AI.

From an AI perspective, I said that the use of cultural studies within AI could lead to new and perhaps better technology. In this thesis, that technology is the Expressivator, an architecture for supporting the user in interpretation of agent behavior by providing narrative cues. This technology is different from current technology because it is based on a different conception of what agents fundamentally are, a conception that stems from cultural studies analyses. Cultural analysis brings in concepts that helped to make the Expressivator possible and that would have been difficult to develop from within the field of AI alone.

From a cultural studies perspective, I described two advantages of using cultural studies in a practice of AI. The first is that by actually practicing AI, the cultural critic has access to a kind of experiential knowledge of science that is difficult to get otherwise, and will deepen his or her theoretical analysis. This increased knowledge is expressed in two ways in this thesis: (1) the analysis of behavior-based AI as a manifestation of industrial culture in Chapter 3, and (2) the analysis of the metaphorical basis of behavior-based AI even into the details of the technology, which occurs throughout the thesis.

The second advantage is that working within AI allows cultural theorists to not only criticize its workings, but to actually see changes made in practice on the basis of those criticisms. The Expressivator reflects the cultural studies analysis in the fundamental changes it makes in how an agent is conceived and structured. This brings home at a technical level the idea that agents are not simply beings that exist independently, but have authors and audiences by which and for which they are constructed.

Finally, the common advantage I peddled for my approach is the potential alteration to the rhetoric of mutual assured destruction that currently seems to be prevalent in interdisciplinary exchanges between cultural studies and science. At the most direct level, the possibilities for communication are enhanced among readers who, whatever their background, now share a common set of concepts which include, on the one hand, AI terms such as behavior-based AI, autonomous agents, and action-selection, and, on the other, cultural theory concepts such as objective vs. subjective technology, schizophrenia, and atomization. But the most fundamental contribution this thesis tries to make toward a cease-fire in the Science Wars is in demonstrating that 'science criticism' is relevant to and can be embodied in the development of technology, so that there are grounds for the two sides to respect each other, as well as a reason for them to talk. My hope is that this thesis can join other similarly motivated work on whatever side of the interdisciplinary divide to replace the Science Wars with the Science Debates, a sometimes contentious and always invigorating medley of humanist, scientific, and hybrid voices.

Appendix A

Technical Details

This appendix gives further details about how the Expressivator is implemented. Section A.1 describes the implementation of signs, signifiers, and the sign-management system. Section A.2 describes the changes to the Hap language that are needed to invoke meta-level controls, and how each language change was implemented. Section A.3 gives the details on the implementation of transition triggers and transition demons. Finally, section A.4 summarizes the changes made to the Hap language in Chapters 5 and 7.

A.1 Details of Sign(ifier) Implementation

As explained in Chapter 5 (pp. 113 - 121), sign management is a technique for structuring the agent in terms of the agent's impression, rather than in terms of internalistic problem-solving. There are three layers to agent structure under sign management — *signs*, which are small sets of physical actions that are likely to be interpreted in a particular way by the user; *low-level signifiers*, which are units of signs, physical actions, and mental actions (arbitrary C code) which communicate particular immediate physical activities to the user; and *high-level signifiers*, which communicate the agent's high-level activities.

Because the interpretation of agent activity depends heavily upon context, signs and signifiers are identified by the designer not when their code is defined, but in the context in which they are used. The same set of physical actions may be a sign in one context, and no sign or a different sign in another context. Similarly, a behavior may be a low-level or high-level signifier in one context, and no signifier or a different signifier in a different context. Signs are identified when they are posted (see below). Signifiers are identified by special annotations in the behavior language when the behavior is invoked: low-level signifiers with `low_level_signifying`, high-level signifiers with `high_level_signifying`. The 'mope-by-fence' signifier, for example, is invoked as `(with low_level_signifying (subgoal mope_by_fence))`.

These annotations mark the given behavior as a low- or high-level signifier, enabling their proper manipulation by other special forms. For example, since the same behavior can be a low-level signifier in some uses, and a regular behavior in others, the form that posts the low-level signifier checks to make sure the behavior *is* a low-level signifier in the current usage before it posts it. Marking behaviors as signifiers also enables the designer to write code that tests whether behaviors are low- or high-level signifiers (see section A.1.2 below). This property will become crucial in later code examples.

A.1.1 Posting Signs and Signifiers

In addition to allowing the designer to structure the agent according to these units, the sign-management system supports structures so that the *agent* can keep track of the signs and signifiers it has communicated to the audience. Signs and signifiers are stored in special data structures, described below. The agent *posts* its signs and signifiers when it is confident they have been communicated. It does this through special `post_sign`, `post_low_level_signifier` and `post_high_level_signifier` forms, which modify the sign and signifier data structures.

Sign / Signifier Data Structures

At any point in time, the agent will have at most one high-level signifier posted. Which high-level signifier is currently posted (i.e., has been demonstrated to the user) is noted in global memory in the working memory element called `CurrentHighLevelSignifier`, which has two fields: `name` (the name of the signifier) and `time` (the time when the signifier was posted).

The agent will usually have only one high-level signifier running.[1] But since high-level signifiers can only be posted once they have been communicated to the user, the currently *running* high-level signifier is often not the same as the currently *posted* high-level signifier. A high-level signifier may be active for quite some time before it is posted as the CurrentHighLevelSignifier (or may, if interrupted, never be posted at all).

Since signifiers are behaviors, both low-level and high-level signifiers are stored just like any other Hap behavior, in special working memory elements called 'Goal' with pointers to their parents and children. The sign and signifier data structure is an addition to this already-existing structure, in order to allow related high-level signifiers, low-level signifiers, and signs ready access to one another.

Signs and signifiers are stored in memory as shown in Figure A.1. A high-level signifier stores the name of its currently-posted low-level signifier and a pointer to its currently-posted sign.[2] A high-level signifier may have more than one active low-level signifier as a child (for example, during transitions), but each currently active high-level signifier only stores the name of *one* of those low-level signifiers, i.e. the one that has been most recently posted.

Low-level signifiers, in turn, store pointers to the high-level signifier of which they are a part — whether or not either signifier has been posted. This makes it easier to implement the posting of low-level signifiers, since

[1] It sometimes has more than one, for example during transitions.

[2] Logically, it would have made more sense to store the sign on the low-level signifier. I did not do this because I implemented signs before I realized I needed low-level signifiers.

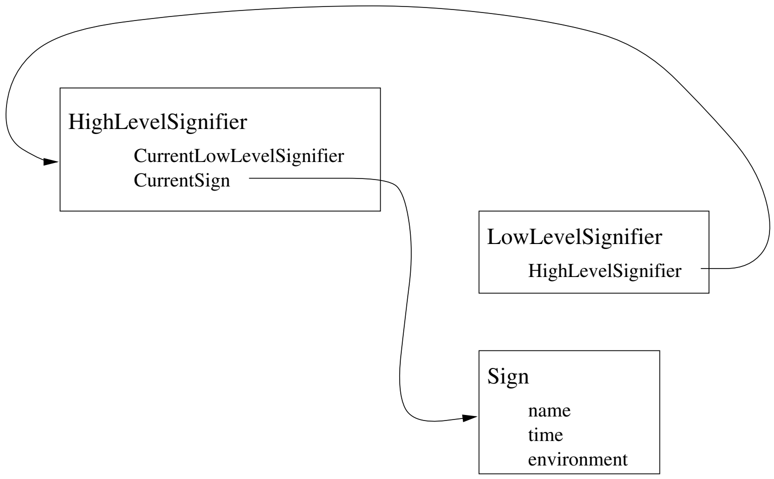

FIGURE A.1: Sign and Signifier Data Structures

```
(sequential_production read_lines (bottom current
                                   increment)
  (subgoal read_line $$current)
  (post_sign read_line ((line_read $$current))
  (subgoal continue_read_lines $$bottom
           "$$current + $$increment"
           $$increment))
```

FIGURE A.2: Example of `post_sign`.

they can easily find the high-level signifier to which they belong, even when (as is regularly the case) that high-level signifier is not posted to global memory yet. Signs simply store their own information: their name, the time they were posted, and a field, `environment`, that stores their arguments as a first-class environment.

Special Forms for Posting Signs and Signifiers

There are three forms for posting signs and signifiers: `post_sign`, `post_low_level_signifier`, and `post_high_level_signifier`. They are responsible for updating the data structures described in the previous section so that they remain a consistent picture of what the user has seen the agent do.

`post_sign`

The `post_sign` form takes as argument an arbitrary label and an optional first-class environment that contains the arguments of the sign. For example, Figure A.2 shows how the Patient reads the lines of the schedule. After each line is read, the Patient posts a sign that reminds itself which line the user has seen the Patient read.

When invoked, `post_sign` creates the 'Sign' data structure, a working memory element which includes the sign's name, its arguments, and

```
(parallel_production goto_spot (x y)
  (subgoal face_then_goto $$x $$y)
  (with persistent
    (demon
        ;;
        ;; check if 'goto_spot' is a low-level
        ;; signifier
        ;;
        (("G (Goal name == goto_spot;
                   low_level_signifying_p
                                    == true;);"
          ;;
          ;; check that 'goto_spot' is not posted
          ;; as a low-level signifier
          ;;
          "- (Goal CurrentLowLevelSignifier
                      == goto_spot;);"
          ;;
          ;; check that the 'walking_to' sign has
          ;; been posted
          ;;
          "CS (CurrentSign
                   name == walking_to;);"))
      (post_low_level_signifier goto_spot))))
```

FIGURE A.3: Example of `post_low_level_signifier`

a time stamp. It then notes the sign on the high-level signifier which invoked the `post_sign` form (this may be the CurrentHighLevelSignifier, but may also be a different, as-yet-unposted signifier). In this case, the 'read_lines' behavior is part of the 'read-schedule' high-level signifier, so it will make 'read_line' the current sign for 'read-schedule,' replacing whatever sign had previously been stored.

`post_low_level_signifier`

The `post_low_level_signifier` form works similarly, but it only takes the low-level signifier's name (no arguments). Its responsibility is to update the current high-level signifier's data structure so that its `CurrentLowLevelSignifier` field has the name of this low-level signifier. For example, the code fragment that implements going to a particular spot in Figure A.3 uses `post_low_level_signifier` to post the 'goto_spot' low-level signifying behavior on its parent high-level signifier (which happens to be 'explore_world') after the 'walking_to' sign has been posted.

An important note: it is not enough for a signifier to post itself once, when it is first demonstrated to the user. This is because behaviors can be interrupted; and a signifier that is interrupted may no longer be posted when control returns to it. It will need to post itself again after it is, once again, demonstrated to the user. Signifiers therefore continuously repost themselves whenever they see the appropriate signs or signifiers

```
(parallel_production mope_by_fence ()
  (with persistent (priority_modifier 100)
    (subgoal sad_looks_through_fence_to_sigh_demon
             $$this_plan))
  (with low_level_signifying
    (subgoal sad_looks_through_fence))
  (with persistent
     (demon
          ;;
          ;; check that sad_looks_through_fence is
          ;; the current low-level signifier
          ;;
        (("G (Goal CurrentLowLevelSignifier
                    == sad_looks_through_fence;);"
          ;;
          ;; check that mope_by_fence is not posted
          ;;
          "- (CurrentHighLevelSignifier
                   name == mope_by_fence;);"))
        (post_high_level_signifier mope_by_fence)))
  (wait))
```

FIGURE A.4: Example of `post_high_level_signifier`

and notice they are not currently posted.

`post_high_level_signifier`

`post_high_level_signifier` works in an way that is analogous to `post_low_level_signifier`. It modifies the working memory element `CurrentHighLevelSignifier` to hold the name of this high-level signifier. For example, the code fragment for 'mope_by_fence' in Figure A.4 waits until the 'sad_looks_through_fence' low-level signifier has been posted, and then posts the high-level signifier 'mope_by_fence.'

Final Note

I set up the system so that information about signs and low-level signifiers are stored on the high-level signifier of which they are a part. After implementing the Patient, it became clear that they should be posted on global memory instead, since you sometimes want to know what the last sign or low-level signifier was even after the high-level signifier that posted them is gone. Certainly, it is possible to have multiple low-level signifiers and signs be posted simultaneously to different high-level signifiers, but in practice this property was not particularly relevant to the running of behavior. It seemed that merely overwriting the most-recently-posted signifier or sign — no matter which signifier it had originated from — would have been simpler to implement and just as useful.

```
(sequential_production show_reaction_to_line ()
  (locals (current_line 0))
  ;;
  ;; find out what line the user saw me read
  ;;
  (with (success_test
            ("CS (CurrentSign name == read_line;
                              $$current :: c;);")
                  ;;
                  ;; store it in a local variable
                  ;;
            (:code "$$current_line = c; "))
      (wait))
  ;;
  ;; show a reaction to that line
  ;;
  (subgoal show_reaction $$current_line))
```

FIGURE A.5: Sign variables can be matched as part of the `CurrentSign` wme by preceding the name of the variable with `$$`
.

A.1.2 Matching on Signs and Signifiers

As you may have noticed from the previous code examples, behaviors can match on signifiers just as on anything else in memory. The name of the high-level signifier can be found on the `CurrentHighLevelSignifier` WME, and the name of the high-level signifier's low-level signifier can be found as a field of that high-level signifier. Since signifiers are behaviors, they can also be matched as any other behavior can; they can be distinguished from other behaviors using the flags `low_level_signifying_p` and `high_level_signifying_p`.

Signs can be found on the `CurrentSign` WME as described above. A special property of signs is that they include not only a name but a first-class environment which represents their arguments. These arguments can be matched straightforwardly through a special syntax which is shown in FigureA.5. This code fragment is taken from a transition; it checks which line of the schedule the user has seen the Patient read, then shows a reaction to that line. The `$$` syntax is unpacked by the Expressivator compiler and used to generate matching code for the proper variable in the `CurrentSign`'s first-class environment.

A.2 Details of Meta-Level Control Implementation

Meta-level controls are introduced in Chapter 5 (pp. 124 - 127). They are special powers that behaviors can use to find out about and coordinate with each other. Meta-level controls are implementable in many behavior-based architectures. This section describes how they are implemented on top of Hap for the Expressivator. They involve the following changes to the way Hap works:

1. *Querying behaviors*: I make use of the as-yet-underutilized Hap behavior matching as a regular part of the Expressivator. I add `low_level_signifying_p` and `high_level_signifying_p` as fields to behaviors, so that other behaviors can test for them.

2. *Deleting behaviors*: I add the primitives `succeed_behavior` and `fail_behavior` to allow behaviors to terminate other behaviors either successfully or unsuccessfully.

3. *Invoking higher-level behaviors*: I generalize Loyall's `breed_goal` function to allow behaviors to add new subbehaviors to any other behavior, or to the agent's top level.

4. *Adding new subbehaviors to other behaviors*: Loyall's `breed_goal`, as adapted for invoking higher-level behaviors, is also used to add new subbehaviors to other behaviors.

5. *Changing internal variables*: I add the concept of Communicative Features and a data structure to store them. Communicative Features allow behaviors to coordinate their presentation in order to present a coherent picture to the user.

6. *Paralyzing behaviors*: I add the primitives `turn_on_muscles` and `turn_off_muscles` to allow behaviors to paralyze and unparalyze other behaviors.

7. *Moving running subbehaviors*: I add `succeed_and_strip_behavior` and `fail_and_strip_behavior` primitives to allow subbehaviors to be switched to another behavior, while causing the old behavior to believe the now-missing subbehavior either succeeded or failed, respectively.

Each of these changes is discussed in more detail in the sections that follow.

A.2.1 Querying Behaviors

In Hap, behaviors sense the environment and check structures in memory by matching against RAL working memory elements (for more information on RAL, see [Forgy, 1991]). For example, the RAL test

```
"B (BelievedLocation x == $$x1;
                     y == $$y1;);"
```

checks the x and y values of the `BelievedLocation` WME against the value of the variables `$$x1` and `$$y1`. The RAL test

```
"P (PositionSensor who == I;
                   valid == true;
                   who_x == $$x;
                   who_y == $$y;
                   xydist != -1;
                   xydist <= RADIUS_ERROR);"
```

senses the position of the agent and checks whether it is within a small distance of the target point `($$x,$$y)`.

In order to be able to sense other behaviors, then, behaviors need to be represented as RAL WMEs against which behaviors can match, in the same way they do for data in memory or environmental conditions. This turned out to serendipitously already be the case in Hap, as a side-effect of its implementation. The compiler turns behaviors into WMEs called 'Goal,'[3] which include the field `name`, which is used most often in testing.

This attribute of Hap was not actually used for anything (or, for that matter, common knowledge among Hap users) until Bryan Loyall used it as a basis for adding a meta-level control, `breed_goal` (which will be discussed below) in the version of Hap he implemented for his thesis [Loyall, 1997a]. In order to be able to sense behaviors in the Expressivator, the only change that was necessary was to add the fields `low_level_signifying_p` and `high_level_signifying_p` to the 'Goal' WME so that behaviors can sense whether behaviors are signifiers. The `CurrentSign` and `CurrentLowLevelSignifier` fields mentioned above are also implemented as part of the Goal WME.

Once behaviors are matched, they often need to be stored and passed around. For example, a behavior may try to find out which low-level signifier is currently running, then tell one of its subgoals to delete that low-level signifier. The 'Goal' WME, which represents a behavior, stores an integer pointer to itself in the field `self`. This integer is used to refer to behaviors by the meta-level controls that follow (for an example, see Figure A.6).

A.2.2 Deleting Behaviors

The underlying Hap architecture has always needed to terminate behaviors; the change in the Expressivator is to make this internal function also available for behaviors to call directly as part of the behavior language. Specifically, I added `succeed_behavior` and `fail_behavior` primitives that took as arguments a behavior pointer, and would terminate that behavior either successfully or unsuccessfully (see Figure A.6).

It turned out in practice that being able to terminate a particular, specified behavior was not that useful for transitions. This is because transitions take place from one externally-seen signifier to another. This externally-seen signifying behavior may not be the same as the internal behavior the agent is engaging in, for example if the agent just changed to a new signifier but has not emitted its signs yet. In this case, the transition does not want to kill the externally-seen but no-longer-existent signifier from which it ostensibly comes. Rather, it will want to kill the newly-begun-but-not-yet-announced signifier which the transition will replace.

The solution to this problem is to introduce new forms which terminate, not a particular signifier, but *any* signifier which is in conflict with this one. Specifically, `kill_low_level_signifier`, when called within a particular low-level signifier, terminates (successfully) all other

[3] Hap makes a distinction between 'goals' (the name of a behavior) and 'behaviors' or 'plans' (the way in which behaviors are implemented), which is not pertinent to the current discussion. I have left it out for fear that it will hopelessly muddy the discussion.

```
(sequential_production
      freeze_in_place_interrupt (parent_plan)
  (locals (g 0))
  ;;
  ;; whirl around
  ;;
  (subgoal whirl_around)
  ;;
  ;; add low-level signifier 'freeze_in_place'
  ;; to the high-level signifier
  ;;
  (breed_goal $$parent_plan freeze_in_place)
  (subgoal wait_for "random_range(1000,6000)")
  ;;
  ;; finish freezing after you have waited a
  ;; while
  ;;
  (demon (("G (Goal name == freeze_in_place;
                  self :: s;);")
          (:code "$$g = s;"))
    (succeed_behavior $$g)))
```

FIGURE A.6: Example of use of matching on behaviors, `breed_goal`, and `succeed_behavior`

low-level signifiers — whether posted or not — that are part of the same high-level signifier as the calling behavior (i.e., all of that behavior's low-level siblings). `kill_high_level_signifier` similarly terminates (successfully) all other high-level signifiers. Both of these forms are implemented as behaviors.

A.2.3 Invoking Higher-Level Behaviors

Invoking new behaviors is a normal function of Hap. Any behavior can generate new subbehaviors. But transition behaviors need to add, not subbehaviors, but new behaviors at *higher* levels.

Specifically, if a transition behavior starts up a new low-level signifier as a subbehavior, the low-level signifier will be the 'child' of the transition behavior rather than of its 'real' high-level signifier parent. Because of the semantics of Hap, this also means the transition behavior needs to stick around until the signifier it invokes terminates, which seems wrong; the transition behavior should be done as soon as the new signifier begins, not when it ends.

These concerns mean that the transition should invoke the new subbehavior, not as part of itself, but as part of its parent high-level signifier (if it is a low-level signifier) or part of the agent's top-level behavior (if it is a high-level signifier). Fortunately for me, Loyall implemented a `breed_goal` form that does this as part of his thesis work. It is limited, however, in that it only works for adding subbehaviors to parallel behaviors, i.e. behaviors whose subbehaviors all run simultaneously. For the Expressivator, `breed_goal` is generalized to work for all behaviors,

```
(parallel_production
     read_sign_to_exercise_transition_demon ()
  (locals (exercisep 0))
  ....
  (breed_goal $$apt_plan sb_exercise)
  (demon (("GE (Goal name == exercise;
                    has_child == true;
                    self :: s;);")
          (:code "$$exercisep = ...."
                 ;; plan for exercise behavior
          ))
    (breed_goal $$exercisep watch $$overseer)))
```

FIGURE A.7: Example of using `breed_goal` to add a new subbehavior to another behavior.

whether parallel or sequential; the new subbehavior will run in parallel with the behavior's original subbehaviors. You can see an instance of `breed_goal` in practice in Figure A.6.[4]

A.2.4 Adding New Subbehaviors

Loyall's `breed_goal` can be used in the straightforward way for adding new subbehaviors to a specified behavior. Figure A.7 shows how the transition from reading the schedule to exercising uses `breed_goal` to add to the new exercise behavior a subbehavior to watch the Overseer.

A.2.5 Changing Internal Variables

Neal Reilly [Neal Reilly, 1996] added Behavioral Features to Hap in order to ease the problem of behavioral coordination. Behavioral Features are variables like "aggression" or "fear" that behaviors share. They are somewhat like emotions, but rather than representing how the agent 'feels,' they represent how behaviors should display the agent's emotions. For example, one agent, when afraid, may become aggressive; another may become quiet and shy.

I used the same basic mechanism as Behavioral Features, but I termed them Communicative Features to make clear that they are things that the behaviors need to communicate to the audience. Communicative Features are stored in a special data structure on global memory which includes two fields: an arbitrary label, `type`, and an integer `intensity`.

In the Industrial Graveyard, I used two Communicative Features: fear and woe. Although this was not my original intention, they correspond basically to a kind of simple emotional system. Whenever a traumatic event happens, the transition into the event calls a function "traumatize" that increases the agent's fear. After a traumatic event, as the fear subsides, woe increases. If the agent is left alone for a long time, woe goes back down. Sadly for the little agent, woe is usually maxed out by the

[4] Technical detail that is meaningless to all but Hap cognoscenti: `breed_goal` takes as an argument, not an integer behavior pointer, but an integer plan pointer.

```
;;
;; This behavior causes the Patient to tremble
;; when the Overseer is near it.
;;
(sequential_production
                 tremble_overseer_when_close ()
  (locals (feardist "0"))
  ;;
  ;; find out how scared I am supposed to look
  ;;
  (demon (("CF (CommunicativeFeature
                   type == fear_of_overseer;
                   intensity :: i;);")
          ;;
          ;; make distance at which to tremble
          ;; short if not scared, long if scared
          ;;
          (:code "$$feardist = 350 * (i / 5);"))
    ;;
    ;; tremble when Overseer is less than this
    ;; distance away from me
    ;;
    (subgoal tremble_overseer_at_dist
             $$feardist))))
```

FIGURE A.8: Example of use of Communicative Features

end of the story. I also used distance from Overseer to influence fear, but it did not affect the woe.

Despite the similarity with Neal Reilly's system, for me the function of these 'emotions' is not so much *as* emotions — although they do influence the agent's behavioral choices — but as a way to knit together disparate behaviors. That is, the Communicative Features act as a kind of behavioral smoothing between behaviors. Without the Communicative Features, the agent might go from a totally miserable round of moping to a very cheerful hop across the room, which looks very wrong. With my two features influencing most of the behaviors (this took about two days to add — for an example see Figure A.8), the behavioral consistency looked much better.

Having gotten this 'emotion system' to work by making it maximally simple, I suspect that a complex emotional system is not appropriate for really expressive agents. This may sound like a paradox. But just as behaviors can be hard to understand if you cannot see what is motivating them, subtle emotions are difficult to understand unless you clearly explain what is making the emotions arise. For example, originally I just had the agent's fear go up when the Overseer came close, and the fear go down when the Overseer went away. I found this made the system hard to understand, because the agent's emotions would change without the agent necessarily displaying any reaction to the event causing the emotional damage. This is why I choose to traumatize the agent when

```
(sequential_production
         paralyze_high_level_signifier ()
  ;;
  ;; find a behavior which is a high-level
  ;; signifier
  ;;
  (precondition
      ("G2 (Goal high_level_signifying_p == true;
                 self :: s);"))
  (locals ("g" "s"))
  (turn_off_muscles $$g))
```

FIGURE A.9: Example of `turn_off_muscles`

it is reacting to the Overseer, rather than when it senses the Overseer — this links the emotional change clearly with what the user is seeing.

A.2.6 Paralyzing Behaviors

By 'paralyzing' a behavior, I mean allowing a behavior to run while intercepting all of its muscle commands. This means behaviors can have effects in Communicative Features, but not in actual movement. I implemented this by using dummy movement commands that check to see if a behavior has its muscles turned on before actually doing the movement. Any behavior can turn on or off the muscles of any other behavior using the constructs `turn_on_muscles` and `turn_off_muscles` (for an example, see Figure A.9).

A.2.7 Moving Running Subbehaviors

Conceptually speaking, moving subbehaviors while they are running is straightforward. The behavior is simply taken from its parent and reinstalled under a different behavior. The `succeed_and_strip_behavior` and `fail_and_strip_behavior` primitives do just this: they move a given subbehavior from one behavior to another, while causing the former parent behavior to believe the suddenly disappeared subbehavior has succeeded or failed, respectively.

While this is conceptually simple, it was technically the most complex meta-level control to add. It basically corresponds to doing brain surgery on the agents. Since the compiler never expected behaviors to move around while they were running, when subbehaviors are taken out from one place and moved to another there is a large and not clearly marked (not to mention largely uncommented) group of pointers that need to be reinitialized to their new, proper values.

In practice, I found that this meta-level control was not really worth the enormous effort it took. Moving subbehaviors was better dealt with by simply deleting the old subbehavior and starting a new version of the same subbehavior in the new spot. In fact, after all the work I put in it, I did not end up using this meta-level control at all!

A.2.8 Related Work

As mentioned on page 125, a number of meta-level controls already exist in other behavior-based architectures. Brooks introduces the idea of subsuming behavior's action commands; Neal Reilly introduces Behavioral Features; Blumberg has Internal Variables and meta-level commands. The meta-level control system here attempts to bring some order to these features by finding a small set that will support behavior transitions.

Meta-level controls are reminiscent of metalevel plans in PRS[Georgeff and Ingrand, 1989]. Like metalevel plans, meta-level controls are intended to allow behaviors to use and manipulate meta information about the system's processing. However, PRS's metalevel plans are intended to be used to allow the system to plan its otherwise reactive behavior, and concentrate on formalizing the system's self-knowledge. Meta-level controls are intended to help designers coordinate behaviors, and focus on adding just enough power so the designer can write behaviors that explicitly refer to one another.

A.3 Details of Transition Implementation

Once meta-level controls are implemented, most of what you need to implement transition triggers and transition demons is already available. In addition to the meta-level controls, I made the following changes to Hap for the Expressivator:

- I added a data structure so transition triggers and transition demons could share information about transitions.
- I added `create_mini_transition` and `create_maxi_transition` primitives to create the transition demons.

Here, I will describe the transition data structure and the implementation of transition triggers and transition demons.

A.3.1 Transition Data Structure

Transition data is stored in the `Transition` WME. Mini-transitions are created with and stored on the high-level signifier to which they belong; the one and only maxi-transition is stored on global memory. The following data is stored in the `Transition` WME:

- **to**: which signifier is being switched to
- **from**: which signifier is being switched from
- **reason**: an arbitrary label which is selected by the designer to represent the reason for the transition (and, hence, what the transition demon must demonstrate)
- **valid**: has value 1 iff the transition has been triggered, but has not been implemented by a transition demon yet
- **switching**: has value 1 iff the transition is in process. It is automatically turned off when the next signifier is posted.

- **type**: whether it is a mini- or maxi-transition

- **high-level signifier**: for mini-transitions, lists the name of the high-level signifier to which this mini-transition belongs.

A.3.2 The Gritty Details of Transition Trigger Implementation

At their most basic, the job of a transition trigger is to notice when it is time to change behaviors for a particular reason. A transition trigger generally runs in the background, waiting for the right combination of environmental factors, signs, signifiers, etc. When a transition trigger notices that it is time to change behaviors, it notifies the rest of the system by altering the `Transition` data structure. Transition demons will check those data structures and fire to implement the transition. An example of a transition trigger is shown in Figure A.10.

Triggers generally want to fire only when particular behaviors are or are not being engaged in. Sigh, for example, only wants to fire when the agent is feeling sad, not feeling very afraid, and is not engaging in react-overseer or a similarly urgent, hyper behavior. Typically, then, triggers are complex demons that go on the alert when an appropriate behavior to switch from is happening, and then have conditions that abort the alert when the behavior is no longer happening.

Triggers turn out to be complicated at times because signifiers become internally active before the user notices them (i.e., before they are posted). Sometimes, triggers need to fire off of what is going on internally, while at other times, what matters is what the user has seen. The actual conditions under which the trigger should fire must be thought out carefully.

For example, when headbanging, if the light goes on the Patient should kill the smack-head behavior immediately, even if it has not been posted yet. This is because smack-head will hit the Patient's head on the floor before it can post its first signifier, and the lamp looks pretty unreactive if it hits its head when the light is on. So even before smack-head has been posted, the transition trigger must be on the lookout for possibly transitioning out of it.

On the other hand, the transition sequence *itself* needs to move from user-seen behavior to user-seen behavior. If smack-head is active, but has not been posted yet, the user will still think the Patient is in its wait-for-light-on behavior. The transition that will be demonstrated must go from that externally seen wait-for-light-on behavior, not the internally active smack-head behavior!

I found it helpful in designing triggers to think in terms of durations under which different conditions are true:

```
;; This is the trigger for the transition from
;; headbanging to being killed.  It fires when
;; the Overseer comes near.
(sequential_production
       monitor_overseer_approach_to_be_killed ()
  (with (success_test
         ;; check that this transition has not
         ;; already fired
         ("TT (Transition type == maxi;);"
          "- (Transition
                 type == maxi;
                 from == head_banging;
                 reason == overseer_approached;
                 switching == 1;);"
          "S (Self me :: I;);"
          ;; check that it is time for the
          ;; patient to die
          "SS (StoryStage stage == SS_DIES;);"
          ;; check that the user knows I am
          ;; headbanging
          "CHS (CurrentHighLevelSignifier
                      name == head_banging;);"
          ;; check that the Overseer is near me
          "PS (PositionSensor
                     who == I;
                     valid == true;
                     target_who == $$overseer;
                     xydist > -1;
                     xydist < 150;);")
          "ESPosition, make_position_wme,
              modify_position_wme, 6, self, -1,
              -1, -1, $$overseer, -1"
          ;;
          ;; Trigger the transition to be_killed
          ;;
          (:code
            "modify TT t {
               t->to = be_killed;
               t->from = head_banging;
               t->reason = overseer_approached;
               t->valid = 1;};"))
        (wait)))
```

FIGURE A.10: An Example of a Transition Trigger

```
;;
;; This is a transition from reacting to the
;; overseer to stepping around the environment
(parallel_production react_to_step_demon
                     (parent_plan)
  ;;
  ;; transition trigger: wait for me to be
  ;; reacting to the Overseer, and for the
  ;; Overseer to go away
  ;;
  (with (persistent when_fails)
    (subgoal check_when_overseer_goes_demon))
  ;;
  ;; transition demon:
  ;;
  (create_mini_transition
        (step parent_plan
              "reason == overseer_goes;"
              :from react_overseer)
    ;;
    ;; kill whatever signifier came before me
    ;;
    (subgoal kill_low_level_signifier)
    ;;
    ;; stop cowering
    ;;
    (act "AStopTremble")
    (act "AStopLook")
    (act "AStopFace")
    (subgoal stop_crouching)))
```

FIGURE A.11: An example of `create_mini_transition`

<table>
<tr><td>Old signifier running</td><td>Transition triggers</td><td>Transition demon starts</td><td>New signifier starts</td><td>New signifier posts</td></tr>
<tr><td colspan="4">Old signifier posted</td><td>New signifier posted</td></tr>
<tr><td></td><td>Transition valid</td><td colspan="3"></td></tr>
<tr><td colspan="2"></td><td colspan="2">Transition switching</td><td></td></tr>
<tr><td colspan="2">Old signifier runs</td><td>Transition runs</td><td colspan="2">New signifier runs</td></tr>
</table>

Depending on the situation, it would be appropriate to trigger off of almost any of these changes in state. This definitely adds a level of complexity to designing the triggers properly.

A.3.3 The Similarly Horrendous Details of Transition Demon Implementation

In order to implement mini-transitions, I added a new form to Hap called `create_mini_transition`. This form is used to automatically set up most of the bookkeeping details that are involved with transition demons. The create-mini-transition form takes the following arguments:

- a variable representing the high-level signifier of the mini-transition (to which the new subbehavior should be attached),
- the name of the behavior to which the mini-transition switches,
- a piece of RAL code which tests the reason for the transition,
- the set of steps that make up the transition sequence.
- a set of optional, keyworded arguments, including
  - `:old_beh` for the behavior the transition is from
  - `:interrupt` if the transition is an interruption

(see Figure A.11 for an example).

The mini-transition then sets up a demon which checks for the transition to fire for the correct behavior and reason. This demon then calls another behavior which implements the transition. The transition can be implemented in one of two ways: (1) actually do the given transition sequence or (2) just kill the old behavior and jump directly to the new one (the 'sudden break' which is the norm in other agent architectures). The system does the first option most of the time, but will use the second option when transitions are turned off, or when the user is not actually looking at the agent.

The same basic technique is used for maxi-transitions (see Figure A.12).

A.4 Summary of Expressivator as Agent Language

Implementation of the Expressivator is spread through Chapters 5 and 7. Here, I summarize the changes made in both chapters to Hap as an agent programming language.

- Signs, Signifiers, and Sign Management
  - The markers `low_level_signifying` and `high_level_signifying` are added to the language in order to allow the declaration of low-level and high-level signifiers.
  - The form `post_sign`, along with an arbitrary list of variables and their values, allows signs to be posted in common memory with a timestamp and their variable list.
  - The forms `post_low_level_signifier` and `post_high_level_signifier` are added to the language. When invoked, they store the name of their enclosing low-level (respectively, high-level) signifier.

```
;; this is the transition from headbanging
;; to be-killed
(parallel_production
  headbanging_to_be_killed_transition_demon ()
  ;; transition trigger: fire when the Overseer
  ;; is coming to kill me
  (with persistent effect_only
    (subgoal
        monitor_overseer_approach_to_be_killed))
  ;; transition demon:
  (create_maxi_transition
     (be_killed "reason == overseer_approached;"
                :old_beh head_banging)
    ;;
    ;; kill whatever high level signifier is
    ;; running
    (subgoal kill_high_level_signifier)
    ;;
    ;; make sure my eyes are shut
    (par
     (subgoal close_eyes)
     ;;
     ;; stop - do you hear someone coming?
     (subgoal wait_for 500))
    (par
     ;;
     ;; traumatize myself
     (with effect_only (priority_modifier -5)
   (subgoal traumatize 5))
     (seq
      ;;
      ;; whirl around blindly
      (subgoal whirl_around)
      (subgoal wait_for 800)
      (subgoal whirl_around)
      (subgoal wait_for 800)
      ;;
      ;; switch to new behavior
      (breed_goal $$apt_plan be_killed)))))
```

FIGURE A.12: Example of create_maxi_transition

- Signs can be tested by checking the `CurrentSign` wme, which is attached to the high-level signifier of which the sign is a part. The compiler is changed to allow the values of the sign's variables to be tested in the same way as any other memory element.
- Low-level signifiers can be tested by checking the wme named `CurrentLowLevelSignifier`, which is attached to the high-level signifier of which the low-level signifier is a part.
- High-level signifiers can similarly be tested by checking the `CurrentHighLevelSignifier` wme, which is a global variable.

• Meta-Level Controls

- The ability to sense behaviors is already a part of Hap; the Expressivator includes the addition of a number of fields to the behavioral data structure: `low_level_signifying_p`, `high_level_signifying_p`, `CurrentLowLevelSignifier`, `CurrentSign`. These allow various additional aspects of the behaviors to be tested.
- Behaviors can delete other behaviors, causing them to either fail or succeed, by calling either `fail_behavior` or `succeed_behavior`, respectively.
- Behaviors can add subbehaviors to other behaviors or at the agent's top level by calling `breed_goal`. This functionality is already present in Hap and allows subbehaviors to be added to behaviors whose subbehaviors run in parallel. It is expanded in the Expressivator to be applicable to behaviors whose subbehaviors run in parallel or sequentially (the new subbehavior will always run in parallel).
- Behaviors can move around running subbehaviors, switching them from an old behavior to a new one, by calling `succeed_and_strip_behavior` or `fail_and_strip_behavior`. The first construct causes the old behavior to believe the subbehavior succeeded; the second construct causes the old behavior to believe the subbehavior failed. Since the old behavior is usually deleted right away, it generally does not matter which one of these are chosen.
- Behaviors can paralyze and unparalyze other subbehaviors by using the `turn_on_muscles` and `turn_off_muscles` constructs.

• Transitions

- `create_maxi_transition` is called with the name of the new high-level signifier, a token representing the reason for the transition, the behavioral steps that express the reasons for the transition, and a set of optional keywords including the name of the old behavior and a keyword designating the transition as an interruption. It generates the code to trigger the demon's steps when the given reason is cited for behavioral change to the given new high-level signifier.

- `create_mini_transition` works on exactly the same principle, but for low-level signifiers.

Appendix B

Detailed Analysis of *Luxo, Jr.*

This appendix contains the details of the analysis of *Luxo, Jr.* in terms of the behaviors and transitions that can be seen in Luxo's actions. Note that this division into behaviors and transitions is not written in stone; it is just one reasonably good match.

Senior		Junior		Ball
Transitions	Behaviors	Transitions	Behaviors	
	stands still			
starts slowly, slowly puts more movement in	when ball stops, turns to look at it			comes in and bounces off of senior
	examines ball			
no transition, except slight stop	smacks ball off screen			

Senior		Junior		Ball
Transitions	Behaviors	Transitions	Behaviors	
here, "watching" is a kind of transition	watches			comes back
	stops smacks ball again			
				comes back again, rolls past Senior
	Senior follows ball, smoothly turns back to Junior (offscreen)			
Shock reaction and scoots back while looking off-screen				
			hops onto stage	
		wiggles butt, looks at ball		

Senior		Junior		Ball
Transitions	Behaviors	Transitions	Behaviors	
	looks at ball		looks at S	
	looks at J	alternation	looks at ball	
	watches J		wiggles butt and hops off	
	watches ball			comes back
			comes back	
		looks at ball		
		gets in position		
			smacks ball	
				hits cord
			“struck” reaction	
	hits ball away		looks at ball	
		stops		
		looks at ball		
		hunkers down		
	looks		jumps on ball	
	surprised			
			rides ball	
				pops

Senior		Junior		Ball
Transitions	Behaviors	Transitions	Behaviors	
	leans in more	looks around as though wondering what's going on		
		decides what to do (looking)		
	looks at ball		rolls back	
		looks at ball		
			flips flat ball over	
		sits back and looks		
	shakes head		looks at S	
			looks deflated sighs	
alternation as a kind of transition	looks at ball	sighing hop		
	looks offscreen		hops off screen	
shock reaction	hops back	off screen (transition from S's behavior!)		

Senior		Junior		Ball
Transitions	Behaviors	Transitions	Behaviors	
	watches (supervise)			big ball comes on stage
			hops after (double hop in middle)	
looks at screen	shakes head			

Appendix C

Case Study: Full Design of the Patient

In this appendix, I step through the entire design process for the Patient character of the Industrial Graveyard.

C.1 Selecting High-Level Signifiers

As described in Chapter 7, the first step in the agent design process is to decide on the agent's high-level signifiers. The Patient was designed in the context of the Industrial Graveyard; its behaviors needed to support the plot of the story (as described in Intermezzo I on pp. 87-88), as well as enhance the user's understanding of the point of the system.

The Patient's high-level signifiers roughly parallel the story plot.

- *In Monitor*: Initially, the user needs to understand that the Patient is being processed mechanically. This is represented in the Industrial Graveyard by having the Patient be examined by the Overseer in a machine called the Monitor. The Monitor reduces the Patient's subjectivity to simple numerics: the user is notified that the Patient has an identification number and a numerically identified 'disease' (short circuit), and that its demerits are being tracked by the system. The Patient's first high-level signifier represents its behavior as it is being processed into the system.

- *Explore World*: Once the Overseer is done processing the Patient, it leaves and the Patient can begin exploring the 'world' (i.e. the junkyard). While exploring the world, the Patient is constantly sanctioned by the Overseer whenever its movements exceed proper bounds. This behavior in connection with the Overseer's reactions to it demonstrates to the user the Patient's helpless position in the world.

- *Read Sign*: There is a schedule of daily activities displayed in the junkyard. As the Patient wanders around, it notices the schedule and goes up to read it. The schedule again is intended to make clear to the user that the Patient's activities are structured for it, and that it has no choice but to do what is on the schedule.

- *Exercise*: Once 10:00 strikes, the Patient must exercise. This consists simply of rapidly bobbing up and down.

- *Mope By Fence*: After some time of being bullied by the Overseer, the Patient becomes depressed. The Patient engages in the Mope by Fence behavior by slowly walking over to the fence of the junkyard, and sadly looking out at the outside world, now forever beyond its grasp. This should be a behavior chock full of pathos.

- *Head-Banging*: The Patient has a short-circuit. This means its light goes out from time to time, leaving it blind. In order to remedy the situation, the Patient may shake its head; if that fails, the Patient will start smacking its head on the ground in order to fix the short. This behavior is designed to be as negative as possible in the eyes of the Overseer; it involves the most jerky body movement.

- *Turned Off*: Whenever the Patient has been misbehaving, the Overseer will come over and turn it off. The Turned-Off behavior consists of the Patient collapsing onto the ground into an unnatural position. After a few seconds, the Patient gets up again and continues on its way as the 'sedatives' wear off.

- *Be Killed*: After the Patient has gotten in enough trouble, the Overseer decides that it is more efficient to turn the Patient off than to continue to monitor it. While the Patient is being killed, it needs to act very frightened so that the user knows something unusual is happening.

- *Unknown Behavior*: This behavior is designed to test one of the transition types, the unknown transition (p. 124 of Chapter 5). The Unknown Behavior consists of simple and relatively meaningless background activity that the lamp can engage in when it is not sure what it should be doing.

C.2 High-level Signifier Decomposition

Once the high-level signifiers are identified, they need to be decomposed. High-level signifiers are broken up into a set of low-level signifiers, which represent the major activities that make up the high-level signifier. These low-level signifiers will later be connected with mini-transitions.

- In Monitor
  - *Be Mechanical*: In the beginning, the Patient reinforces the mechanistic propaganda the user has just read by acting completely mechanical. The Patient doesn't blink; it moves slowly and mechanically, and it does not visually track objects or the Overseer.
  - *Tremble and Watch Overseer*: Once the Patient notices the Overseer, it 'comes to life.' It tracks the Overseer's movements and trembles nervously.

  - *Look Around Scared*: When the Patient is not watching the Overseer, it starts to examine its environment. It is, however, still frightened, so it still trembles now and then and uses quick, jerky looks.
  - *Look Around Curiously*: Once the Patient has gotten used to the Monitor, it becomes curious. It gets closer to the front of the machine, and looks out into the junkyard. Its looks are slower and longer, and its gaze follows things in the environment.

- Explore World
  - *Looking Around*: This is the behavior the Patient uses when it is trying to decide where in the world it should go. It looks around for interesting spots. It should not pick such spots near the Overseer.
  - *Go To Spot*: The Patient walks determinedly, if fearfully, to the spot it has chosen. It looks mostly at spot but checks out the rest of the environment, too.
  - *Look Around*: Once it has gotten to a particular spot, it looks it over. This behavior has a focus of interest.
  - *Sigh*: Overcome with sadness, the Patient occasionally interrupts other behaviors with a sigh.
  - *React to Overseer*: Whenever the Overseer comes close, the Patient reacts to it by trembling and acting fearful.
  - *Freeze in Place*: Occasionally, the Patient's paranoia gets the better of it, and it interrupts its behavior to freeze in place and look around for danger.

- Mope By Fence
  - *Look Out At World*: The Patient sadly stands at the fence and slowly moves its gaze around the outside world.
  - *Sigh*: Ah, what pathos! Let your sadness escape, little creature!
  - *Walk Up and Down Fence*: Sometimes the Patient will move up and down the fence a little to find a better viewing position.

- Read Sign
  - *Read lines*: The Patient moves its head from left to right in a reading motion.
  - *React to lines*: Sigh, shake its head, or read a line more than once.

- Exercise
  - *Bob up and down*: Exercising consists simply of this bobbing up and down motion.

- Head-Banging
  - *Hit head on ground*: The Patient flings its head back and then whacks it into the floor of the junkyard.

  - *Wait to see if light went out*: The Patient pauses in its head-banging to see if its light has come on yet.
- Unknown Behavior
  - *Sigh*: As always, sighing is an essential part of the Patient's existence.
  - *Look Around*: Look around aimlessly, seeing what is going on around the Patient.
  - *Watch Overseer*: Keep an eye on the Patient's evil enemy.
- Be Killed
  - *Fear City*: The Patient needs to show that it is extremely frightened. This is like the trembling at the Overseer mentioned earlier, but even more extreme.
  - *Die*: When the Patient dies, it turns into a cardboard cut-out.
- Turned Off
  - *Be Turned Off*: Collapse and stay turned off for a while.

Each of these low-level signifiers was implemented separately. Note, however, that some of the high-level signifiers share the same low-level signifiers; in this case, the code for them was shared as well.

Composing Low-Level Signifiers with Mini-Transitions

Once I selected the signifiers for the Patient, it was time to connect them to form the Patient's complete behavior. The first step was to synthesize the low-level signifiers with mini-transitions in order to generate the high-level signifiers. In order to do this, for each high-level signifier I made a list of all possible mini-transitions between its low-level signifiers.

Fortunately, many of the possible transitions turned out to be impossible. For example, Be Mechanical is always the first behavior, and always leads to noticing the Overseer and becoming frightened. Therefore, there is no need to implement transitions from Be Mechanical to any other behavior.

Once the mini-transition list was whittled down, I enumerated reasons for each behavior change. For each reason, I also listed how that reason could be concretely be communicated to the user. These two aspects form the basis for the design and implementation of each mini-transition[1].

The synthesis of each high-level signifier through mini-transitions is laid out in Figures C.1 through C.10.

[1] In the case of the Patient, I contented myself to have only one reason for each behavior change, but there is no reason to limit oneself this way in general.

For **In Monitor**:
Low-level signifiers:

1. Be Mechanical
2. Tremble and Watch Overseer
3. Look Around Scared
4. Look Around Curious

Transitions:

From	To	Reason	How
1	2	see Overseer	shock reaction; back up
2	3	less scared (Overseer turns or goes away)	blend looks at Overseer and around world set Communicative Feature fear to maximum
3	2	more scared (Overseer turns or comes back)	quick jerk to Overseer; maybe back up
3	4	even less scared (Overseer has been far away for a while)	notice something interesting start looking at it
4	2	scared again (Overseer comes back)	scurry to back

FIGURE C.1: Mini-transitions that make up In Monitor

For **Explore World**:
Low-level signifiers:

1. Looking Around
2. Go to Spot
3. Look Around
4. Sigh
5. React to Overseer
6. Freeze in Place

Transitions:

From	To	Reason	How
1	2	Picked a spot that looked interesting or plausible	Focus on spot Look left, right Focus on spot again Go for it
1	5	Overseer came nearby	Whirl to face Overseer Back up Tremble
2	1	Overseer approached chosen spot (but not agent)	Shock reaction Watch Overseer Turn in opposite direction to pick something there
2	3	Got to spot	As approaches spot, look intently at object of interest
2	4	How sad! I miss the outside world!	Pause a moment in reflection
2	5	Overseer came nearby	Same as 1→5
2	6	I hallucinated the Overseer might be nearby	Glance around very quickly Turn and look at spot behind me.

FIGURE C.2: Mini-transitions that make up Explore World