Designing Comprehensible Agents

Phoebe Sengers
Carnegie Mellon University
phoebe@cs.cmu.edu

Abstract

For many applications, it is important that agents be not only correct, but also comprehensible to human users. Typically, people have tried to make agents' behavior and reasoning understandable by adding post-hoc special-purpose explanation systems, with often disappointing results. Here, I instead take the comprehensibility of agent behavior as a central agent design consideration from the start. I describe an agent architecture, the *Expressivator*, that supports comprehensibility on top of a behavior-based framework, using four technical innovations: (1) structuring the agent's behavior according to the *signs* and *signifiers* it is intended to communicate; (2) allowing the agent to keep track of its impression on the user with *sign management*, (3) using behavioral *transitions* to explain the reasons for agent, behavior, and (4) expressing behavioral interrelationships directly using *meta-level controls.*

1 Introduction

No matter how powerful an AI system is, it may be useless if its actions and reasoning are incomprehensible to users. Expert systems that suggest investing one's life savings in an obscure Caribbean manufacturer or treating a patient with a little-used drug with potent side-effects without making clear *why* will quickly be shelved in favor of old-fashioned, questionable human experts. Tutoring systems will fail if students cannot figure out what the agent demonstrating the proper solution is doing or why. Humans will have difficulty guiding teleautonomous robots whose behavior and reasoning is only understandable after time-consuming analysis of activity logs. Believable agents are pointless if their personality, behavior, and thinking are unclear.

In all these cases, the usability and usefulness of the AI system depends heavily on its behavior being easy to understand by human users. But most techniques available for building such systems focus on problem-solving, with understandability as an afterthought. As a result, previous approaches to this problem have generally focused on taking already-built but inadequately understandable systems and adding explanation systems to make them more comprehensible [Johnson, 1994] [Diederich and Tickle, 1995]. In essence, with these approaches the system's behavior and reasoning remain as ineffable as always, but a special-purpose system is added to reconstruct and translate it for human users often with disappointing results.

In this paper, I instead explore the possibility of designing agents to be comprehensible *from the start* That is, agents can be structured so that their behavior and reasoning is comprehensible all along, eliminating the need for a special-purpose explanation system to improve the agent's comprehensibility after the fact. In order to do this, I first lay out the requirements for comprehensibility from a psychological perspective. I then describe an implemented agent architecture, the *Expressivator*, which provides support for comprehensible behavior on top of a behavior-based framework. The Expressivator works by designing agents in terms of communicated signs rather than internal problem-solving behaviors, and by using behavioral transitions to explain the connection between the agent's activities to users.

2 Prerequisites for Comprehensibility

The groundwork for comprehensible systems has been laid out by believable agents research. Because comprehensibility is so essential to believable agents, BA systems have often included user understandability as one central agent design consideration [Loyall, 1997] [Neal Reilly, 1996] [Blumberg, 1996] [Lester and Stone, 1997], but one that is usually applied in an ad hoc manner. The Expressivator is instead built on a systematic analysis of the requirements for comprehensibility from a psychological point of view.

Basically, an agent is comprehensible if, based on the agents' observable actions, users can build an accurate interpretation of the agent's beliefs, reasoning, knowledge, and so on. This means that an agent will be particularly understandable if it gives off the kinds of behavioral cues from which people find it easy to construct meaning. In order to know what kind of cues these are, we need to understand how people go about interpreting agents. We can then tailor our agents to give the kind of behavioral cues that are easy for people to understand.

The way in which people go about interpreting the behavior of complex, autonomous systems is the subject of the field of *narrative psychology* [Bruner, 1990]. Narrative psychology argues that people construct meaning from observed actions by organizing them into *narrative*, or stories. Narrative theory has uncovered numerous properties of this kind of organization [Bruner, 1991], including centrally the following two:

- People try to discover not only *what* the agent does, but also *why*. The reasons or motivations behind actions are just as important as what is done.
- Events are not seen in isolation. Users try to interpret events as they relate to each other.

Narrative psychologists argue that people comprehend intentional activity by trying to interrelate observed actions and to find out the reasons for the agent's behavioral choices. If this is so, then an agent, to be maximally understandable, must have the following properties:

- The agent must clearly express *what* it is doing. These individual actions must be clear to the user, since they form the building blocks for any future interpretation.
- The agent must clearly express *why* it is doing what it does.
- The *relationships* between the agents' activities must be made clear. The agent cannot simply jump between different activities, but should suggest to the user how the activities fit together into a logical whole.

In addition, for maximum comprehensibility, behavior cannot be arbitrarily complex. Behavior in which an agent can take account of many perhaps barely noticeable environmental conditions and do complex reasoning instantaneously simply cannot be communicated adequately. At the same time, since users can infer a great deal about the agent's motivations and personality from simple actions, simple, well-chosen behaviors can be enough.

The properties which comprehensible agents must have lead to the following heuristic for comprehensible agent construction:

> Behaviors should be *as simple as possible*. The agent's comprehensibility comes from thinking out the *connections* between behaviors and *displaying* them to the user.

3 Building Comprehensible Agents

This heuristic forms the basis of the *Expressivator*, an agent architecture that focuses on the expression of agent actions and their interrelationships to users. The Expressivator is built on top of Hap [Loyall and Bates, 1991], a behavior-based language designed for believable agents. The Expressivator provides systematic support for narrative comprehensibility through the following mechanisms:

- Expressing *what* the agent does: The agent's design is based not on internally-defined behaviors, but on *signs* and *signifiers* which are directly communicated to the user. The agent uses a *sign-management system* to keep track of the signs and signifiers that have been communicated.
- Expressing *why* the agent does it: Behavioral *transitions* communicate the reasons for agent activity.
- Expressing the *relationships* between activities: Transitions use *meta-level controls* to know about and influence other behaviors, so that their relationships are expressly communicated to the user.

Each of these mechanisms is described in more detail below. Examples come from the Expressivator's use in the *Industrial Graveyard.* In this virtual environment the Patient, a discarded lamp character implemented with the Expressivator, attempts to eke out a miserable existence while being bullied about by the Overseer, an agent implemented in Hap.

3.1 Signs, Signifiers, and Sign Management

The first prerequisite for behavior comprehensibility is that the user should be able to clearly tell what the agent is doing. Clear communication of the basic actions and behaviors of the agent is essential if the user is to comprehend the agent's activity. Signs and signifiers support the construction of clearly communicated behavior; sign management allows the agent itself to keep track of what has been communicated, so it can tailor subsequent behavioral communication to the user's current interpretation.

Signs and Signifiers

Current behavior-based approaches are based on an internal, problem-solving approach, and generally divide an agent into activities in which the agent likes to or needs to engage. Typical behavior-based systems divide an agent into three parts: (1) physical actions in which the agent engages, (2) low-level behaviors, which are the agent's simple activities, and (3) high-level behaviors, which combine low-level behaviors into high-level activities using more complex reasoning. Because these activities are implemented according to what makes sense from the agent's internal point of view, there is no necessary correlation between the agent's behaviors and the behaviors we would like the user to see in our agent.

But for comprehensible agent design, it may make more sense to design the agent according to the things we would like to communicate to the user, i.e. making the internal behaviors exactly those behaviors we want to communicate to the user. Then, communicating what the agent does reduces to the problem of making sure that each of these behaviors is properly communicated. For this reason, the Expressivator structures an agent not into physical actions and problem-solving behaviors, but into signs and signifiers, or units of action that are likely to be meaningful to the user. This structure involves three levels, roughly corresponding to those of generic behavior-based AI: (1) *signs*, which are small sets of physical actions that are likely to be interpreted in a particular way by the user; (2) *low-level signijiers*, which combine signs, physical actions, and mental actions to communicate particular immediate physical activities to the user; and (3) *high-level signifiers*, which combine low-level signifiers to communicate the agent's high-level activities.

There are several differences between these structural units and the default behavior-based ones. Unlike physical actions and behaviors, signs and signifiers focus on *what the user is likely to interpret*, rather than what the agent is 'actually' (i.e. internally) doing. In addition, signs and signifiers are *context-dependent;* the same physical movements may lead to different signs or signifiers, depending on the context in which the actions are interpreted. Most importantly, signs and signifiers carry an *explicit commitment* to communication; they require the agent designer to think about how the agent should be interpreted and to provide visual cues to support that interpretation.

Signs and signifiers are not simply design constructs; they also have technical manifestations. Formally, 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 consists of an arbitrary label and an optional set of arguments. The label, such as "noticed possible insult", is meaningful to the designer, and represents how the designer expects that physical behavior to be interpreted. The arguments (such as "would-be insulter is Wilma") give more information about the sign. This token is stored by the sign-management system described below, so that the agent can use it to influence its subsequent behavioral decisions. A low-level signifier is a behavior that is annotated with the special form (with lowJeveljsignifying...); a high-level signifier is similarly annotated (with highJevel_signifying....). Signifiers can also generate tokens for the sign-management system, as described below.

Sign Management

Once a designer has structured an agent according to what it needs to communicate, agents can reason about what has been communicated in order to fine-tune presentation of subsequent signs and signifiers. That is, by noting which signifiers have been communicated, agents can reason about the user's likely current interpretation of their actions and use this as a basis for deciding how to communicate subsequent activity.

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.* For example, if the agent is currently running a "head-banging" signifier, the user will need to see the agent smack its head a few times before realizing that the agent is doing it.

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. 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. Similarly, once signs have been posted that express a low-level signifier, behaviors use postJowJevel to post that that low-level signifier has been successfully expressed. Once the right low-level signifiers have been posted to express a high-level signifier, post-high Jevel is used to post that high-level signifier.

Each of these commands causes a token to be stored in the agent's memory listing the current sign, low-level signifier, or high-level signifier, respectively, along with a time stamp. Once signs and signifiers have been posted, other behaviors can check to see what has been posted recently before they decide what to do. The result is that the signs and signifiers the agent has expressed can be used just like environmental stimuli and internal drives to affect subsequent behavioral presentation, tuning the agent's behavior to the user's interpretation.

3.2 Transitions

The second requirement of behavior comprehensibility is that the user should be able to tell *why* the agent is doing what it is doing. In behavior-based terms, every time an agent selects a particular behavior, it should express to the user the reason it is changing from the old behavior to the new one. This is difficult to do in most behavior-based systems because behaviors are designed and run independently; when a behavior is chosen, it has no idea who it succeeds, let alone why.

In the Expressivator, behavioral *transitions* are used to express the agent's reasoning. Transitions are special behaviors which act to 'glue' two signifying behaviors together. When a transition notices that it is time to switch between two signifiers, it takes over from the old signifier. Instead of switching abruptly to the new signifier, it takes a moment to express to the user the reason for the behavioral change.

Transitions are implemented in two parts, each of which is a full-fledged behavior: (1) *transition triggers*, that determine when it is appropriate to switch to another behavior for a particular reason, and (2) *transition demons*, that implement the transition sequence that expresses that reason to the user. Transition triggers run in the background, generally checking which behaviors are running (e.g. exploring the world), and combining this information with sensory input about current conditions (e.g. the Overseer is approaching). When its conditions are fulfilled, the transition trigger adds a special token to memory, rioting the behavior which should terminate, the behavior which should replace it, and a label which represents the reason for the replacement (e.g. afraid _of_overseer).

Transition demons monitor memory, waiting for a transition for a particular reason to be triggered. They then choose an appropriate behavioral expression for the reason for change, according to the current likely user interpretation and conditions in the virtual environment. Expressing the reasoning behind behavioral change often requires changes to subsequent behaviors; for example, if the Patient starts doing some odious task because it is forced to by the Overseer, it should include some annoyed glances at the Overseer as part of the task-fulfilling behavior. Transitions are able to express these kinds of interbehavioral influences using the meta-level controls described below.

3.3 Meta-Level Controls

The third requirement of narrative comprehensibility is that the relationships between the agent's activities are made clear. Instead of jumping around between apparently independent actions, the agent's activities should express some common threads. But these relationships are difficult to express in most behavior-based systems because they 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. Because the mechanisms that handle relationships between behaviors are part of the implicit architecture of the agent, they are not directly expressible to the user.

The Expressivator deals with this problem by giving behaviors *meta-level controls,* special powers to sense and influence each other. 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 nonactive behaviors to suggest actions for the currently dominant behavior to do on the side [Blumberg, 1996]. In the Expressivator, behaviors can (1) *query* which other behaviors have recently happened or are currently active; (2) *delete* other behaviors; (3) *add* new behaviors, not as subbehaviors, but at the top level of the agent; (4) *add new sub-behaviors* to *other* behaviors; (5) *change the internal variables* that affect the way in which other behaviors are processed; (6) *turn off a* behavior's ability to send motor commands, and (7) *move running subbehaviors* from one behavior to another.

The most important function for these meta-level controls in the Expressivator is to allow for the implementation of transitions. Transitions, at a minimum, need to be able to find out when an old behavior needs to be terminated, delete the old behavior, engage in some action, and then start a new behavior. This means 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. Ideally, they should also be able to alter the new behavior's processing to reflect how it relates to what the agent was doing before. In the Expressivator, transitions can do all these things with meta-level controls.

More generally, meta-level controls make the relationships between behaviors explicit, as much a part of the agent design as behaviors themselves. They allow behaviors to affect one another directly when necessary, rather than making interbehavioral effects subtle side-effects of the agent design. Meta-level controls give agent builders more power to expose the inner workings of agents by letting them access and then express aspects of behavior processing that other systems leave implicit.

3.4 Putting It All Together

Narrative psychology suggests that comprehensibility requires agents to clearly express what they do, why they do it, and the interrelationships between their activities. The Expressivator supports comprehensibility by expressing the agent's actions through signs and signifiers, the reasons for agent activity through transitions, and the interrelationships between activities through meta-level controls.

These architectural mechanisms are described separately, but used together in the agent design process, changing the way in which agents are designed. In a typical behavior-based system, an agent is defined in 3 major steps: (1) deciding on the high-level behaviors in which the agent will engage; (2) implementing each high-level behavior, generally in terms of a number of low-level behaviors and some miscellaneous behavior to knit them together; (3) using environmental triggers, conflicts, and other design strategies to know when each behavior is appropriate for the creature to engage in. With the Expressivator, the choice and expression of these structural 'units' for the agent is not enough; in order to support the user's comprehension, the designer must also give careful consideration to expressing the reasons for and interrelationships between those units. These interrelationships are designed and implemented with transitions, which alter the signifiers they connect in order to make them clearer. In practice, transitions are the keystone of the architecture, combining signifiers in meaningful ways through the use of meta-level controls.

4 Results

The best way to see how the Expressivator changes the quality of agent behavior is to look at how its transitions work in detail. Here, I will go over one point where the agent switches behaviors, and explain how transitions make this switch more comprehensible. One example does not proof make, but it does take up a lot of space; the sceptical reader can find more in [Sengers, 1998].

As our excerpt begins, the Patient notices the schedule of daily activities which is posted on the fence. It goes over to read the schedule. The Overseer, noticing that the Patient is at the schedule and that the user is watching the Patient, goes over to the schedule, changes

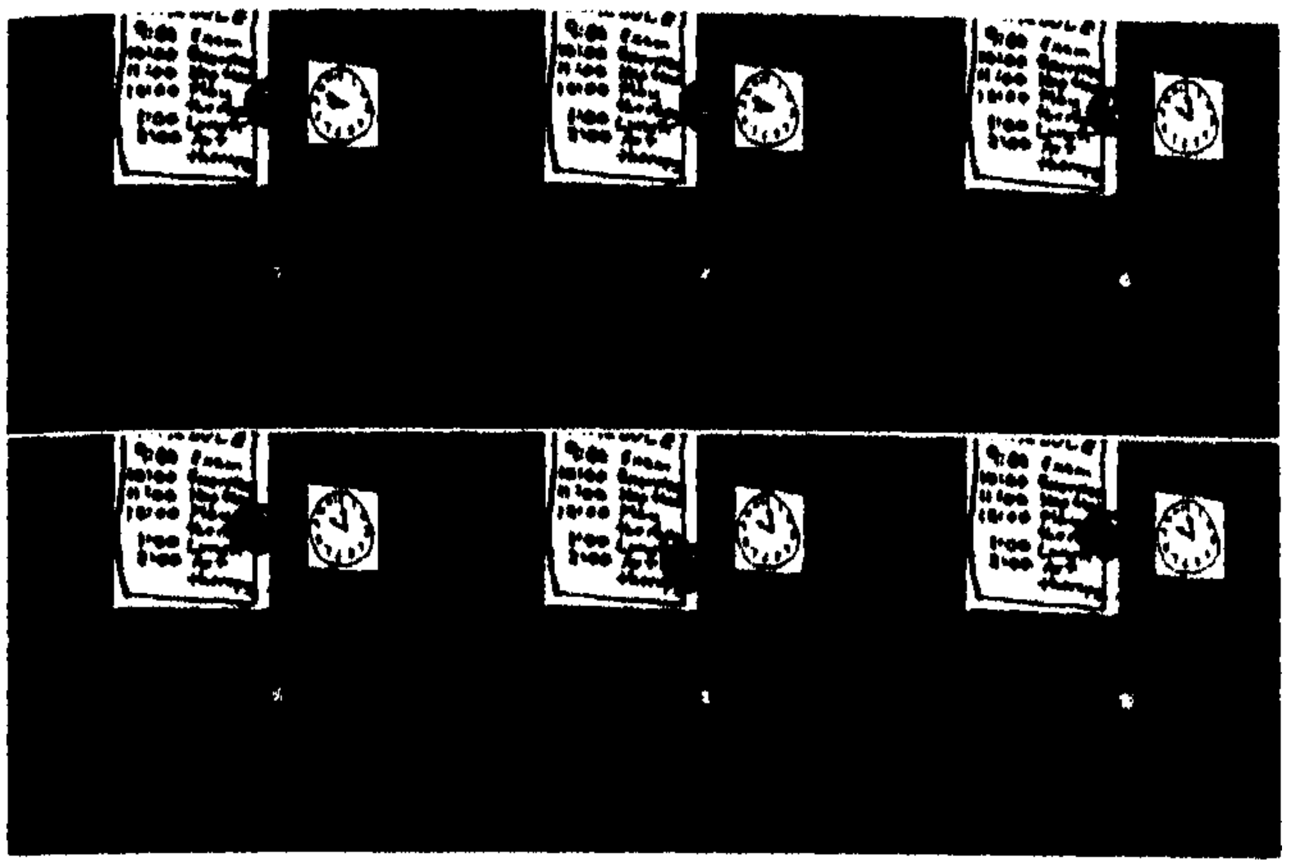

Figure 1: Response without transitions

the time to 10:00, and forces the Patient to engage in the activity for that hour: exercising.

The goal of this part of the plot is to communicate to the user the daily regime into which the Patient is strapped. The Patient does not have autonomy over its actions; it can be forced by the Overseer to engage in activities completely independently of its desires. The specific behavioral change from reading the schedule to exercising, then, should show the user that the agent changes its activity because (1) it notices the Overseer, (2) the Overseer enforces the scheduled activities; (3) the activity that is currently scheduled is exercising.

Without transitions, the Patient's response to the Overseer is basically stimulus-response (Figure 1). The Patient starts out reading the schedule. As soon as the Patient senses the Overseer, it immediately starts exercising. This reaction is both correct and instantaneous; the Patient is doing an excellent job of problem-solving and rapidly selecting optimal behavior. But this behavioral sequence is also perplexing; the chain of logic that connects the Overseer's presence and the various environmental props to the Patient's actions is not displayed to the user, being jumped over in the instantaneous change from one behavior to another.

With transitions, attempts are made to make the logic behind the behavioral change clearer (Figure 2). Again, the behavior starts with the Patient reading the schedule. This time, when the Overseer approaches, the Patient just glances at the Overseer and returns to reading. Since the Patient normally has a strongly fearful reaction to the Overseer (and by this time the Overseer's enthusiasm for punishing the Patient has already generally aroused sympathy in the user's mind), the user has a good chance of understanding that this simple glance without further reaction means that the Patient has not really processed that the Overseer is standing behind it.

Suddenly, the Patient becomes startled and quickly looks back at the Overseer again. Now, the user can get the impression that the Patient has registered the Overseer's presence. W'hatever happens next must be a reaction to that presence. Next, the Patient checks the

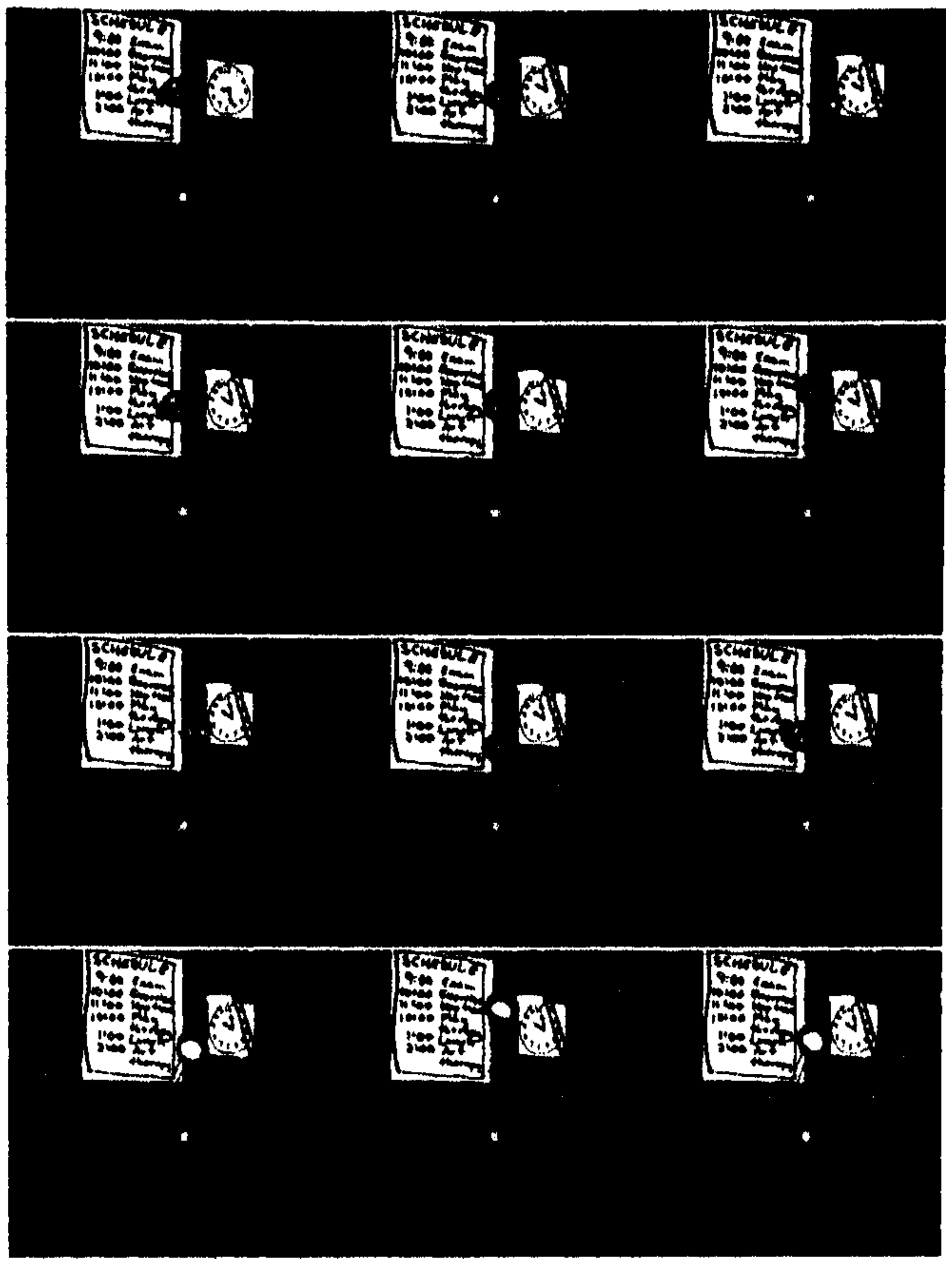

Figure 2: Response with transitions

time and the schedule of activities to determine that it is time to exercise. Then the Patient whirls to face the Overseer and frantically and energetically begins exercising, tapering off in enthusiasm as the Overseer departs. This transition clearly communicates that the change in behavior is connected to several factors: the presence of the Overseer, the clock, and the schedule. This is in contrast with the transition-less sequence, in which there is no clear connection between any of the environmental factors and the Patient,'s behavioral change.

4.1 Evaluation

How good is the Expressivator? The kind of detailed transition analysis given here suggests that, with the Expressivator, the user is given more information on which to judge both the agent's behavior and the reasons for the agent's behavioral changes. This is certainly a basis for improved user understanding, but does not necessarily imply actual improvement. In particular, the quality of the animation is not up to snuff, which means users sometimes have trouble interpreting the simple movements of the agent. All the innovations the Expressivator introduces are wort hless if individual signs are not clearly animated; everything rests on the substantial animation problem of getting a sigh to look like a sigh and not like a cough or a snort. This problem is exacerbated when, as in Hap, there is a mind-body split, with the mind generating actions that are implemented autonomously by the body. The resulting divide between command and execution makes accurate timing and therefore effective control of animation impossible. This problem of generating expressive animation, while not a straightforward "AI problem," must be addressed by any architecture that is going to implement graphically presented, comprehensible agents.

Certainly, the difficulty of communicating agents' thinking could be lessened by allowing them to talk. Then, the framework designed here could be applied simply by using natural language to express what the agent is doing and why. At the same time, *constant* explanatory commentary ("Now I am going to hide because the Overseer is coming and I am afraid!") seems unnatural and distracting, leaving a role for physical action to back up and expand on the agent's utterances.

The *Industrial Graveyard* is an entertainment application, but the constructs of the Expressivator are not limited to believable agents. The concept of a narrative structure for behavior can be just as important for tele-autonomous robots, semi-autonomous avatars, or pedagogical agents. However, the Expressivator's focus on behavior and concrete action probably does not adequately support systems like automatic theorem provers that engage in complex, abstract reasoning.

The greatest conceptual problem with the Expressivator is the potential explosion of the number of transitions needed between signifiers; but this turned out not to be a problem in practice. For the Patient's 8 high-level signifiers there were only 15 transitions, and for the Patient's 16 low-level signifiers, there were only 25 transitions. This is for several reasons. First of all, transitions are only needed between high-level signifiers, and between low-level signifiers that share the same high-level signifier — *not* between low-level signifiers in different high-level signifiers.[1] I also cut out many transitions by writing several generic transitions, that could go from any behavior to a particular behavior. Most importantly, I found in practice that many of the possible transitions did not make practical sense because of the semantics of the behaviors involved.

The greatest advantage of the Expressivator for the behavior programmer is that it makes it much easier to handle interbehavioral effects. The coordination of multiple high-level behaviors is one of the major stumbling blocks of behavior-based architectures [Brooks, 1990]; since interbehavioral factors are implicit in the architecture they are hard to control, leading to multiple behaviors battling it out over the agent's body, and hours of tweaking to get each behavior to happen when and only when it is supposed to. This is much easier to handle when behaviors can simply kill other behaviors that are not appropriate, and when the trigger conditions for each behavior can be explicitly set.

[1] This would be implemented instead with a transition between the respective high-level signifiers.

5 Conclusion

For many applications, agents will be much more usefill if their behavior and reasoning are understandable to human users. The work done here is based on the idea that agents will be more comprehensible if they are tailored to support human, narrative interpretation from the start. This interpretation requires that agents should show what they are doing, why they are doing it, and the interrelationships between their activities. In practice, this means making behaviors maximally simple and expressive, and explicitly showing the connections between them. The Expressivator supports this style of agent construction by using transitions to relate signifying behaviors, expressing their interrelationships through meta-level controls.

References

[Blumberg, 1996] Bruce Blumberg. *Old Tricks, New Dogs: Ethology and Interactive Creatures.* PhD thesis, MIT Media Lab, Cambridge, MA, 1996.

[Brooks, 1990] Rodney A. Brooks. Elephants don't play chess. In Pattie Maes, editor, *Designing Autonomous Agents.* MIT Press, Cambridge, MA, 1990.

[Bruner, 1990] Jerome Bruner. *Acts of Meaning.* Harvard University Press, Cambridge, MA, 1990.

[Bruner, 1991] Jerome Bruner. The narrative construction of reality. *Critical Inquiry,* 18(1):1 21, 1991.

[Diederich and Tickle, 1995] Joachim Diederich and Alan B. Tickle. Explanation and collective computation. *Complexity International,* 2, 1995.

[Johnson, 1994] W. Lewis Johnson. Agents that learn to explain themselves. In *Proceedings of AAAI-94,* pages 1257 1263, Seattle, Washington, 1994. AAAI Press.

[Lester and Stone, 1997] James C. Lester and Brian A. Stone. Increasing believability in animated pedagogical agents. In W. Lewis Johnson, editor, *Proceedings of the First International Conference on Autonomous Agents,* pages 16 21, NY, February 1997. ACM Press.

[Loyall and Bates, 1991] A. Bryan Loyall and Joseph Bates. Hap: A reactive, adaptive architecture for agents. Technical Report CMU-CS-91-147, Carnegie Mellon University, 1991.

[Loyall, 1997] A. Bryan Loyall. *Believable Agents: Building Interactive Personalities.* PhD thesis, Carnegie Mellon University, Pittsburgh, May 1997. CMU-CS-97-123.

[Neal Reilly, 1996] Scott Neal Reilly. *Believable Social and Emotional Agents.* PhD thesis, Carnegie Mellon University, 1996. CMU-CS-96-138.

[Sengers, 1998] Phoebe Sengers. *Anti-Boxology: Agent Design in Cultural Context.* PhD thesis, Carnegie Mellon University Department of Computer Science and Program in Literary and Cultural Theory, Pittsburgh, PA, 1998.