Believable Agents researchers have long pointed out that the nature and utility of various technological mechanisms may change radically when the intelligibility of agents is seen as equally important to — or more important than — their reasoning abilities in abstract. Taking this point of view changes, for instance, what behaviors fundamentally mean. In alternative AI, behaviors are assemblages of actions that help the agent to fulfill its goals with respect to the environment, e.g. to navigate around the room (Brooks), avoid getting too hungry (Blumberg), or to kill enemies and win the video game (Agre and Chapman). Behaviors are defined in terms of their correctness in helping agents to achieve their goals.

In socially situated AI, however, behaviors are fundamentally the designer's vehicle for communicating an idea of agent activity to the audience. Behaviors need to be designed, not just in terms of fulfilling the internal goals of the agent, but in terms of what the agent is communicating to the audience. It is not enough to just do something; the audience must be able to tell the agent is doing it. This means a behavior includes the intention to communicate that behavior to the audience. 'Behaviors' therefore explicitly become something more like 'understandable aggregates of action' than the a priori, problem-solving modes of behavior in most behavior-based AI applications.

Many behavior-based researchers have focused on action selection, i.e. determining when an agent should switch to another behavior. Again, action-selection takes a problem-solving view of agents in that it focuses on correctness: when the agent should, for the sake of correctness, switch to a different behavior. The focus on agent *presentation* that is part and parcel of socially situated AI means that the question of what behavior the agent should pick is less important than how well the agent communicates through that behavior. For socially situated AI, then, the fundamental problem is better rephrased as what Tom Porter terms *action-expression* [Porter, 1997] [Sengers, 1998]:

> How can the agent at every point choose an action that best communicates the goals, activities, and emotions the designer has selected to its audience?

But even this point of view is too limiting, since it causes us to focus on the mechanics of agent action choice. The point here is not doing the "correct" behavior, but doing the behavior well. For human understanding, the manner in which the agent *does* the what it has chosen is just as important, if not more so, then whether or not the agent has chosen the optimal thing to do. These conceptual differences are summarized in Figure 5.1. These differences form the foundation for addressing schizophrenia.

Schizophrenia Revisited

In Chapter 2, we learned that schizophrenia comes about when the agent's behaviors are so atomized that they become easy for the user to pick out. Schizophrenic behavior has one or more of the following properties:

	Alternative AI	Socially Situated AI
Concept of agent	Autonomous	Communication
Concept of behaviors	Chunks of problem-solving	Chunks of meaning
Fundamental problem	Action-selection	Action-expression (and more)

FIGURE 5.1: Differences between alternative and socially situated AI

1. The agent, rather than engaging in a fluid stream of activity, jumps abruptly from behavior to behavior.
2. The agent combines actions from different behaviors in a way that makes no overarching sense.

These properties happen because the agent's behavior is atomized into meaningful units, with very little intercoordination of each unit.

A natural instinct when faced with schizophrenia is to hope it can be resolved by getting rid of atomization. It turns out that this is probably not a very practical solution for complex agents with a variety of high-level behaviors. Atomization, in the form of modularization, is what allows us to build these complex systems in the first place, since unmodularized systems beyond a certain size become an interrelated, undebuggable mess.[1] There are natural limitations to the size of these unmodularized systems because people simply cannot keep track of what is going on in the code without some level of abstraction.

In Chapter 2, we came to the conclusion that schizophrenia is therefore unsolvable. This is, in fact, the case, as long as we look at the agent in isolation. A humanly constructed agent will almost certainly be atomized, and therefore also schizophrenic. However, the problem of schizophrenia changes in some interesting ways when looked at in the context of agent and designer.

Situating Schizophrenia in Context

Here, I describe the fundamental philosophy motivating the technical choices I make later. You might have thought that fundamental philosophy is in Chapter 4, and how right you are! Here, the goal is to bring that philosophy close to the technology so that it can be instantiated.

From the designer's point of view, atomization is necessary in order to maintain a manageable system. Constructed agents don't spring out of the air; they are constructed by someone who needs to be able to understand and control how they work. In order to be effective, the agent architecture must be simple enough that the designer can understand and, to a reasonable extent, control the effect of the agent. This leads to the first heuristic we will use in addressing schizophrenia:

Remember the designer
Support modularized code to make the programming job easier and more understandable.

[1] It is possible that such systems could be learned automatically. The exploration of mechanisms which could automatically generate complex, expressive, and deeply interrelated behavior is still in its infancy. I suspect (but certainly cannot prove) that systems that are truly so complex will also have to be learned step-by-step in a modularized fashion that may undermine the possibility for truly interrelated, learned behavior. The argument in this thesis limits itself to systems which are (mostly) humanly designed and built.

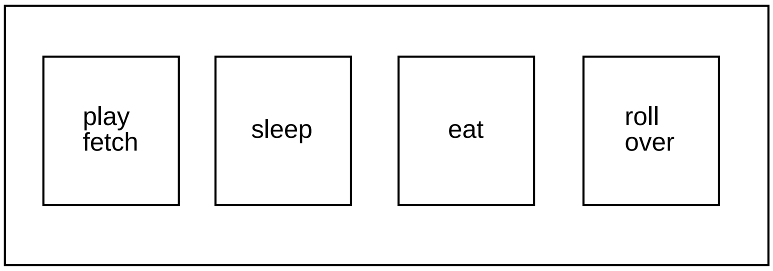


FIGURE 5.2: Atomized behaviors leave gaps that are obvious to the user.

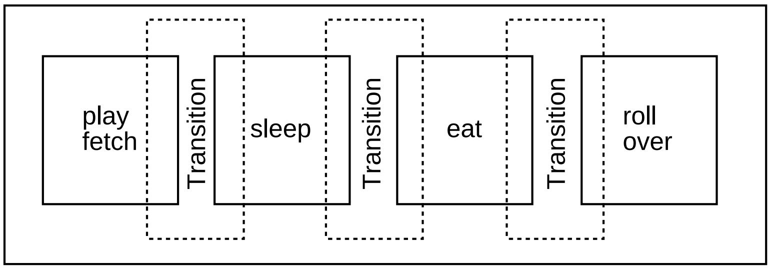


FIGURE 5.3: Atomized transitions cover up the breaks left by atomized behaviors.

As noted before, while atomization is good for the designer, it is bad for the user; the agent jumps abruptly from behavior to behavior, or mixes its actions together in an incoherent mess. If we do not want to give up atomization, we need to find a way to mitigate its effect. Specifically, since the agent is a form of communication, our goal will be to integrate the *effect* of the agent, rather than the agent per se. This forms the basis of our second heuristic:

> **Remember the audience**
> Integrate the *impact* of the behavior, not behaviors themselves.

This observation holds the key to solving schizophrenia. From the user's point of view, atomization is bad because it makes it too easy to see the 'breaks' in the system. The problem for the user is that he or she can see the 'lines' the programmer has drawn in the agent. Those lines are obvious, since they are drawn between the behaviors, i.e. the high-level activities we expect and hope the user will be able to recognize. These considerations lead to an obvious conclusion: *if we draw the lines somewhere different* — somewhere where the user is not trained to look, and hence has more difficulty recognizing them — *the agent may not appear as schizophrenic*.

In particular, if users are good at recognizing behaviors, abrupt switching between behaviors will be obvious to them. Instead, we should switch *during* a behavior. When the switches occur during a behavior, not between behaviors, they will be less obvious to people watching the agent, since even after the switch the agent is, from the point of view of the audience, still doing the same thing. The way to do this is to make behavior transitions — which traditionally fall through the architectural cracks — into full-fledged modules or components of the agents, i.e. *atomize the*

> Of course, these figures don't prove anything. They rely on a visual metaphor to make the basic argument plausible. Such diagrams have a venerable tradition in AI... as well, it seems, as everywhere else.

behavior transitions themselves. This concept is graphically represented in Figures 5.2 and 5.3.

Since this is the key to all the technical work in this chapter, I will leave the reader a moment of silence to contemplate this changed viewpoint.

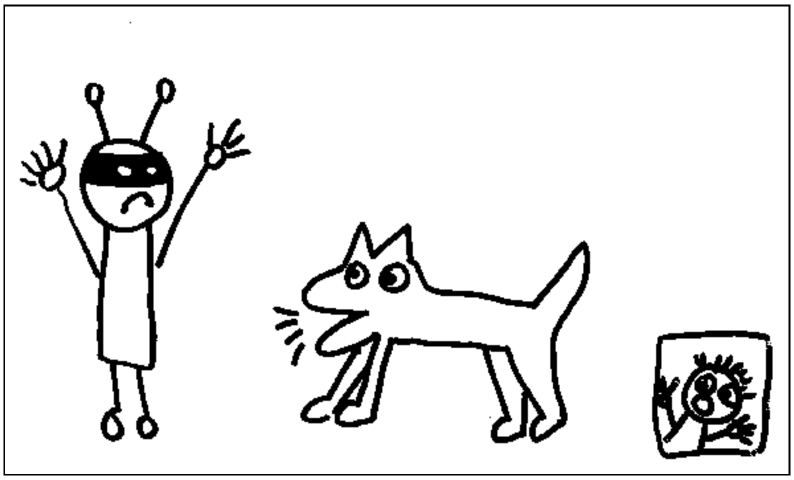


FIGURE 5.4: Our dog in action

Treating Schizophrenia in Attack Dogs

For example, suppose we want to build an artificial "guard dog" Following the behavior-based approach, we'll pick a selection of behaviors for it, such as "eat," "sleep," "chew on bone," and, since it is a guard dog, "bark at intruder." Then, we'll try to find the circumstances under which each behavior is appropriate: if you're hungry, eat; if you're tired, sleep; if there is a bone, chew on it; if there is an intruder, bark.

Now imagine that one day our dog has found a burglar to bark at (Figure 5.4 — the user is represented in the box in the corner). In this case, having been properly programmed, the dog starts barking. The observer, having some background knowledge of dogs and burglars, is likely to understand that the dog is trying to scare away the intruder.

Suddenly, the dog realizes that it has gotten very tired (Figure 5.5). What this means in technical terms is that the internal counter for "tired" has reached a threshold that outweighs the importance of scaring away the burglar (maybe the dog has been barking at various intruders all day, or had a particularly thrilling morning at the park).

Since sleeping is now the most important thing to do, the dog immediately stops barking and passes out on the floor (Figure 5.6). This sudden change of circumstances leaves the poor observer stymied: what on earth is that dumb dog doing? is it dead? did the burglar drug it? does the burglar have mystical hypnotic powers? This strange sudden break is, for the observer, the symptom of the dog's schizophrenia.

By adding a transition, we can mitigate the effects of this schizophrenia on the audience. A transition could work like this. As soon as the dog starts noticing that it is getting tired and likely to switch to sleeping, the dog will terminate the bark-at-intruder behavior and start a bark-to-sleep transition. This transition would keep the dog barking, while gradually adding some signs of sleepiness. When the dog becomes very tired, the dog could become more droopy, bark more slowly, lie down, bark a few more times, yawn, and then fall asleep. With this transition, there is no sudden break to confuse the user; the user understands both what the agent is doing (sleeping, not dead), and why the agent did it (was very

In this section, I will use a specific example to make the technical proposal plausible. The idea is to spin a narrative under which the technology seems intuitively correct. It connects a particular technology with a suggested way of experiencing reality. Since AI researchers are also human beings, I like this way of connecting lived experience with technology, in the philosophy of [Varela *et al.*, 1991].
Nevertheless, this strategy is also subject to some abuse (a nice analysis of this phenomenon can be found in [Agre, 1990]). In the 70's, a relatively common technique was to have one or two examples to intuitively ground the technology, and never bother to implement it at all (in all fairness, one can hardly blame people for trying to avoid working on the complicated and slow machinery of that day). Another tactic was to implement only a single example of the basic idea, and proclaim that as some kind of proof. This approach is pleasantly satirized in [McDermott, 1981].
The '90's have, for various reasons including this, seen a kind of backlash against this style of AI. Now, researchers often insist on concrete justification, preferably of an objective, empirical kind inspired by physics. Hopefully, we will one day be able to find a happy medium.

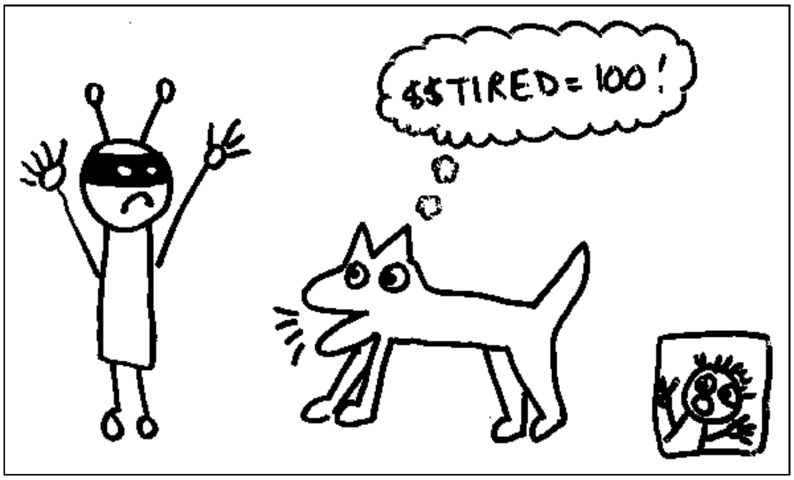


FIGURE 5.5: Rex gets sleepy

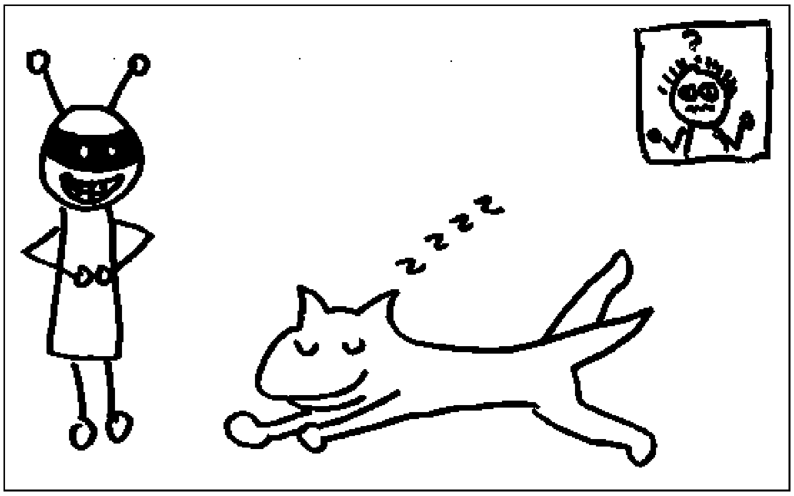


FIGURE 5.6: Rex immediately starts to snooze

tired). The transition "covers up" the break between the behaviors, so the change in the agent's behavior is gradual and natural. It is likely the user will not notice the "real" (i.e. internal) breaks at all (the one between bark and the transition, and the one between the transition and sleeping).

Summary

Behavior transitions can be thought of as a a form of strategic de-atomization. Rather than getting rid of atomization at the code level (where the designer needs it), behavior transitions reduce the apparent atomization of the agent from the audience's point of view. Behavior transitions allow the designer to use the full strength of atomized high-level behaviors without reducing agent activity to an abrupt jumping around from behavior to behavior. Agents with behavior transitions do not have discrete behavior breaks; rather, they *blend* their behaviors together.

Technically speaking, behavior transitions are a straightforward extension of the basic behavior-based framework. Transitions are themselves behaviors that act to 'glue' two distinct high-level behaviors together. When a behavior transition notices that it is time to switch between two higher-level behaviors, it takes over from the old behavior. Instead of switching abruptly to the new behavior, it 'finishes up' for the old behavior and introduces a plausible transition to the new one.

The technical reader may now feel a burst of trepidation at the additional burden of work transitions may introduce. After all, if a transition is needed to connect any 2 behaviors, then for n behaviors we will be forced to write $O(n^2)$ transitions! We will see in Chapter 7 that while we will probably need to write at least $O(n)$ transitions, the actual number of transitions needed is limited, through mechanisms including their localization within high-level behaviors and their generalization (transitions that can go either from or to any arbitrary behavior).

Translation for non-technical readers: if we have to write a transition for every combination of 2 behaviors, then for 10 behaviors we need to write about 100 transitions, for 20 behaviors we need to write about 400 transitions, and so on. That is too much work to be practical. But it will turn out that the actual number of transitions needed is far less.

The ways in which transitions work and the architectural foundations they need are the subject of the rest of this chapter. We will start with a survey of the support for behavior blending that already exist as parts of various agent architectures. This will provide the basis for the architectural mechanisms — sign-management, transitions, and meta-level controls — that allow designers (1) to build agents with respect to how they will be interpreted, and (2) to use transitions to de-atomize those interpretations.

The Magic Principles

1. Don't integrate the agent; integrate the user's understanding of the agent.
2. Don't stop atomizing; change the choice of what to atomize. Let the designer understand and control the effect of the created agent.

Behavior Blending: State of the Art

In order to blend behaviors, we need to have techniques that allow us to combine behaviors together. In both classical and alternative AI, the technique most commonly used when two behaviors need to be combined is to interleave the agent's actions. For example, planning approaches for conjunctive goals integrate behavior by interleaving activities for each goal without any smoothing between them. The subsumption architecture, Pengi, and ANA all rely on interleaving actions to combine behaviors.

Here, we want to actually blend behaviors together. In order to find tools for this, we need to find ways in which you can combine behaviors that can smooth, average, or otherwise compromise between two behaviors, turning a discrete behavior break into a smooth transition from one to the other. While this smoothing has not previously been done on complex high-level behaviors, there are a number of techniques already available for smoothing between lower-level actions.

Ideas from Graphics

Blending is common, for example, in computer animation. In animation, it is clearly not appropriate for a character (or inanimate object, for that matter) to jump jerkily from pose to pose. At the same time, animation studios do not want to waste money and time by having highly-skilled animators draw all 24 frames a second in order to generate a half hour of animation. One of the common techniques to handle this is to use keyframes to specify a character's actions, and then use a process called "in-betweening" to provide smooth transitions from keyframe to keyframe. "In-betweeners" used to be humans, but they are now mostly replaceable by programs that can do the same thing. These programs can do various kinds of interpolation (averaging) between frames to smooth them out.

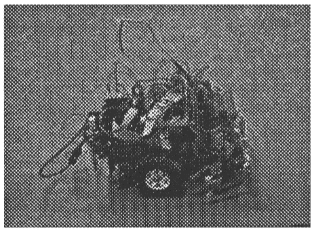
The related work section is essential for placing the developed technology in the context of a community. This section gives credit where credit is due for ideas that inspire the current work. It can be used for reference by people trying to do something similar to your work. It often also includes a component of explaining how your developed technology is different from (and by implication better than) what other people do. I managed to withstand this temptation here since I go so far as to devote several entire chapters to this argument elsewhere.
If you are not technically trained, this section may be hard to follow, since fully explaining each related technology for a non-technical audience would double the size of this already bloated thesis. Nevertheless, I would encourage you to hang in there and try to read this section at a high level, so you get some flavor of how these problems are thought about and some idea of how the technology I develop relates to AI technology in general.

Other graphics systems allow you to specify the animation by providing various key poses, and using physical simulation to figure out how the object should move between the poses. Jerks that remain at the low level can be worked out by a process called "time domain supersampling." With this technique, the computer system generates twice as many frames as necessary and then blurs between them instead of jumping from discrete state to discrete state. More details on these graphical approaches to transitions can be found in [Watt, 1993].

Ideas from Low-level Action

While these graphical techniques do not map directly to agent action, they introduce the idea that you can smooth between two discrete states by doing various kinds of averaging between them. This idea has been applied to agent action as well, resulting in various techniques that average between actions to create smooth transitions.

Ken Perlin, for example, has built a system representing a human dancer [Perlin, 1995] who can follow discrete commands ("rhumba," "walk," "run," etc.) while moving smoothly from one behavior to the next. The "actions" in Perlin's system represent joint angles (e.g. "move left knee 30 degrees"). Each behavior consists of a set of actions over time. When switching from one behavior to the next, the weight of the "finishing" behavior is gradually reduced to 0, and the weight of the "starting" behavior is simultaneously gradually increased to 1. To determine the actual actions that the dancer does, it multiplies the weight of the behavior by the magnitude of the action, so that the dancer's behavior is gradually, for example, less rhumba-esque and more like walking. Perlin also adds some additional constraints to make sure that the combined activity actually makes sense. Interestingly, Perlin also mentions the value of having smooth activity be visible to the user, while the programmer can think purely in terms of the atomized, discrete behaviors.


Steels's robot

Luc Steels' agent architecture, which is used to run robots, works on a similar principle [Steels, 1994]. Like Perlin, Steels explicitly states that smooth behavior switching is one of his goals. Steels criticizes the concept of action-selection as being incapable of generating smooth behavior because it implies jumping from action to action. Instead, Steels has all behaviors running all the time, with the resulting action commands being added together to generate the robot's final activity. For example, if one behavior wants to turn left, and one wants to turn right, the result will be that the robot goes straight ahead. Since this clearly could result in nonoptimal behavior (for example, if the robot wants to turn either left or right because there is a wall right in front of it), behaviors need to be developed hand-in-hand so that the additive principle works out correctly (rather than the independent behavioral development of many others in alternative AI).

Ideas from blending low-level actions for high-level behaviors

These systems focus on relatively low-level action (mostly moving around). Systems that are going to combine high-level behaviors will necessarily be more complex. These systems often have a motor-level component that is in charge of action (e.g. "go 3 feet to the left") with a high-level component that takes care of high-level behaviors (e.g. "go to the store") and sends orders to the motor system. In order for the action to look plausible, these systems, too, have various techniques for combining actions.

Blumberg's Hamsterdam [Blumberg, 1996], for example, has a motor level system which takes care of the low-level details of the agent's activity. An agent's body has various "Degrees of Freedom" that represent things an agent can move independently (for example, wagging its tail is usually independent of sticking its tongue out). Motor Skills are various low-level physical actions the creature can engage in that effect some of the agent's Degrees of Freedom, like "walking," "wagging tail," "putting ears back," etc. Motor Skills can be blended in two ways:

1. If Motor Skills affect different Degrees of Freedom they can happen simultaneously (you can walk and chew gum).

2. Consecutive Motor Skills can be smoothed by always putting the body in the same posture between the Skills. Silas the dog always stands up between actions; this makes sure that he doesn't, for example, spring from a lying-down behavior into a begging position. Unfortunately, this also means that he will stand up between lying-down and sitting down, which Blumberg points out doesn't seem quite right.

Both the Woggles [Loyall and Bates, 1993] and the Industrial Graveyard have a motor system that is at its most basic level surprisingly similar to Hamsterdam, given that they were developed separately. These agents have "body resources" which correspond to Hamsterdam's "Degrees of Freedoms." For these non-biologically-inspired agents body resources include such things as the bottom of the agent, the top of the agent, and the angle the agent is facing. Agents have a set of low-level physical actions that they can engage in; things like "squash," "spin," and "jump."

In these Hap-based systems, agents' actions are physically simulated. One benefit of this is that the graphics system that runs this simulation takes care to smooth the actions together appropriately. Between two consecutive actions, the system calculates an appropriate intermediate state based on the physics of the world. An agent, for example, that strings together two jumps will take care to land the first jump to transfer its momentum into the second; an agent that is jumping once and then stopping will land in a way to stop its momentum (otherwise it would fall on its face). This means that, unlike in Hamsterdam, there are no stereotypical in-between states the agents always engage in to move from one behavior to another.

One of the most complex and interesting methodologies for combining low-level action for high-level behaviors is explored by Gerald Payton and his colleagues. In both Hamsterdam and Hap, behaviors can ask for an action; if they conflict, the most important behavior's action actually happens, while the other behaviors have to wait until the important behavior is done. In Payton's system, behaviors give, not a single action command, but a range of preferences for various actions. The preferences of all behaviors are combined according to the importance of each behavior, and the best resulting action is selected. Behaviors can say both which actions they want, and which they don't want; Payton's system therefore avoids Steels's system's problem of the robot running into the wall, since both behaviors will say they do not want the robot to go straight. Additional details of the action specification mechanism can be found in [Payton *et al.*, 1992].

Ideas from high-level behaviors

It should be clear at this point that there already are a number of reasonable solutions to the problem of low-level behavior blending. There are several useful techniques for behavior blending, from various forms to averaging, to simultaneously engaging in both behaviors, to moving to set in-between states, to using physical simulation to determine how the actions can be combined properly. However, these techniques are not always appropriate for high-level behaviors. How can you average between going to the store and staying at home? Should an agent always stand stock-still, looking straight ahead, between any two high-level behaviors? Can physical simulation tell you how to move from dancing the rhumba to eating dinner? At some point as behaviors become more complex, the meaning of a behavior becomes more than the physical actions of which it consists (including, for example, groups of conditions under which different actions are appropriate). At this point simple averaging or weighting schemes no longer suffice to blend one behavior appropriately into the next one.

Clearly, the first step in blending behaviors is being able to blend the actions of one behavior into that of the next; for this we can use some of the techniques of the previous section. Now, we will take a look to see what support we currently have for blending together high-level behaviors themselves, and not just the actions they output.

There was no direct support for interbehavioral effects in the original version of Hap [Loyall and Bates, 1991]; the only tangential support was the availability of global memory.[2] The resultant difficulties in creating coherent behavior were noticed by both Bryan Loyall and Scott Neal Reilly, who add new mechanisms for interbehavioral support in their respective theses [Loyall, 1997a] [Neal Reilly, 1996]. Loyall adds dynamic variables, which allow different behaviors to share information about what they are doing; these variables can then be used by behaviors to coordinate what they are doing. A more direct support for behavioral

[2] Similarly, the subsumption architecture provides message passing, but as far as I know this is not used to support behavior blending.

coherence is provided by Neal Reilly, who introduces the concept of "behavioral features." A behavioral feature is an overall emotional attribute that the agent's behaviors should display (for example, "fear," "anger," "happiness"). Behavioral features are used in many behaviors to modify their action in order to create the overall impression of a coherent and identifiable emotional state.

Blumberg uses "Internal Variables" to allow his behaviors to share information in a way somewhat analogous to behavioral features. An Internal Variable is information that is local to a particular behavior, but will be shared with another one. For example, an Internal Variable could be "Focus of Attention;" behaviors sharing this variable will make sure the agent's activity, though switching from behavior to behavior, remains focused on the same object in the agent's environment.


Herman the Bug

An additional twist Hamsterdam makes is to allow behaviors to make different kinds of action commands, which can be blended in different ways. A behavior may issue a "primary" command, which basically means "do it!" A behavior that merely wants to make a recommendation can issue a "secondary" command, which means "do it unless someone more important objects." Or, a behavior can make a "meta-level" command, which means "if anyone wants to do it, they should do it this way" (e.g. "if I am going to walk, then it had better be slowly!"). This last kind of command can be used to create an effect like behavioral features, by getting the behaviors to generate a style of action that is coherent over the different behaviors that may control the body.

These systems add some tools into the behavior blending mix. The system that currently has by far the greatest level of blending and transition support, though, is Lester and Stone's Behavior Sequencing Engine [Stone, 1996] [Lester and Stone, 1997]. The "Behavior" in this title is something of a misnomer, since their system actually sequences not programmed behaviors but hand-made animation clips of their character, Herman the Bug. While some of their techniques are limited to sequencing clips, others can be generalized to more complex behaviors as well.

Herman is a pedagogical agent, whose role is to supervise students in an educational simulation, stepping in with advice when students seem to be getting lost. Because children are impatient with characters that are supposed to be alive but seem wooden and mechanical, Lester and Stone's system is specifically focused on generating *visually coherent* activity for their agent. At the low level, film clips are sequenced seamlessly by using a technique called "visual bookending." This means that the start and end frame of each clip is chosen from a small set of possible "home" frames, and only clips with the same home frame are sequenced together. This system is analogous to Silas's movement to standing between his behaviors, although the use of a variety of "in-between" states reduces the danger of stereotypicity. If clips that must be sequenced have different keyframes, a transition animation is played in between to move from one to the next.

At a higher level, since Herman spends a lot of time explaining concepts, much attention is paid to making these explanations coherent. Rather than jumping from topic to topic, Herman uses 'topical transitions' between different explanatory behaviors. In addition, when Herman has been quiet and now wants to launch into a very noticeable behavior, he uses an anticipatory action to alert the student he is about to do something he or she should notice. For example, if he has been lying down, he will sit up before he launches into an explanation.

At this point, we have various tools in various frameworks for supporting behavior blending. Each of these provides part of the answer. Turning this into an adequate AI technology requires a few more pieces:

1. We need to have some conception of the full range of kinds of transitions, so we have some idea of what the architecture needs to support.
2. We need a common framework that will allow us to provide support for these different kinds of transitions in a single system.

This is the goal of the rest of the chapter.

Design of the Expressivator

The architecture designed here is called the *Expressivator*,[3] since, unlike most current systems, it focuses on the ways in which the agent expresses its designer's intentions to the audience, rather than on what the agent is doing internally from moment to moment. The goal of the technology developed in this chapter is to be able to de-atomize the agent from the user's point of view, by introducing techniques for smoothing between observed behaviors using transitions. This will involve three major components:

1. We need to provide the agent author with a way of being able to program the agent with respect to what the user sees the agent do (not just what the designer thinks the agent is doing). Agents built under the Expressivator are structured using *signifiers*, which are behaviors that are explicitly communicated to the audience through the use of low-level *signs*. The agent uses a *sign-management system* to keep track of signs and signifiers that have been communicated, allowing it then to decide what to do based not only on sensing and its internal state, but also on what has been communicated to the user.
2. We need to get some idea of the range of possible kinds of behavior-blending *transitions*, so that we have some idea of the kinds of things the architecture needs to support. These transition types specify different ways in which high-level behaviors can be smoothed together.
3. We need to add structures to the architecture that will allow it to support this range of transition types. The Expressivator does this through the use of *meta-level controls*, or special mechanisms which transition behaviors can use in order to sense and alter the behaviors they connect. In addition to supporting transitions, these controls allow the agent designer to explicitly coordinate and communicate the relationships between behaviors, rather than leaving the coordination of behavior as an implicit property of the agent architecture.

[3] Yes, this name *is* supposed to evoke images of 60's optimistic futuristic culture.

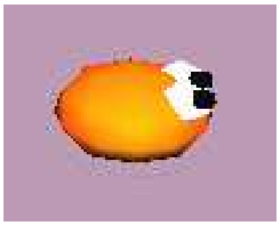


FIGURE 5.7: A Woggle that is 'clearly' moping.

These pieces together will form the structure of the Expressivator.

Signs, Signifiers, and Sign Management

As mentioned in Chapter 2, in 1992, a group including many members of the Oz Project built the Edge of Intention, a system containing small, social, emotional agents called Woggles that interact with each other and with the user. While building the agents, we took care to include a wide variety of behavior, which ranged from simple behavior like sighing and moping to relatively complex social behavior like follow-the-leader and fighting. At the same time, we made sure that the agents did not blindly follow the user but had a 'life of their own;' we hoped that this would make them more compelling personalities to get to know.

At the time, we believed that the individual behaviors of the agents were reasonably clear. After all, we — their builders — could usually tell what they were doing ("A-ha! It's small and flat! That means it is moping!" — see Figure 5.7 and judge for yourself). Soon, however, we found that it was difficult for other people to be able to understand the behaviors and emotions we were trying to communicate through the Woggles. Users were at a disadvantage because, unlike us, they did not actually have the code memorized while they were watching the agents. Because we — the builders — thought in terms of the underlying behavior names in the code, we had thought the agents' behavior was clear. This had led us to neglect to some extent the *external* behavior of the agents. Behaviors were not always programmed with enough observable actions that the audience could actually tell what the agent was doing.

For de-atomizing the user's impression, allowing the designer to control the impact, the external appearance, of the behavior is extremely important. But our lack in this department was not due (merely) to a perverse attitude about how agents should be programmed. Most current behavior-based architectures only allow designers to write code more or less purely based on an internalistic perspective. Agents make decisions based on what they perceive, on what they have recently done, or on their current mood — but not based on *what the user thinks the agent is doing*. This is partly a consequence of the attitude described in Chapter 4 that agents are fundamentally autonomous problem-solvers, and that therefore any impression the user may have of them irrelevant. But it is also partly a consequence of the difficulty of figuring out what on earth the user is thinking.

For many computer scientists, there are two main strategies that immediately suggest themselves for figuring out what is going on with the user: (1) perceive what the user is doing and try to figure out from that what the user is thinking; or (2) make a general model of what a typical user would be thinking, and use that to predict what the current user thinks of the agent. Neither of these options is in itself particularly compatible with behavior-based AI, since they both require a substantial amount of modeling and reasoning. Chances are also fairly good that either approach will be wrong a lot of the time — mind-reading is not well-developed among humans, let alone among computers.

Believable Agents research suggests that there is a third way out. In this view, there *is* a way to give the designer access to the presentation of the agent as comprehended without having to model or perceive the user, and that is by turning the tables on the user. The user could really be thinking anything; but the *designer* knows what he or she *wants* the user to think. The goal, then, is not to have the agent try to figure out what the user thinks, but rather to provide the designer with support for communicating as clearly as possible through the agent what the user should be picking up. Since designers are generally much more savvy about cues a human observer might pick up than an agent can be, this puts the most competent agency in the driver's seat.[4]

Non-technical readers may recognize this strategy from the arts. Directors of films, composers of music, and authors of books (and technical reports, for that matter) also do not know exactly what the 'user' of their works is going to pick up on, but they generally do not feel the need to develop a scientific, testable model to find out what the observer is thinking. Rather, they rely on their intuition, a tradition of techniques, trying things out on themselves, friends, and test audiences, and a preoccupation with presentation in order to communicate their concept successfully to the audience. The argument Believable Agents researchers make is merely that these sorts of things can also be tapped for AI.

Agent Structure for Communication

The goal of sign management is to provide support for communication within the agent design. The Expressivator implements sign management through the following three mechanisms:

- The agent's low-level activities are structured into *signs*, which communicate the meaning of the agent's actions directly to the user.
- The agent's high-level activities are structured into *signifiers*, i.e. behaviors *which are explicitly intended to be communicated to the audience*.
- The *sign-management system* keeps track of the signs and signifiers that have been communicated to the audience. Signs and signifiers are posted to memory when they have been communicated. This allows the agent to base its activity, not only on what it sees around it and where it is in its internal code, but also on *what the user has seen the agent do*.

[4]This philosophy is similar to the "Inverse User Model" suggested by Michael Mateas [Mateas, 1997] to manipulate users into 'seeing' the world in an author-chosen way.

At a high level, the motivation for sign management can be understood in the following manner. Typically in alternative AI, the behaviors with which the agent is programmed are activities which allow the agent to achieve its goals. Here, behaviors are better thought of as 'activities to be communicated to the user.' The Expressivator therefore structures agents according to *levels of meaning-generation*. Behaviors are not problem-solving units, but units of meaning to be communicated to the user, and they are organized according to the kind of meaning they communicate.

In Hap, for instance, behaviors are at the most fundamental level designed out of physical actions — such as "jump," "squash," or "spin" — and mental actions — such as "calculate a good angle for me to face".[5] Actions are combined into low-level behaviors, such as "say hi," "watch out for insults," or "walk to bed," which are small units of useful behavior. These units are then combined into high-level behaviors, such as "play follow-the-leader," "have a fight," or "take a nap," which represent what the agent is basically doing. The lines between low-level and high-level behaviors are not clearly drawn, but they provide a useful framework for thinking about behavior design.

In the Expressivator, the fundamental units of behavioral design are not physical actions that have effects in the world, but *signs* that have effects on the user. Signs, physical actions, and mental actions can be combined to form *low-level signifiers*; these are behaviors, which are differentiated from low-level behaviors only in that they are explicitly intended to be recognized by the user. Low-level signifiers can in turn be combined into *high-level signifiers*, which are behaviors which communicate the fundamental activities the user should be able to recognize in the agent. A *sign-management system* keeps track of when each sign and signifier has been communicated to the user. Now, we will take a look at each of these mechanisms in more detail.

Signs

The most basic unit of agent structure for most behavior-based architectures is also the most basic unit of physical activity, the physical action. Physical actions are commands to the motor system like "move hand left," "raise head," etc. While the Expressivator certainly composes behaviors out of physical actions, the design of the agent is not so much focused on what the agent is physically doing, but *how the agent's action will be interpreted*. This means that, at the design level, the most basic unit through which an agent is structured for the Expressivator is not the physical action but the *sign*.

A sign is a token the system produces after having engaged in physical behavior that is likely to be interpreted in a particular way. This token includes an arbitrary label (like "sigh") that is meaningful to the designer, and represents how the designer expects the physical behavior (like "stretch up for 100 milliseconds and then squash down for 100 milliseconds") will be interpreted. This token is stored by the sign-management system, so that the agent can use it to influence its subsequent behavioral decisions.

[5] Mental actions are expressed in C or Lisp code.

Behavior: *Harass patient to follow scheduled activity*

1. Go to schedule
2. Read schedule
3. Look at clock
4. Look at schedule
5. Look at patient
6. Wait a moment for patient to comply
7. Look at schedule
8. Look at patient
9. Shake head
10. Approach patient menacingly ....

FIGURE 5.8: Example of a behavior and its signs

Formally, a sign is an arbitrary label (such as “saw possible insult”) and an optional set of arguments that give more information about the sign (such as “would-be insulter is Wilma”). A behavior can ‘post’ a sign each time it has engaged in some physical actions that express that sign, using the `post_sign` language mechanism. For example, after moving its head slowly from left to right, the agent may post a sign “read_line” with an argument of the number of the line it just ‘read.’

Figure 5.8 shows an example of a behavior and the signs that are emitted during it. At first glance, these signs look like low-level physical actions, but there are important differences. Rather than corresponding to simple movements an agent can engage in, a sign corresponds to a *set* of such movements that *carries meaning to a user*. The “reading” sign, for example, combines a set of low-level actions as the lamp’s head moves from left to right across each line of the schedule. More fundamentally, signs are different from both actions and traditional behaviors in that they focus on *what the user is likely to interpret*, rather than what the agent is ‘actually’ doing. When “reading,” for example, the agent does not actually read the schedule at all (the locations of the lines and their contents are preprogrammed); it merely needs to give the *appearance* of reading.

Figure 5.9 shows how the `post_sign` language construct is used while the agent is walking; after each step, it posts that the user has seen it take a step towards a particular goal point.[6] Signs are context-dependent in the sense that the designer notes the meaning of physical actions within the context of the behavior in which the action appears. This means that the same physical actions might result in quite different

[6] It needs to keep posting the sign, even after the first step, in case the behavior is interrupted.

```
(sequential_production walk_towards (gx gy)
  (with (success_test
        (... agent has reached goal point ....))
     (with persistent
       (seq
         (subgoal take_step_to $$gx $$gy)
         (post_sign walking_to
                    ((x $$gx) (y $$gy)))))))
```

FIGURE 5.9: The 'walk_towards' behavior and the sign ('walking_to') it posts.

signs, depending on context: for the lamp, while walking, jumping to a new spot results in a "taking a step" sign, while during headbanging, the same physical action leads to a "hop around" sign.

Signifiers

Physical actions, mental actions, and signs are combined into low-level signifiers. Signifiers are behaviors that are *explicitly intended to be communicated*. Low-level signifiers correspond to low-level behaviors; they are relatively simple behaviors that convey a particular kind of activity to the user. In the Industrial Graveyard low-level signifiers include things like "hit head on ground," "tremble and watch the Overseer," "look around," and "go to an interesting spot." Low-level signifiers differ from low-level behaviors in ordinary behavior-based architectures in that users should be able to identify the low-level signifiers more or less correctly — which is otherwise not necessarily the case.

For example, a low-level behavior for the Woggles might be "watch out for insults." This behavior would consist mainly of sensing to make sure that no one is coming nearby and engaging in the "In Your Face" activity, which is the highest insult one Woggle can pay another. This sensing, however, does not have any component that is visible to a user. There is no way for the user to know that the agent is trying to avoid being insulted — the only way for the user to get this idea is to see the agent being insulted, watch it react, and then hypothesize that the agent was watching out for insults all along.

Turning "watch out for insults" into a low-level signifier means adding signs to it that communicate what the agent is doing to the user. An agent watching out for insults in this sense might glance around now and then, becoming nervous when it notices a frequent insulter coming nearby.[7] Now the user knows that the agent is paying attention for something — and, incidentally, is not caught off guard when the Woggle goes into a state of frenzy upon finally actually being insulted.

Low-level signifiers are identified by marking behaviors when invoked. This is done using a special marker, `low_level_signifying`, which has been added to the behavior language. The behavior 'smack_head' would be invoked as `(subgoal smack_head)`; to make it a low-level

[7] A Woggle might do these things too, but they will be components of other behaviors that are coincidentally displayed, not part of the watching for insults behavior itself.

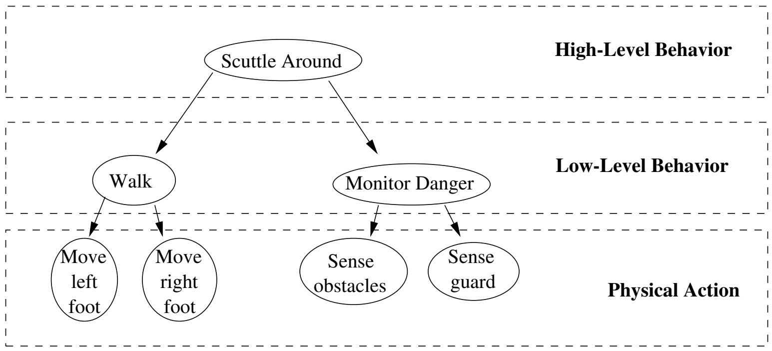


FIGURE 5.10: A typical behavior structure divides the agent into objective units of activity.

signifier, it is invoked as `(with low_level_signifying (subgoal smack_head))`.

Low-level signifiers can be combined to build up *high-level signifiers*. High-level signifiers are, like low-level signifiers, behaviors that are intended to communicate the agent's activity. High-level signifers are composed of low-level signifiers to form a complex, high-level activity. The Industrial Graveyard includes high-level signifiers like "head-banging," "exercise," and "be killed," These, like low-level signifiers, are intended to be communicated to the user. The rule of thumb is that low-level signifiers are groups of actions that can be grasped and comprehended as what the agent is doing on a moment-by-moment basis. High-level behaviors are what the agent should be thought of as doing at a whole. They extend over time and are composed of various low-level behaviors, which they organize into an intentional unit. The high-level signifiers, in turn, combine to form the complete activity of the agent.

High-level signifiers are identified in the analogous manner to low-level signifiers. A special marker, `high_level_signifying`, is added to the language. The 'headbanging' high-level signifier, for example, can then be invoked this way: `(with high_level_signifying (subgoal headbanging))`.

Summary: Signs and Signifiers

To summarize, a typical behavior-based architecture structures the agent according to its objectively determinable activities. To build a behavior like "scuttle around," in which the Patient wanders around the graveyard while trying to avoid danger, the high-level "scuttle" behavior may be broken into low-level walking-around and danger-sensing behaviors, which are in turn broken up into the physical actions (including sensing) of which they are composed (Figure 5.10). In the Expressivator, on the other hand, "scuttle-around" is a high-level signifier, which is broken into low-level signifiers, which are then broken into signs (Figure 5.11). Because signifiers and signs are explicitly intended to be communicated, the structure of the agent may change; for example, instead of simply sensing danger, the Patient actually moves its head and eyes around to look for danger, to be sure that the user will know what it is sensing and

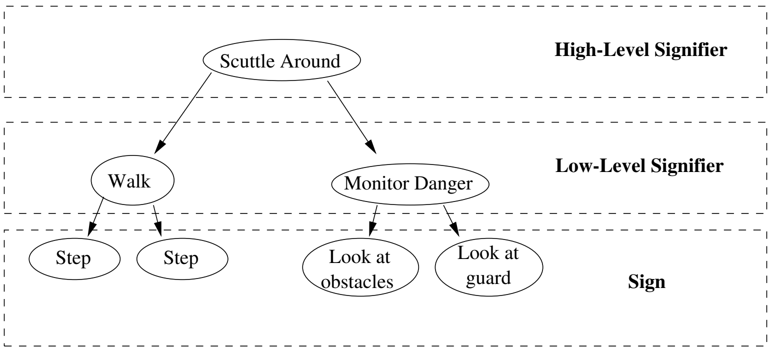


FIGURE 5.11: The Expressivator behavior structure divides the agent into subjective units of meaning to be communicated.

be able to identify the "monitor danger" behavior. In this sense, signs and signifiers help the designer to design the agent so that the designer's chosen behaviors actually are communicated to the user.

Sign Management

So far, I have described signs and signifiers in terms of how the designer can use them to structure their agent with respect to eventual user interpretation. But it would also be nice if the agent itself could reason about how the user is currently interpreting it. For example, if the agent is about to walk across the world, but the user most recently saw it hiding from the Overseer, the agent can modify its walking behavior to include glances at the Overseer so that the change in behavior seems less jarring. The *sign-management system* helps the agent to keep track of the user's current likely interpretation, so what the user is likely to be thinking can influence behavioral decisions in the same way as environmental sensing and internal state do.

The most obvious way for the agent to keep track of what the user thinks is for it simply to notice which signs and signifiers are currently running. After all, signifiers represent what is being communicated to the user. But it turns out in practice that this is not correct *because the user's interpretation of signs and signifiers lags behind the agent's engagement in them*. That is to say, if the agent is currently running a "headbanging" signifier, the user will need to see the agent smack its head a few times before realizing that that, in fact, is what the agent is doing.

The sign-management system deals with this problem by having the agent *post* signs and signifiers when it believes the user must have seen them. As mentioned about, the `post_sign` language construct is used to remember that a particular sign has been displayed. Similarly, the `post_low_level_signifier` and `post_high_level_signifier` constructs are used to remember that particular signifiers have been displayed. The question, then, is how the agent knows when the sign or signifier has been displayed and can therefore be posted.

Signs have been displayed — and are therefore posted — whenever the agent has done some physical activity that expresses the sign (Figure 5.12). "Posting" means the agent stores the sign and its arguments

```
(sequential_production smack_head_emotionally ()
  (locals (time "random_range(350,800)"))
  (par
   (subgoal snap_head $$time)
   (subgoal swing_head $$time)
   (subgoal squish_body $$time
            "random_range(-10,10)"))
  (par
   (act "ASquashHold" 0 "$$time / 2")
   (act "AElevateEyesTo" 0 "$$time / 2"))
  (post_sign smack_head_once))
```

FIGURE 5.12: Signs are posted once their physical actions have been engaged in.

```
(parallel_production hit_head ()
  ....
  (with effect_only
      (demon (("G (Goal CurrentSign
                            == slap_head_once;);"))
 (post_low_level_signifier hit_head)))
```

FIGURE 5.13: Low-level signifiers are posted after a demon notices that appropriate signs have been posted.

in memory; the agent also notes the time the sign was expressed. Now, other behaviors that want to know what the agent has been doing from the perspective of the user can check memory to see which sign has recently been posted.

Low-level signifiers, in turn, can be assumed to have been displayed when some key signs have been emitted. They therefore watch the signs that are emitted to find out when they have been expressed (Figure 5.13). For example, “look around scared” watches for a “scared glance” sign to be posted. When the “scared glance” sign appears in memory, the agent can start having some confidence that “look around scared” is starting to be communicated, too. The agent then posts that “look around scared” is being communicated, using a mechanism analogous to posting signs. In general, when the right signs have been posted, low-level signifiers post themselves, in effect announcing that the user should have seen them, too.

High-level signifiers, in turn, have probably been displayed when key low-level signifiers are expressed. They therefore watch for the posting of their low-level signifiers. When the right combination of low-level signifiers have been posted, the high-level signifier is posted as well (Figure 5.14). In this way, the agent can keep track of the impression it is making on the user, from the details of signs to the overall impression of high-level behaviors. More technical details of how this works can be found in section A.1 of the Appendix.

Once signs and signifiers have been posted, other behaviors can check

```
(parallel_production head_banging ()
  ....
  (with effect_only
      (demon
         (("G (Goal CurrentLowLevelSignifier
                        == hit_head;);"))
         (post_high_level_signifier head_banging)))
}
```

FIGURE 5.14: High-level signifiers are posted after a demon notices that appropriate low-level signifiers have been posted.

to see what has been posted recently before they decide what to do. Behaviors can check for arbitrary sequences of signs and signifiers. The end result is that *the signs and signifiers the agent has expressed can be used just like environmental stimuli and internal drives to affect subsequent behavior*. This means that in the Expressivator, behavioral effects on the user have the same status as action memory and perception in other systems. For example, a *watch-guard* behavior may check recent signs and notice that a *hide-from-guard* sign was posted; in this case, it would know to maintain behavioral coherence by peering at the guard carefully, rather than walking right up to the guard to see what it is doing.

Summary of signs, signifiers, and sign management

One nice property of this hierarchy of meaning-production is that it follows our principle of maintaining modularization in order to simplify agent design. Signs, low-level signifiers, and high-level signifiers can still be designed separately. When combining them into the full system, each level only needs to worry about the level directly under it. Signs only need to be concerned with the physical actions that express them; low-level signifiers only care about signs, not about physical actions; and high-level signifiers only need to worry about low-level signifiers, not signs.

Signs, signifiers, and sign management also provide the basic support for our other principle, i.e. designing agents with respect to their impact. In fact, the sign-management system improves not only the impact of the agent's behavior but also that of the agent-builder's! This is because, in addition to allowing agents to reason about what the user sees, it also forces the *designer* to reason about those things. By noting every time a sign or signifier is supposed to have been communicated by a behavior, builders' attention is focused on the problem of breaking a behavior into signs and signifiers and then making sure that they are expressed. The structure of the sign-management system encourages them to think about behavior in terms of signs and signifiers, and to construct appropriately expressive low-level behaviors to display those signs and signifiers.

Behavior Transition Types

Sign management provides the foundation for de-atomizing the agent's impact, since it allows us to design the agent with respect to its probable interpretation by the user. With this under our belt, we can turn our attention to providing support for behavior blending. The first step is to try to get a handle on the range of possible ways that behaviors can be combined. In this section, we will look at a variety of ways in which this can be done.

Generality is a great virtue in Artificial Intelligence (as in other sciences). Even researchers whose goal is to construct technology that is specific to a particular environment want to give the general rules of specificity. Ian Horswill gives an elegant example of how to do this. He builds an architecture radically specific to an environment — and then shows exactly which parts are specific to which properties of the environment, and therefore need to be replaced for the robot to run in another environment [Horswill, 1993]. In this section, then, I try to get as broad an idea of transition types as possible so that the Expressivator can be built to support as many of them as possible.
It may seem to you that I am basically making most of the stuff in this section up. If so, it is because I am. This is basically a form of brain-storming based on current architectures and on where they could go with the ideas of socially situated AI. I have no proof that this section is comprehensive, and I rather doubt that it is. But it is at least a place to start.

The analysis of already existing support for behavior blending suggests a number of transition types as a starting point:[8]

- *Parallel Behavior Blend*: Both Hap and Hamsterdam allow two behaviors to run simultaneously, sharing control of the agent's body. This is a meaningful form of blending when the behaviors use non-conflicting body resources (e.g. walking and talking).
- *Virtual Behavior Blend*: The subsumption architecture allows two behaviors to run simultaneously, while disabling one behavior's muscle commands. This means the disabled behavior can still perceive the world and influence the creature's emotions, but cannot move the agent's muscles. (It might be an interesting variation to allow the disabled behavior only to move the eyes; this way, the focus of attention of the disabled behavior can still peek through).
- *Average Behavior Blend*: The architectures for low-level action suggest that an interesting way of combining behaviors may be to average their action commands. It remains to be seen if this technique is meaningful for high-level behaviors.
- *Interruption*: If an agent intends to engage in a behavior for a very short time, it may make sense to merely interrupt one behavior with the other, then return to the first behavior when the second has completed. This is supported by nearly all current architectures, including Hap.
- *Sudden Break*: At times, the most appropriate way to combine behaviors is to jump from one behavior to another without transition. This can communicate that something sudden has happened to force the agent to switch rapidly, or that the agent has a highly reactive personality. You may have already noticed that this is the default in nearly all architectures — it is the definition of schizophrenia. But just because it is not so good to have sudden breaks *all* the time, this does not mean that it is *never* the right policy.

The example of the guard dog earlier suggests that one function of the transition is to make the reason for the switch to the new behavior plausible to the user. This means an important novel kind of transition can be the *Explanatory Changeover*. This transition is the default transition proposed when I introduced the concept of transitions on page 106: finish up the old behavior, engage in a sequence of actions that explains why the new behavior is being started, then start the second behavior.

[8] These categories were also inspired by my analysis of Luxo, Jr., which appears in Intermezzo II. While rhetorically it made sense to present them in this order, in practice the development of the ideas in this thesis was never so linear.

Finally, the *Accidental Transition* turns the Explanatory Changeover on its head by watching for and capitalizing on what the user would find plausible. The agent watches its recent behaviors for patterns of behavior that might seem reasonable to the user. When a particular pattern is launched, the agent can 'switch gears' to follow that pattern, instead of whatever it was planning to do originally. This is a technique frequently used by my cat in moments of embarrassment: rather than admit that falling off the windowsill was an accident, he finds some way of recovering so that it looks like he meant to do it all along. It is also akin to something that can be observed in split-brain patients, who manage with one side of the brain to spin narratives (albeit patently false ones) that structure actions taken by the other side.

I had gotten to this point in my analysis of transition types when I noticed there was something strange at work. Even though I had repeated my magic mantra of de-schizophrenization hundreds of times, I still found myself slipping back into my straightforward, technical, agent-as-autonomous mindset. Perhaps you have noticed the flaw in this line of reasoning already: *all the behavior transition types mentioned so far work with respect to the agent's internally-defined behaviors,* not *with respect to what the user sees*. The real question is not how behaviors can be combined, but how the user can be given the *impression* that behaviors are being combined. It turns out this re-formulation can make the problem much simpler — by avoiding the complexity of actually having two full-blown behaviors running simultaneously.

With this lesson firmly ingrained (or so I thought — the Doctrine of Agent Autonomy turns out not to be so easily erased from an AI researcher's world view), I went on to design several 'impressionistic' transition types:

- *Subroutine Behavior Blend*: Don't run both behaviors simultaneously; rather, take some 'representative' subbehaviors of one behavior and combine them with the other behavior. The idea here is to still give the user the 'flavor' of the behavior, without actually having the complexity of doing both behaviors simultaneously.

- *Principled Subroutine Behavior Blend*: Why stop at reducing only one behavior? Pick just a few subbehaviors of both behaviors, and combine them in a single blended behavior. This has the advantage of letting you weed out the subbehaviors of the 'dominant' behavior that conflict with the subbehaviors you would like to add to it.

- *Symbolic Reduction*: When it comes down to it, you don't even need to use *any* of the subbehaviors of the first behavior. Rather, the behavior can be reduced to a simple symbol or sign — a tick, a focus of attention, a particularly poignant movement — that is easy to incorporate in the second behavior. Note that this is similar to the use of Internal Variables in Hamsterdam, though with a different goal.

- *Reductive Behavior Blend*: We can make things yet simpler again. The Reductive Behavior Blend reduces the first behavior to an attribute whose value can vary — "mope" can be reduced to slowness; "hide from Overseer" can be reduced to fear; "escape from Overseer" can be reduced to agitation. This attribute is then used to modify the second behavior. We can now combine two behaviors with various emphases between them simply by varying this attribute's value from 0 to 1.

- *Off-screen Transition*: Since the goal is to blend the user's impression of the agent's behaviors, if the user is not looking at the agent at all, the agent can simply jump from one to the next.

- *Unknown Transition*: Sometimes, none of the agent's behaviors are appropriate. Rather than sitting around like a lump of silicon, the agent should fill in the 'lulls' between behaviors. A good way of doing this is to add a behavior that merely looks around the world or at the most recent object of attention.

Taken together, these 12 transition types almost certainly do not tell the full story of all the ways in which behaviors can be combined. They do, however, provide the groundwork for the kinds of ways of combining behaviors that the Expressivator should support. In the next section, I will introduce *meta-level controls* as a way to support these transition types — in addition to providing a form of de-atomization themselves.

Meta-Level Controls

At this point, the Expressivator is equipped with techniques for designing agents' impressions, and we have some idea of the kinds of transitions we would like the Expressivator to support. Now all we need to do is actually implement them.

It turns out that this is not entirely straight-forward. Most transition types depend on the agents' behaviors to know about and coordinate with one another. However, most behavior-based architectures are based on the idea that behaviors should be shielded as much as possible from one another. Because behaviors engage in minimal communication, it is difficult for behaviors to know enough about each other to coordinate.

There are good reasons for this kind of black-boxing. Making behaviors highly interrelated makes them harder to program, and makes it harder to add new behaviors to an already-built system. The image Brooks produces of being able to add new behaviors without making any changes to the old system is therefore highly attractive.

The question that faces us, then, is the following: *what is the minimum amount of de-modularization we can do and still have behavior blending work*? We will investigate this question by finding a small set of meta-level controls that will support the full range of behavior transition types listed here. It will turn out that, with the exception of the average behavior blend, the set we need is small, reasonable to implement, and useful for things besides transitions, as well. In particular, it will turn out that meta-level controls add to the expressiveness of behavior-based architectures in ways that will turn out to be crucial in Chapter 7 — by making explicit, and therefore expressible, the formerly implicit interactions between behaviors.

Meta-level controls to implement transitions

Transitions at their most basic work as glue between an old behavior and a new behavior. Generally, they need to know when the old behavior needs to be terminated, delete the old behavior, engage in some action, and then start the new behavior. This means, at a minimum, that transition behaviors need to have all the abilities of a regular behavior, and a few more: (1) they need to be able to know what other behaviors are running; (2) they need to be able to delete an old behavior; and (3) they need to be able to begin a new behavior.

These abilities to know about and affect other behaviors I call *meta-level controls*. Because meta-level controls are explicitly intended for communication and coordination between behaviors, they are in some sense a violation of the behavior-based principle of minimal behavioral interaction. Nevertheless, meta-level controls are so useful for coordinating behavior that several have already found a home in behavior-based architectures. An example is Hamsterdam's meta-level commands, which allow non-active behaviors to suggest actions for the currently dominant behavior to do on the side.

The Expressivator attempts to systematize this use of meta-level controls. The goal for the Expressivator is to find a small set of meta-level controls that will support the full range of transition types. This set of meta-level controls, then, provides a common framework under which transition types can be implemented and combined.

A stroll through the behavior transition types reveals the meta-level controls sufficient to implement all these transition types:

- *Parallel behavior blend*: The behaviors run simultaneously. This needs no meta-level controls. It is currently supported by behavior-based architectures.

- *Average behavior blend*: For the average behavior blend to work, all physical actions need to be averaged before they are sent to the agent's body. This requires re-routing the action commands that behaviors make through the transition behavior, which then averages them before sending them to the body.

- *Subroutine behavior blend*: The transition adds a subroutine to an already-running behavior. Transitions need to have the power to take some subbehaviors and add them to other behaviors.

- *Virtual behavior blend*: Transitions 'paralyze' one of the two behaviors being combined. Transitions need to be able to turn off muscles of a particular behavior.

- *Reductive behavior blend*: Transitions need to be able to change the internal variables that affect how other behaviors are processed in order to make one behavior reflect the addition of another.

- *Symbolic reduction*: The transition adds a subroutine to express a simple version of a behavior another already-running behavior. This can be done using the same techniques as subroutine behavior blend.

- *Principled subroutine behavior blend*: The transition makes a new behavior by combining either already-running or new subbehaviors. Transitions need to be able to construct new behaviors out of subpieces that already exist.
- *Interruption*: This is do-able in current agent architectures.
- *Sudden break*: This is also do-able in current agent architectures.
- *Off-screen transition*: This needs no special powers — the transition only needs to know if the agent is visible.
- *Accidental transition*: Transitions need to have access to a memory of previous behaviors and to be able to match patterns of behaviors against it.
- *Explanatory transition*: As above; delete the old behavior, do some action, and start the new behavior.
- *Unknown transition*: Transitions need to be able to tell that there are no other behaviors active, and fill this time in with default behavior.

Summing these needed controls up gives us a complete set of meta-level controls, which will allow transitions to be built on top of almost any behavior-based architecture. Transition behaviors need to be able to do the following:

1. to *query* which other behaviors have recently happened or are currently active,
2. to *delete* other behaviors,
3. to *add* new behaviors, not as subbehaviors of the transition, but at the top level of the agent,
4. to *add new sub-behaviors* to *other* behaviors,
5. to *change the internal variables* that affect the way in which other behaviors are processed (I call these "Communicative Features"),
6. to *turn off* a behavior's ability to send motor commands, and
7. to *move running subbehaviors* from one behavior to another.

The average behavior blend might be easy to implement in an architecture like Payton's or Perlin's that supported action blending. It turned out to be nearly impossible to do in Hap because of the way Hap divides action implementation (the level at which averaging should happen) from behaviors (the level at which the transition should be able to invoke the averaging). The more I thought about this transition type, though, the less sense it made to me. How often does it make semantic sense to combine high-level behaviors like "eat" and "sleep" by averaging their muscle commands? It is possible that someone more creative than me will come up with a good use for the average behavior blend, but on the surface it did not seem to warrant a great deal of architectural effort.

The implementation of these meta-level controls in the Expressivator and their relationships with other schemes for meta-level reasoning is discussed in more detail in section A.2 of the Appendix. This is a must-read for the technically oriented, but I did not want to torment the humanists any more than necessary.

Meta-Level Controls In General

I originally foresaw meta-level controls purely as a way to implement behavior transitions. It turns out, however, that they have interesting properties in themselves. Most fundamentally, meta-level controls provide support for building expressive, communicative agents because they make explicit — and therefore expressible — parts of the agent that were formerly implicit in the architecture.

Specifically, most behavior based systems treat individual behaviors as distinct entities which do not have access to each other. Conflicts and influences between behaviors are not handled by behaviors themselves but by underlying mechanisms within the architecture. Expressing the reasons for the behavioral decisions the agent has made is difficult, when, for instance, the agent decides what to do by reducing behavioral appropriateness to a number and then choosing the behavior with the highest numerical value. In these cases, the designer may not even be able to articulate why the agent does what it does, let alone the agent itself. *Because the mechanisms by which the agent decides what to do are part of the implicit architecture of the agent, they are not directly expressible to the user.*

Meta-level controls make the relationships between behaviors explicit, just as much a part of the agent design as the behaviors themselves. They allow behaviors, when necessary, to affect one another directly, rather than having inter-behavior effects be subtle side-effects of the agent design. Meta-level controls give the agent builder more power to expose the inner workings of the agent by letting them access and therefore express aspects of behavior processing that other systems leave implicit. Behaviors in this framework can check on and coordinate with each other, increasing their ability to create a coherent impression on the user.

Putting It All Together: The Expressivator In Action

Now that we have sign management and meta-level controls, behavior transitions between user-identified behaviors become easy to write. Here, I will give some examples of how transitions are implemented in the Patient of the Industrial Graveyard, to give a flavor for how the architecture is used in practice.

Single Transition Type

When the Patient is in trouble, the Overseer comes over to 'administer meds.' It does this by striking the Patient on the head, which causes it to collapse and turn off for a period of time. This is a virtual behavior blend, which is implemented as shown in the pseudo-code in Figure 5.15. The virtual behavior blend uses the 'paralyzing' meta-level control in order to allow emotional processing (particular with respect to fear of the Overseer) to continue, while overriding muscle commands so that

Transition behavior: *Anything to turned off*
Precondition: *Patient is struck by Overseer*

1. **Paralyze** all behaviors
2. Close eyes
3. Find a direction in which I can fall down without hitting anything
4. Collapse in that direction
5. **Add the 'turned_off' high-level behavior**

FIGURE 5.15: Example of a virtual behavior blend (meta-level controls are in bold)

the Patient appears passed out. The 'adding new behavior' meta-level control is used to start the 'turned_off' behavior when the transition is complete.

Combining Transition Types

In practice, I often found it useful to *combine* transition types. In my experience, meta-level controls provide a flexible framework in which those types can be combined to produce whatever transition makes the most sense for the current behavioral change. For example, the Patient has a 'reading' behavior, in which it appears to be reading the daily schedule of events in the Junkyard, and an 'exercise' behavior, in which it does aerobics. When the Patient is reading the schedule during exercise time and the Overseer menacingly approaches, the Patient should switch from reading to exercising. Rather than switching abruptly, the Patient shows its reaction to the Overseer and switches to a panicking version of exercising. As the Overseer goes away, the Patient calms down and the exercise behavior reverts to normal.

This is implemented using a mixture of meta-level controls as shown in Figure 5.16. This transition combines an explanatory changeover (the Patient is switching because it notices the Overseer) with a symbolic reduction (the shock of being caught by the Overseer is reduced to the gesture of looking at the Overseer) and a reductive behavior blend (the exercise behavior is modified by the "energy" Communicative Feature which is at first set high to reflect the Patient's shock at being caught reading by the Overseer, then diminished as the Overseer leaves).

The Story So Far

In this chapter, we have looked at transitions as a form of de-atomizing the user's perception of the agent. I introduced the idea of structuring

Transition behavior: *Reading to exercising*
Precondition: **Reading behavior is active**
Overseer has approached

1. **Delete reading behavior**
2. Look at Overseer
3. Look at sign
4. Show sudden shock reaction
5. Look at Overseer again
6. Do some quick, sloppy exercises
7. **Spawn** exercise behavior **with high energy**
8. **Add "Watch Overseer" subbehavior to exercise**
9. When Overseer leaves, **gradually reduce energy level of exercise**

FIGURE 5.16: Example of a behavior transition using meta-level controls to combine multiple transition types (meta-level controls are in bold).

agents according to the *signs* and *signifiers* they express instead of the physical actions and behaviors that reflect their internal structure. The agent keeps track of what has been communicated to the user by using the *sign management system*. I surveyed the range of behavior transition types one might want to support, and developed *meta-level controls* to support these transition types by allowing behaviors to refer to one another directly. These controls also allow the designer to express aspects of behavioral interrelationships by making explicit formerly implicit behavioral interactions.

At this point, you should be desperately wondering how these techniques actually worked out in practice. Initial results with them were good or bad, depending on your viewpoint. The transitions clearly reduced the apparent atomization of the agents. Since this was my goal for them, it seemed like I was well on the road to success. However, I did not need to do any fancy user studies to see that straightforward use of transitions per se did *not* improve the comprehensibility of the agent. The agent's behavioral changes were smooth and flowing, but remained just as enigmatic as before.

For example, two of the Patient's low-level signifiers are "watch the Overseer" and "glance around curiously." To change from watching the Overseer to glancing around, I tried using an alternating transition: interleave glances at the Overseer with glances around the junkyard, changing the proportion of glances from each behavior until the Patient was looking only around the junkyard. Clearly, this made the transition between the behaviors smooth; you could not tell when the "looking at

Overseer" behavior ended and the "glancing around" behavior began. On the other hand, you also could not tell *why* the agent was doing that sequence of glances. Watching the Patient, it seemed that its choice of what to look at was pretty arbitrary and not motivated by anything in its environment or, for that matter, in its personality. In fact, it *was* pretty arbitrary, but that was not supposed to be communicated!

In essence, transitions as a form of behavior blending means that the agent changes from randomly jumping between behaviors to randomly morphing between behaviors. While this is certainly less jarring — the user is not constantly notified of random changes by sudden radical changes in agent behavior — it does not fundamentally solve the problem that behavioral choice seems random, not a result of intentional thought. Nevertheless, it seems like transitions such as the guard dog example on page 105 really *should* be able to make the agent's behavioral choices clearer. Where did I go wrong?

For one thing, merely hiding the agent's inadequacies from the user is not enough. The goal for our agents is to be understandable as intentional beings to their audience, for whom these agents should be, according to my own philosophy, explicitly designed. But so far, I have been treating this audience as a bunch of TV-watching couch potatoes who just need to be insulated from the sticky details of agent implementation. That is to say, so far, I have been using transitions merely to hide the agent's atomization from the user, who is seen as a passive observer of the agent's behavior.

In my own defense, I would like to note that I was merely following a grand tradition of post-Eliza AI.[9] Eliza is an extremely simple program intended as a study in natural language communication. It plays the part of a Rogerian psychoanalyst, and basically repeats everything the user says in the form of a question [Weizenbaum, 1965]. To the shock of its programmer and indeed much of the AI community, who knew that Eliza was little more than a language recording and playback device, human users often imputed extraordinary intelligence to Eliza, treating it as a human confidant. The conclusion that many AIers drew from this incident is that human perception of the intelligence of agents is a wildly inaccurate measure of their actual intelligence.

Unfortunately, though, many AI researchers unconsciously go a step further. They conclude that if Eliza's apparent intelligence is a result of a few simple measures, then *any* attempt to be comprehensible to the audience probably merely involves a bunch of 'tricks' that hide the actual stupidity of the agent from the naive and gullible common masses. I must confess shame-facedly that my use of transitions to hide atomization is simply a slightly subtler extension of this attitude. In general, the result of 'Eliza backlash' is research strategies which focus solely on internal or functional aspects of the agent, ones that can be demonstrated to show intelligence without reference to user interpretation. In the end, the user as an active constructor of understanding of the agents is forgotten.

But this minimization of audience involvement is bad for AI because it hinders the development of creatures that truly appear intentional. It turns out, as generations of psychologists, literary critics, and artists

[9] This is another example of the incredible ability of AI Doctrine to hijack my mind despite my explicit anti-Doctrine philosophy

understand, that audiences are not merely passive. They are *actively* constructing understandings of the intentional and pseudo-intentional beings they encounter. Hiding the things that hinder this construction is good; but even better would be *providing tools that support the user in his or her attempt to find meaning in what the agent is doing*. This does not mean cheap tricks that make the agent falsely seem intentional, but support for the user to understand specifically the impression of agenthood (including goals, decision-making processes, thoughts, and feelings) that the designer is trying to get across. The development of architectural mechanisms that support user interpretation will be the technical goal for the rest of the thesis.

In order to support user interpretation, we first need to have a better understanding of how users come to interpret intentional behavior in the first place. We will spend Intermezzo II looking at a case study of how animators use transitions to make their characters come alive. Chapter 6 will look at how the humanities and psychology describe the construction of knowledge about intentional beings. We will use these two sources to figure out how transitions can be used not just to hide the flaws of atomized agents, but to actively support the user's perception of them as intentional beings in the way the designer intends. It will turn out that with a little re-thinking of the nature of transitions, the mechanisms developed in this chapter — signs, signifiers, and sign management; transitions; and meta-level controls — are not only useful for behavior blending, but can also be used to support user interpretation in the way I have described here. I will build on the technical mechanisms introduced in this chapter in the full development and evaluation of the Expressivator in Chapter 7.

Intermezzo II

Luxo, Jr.: A Case Study of Transitions in Animation

In Chapter 5, I pushed the technical understanding of schizophrenia as far as it would go. The result was some interesting technology that helped to reduce schizophrenia, while miraculously avoiding making agents seem more intentional. You should note that this handily solves the technical problem, but manages to do it while ignoring or even subverting the big picture that motivated the technical problem.[1]

Let's take a moment to go back to the basics. The dream is to be able to create artificial creatures, whether built as robots or rendered by computer graphics, that are not merely smart but really seem alive and intentional. These agents would come to life like characters in a novel or film, that, although human creations, seem to have a life of their own. Although we know they are in some sense 'fabrications,' we listen to them, sympathize with them, laugh at them, hate them, fall in love with them, without a sense of being deluded. Their concerns, worries, and life dilemmas are not simply factual; they are at times ridiculous, at times meaningful, but always to be interpreted within the full context of human life.

What would such artificial creatures look like? One way of finding out is to do a thought experiment. We already know that such creatures can be generated, not by an AI program, but by a character animator. What if we pretend that this animation is actually the result of a behavior-based AI program? Could we reverse-engineer the program that generated it?

The idea that character animators have something to tell computer scientists about how to build agents is not novel. This idea has already been explored by several AI researchers starting with Joseph Bates [Bates, 1994]. In this Intermezzo, I will add to this tradition by looking at a particular animated sequence as though it were generated by an AI program, and then imagine how behaviors and transitions were used to create the feeling that the character is really intentional. How are the different 'behaviors' of the 'agents' connected? How do these connections help to make the agent alive?

Clearly, it is unlikely that animators actually think in terms of 'behaviors' and 'transitions,' as an AI researcher would. Nevertheless, we can

[1] This is perhaps a larger (though unwanted) tradition in science.

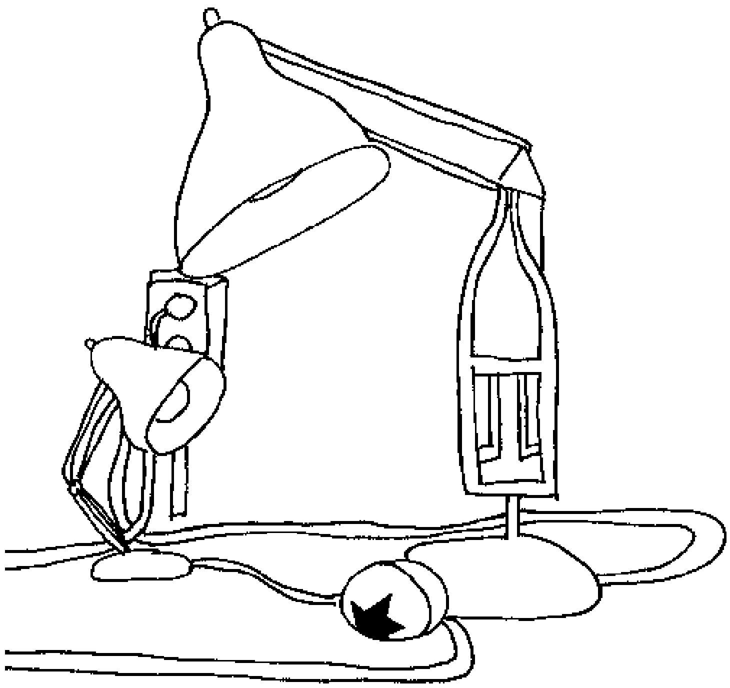


FIGURE II.1: Luxos Senior and Junior (artist's rendition)

learn something by provisionally viewing animation through the lens of AI architecturology. It turns out that animation brings an interesting, new perspective to the table, in the ways that it both is and is not adequately described by the behavioral metaphor.

Introduction to *Luxo, Jr.*

> "Whether it is generated by hand or by computer, the first goal of the animator is to entertain. The animator must have two things: a clear concept of what will entertain the audience; and the tools and skills to put those ideas across clearly and unambiguously. Tools, in the sense of hardware and software, are simply not enough. The principles discussed in this paper, so useful in producing 50 years of rich entertainment, are tools as well... tools which are just as important as the computers we work with." ([Lasseter, 1987], 43)

The animation we will be looking at is John Lasseter's short film *Luxo, Jr.*[Pixar, 1986], an artist's rendition of which appears in Figure II.1.[2] This film was one of the first computer animations to focus on developing character and intentionality, rather than on creating mechanical photorealism. Lasseter's explicit goal is to use traditional (hand-drawn) animation techniques to communicate personality, emotion, and intentionality clearly through his computer-generated images [Lasseter, 1987]. The success of *Luxo* and subsequent films such as *Toy Story* suggests that he has been effective.

The movie itself centers on two characters, Luxos Junior and Senior, and a ball. Luxo Junior comes on stage, playing with the ball. After some time, the ball breaks. Luxo Junior is at first disappointed, but soon finds a new ball. Despite (or perhaps because of) the utter simplicity of the plot, the characters are strongly portrayed, clearly emotional and intentional, and fun to watch.

[2]Permission to use an actual still from the film was not given.

You may note a striking family resemblance between the Luxos and the Patient in the Industrial Graveyard.[3] More importantly, the relatively simple structure of the lamps in *Luxo*, the simplicity of its plot and the agents' behavior, the absence of natural language, and the fact that it is all rendered by computer mean that, perhaps, the goal of automatically generating similarly affective characters is not entirely implausible, though perhaps far beyond the state of the art. Let's imagine that they *are* created by a behavior- and transition-based architecture. What can this tell us about how transitions work?

Luxos As AI Agents

Detailed analysis of behaviors and transitions in *Luxo* can be found in Appendix B. The general trend is that *agents communicate what they will do before they do it*. This means they stop whatever they are doing and engage in some pre-behavior activity to tell you what they are going to do next. This use of transition corresponds to the animation technique of anticipation.

> Anticipation is... a device to catch the audience's eye, to prepare them for the next movement and lead them to expect it before it actually occurs. Anticipation is often used to explain what the following action is going to be. Before a character reaches to grab an object, he first raises his arms as he stares at the article, broadcasting the fact that he is going to do something with that particular object. The anticipitary moves may not show *why* he is doing something, but there is no question about *what* he is going to do next. ([Lasseter, 1987], 38)

This is different from the default transition theory of Chapter 5. There, we used transitions to blend together two behaviors. In this mindset, the important thing is to finish up the old behavior cleanly and begin the new behavior in an unobtrusive way. But with Luxos the old behavior is at least somewhat irrelevant. The point of transitions here is that the character must do some communication before it starts a behavior. This communication tells the audience that the Luxo has made a decision to do something different, as well as letting the audience know how the behaviors interrelate.

Transitions communicate a variety of such relationships in *Luxo, Jr.*:

- That didn't work; I have a new idea.
- Hey, what just happened?
- Oh no! Let me get out of here!
- I wonder what *that* will do?

These various relationships are largely communicated through a small set of basic tools.

[3] The Patient is for this reason sometimes nicknamed "Lixo," or, in moments of hacking frustration, "Suxo."

- *Eye movement* — This is probably the single most important way Luxos communicate behavioral transition. They stop, look at what they are going to do, and then do it. The moment of looking is important as it communicates that the lamp is making a decision.

  Just watching by itself can also be a behavioral transition. As a default, if the character does not know what to do, it can just watch what is going on. When something new has caught the character's attention, it can change to a behavior involving that object.

- *Behavioral blend* — Where behaviors correspond to movements, the two behaviors can be blended using low-level action blending techniques like those presented in the previous chapter. For example, when Luxo moves from standing still to examining an item, it starts out very slowly (almost still), then gradually speeds up. When Luxo moves from sighing to hopping, it does sad, sighing hops. In these cases, the animator has found a defining characteristic of one behavior, and blends the behaviors by applying that characteristic to the other behavior.

  Again, what is important here is not to blend the behaviors per se but the impression of that behavior on the user. If some behaviors can be fundamentally defined rather simply, it will be easy to mix them in with other ones. You are not including the whole behavior, but an *image* of it.

- *Alternation* — At times, Luxo transitions between behaviors by switching between parts of them. For example, when Luxo Senior switches from watching the ball to watching Luxo Junior, it alternates glances between them.

- *Shock reaction* — a common transition. The agent engages in some behavior, then shows a shock reaction to something in the environment and switches to a different behavior. This shows clearly that the agent is reacting to something unexpected rather than just changing on a whim.

- *Shared object* — Often the old and new behaviors share an object of interest. Transitions are frequently predicated on external objects upon which the character focuses during a transition. This makes the transition clearly not internal or arbitrary, but a reaction to observable events.

- *Off-screen* and/or *non-individualistic* — At times Luxo will switch behaviors off-screen. Here the change in behavior will be reflected in the reaction of the character left on the screen. This means not all behavioral transitions take place in the creature him/herself — some transitions are communicated by the reaction of the other character. This in turn implies that transitions are not just about individual behavior, but (at least in Luxo) are important in terms of story — they are about advancing the story, and can therefore appear in either character. Additional support for this is in the fact that Junior and Senior generally do not transition at the same time — while we are watching Junior play Senior just stays in one (simple background) behavior.

In general, unlike the transitions of Chapter 5, in Luxo most transitions are not 'internal' or 'arbitrary.' They are *reactions* to observable events: a result of a previous action or something another creature did. Transitions relate the events of the story to one another by expressing the relationships between behaviors and explaining why the creature is moving from one activity to the next. They help the audience to understand what Luxo is doing by anticipating and explaining the reasons for the behavior in which it engages.

Luxos Exceed AI

While viewing *Luxo* through the lens of behaviors and transitions is illuminating, there are clearly some ways in which this paradigm does not do justice to the film. These areas point to some fundamental limitations in the behavior/transition metaphor. These limitations are not all addressed in this thesis, but are mentioned here to provide a roadmap to changes which may need to be made to generate truly expressive agents.

Inadequacies of Behaviorism

The first step in analyzing Luxo's transitions is to identify the behaviors and transitions Luxo uses. But a number of behaviors cannot be classified easily as 'behavior' or 'transition'. The most obvious one is 'watch,' in which Senior engages for much of the film. 'Watch' is a transition because it fills in spaces between activities, telling you what Senior is thinking about and deciding to do. It is also a behavior because it is so long, and because it really seems to be an activity in and of itself.

In addition, some 'behaviors' seem to exist only in a transitory phase. A good example of this is when Junior hops on stage for the first time, playing with the ball. It then spends some time alternating between looking at Senior and looking at the ball (the 'transition'), and hops off stage to go play with the ball again (the 'original behavior').

More fundamentally, while you could provisionally call much Luxo activity "behaviors," Luxo's behaviors are clearly different from behavior in the behavior-based sense. For example, Luxo's behaviors are not repeatable; when he engages in the 'same' behavior twice it is often quite different in its presentation and context. It seems inaccurate to call them "really" or "fundamentally" the same thing.

In general, AI-style behaviors carry with them a load of intellectual baggage that animators do not seem to want.

- For AI researchers, a behavior fundamentally *is* the name or concept of the behavior. For animators, behavior is movement that may or may not be described with a particular name; while this name may repeat ("doing the same behavior twice"), the action itself may not.

- For AI researchers, an agent moves from behavior to behavior, and is always running at least one. For animators, an agent is always engaging in movement, which is interpretable as an activity, or shows the agent's emotions, or reveals the agent's motivations, or...

Fundamentally, for AI researchers, the behavioral level is reality, with the actions a surface impression of this deeper level. For animators, the actions are reality, with the behaviors an abstract description of the real. AI thought seeks out the deep structure of agent action and finds it in behaviors; animation thought seeks out the clear communication of ideas and finds it in all the details of character movement.

Inadequacies of Transitionism

If behaviors are not completely adequate to understanding Luxo's activity, then it should come as little surprise that transitions are not, either. The non-individualistic transitions mentioned on page 136 are one interesting way in which the idea of behavior transitions needs to be 'bent' in order to fit Luxo behavior. Behavior transitions as conceptualized so far have resided purely in one individual, i.e. one behavior changing to another in a clear way. Non-individualistic transitions expand this notion for when a group of agents is meant to have a cumulative effect, rather than focusing on each individual agent.

Non-individualistic transitions, by exceeding the definition of transitions made in the last chapter, reveal an inadequate assumption underlying this definition. This assumption, which comes from the behavior-based AI tradition, is that all behavior is somehow fundamental to the individual, rather than to the group to which the individual belongs. In this tradition, even multi-agent systems that engage in group coordination tend to work by figuring out how to program the individual agents so that the correct global behavior emerges from local interactions based on local knowledge. In contrast, for animation, the *story* is fundamental; the characters are secondary. The decision of which behavior a character should present is not based primarily on its plausibility for that character but on how it fits into the overall plot.

This suggest that animation has a fundamentally different understanding of the relationships between the parts and the whole. In AI, the 'parts' (agents) are primary, with the whole being the simple sum of the parts. This corresponds exactly with the whole agent being the simple sum of the individual behaviors. In animation, on the other hand, the 'whole' is primary, with the 'parts' (characters) being instantiations of and motivated for the whole. This different way of conceptualizing the relationship between part and whole is a fundamental difference between humanistic and scientific worldviews. It will become key in Chapter 6.

Transitions from an Animator's Perspective

These differences between the AI and animation worldviews suggest that someone trained in animation may come to quite different conclusions about how the idea of transitions applies to *Luxo, Jr.*. To fill out this analysis, I asked a professional animator, Steve Curcuru, to do an informal analysis of Luxo in parallel with mine [Curcuru,]. Since Curcuru had at that point not yet become infected with any knowledge of behavior-based AI, his impressions are based on how an animator might think of behaviors and transitions, and are therefore, unsurprisingly, quite different from mine.

In general, Curcuru focuses much more on the actual form and structure of Luxo's motion, whereas I — with my AI intellectual baggage firmly in tow — tend to focus on what the character is 'fundamentally' doing. Curcuru therefore, unlike me, tends to find transitions within the details of Luxo's movements. For example, Curcuru points out that a number of times, Luxo 'settles' from one behavior to another. That is, if a new behavior is relatively static in terms of motion (e.g. Senior looking off-screen), the character will slowly move from the position of the old behavior into its new final position. Also, Curcuru describes how a quick motion *contrasts* with a slow motion the character just engaged in, and that this contrast is essential to understanding what the character was doing (a change in thought). This suggests that the relationship between the agent's behaviors may be more complicated that a simple transition that can be inserted between them; in these cases it is a relation between the *forms* of the two behaviors.

Curcuru additionally makes clear that the idea that behavior is intended to communicate permeates not only transitions but also behaviors themselves. He identifies many aspects of Luxo's behavior that are there simply to show what the agent is thinking. He mentions two major tools in particular for showing what a character is thinking: anticipations and holds. Anticipations may be helpful to get the audience to understand what the character is doing and / or to make the agent seem more intentional. Holds are used to depict that the character is thinking.

Curcuru believes that transitions are fundamentally there to show *why* the character's behavior changes. He describes Luxo's transitions as having the following general form:

1. The character does something.
2. Hold; the character must be thinking about something.
3. The character does something different; hence, it must have changed its mind.

During the transition, the character shows that it is considering something, usually an observable object or event. When the behavior changes, the audience assumes that the change is due to this moment of thought.

The fundamental insight from Curcuru's analysis is that *transitions show that the character is making behavioral changes reflectively, rather than reflexively*. The character is not instinctively or arbitrarily moving from action to action, but is considering what it does. Transitions allow the animator to make clear that the character is noticing the world around it and reacting to it in its own idiosyncratic ways. Done well, this thoughtful interactivity makes the character come alive.

Lessons Learned from Luxo

This analysis of Luxo leads to some new conclusions about transitions. In Chapter 5, transitions were intended to make sure that behavior change is not abrupt. The idea was that, if abrupt and sudden behavior switching is confusing to the user, then we should disguise the behavioral change so the user does not notice it.

But this analysis of Luxo shows that transitions do not merely blend together behaviors in a seamless whole. In Luxo, transitions are needed, not to hide behavioral change, but to set the stage for *new* behavior. They prepare the audience for the new behavior by anticipating it and by showing the reasons for the switch. They let the agent unmistakably show that its behavior is affected by what is happening around it.

For Luxo, transitions are about the *reasons* for behavior change. They show why the agent is moving from activity to activity. Transitions show that the agent is making its behavioral choices reflectively, not instinctually, by revealing the agent's thinking processes. They are therefore essential to giving the agent the aura of being a conscious being, rather than an automaton.

Luxo shows that transitions are intended to communicate; and *so are behaviors*. Many aspects of Luxo's behaviors are there purely to show what Luxo is thinking. Therefore, design choices that let transitions communicate better may also be useful for improving regular behaviors. Disciplined use of transitions and the architectural mechanisms that support them may help make *all* behavior clearer, not just the behaviors that are directly related with transitions.

Chapter 6

Narrative Intelligence

At this point, let's take a moment to review where we are. In Chapter 2, we started with the problem of incoherence and mechanicity in autonomous agents. As agent builders combine more and more behavior, the overall activity of the agents tends to degrade into a jumping-around between separately defined behaviors, a problem we have been calling *schizophrenia*.

In Chapters 3 and 4, we looked at schizophrenia and AI in culture, suggesting that one way of addressing the problems of schizophrenia is by looking at agents in their social and cultural context. This motivated us to redefine the problem of schizophrenia in Chapter 5 in terms of the user's perception. Instead of asking "how can an agent be coherent?" we ask, "how can an agent appear coherent to the user?"

This reformulation suggests that we should use transitions to smooth between behaviors, i.e. to *hide* the breaks between behaviors from the user. However, as we noted at the end of Chapter 5, transitions as de-atomization do not really address the fundamental problem of schizophrenia. They do hide the breaks between behaviors, but they do not do so in a way that makes the agent seem any more intentional.

In Intermezzo II, we saw that character animators have a fundamentally different way of thinking about transitions between behaviors. Instead of using transitions to hide or smooth over a behavioral change, transitions are used to help the user understand the *reasons* for behavioral change. Far from hiding behavioral switches, transitions call attention to them, but they do so in such a way that they help the viewer to figure out what the agent is doing. Transitions are one tool among many that animators use in order to send cues to the viewer about how they should interpret the character.

This animation viewpoint suggests that we have been looking at the problem of agent construction from the wrong end. Rather than focusing on the agent — "how can we fix the agent so that the user will not notice it is actually incoherent?" — we should focus on the *user*. This leads to yet another re-formulation of the problem statement: "*how can we support the user in constructing coherent interpretations of the agent?*"

This will be the final reformulation of the problem statement, leading to the full-blown Expressivator architecture of Chapter 7. In this chapter, we will explore the ramifications of thinking about the problem in this way. First, we will review how this problem statement diverges from traditional AI thinking. Then, we will find out how people interpret intentional behavior through a foray into narrative psychology. Finally, I will present principles of *Narrative Intelligence*, i.e. a way of designing agents to support the user's construction of narrative interpretations of their behavior.

Interpretation and Agent Transparency

Many AI systems that try to make agent behavior clear are based on what I call the Agent Transparency Assumption. In AI, the agent is not thought of as the viewer's interpretation, but exists 'objectively', i.e. is an independent object to be observed passively. The agent is thought to primarily exist on its own, with its presentation an afterthought. Therefore, in AI communication of the 'idea' of the agent is thought to be best achieved, not by tailoring the visible presentation of the agent towards particular interpretations, but by allowing the user to see what the agent is 'actually' doing. Here 'actual' means 'in the agent's code' — i.e., the way the designer thinks of the agent. Blumberg, for example, defines the expression of intentionality as "allow[ing] the observer to understand on which goal the system is working, how it views its own progress towards that goal, and what the system is likely to do next" ([Blumberg, 1996], 25). For AI, the character actually, independently exists — as represented in the body of its code — and the interpretation of the viewer is not a creative act but a passive observation or correctable reconstruction of the agent's code.

But even in AI, users are involved in a complex process of interpretation of the agent's behavior. This is because the user's view of the agent is quite different from the designer's. Agent designers tend to think of agents in terms of the code we use to write them. We choose particular goals, emotions, or plans for the agent, and when we watch the agent, we interpret its activity according to those components. We are on the lookout for the behaviors and emotions we know it must have, since we put them in the code.

However, users who do not know how the agent was designed do not have the internal structure of the agent as a resource in interpreting the agent's activity. *All the user can go on is the agent's physical actions*. The agent's "actual" structure (goals, behaviors, and so on) must be inferred from the movements the user sees the agent use.

Given this relative poverty of information, *it is amazing users understand agents at all*! It is fairly incredible that, when users observe two spheres with eyeballs flattening and moving towards each other, they quite frequently say "hey! look! They're getting into a fight!" Extremely simple physical cues often lead users to infer complex motives and behavior that may or may not be warranted by the code running the agent.

Viewers' understanding of agents is grounded in the fact that people are fundamentally social creatures, specialized in understanding intentional behavior.[1] When people watch our agents, they bring with them

[1] The degree to which this is true can be understood by looking at the great handicaps

sophisticated capacities of interpretation which often allow them to infer or "read in" the intentional behavior we would like them to see in our creatures — even despite the obstacles we agent builders often put in their way! Even when people happen to reconstruct understandings of our agents that correspond to what we designed into their code, they are in no sense passively observing; it is always an active reconstruction.

In AI, we generally feel that this process of interpretation is somewhat dubious. Instead of encouraging the user to interpret to his or her heart's content, we try to ground the user's interpretation of the agent in the 'actual' agent, i.e. its code. That is, we try *to make the user look at the agent in the same way the designer does*. The major problem with this strategy is that it is counterproductive. On the one hand, users are used to interpreting creatures' behavior, and they will resist attempts to 'see' in ways that are different from what they are used to. On the other, users are extremely good at interpreting creatures' behavior, so we are wasting their talents.

Animation suggests a different strategy: maybe we should *try to make the designer look at the agent in the same way the user does*. The animation viewpoint suggests that rather than throwing out this interpretive ability by getting users to simply 'see' the code, we can make our creatures appear maximally intentional by *supporting users in their ongoing drive to interpret the agent as an intentional creature*. That is, we can construct agents so that they give off cues that are easy for users to understand as coherent, intentional behavior.

One way of understanding this reformulation is to go back to the very concept of agent. As mentioned in Chapter 1, we use the notion of 'agent' when we think that it is helpful, informative, or good P.R. to think of our programs as self-contained individuals. The term agent has become mind-numbingly popular recently and has been substantially diluted in the past several years, so that now people use the word 'agent' almost interchangeably with the word 'program' or 'engineered artefact' (as in, "I used my remote control agent to turn on my TV agent"). Autonomous agent researchers such as myself have felt alternately encroached upon and far superior to the competitition, since our usage of the term 'agent' — to refer to a computer-controlled character or artificial creature roughly analogous to living agents — seems to be one of the few actually meaningful uses of the term.

Here I would like to suggest that, despite the moral high ground autonomous agent researchers occupy in this respect, *the usage of the agent metaphor for autonomous agents may actually be unhelpful*. As discussed in Chapter 4, thinking about our programs as 'agents' implies that they are autonomous and self-contained, and that communication of agent activity to users consists of the apperception by external people of an independently and objectively existing object. Animation and narrative psychology suggests that for applications where human comprehension of our agents is essential, it may be more helpful to think of autonomous agents *as narrative*. This implies that an agent is not self-contained, but exists through a process of communication. An agent-as-narrative has an author and an audience, exists in a context that affects how it is understood, and comes to life only in so far as it is adequately communicated

that autistic people face in our society.

to the audience.

In this chapter, we will explore what it means for agents to be structured and communicated as narrative. In the next section, we will start by looking at narrative psychology, which studies how people interpret specifically intentional behavior. Narrative psychology suggests that this process of creating narrative is the fundamental difference between the way people understand intentional beings and mechanical artefacts. This implies that by structuring our agents as narrative, we can make it more natural for people to understand our agents as comprehensible, intentional beings. I will therefore discuss how agents can be built according to the principles of narrative. This forms a style of agent-building I term *Narrative Intelligence*, in which agents give off visual behavioral cues that are easy to assimilate into narrative.

Principles of Narrative Psychology
or How We (Sometimes) Make Sense of Creatures

Artificial Intelligence attempts to generate intentional creatures by blurring the distinction between biological, living beings and automatic processes of the kind that can be run on computers. That is, AI agents should ideally be understandable both as well-specified physical objects and as sentient creatures. But it turns out that human understanding of the behavior of humans and other conscious beings differs in important ways from the way we understand the behavior of such physical objects as toasters. Identifying the distinction between these two styles of comprehension is essential for discovering how to build creatures that are understandable not just as helpful tools but as living beings.

The way people understand meaningful human activity is the subject of narrative psychology, an area of study developed by Jerome Bruner [Bruner, 1986] [Bruner, 1990]. Narrative psychology shows that, whereas people tend to understand inanimate objects in terms of cause-effect rules and by using logical reasoning, intentional behavior is made comprehensible by structuring it into narrative or 'stories.' We find structure, not by simply observing it in the person's activity, but through a sophisticated process of interpretation. This interpretation involves finding relations between what the person does from moment to moment, speculating about what the person thinks and feels about his or her activity, and understanding how the person's behavior relates to his or her physical, social, and behavioral context.

Even non-experts can effortlessly create sophisticated interpretations of minimal behavioral and verbal cues. In fact, such interpretation is so natural to us that when the cues to create narrative are missing, people spend substantial time and effort trying to come up with possible explanations. This process can be seen in action when users try to understand our currently relatively incomprehensible agents!

This sometimes breathtaking ability — and compulsion — of the user to understand behavior by constructing narrative may provide the key to building agents that truly appear alive. *If humans understand intentional behavior by organizing it into narrative, then our agents will be more 'intentionally comprehensible' if they provide narrative cues.* That is to say, rather than simply presenting intelligent actions, agents should give visible cues that support users in their ongoing mission to generate narrative explanation of an agent's activity. We can do this by organizing our agents so that their behavior provides the visible markers of narrative. The remainder of this chapter presents the properties of narrative and explains how they can be applied to agent construction.

Prolegomena to a Future Narrative Intelligence

There has recently been a groundswell of interest in narrative in AI and human-computer interaction (HCI). Narrative techniques have been used for applications from automatic camera control for interactive fiction [Galyean, 1995] to story generation [Elliott *et al.*, 1998]. Abbe Don and Brenda Laurel argue that, since humans organize and understand their experiences in terms of narrative, computer interfaces should be organized as narrative, too [Don, 1990] [Laurel, 1991] [Laurel, 1986]. Similarly, Kerstin Dautenhahn and Chrystopher Nehaniv argue that robots may be able to use narrative in the form of autobiography to understand both themselves and each other [Dautenhahn and Nehaniv, 1998].

The term Narrative Intelligence has been used by an informal group at the MIT Media Lab to describe this conjunction of narrative and Artificial Intelligence. It is also used by David Blair and Tom Meyer to refer to the human ability to organize information into narrative [Blair and Meyer, 1997]. Here, I want to suggest that Narrative Intelligence can be understood as the confluence of these two uses: *that artificial agents can be designed to produce narratively comprehensible behavior by structuring their visible activity in ways that make it easy for humans to create narrative explanations of them.*

In order to do this, we need to have a clear understanding of how narrative works. Fortunately, the properties of narrative have been extensively studied by humanists. Bruner (nonexhaustively) lists the following properties [Bruner, 1991]:

- *Narrative Diachronicity*: Narratives do not focus on events moment-by-moment, but on how they relate over time.
- *Particularity*: Narratives are about particular individuals and particular events.
- *Intentional State Entailment*: When people are acting in a narrative, the important part is not what the people do, but how they think and feel about what they do.
- *Hermeneutic Composability*: Just as a narrative comes to life from the actions of which it is composed, those actions are understood with respect to how they fit into the narrative as a whole. Neither can be understood completely without the other. Hence, understanding narrative requires interpretation in a gradual and dialectical process of understanding.
- *Canonicity and Breach*: Narrative gets its 'point' when expectations are breached. There is a tension in narrative between what we expect to happen, and what actually happens.

- *Genericness*: Narratives are understood with respect to genre expectations, which we pick up from our culture.
- *Referentiality*: Narratives are not about finding the absolute truth of a situation; they are about putting events into an order that feels right.
- *Normativeness*: Narratives depend strongly on the audience's conventional expectations about plot and behavior.
- *Context Sensitivity and Negotiability*: Narrative is not 'in' the thing being understood; it is generated through a complex negotiation between reader and text.
- *Narrative Accrual*: Multiple narratives combine to form, not one coherent story, but a tradition or culture.

While these properties are not meant to be the final story on narrative, they stake out the narrative landscape. Taking narrative agents seriously means understanding how these properties can influence agent design. It will turn out that current AI techniques, which largely inherit their methodology from the sciences and engineering, often undermine or contradict the more humanist properties of narrative. Here, I will explain problems with current agent-building techniques, techniques already in use that are more amenable to narrative, and potential practices that could be more friendly to the goal of meaningful Narrative Intelligence. This will form the theory or philosophy of Narrative Intelligence; its technical manifestation will rear its head in the next chapter.

One note of caution: the goal here is to interpret the properties of narrative with respect to agent-building. This interpretation is itself narrative. Since, as we will see below, the nature of narrative truth is different from that of scientific factuality, this essay should not be read in the typically scientific sense of stating the absolute truth about how narrative informs AI. Rather, I will look at the properties of narrative in the context of current AI research, looking for insights that might help us to understand better what we are doing better and suggest (rather than insist on) new directions.

1.Narrative Diachronicity

The most basic property of narrative is its diachronicity: a narrative relates events *over time*. Events are not understood in terms of their moment-by-moment significance, but in terms of how they relate to one another as events unfold. For example, if Fred has an argument and then kicks the cat, we tend to infer that the cat-kicking is a result of his frustration at the argument. When people observe agents, they do not just care about what the agent is doing; they want to understand the relations between the agent's actions at various points in time. These perceived relations play an important role in how an agent's subsequent actions are understood. This means that, to be properly understood, it is important for agents to express their actions so that the intended relationships are clear.

However, as described in Chapter 2, it is currently fashionable to design behavior-based autonomous agents using action-selection, an agent-building technique that ignores the diachronic structure of behavior.

Action-selection algorithms work by continuously redeciding the best action the agent can take in order to fulfill its goals [Maes, 1989a]. Because action-selection involves constantly redeciding the agent's actions based on what is currently optimal, there is no common thread structuring the actions that are chosen into understandable sequences — this fact is simply schizophrenia rephrased. Schizophrenia undermines the appearance of intentionality because agent action seems to be organized arbitrarily over time, or, at maximum, in terms of automatic stimulus-response.[2]

More generally, as mentioned in Chapter 5, expressing the relationships between behaviors is not well-supported in most behavior-based systems (a complaint also raised in [Neal Reilly, 1996]). While these architectures do provide support for clear, expressive *individual* behaviors, they have problems when it comes to expressing relations *between* behaviors. This is because a typical behavior-based system (e.g. [Blumberg, 1994] [Brooks, 1986a] [Maes, 1989b]) treats each behavior separately; behaviors should refer as little as possible to other behaviors. Because of this design choice, a behavior, when turned on, does not know why it is turned on, who was turned on before it, or even who else is on at the same time. It knows only that its preconditions must have been met, but it does not know what other behaviors are possible and why it was chosen instead of them. In most behavior-based architectures, behaviors simply do not know enough about other behaviors to be able to express their interrelationships to the user.

In this light, classical AI would seem to have an advantage over alternative AI, since it is explicitly interested in generating structured behavior through such mechanisms as scripts and hierarchical plans. However, classical AI runs into similar trouble with *its* modular boundaries, which occur not between behaviors but between the agent's functionalities; for example, the agent may say a word it cannot understand. Fundamentally, agent-building techniques from Marvin Minsky's Society of Mind [Minsky, 1988] to standard behavior-based agent-building [Maes, 1991] to the decomposition of classical agents into, for example, a planner, a natural language system, and perception [Vere and Bickmore, 1990] are all based on divide-and-conquer approaches to agenthood. Being good computer scientists, one of the goals of AI researchers is to come up with modular solutions that can be easily engineered. While some amount of atomization is necessary to build an engineered system, narrative intentionality is undermined when the parts of the agent are designed so separately that they are visibly disjoint in the behavior of the agent. Schizophrenia is an example of this problem, since when behaviors are designed separately the agent's overall activity is reduced to a seemingly pointless jumping around between behaviors. Bryan Loyall similarly points out that visi-

[2]This is unfortunate, since the original idea of constantly redeciding behavior came in work explicitly interested in diachronic structure. Philip Agre and David Chapman focus, not on the design of the agent per se, but on the ongoing dynamics of the agent and the environment [Agre and Chapman, 1987]. The goal is to construct action-selection so that, when put in a particular environment, agents *will* tend to have particular diachronic structure in their behavior. Continuous redecision is part of this work because it keeps the agent's actions closely tied to the agent's context, a property that is also important for narrative, as we will see below. However, the concept of the action-selection algorithm itself tends to undermine diachronic structure, especially when it is used for agent — rather than dynamic — design.

ble module boundaries destroy the appearance of aliveness in believable agents [Loyall, 1997a].

The end result is that the seductive goal of the plug-n-play agent — built from the simple composition of arbitrary parts — may be deeply incompatible with intentionality. Architectures like that of Steels [Steels, 1994], which design behaviors in a deeply intertwined way, make the agent design process more difficult, but may have a better shot at generating the complexity and nonmodularity of organic behavior. In Chapter 7, we will try a less drastic solution, using transition sequences to relate separately designed behaviors.

2.Particularity

Narratives are not simply abstract events; they are always particular. "Boy-meets-girl, boy-loses-girl" is not a narrative; it is the structure for a narrative, which must always involve a particular boy, a particular girl, a particular way of meeting, a particular way of losing. These details bring the story to life. However, details do not by themselves make a narrative either; the 'abstract structure' the details can be ordered into brings meaning to the details themselves. A narrative must be understood in terms of tension between the particular details and the abstract categories they refer to; without either of these, it is meaningless.

This same tension between the abstract and the particular can be found in agent architectures. Agent designers tend to think about what the agent is doing in terms of abstract categories: the agent is eating, hunting, sleeping, etc. However, users who are interacting with the agent do not see the abstract categories; they only see the physical movements in which the agent is engaged. The challenge for the designer is to make the agent so that the user can (1) recognize the particular details of the agent's actions and (2) generalize to the abstract categories of behavior, goal, or emotion that motivated those details. Only with a full understanding at both the particular and the abstract levels will the user be likely to see the creature as the living being the designer is trying to create.

But AI researchers are hampered in this full elucidation of the dialectical relationship between the particular and the abstract by the valorization of the abstract in computer science. As mentioned in Intermezzo II, in AI we tend to think of the agent's behaviors or plans as what the agent is 'really' doing, with the particular details of movement being a pesky detail to be worked out later. In fact, most designers of agents do not concern themselves with the actual working out of the details of movement or action at all. Instead, they stop at the abstract level of behavior selection, reducing the full complexity of physical behavior to an enumeration of behavior names. Maes, for example, uses abstract atomic actions such as "pick-up-sander" [Maes, 1989b].

Similarly, the Oz Project's first major virtual creature, Lyotard, was a text-based virtual cat [Bates *et al.*, 1992]. Because Lyotard lived in a text environment, his behaviors were also text and therefore high level: "Lyotard jumps in your lap," "Lyotard eats a sardine," "Lyotard bites you." Because we were using text, action did not need to be specified at a more detailed level. We did not have to specify, for example, how Lyotard moved his legs in order to jump in your lap.

Lyotard's successors, the Woggles, on the other hand, were graphically represented. As a consequence, we were forced into specifically defining every low-level action an agent took as part of a behavior. The effort that specification took meant that we spent less time on the Woggles' brains, and as a consequence the Woggles are not as smart as Lyotard. But — surprisingly to us — the Woggles also have much greater affective power than Lyotard. People find the Woggles simply more convincingly alive than the text cat, despite the fact that Lyotard is superior from an AI point of view. This is probably in part because we were forced to define a particular body, particular movements, and all those pesky particularities we AI researchers would rather avoid. [3]

Again, as mentioned in Intermezzo II, if we look at animation, the valorization tends to run to the other extreme [Thomas and Johnston, 1981]: the particular is the most essential. Animators tend to think mostly at the level of surface movement; this movement may be interpretable as a behavior, as evidence of the character's emotions, as revealing the character's motivations, or as any of a host of things or nothing at all. Animators make the point that any character is of necessity deeply particular, including all the details of movement, the structure of the body, and quirks of behavior. The abstract comes as an afterthought. Certainly, animators make use of a background idea of plot, emotion, and abstract ideas of 'what the character is doing,' but this is not the level at which most of animators' thinking takes place.

Loyall points out that this focus on the particular is also essential to the creation of effective believable agents [Loyall, 1997a]. A focus on particularity by itself, though, is not adequate for creating artificial agents. Agents are expected to interact autonomously with the user over time. In order to build such autonomous systems, we need to have some idea of how to structure the agent so that it can recognize situations and react appropriately. Because we do not know every detail of what will happen to the agent, this structure necessarily involves abstract concepts in such aspects as the modules of the agent, the classification of situations according to appropriate responses, abstract behaviors, emotions, goals, and so on.[4] We must design agents, at least partially, at an abstract level.

In order to build agents that effectively communicate through narrative, AI researchers will need to balance their ability to think at the abstract level with a new-found interest in the particular details their system produces, an approach that seems to be gaining in popularity [Frank *et al.*, 1997]. Narrative Intelligence is only possible with a deep-felt respect for the complex relationship between the abstract categories that structure an agent and the physical details that allow those categories to be embodied, to be 'read,' and to become meaningful to the user.

3.Intentional State Entailment

Suppose you hear the following:

> A man sees the light is out. He kills himself.

[3] Similar arguments may hold for robots. The Sony robotic dogs at Agents '97 were a compelling demonstration that robots may have much greater affective power than even graphically represented agents [Fujita and Kageyama, 1997].

[4] It may be that one day we can use machine learning to develop this structure instead; whether this learned agent must also be structured abstractly remains to be seen.

Is this a story? Not yet. You don't understand it. After endless questions, you find out that the man was responsible for a light house. During the night, a ship ran aground off shore. When the man sees that the light house light is out, he realizes that he is responsible for the shipwreck. Feeling horribly guilty, he sees no choice but to kill himself. Now that we know what the man was thinking, we have a story.

In a narrative, what 'actually happens' matters less than what the actors feel or think about what has happened. Fundamentally, people want to know not just what happened but *why* it happened. This does not mean the 'causes' of an event in terms of physical laws or stimulus-response reactions, but the reasons an actor freely chose to do what s/he did. The narrative is made sense of with respect to the thoughts and feelings of the people involved in its events.

This means that when people watch autonomous agents, they are not just interested in what the agent does. They want to know how the agent thinks and feels about the world around it. Instead of just knowing what the agent has chosen to do, they want to know *why* the agent has chosen to do it. This is, in fact, the grounds for the strategy animation uses for transitions: as mentioned in Intermezzo II, transitions in animation communicate the reasons for behavioral change.

But in many autonomous agent architectures, the reasons for the decisions the agent makes are part of the implicit architecture of the agent and therefore not directly expressible to the user. Bruce Blumberg's Hamsterdam architecture, for example, represents the appropriateness of each currently possible behavior as a number; at every time step the behavior with the highest number is chosen [Blumberg, 1996]. With this system, the reasons for behavioral choice are reduced to selecting the highest number; the 'actual' reason that behavior is the best is implicit in the set of equations used to calculate the number. The agent simply does not have access to the information necessary to express why it is doing what it does.

This means the strategy of action-expression described in Chapter 5 is more narratively friendly than action-selection. Instead of this emphasis on *selecting* the right action, Tom Porter suggests the strategy of *expressing* the reasons an agent does an action and the emotions and thoughts that underly its activity [Porter, 1997]. This means organizing the agent architecture so that reasons for behavioral change are explicit and continuously expressed. By showing not only what the agent does, but why the agent does it, people may have an easier time understanding what the agent is thinking and doing in general.

A deeper problem with current architectures is that ethologically-based models such as [Blumberg, 1996] presuppose that most of what an agent does is basically stimulus-response. As scientists, we are not interested in the vagaries of free will; we want to develop cause-effect rules to explain why animals do what they do when they do it. We intentionally adopt what Daniel Dennett might call a 'non-intentional stance' [Dennett, 1987]. We therefore develop theories of behavior that are fundamentally mechanistic.

But when we build agents that embody these theories, they often work through stimulus-response or straightforward cause-effect. This automaticity then carries forward into the quality of our agent's behavior.