Animation from Instructions

Norman L. Badler, Bonnie L. Webber, Jugal Kalita, and Jeffrey Esakoff

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pennsylvania

ABSTRACT

We believe that computer animation in the form of narrated animated simulations can provide an engaging, effective, and flexible medium for instructing agents in the performance of tasks. However, we argue that the only way to achieve the kind of flexibility needed to instruct agents of varying capabilities to perform tasks with varying demands in work places of varying layout is to drive both animation and narration from a common representation that embodies the same conceptualization of tasks and actions as natural language itself. To this end, we are exploring the use of natural language instructions to drive animated simulations. In this chapter, we discuss the relationship between instructions and behavior that underlie our work and the overall structure of our system. We then describe in somewhat more detail three aspects of the system—the representation used by the simulator, the operation of the simulator, and the motion generators used in the system.

3.1 Introduction

The enterprise we have embarked on rests on a two-part argument:

- Narrated animations are an engaging and extremely effective medium for instructing agents in task performance. Moreover, coupled with the emerg-

ing technology of virtual reality (Fisher et al., 1986), narrated animations can provide a low-cost way of giving learners “personal” trainers and “on-site” environments in which to train.

- The only way to create the kind of flexible narrated animations needed to instruct agents of varying capabilities to perform tasks with varying demands in work places of varying layout is to drive both animation and narration from a common representation that embodies the same conceptualization of tasks and actions as natural language itself.

Here we explain and elaborate this argument, before describing our own efforts towards creating narrated animations.

To argue the first point—that narrated animations are an engaging and extremely effective medium for instructing agents in task performance—we need to answer three questions:

1. Why narrated animations? Why not just annotated or captioned still images?

2. Why narrated animations? Why not just movies or videotapes of human agents carrying out the tasks?

3. Why narrated animations? Why not animation alone?

Annotated or captioned stills are clearly useful in grounding the referents of terms in instructions such as “the pipe and ball assembly” and “four blade flanges” (to take two examples from a ceiling fan installation manual) and in demonstrating what the end results of particular actions should look like. However, they cannot show the agent how to achieve those results. Narrated animations can do both. As to why animation should be preferred to live-action videotape, it is simply that graphics (and schematic renderings, in general) can abstract away (as well as visually “carve away” what is irrelevant, demonstrating only what is relevant to the task at hand. As to why narrated animation is better than animation alone, researchers studying plan inference have shown just how hard it is to infer an agent’s intentions from his or her observed actions alone (Cohen, 1981). To effectively instruct an agent to do a task, the communication of intentions is as important to effective performance as the communication of actions. Such intentions cannot be effectively communicated through images alone. For example, consider the following excerpt from instructions for installing a diverter spout on a bath tub faucet:

Install new spout. When doing so, DO NOT use lift knob or hose connection for leverage. Damage may result. Tighten by hand only.

With red-slashed icons on warning signs may be effective in reminding people of what behavior is forbidden (“no smoking,” “no wearing high-heeled shoes”), they cannot unambiguously convey the reason for forbidden or otherwise discouraged behavior. Communication of both the hows and the whys of task performance requires a union of both visual presentation and image.
The second part of our argument is that the only way to create the kind of flexible narrated animations needed to instruct agents of varying capabilities to perform tasks with varying demands in work places of varying layout is to drive both animation and narration from a common representation that embodies the same conceptualization of tasks and actions as natural language itself.

Of course, we are not claiming that animation can be driven solely from that common representation: Other types of knowledge are clearly needed as well—including knowledge of motor skills and other performance characteristics (see Section 3.7). Nor are we claiming that these types of knowledge could in any way be provided through natural language. That too is clearly false. For example, the gross underspecification of natural language communication comes out most strongly when attempting to describe motion: The relation between language and the world appears to be at its most tenuous where motion specification is concerned. Existing means of specifying motor skills such as direct manipulation, skill learning, and algorithmic determination are clearly more appropriate.

The second part of our argument raises the following further questions:

1. Why should animation and narration be driven from a common representation? Why not just create them separately—for example, through direct manipulation and text provided by some commentator?

2. Why should that representation embody the same conceptualization of tasks and actions (that is, reflect the same ontology) as natural language itself?

3. Where are we going to get that common representation?

The point of driving animation and narration together from a common representation is flexibility and accuracy. How an agent should carry out a task may depend on both his capabilities and features of the given workspace. Veridical narration must reflect both the agent’s actions and the conditions motivating them. This can be provided by an outside commentator, but she will have to comment afresh (and in a consistent manner) for each “version” of the task simulation. Moreover any changes in the task (for example, those arising from minor model changes in the device whose use is being portrayed) are likely also to require fresh commentary in order to remain veridical. As we and others have argued regarding explanation of expert system behavior, it is desirable that the explanation derive from the same underlying specification as the system’s reasoning, if that explanation is to truthfully represent why the system came to its conclusions.1

As to why that common representation should embody the same ontology of tasks and actions as natural language (including the range of causal and other-wise contingent relations between actions assumed by language), the reasons are separate for animation and narration, but the conclusion is the same. On one hand, if the representation is to drive the generation of natural language, the structure of the representation should be directly related to natural language semantics. On the other, since natural language semantics is directly related to natural cognition, it is likely that a representation that embodies it will be maximally helpful to (human) graphics systems designers developing and modifying the animation side of the system.

Finally, as to where we can get that common representation, we would like to argue against creating it by hand and for acquiring as much as possible through the medium most often used for conveying the whys and hows of actions and tasks—natural language instructions. The alternative of encoding the driving representation directly by hand has all the disadvantages that have pulled the programming language community in the direction of higher level programming languages. The “smarter,” more powerful compilers they demand.2 That such higher level tools should include natural language instructions comes in part from the vast body of such data we have around and in part, what we can learn from them vis-à-vis the production of the narrative that accompanies animation (or even live instructional material) to clarify and explain the action.

This chapter describes some of our work to date on producing narrated animations from natural language instructions. Its structure is as follows:

- Section 3.2 briefly describes previous efforts to connect natural language instructions with computer graphics animations.
- Section 3.3 discusses instructions as given in natural language and characterizes computational models for understanding them.
- Section 3.4 outlines the structure and components of the system we are developing here at the University of Pennsylvania to study instructions and their animation by synthetic (human-like) agents.
- Section 3.5 elaborates the portion of the system between the planned actions and semantically valid primitives that may be characterized computationally.
- Section 3.6 describes the simulator and temporal constraint planner that organizes the primitives into a sequential stream of executable motions.
- Section 3.7 outlines the available motion generators.
- Section 3.8 summarizes agent performance issues that arise at various places in the system.
- Section 3.9 offers some observations and conclusions.

1. This is not to say the reasoning and explanation must be structurally identical: Clarity and communicativeness often argue against such an isomorphism.

2. In the case of animation, higher level tools will widen the user community beyond current manually skilled (or programming-wise) animators: include for example, human factors engineers with tasks to design or evaluate and trainers with new personnel to instruct.
3.3 Instructions

The jumping-off point for our view of instructions—what they are, what it means to understand them and what it means to use them in the context of animation from instructions—is the view of plans we have adopted from the work of Pollack (1986; 1990), Suchman (1987), and Agre and Chapman (1989). This view of plans gives us a simple and uniform way to talk about instructions: it does not, by itself, solve the problems of fully understanding instructions to agent behavior—the behavior we hope to demonstrate through our animated simulations. That linkage we are undertaking gradually, by considering instructions of increasing richness, in the sense described in Section 3.3.3.

3.3.1 Plan as Data Structure

In (Pollack, 1986), Pollack contrasts two views of plan: plan as data structure and plan as mental phenomenon. (The former appears to be the same view of plans that Agre and Chapman have called plan as program.) Plans produced by Sacerdoti’s NOAH system (Sacerdoti, 1977) are a clear example of this plan as data structure view. Given a goal to achieve (that is, a partial state description), NOAH uses its knowledge of actions—their preconditions, the conditions they are able to achieve, and their elaborations in terms of (partially ordered) aggregates of other actions—to create a data structure (a directed acyclic graph) whose nodes represent goals or actions and whose arcs represent temporal ordering, elaboration, or entailment relations between nodes. This data structure represents NOAH’s plan to achieve the given goal.

As Suchman points out, NOAH’s original intent was to provide support for novice human agents in carrying out their tasks. Given a goal that an apprentice was tasked with achieving, NOAH was meant to form a plan and then use it to direct the apprentice in what to do next. To do this, it was meant to generate a natural language instruction corresponding to the action associated with the “current” node of the graph. If the apprentice indicated that he didn’t understand the instruction or couldn’t perform the prescribed action, NOAH was meant to “move down” the graph to direct the apprentice through the more basic actions whose performance would entail that of the original. The result is a sequence of instructions that corresponds directly to the sequence of nodes encountered on a particular graph traversal. This also shows why Agre and Chapman have labeled this the plan as program approach, since the plan effectively corresponds to a sequence of commands in the “instruction set” of the device (here, the human agent) intended to execute them.

Because there has been so little substantive work published (or, to our knowledge, done) on controlling and augmenting animation with natural language (but see Esakov and Badler, 1990; Gangel, 1985; Green and Sun, 1988; Simmons and Zent-Novak, 1975; Takashima et al., 1987), it is important to state clearly why we believe this route will prove successful. Several developments have occurred in animation technology that are enabling us to realize the “natural language connection.”

Task-level specifications are one of the three levels of animation control surveyed by Zeltzer in his insightful analysis (Zeltzer, 1985). He recognized the need for planning and the requirement that tasks be executable as skills known the animated agent. In 1983 we outlined a system designed to translate task specifications into animation (Badler et al., 1983) that has evolved into the structure presented here.

Previous attempts to animate natural language have been weakened by a use of or arbitrary set of motion control schemes available to implement task meanings. Prior to 1978, Badler et al. (Badler, 1978; Weber et al., 1978) tried to design a suitable architecture but were stymied during implementation by large numbers of particular animation process problems requiring clarification and solution (Badler et al., 1980). Later, problems of locomotion control, essential to the animation of a mobile agent, were addressed by others (Bruderlin and Calvert, 1989; Sander and Maciejewski, 1985; Zeltzer, 1982) yielding manageable models.

Verse kinematics for end-effector goal positioning were adopted from robotics invented for biomechanical models (Badler et al., 1987; Girard and Maciejewski, 1985; Korein and Badler, 1982; Korein, 1985; Lee et al., 1990; Rao and Badler, 1989).

More recently, dynamic simulations have allowed objects to move in physically correct motions (Armstrong et al., 1987; Hahn et al., 1988; Hoffman and opcroft, 1987; Isaacs and Cohen, 1987; Moore and Wilhelms, 1988; Wilhelms, 1986; Wilhelms, 1987), and geometric constraints resolved by kinematic, force, and energy considerations have freed animators from having to specify the details of object trajectories (Barzel and Barr, 1988; Brotman and Netervalli, 1988; Sander and Melzer, 1988; Witkin et al., 1987; Witkin and Kass, 1988).

Zeltzer (1982), Reynolds (1987), and Ridsdale (Ridsdale et al., 1986) have explored behavioral models that tried to control the complexity of interaction between any individuals or parts of the same object. By putting together rough animation processes, the Thalmanns have even used “synthetic actors” to imitate the personalities, expressions, and actions of life-like figures of Marilyn Monroe and Humphrey Bogart (Magnenat-Thalmann and Thalmann, 1987).

Coming even closer to natural language-level instructions, high-level description have been compiled into simple animations based on a small number of predefined actions (Drewery and Tsotsos, 1986; Fishwick, 1986; Takashima et al., 1987) or motor skills (Morasso and Tagliasco, 1986; Zeltzer,
3.3.2 Plan as Mental Phenomenon

The alternative view of plans presented by Pollack (1986; 1990) is the plan as mental phenomenon view, which builds on earlier ideas about plans put forth by Bratman (1983). In this view, having a plan to do some action \( \beta \) corresponds roughly to:

- a constellation of beliefs about actions and their relationships;
- beliefs that their performance, possibly in some constrained order, both entails the performance of \( \beta \) and plays some role in its performance;
- an intention on the part of the agent to act in accordance with those beliefs in order to perform \( \beta \).

In order to describe the consequences of this approach for our view of instructions, we need to say more about such beliefs. Pollack draws a three-way distinction between act types, actions (or acts) and occurrences. Act-types are, intuitively, types of actions like playing a chord, playing a D-major chord, playing a chord on a guitar, etc. Act-types, as these examples show, can be more or less abstract. Actions can be thought of as triples of act-types, agents, and times (relative or absolute intervals) like Mark playing a D-major chord last Sunday afternoon on his Epiphone. Because it is useful to distinguish an action from its occurrence in order to talk about intentions to act that may never be realized, Pollack introduces a separate ontological type occurrence that corresponds to the realization of an action. (Pollack represents an occurrence as OCCUR(\( \beta \)), where \( \beta \) is an action. Thus an occurrence inherits its time from the associated time of its argument.)

Agents can hold beliefs about entities of any of these three types:

- Act-types: An agent may believe that playing a D-major chord involves playing three notes (D, F\# and A) simultaneously, or that she does not know how to perform the act-type playing a D-major chord on a guitar, etc. Any or all of these beliefs can, of course, be wrong.

- Actions: An agent may believe that some action \( \alpha \) must be performed before some other action \( \alpha' \) in order to do action \( \beta \) or that \( \alpha \) must be performed before \( \alpha' \) in order to do \( \beta' \). Here too, the agent’s beliefs can be wrong. (It was to allow for such errors in beliefs and the natural language questions they could lead to that led Pollack to this plan as mental phenomenon approach.)

- Occurrences: An agent may believe that what put the cat to sleep last Sunday afternoon was an overdose of catnip. He may also have misconceptions about what has happened.

While Pollack does not discuss necessary or default relationships between beliefs about particular act-types or between particular act-types, actions and occurrences—for example, what beliefs about act-types are inheritable by more specific act-types or by the actions they participate in, this would seem a useful area to explore for its applicability to planning (Tenenbaum, 1989). Plan inference, an instruction understanding.

In contrast with the previous view of instructions as lexicalized graph traversals, the plan as mental phenomenon approach allows one to view instructions as being given to an agent in order that she develops appropriate beliefs, which the agent may then draw upon in attempting to “do \( \beta \)” Depending on the evolving circumstances, different beliefs will become salient at different times. This appears to be involved in what Agre and Chapman and what Suchman mean by using plans as a resource. Beliefs are a resource an agent draws on in deciding what to do next.

3.3.3 Behavior

Given this view of plan as mental phenomenon, we can now consider possible relationships between instructions and an agent’s behavior. At one extreme is a direct relationship, as in the game “Simon Says,” where each command issued by the leader (“Simon says put your hands on your ears”) is meant to lead directly to particular behavior on the part of the player. That is,

Instruction \( \rightarrow \) Behavior

The fact that such instructions are given in natural language is almost (but not quite) irrelevant. That one can drive animated simulations from such instructions has been demonstrated by Esakov and Badler (1989). Key-frames from an animated simulation of two agents (John and Jane) at a control panel following an instruction sequence that begins

```
3. Since instructions may be presented as applicable until a particular event occurs (“Twist the blade until it clicks into place”) or after a particular event has occurred (“After 1988, you have to deprecate X using revised schedule Y”), one might also assume they are meant to instill beliefs about particular occurrences as well.
```
1. The scope or manner of an action may become clear only through understanding several sentences in the instruction set. For example, the intended culmination of an action may not be intrinsic to that action, but only contingent on the start of the action prescribed next. A simple example of this appears in the instructions that Agre and Chapman (1989) gave to several friends for getting to the Washington Street subway station.

Left out the door, down to the end of the street, cross straight over Essex then left up the hill, take the first right, and it'll be on your left.

While the action description "go up the hill" may have an intrinsic endpoint (that is, when the agent gets to the top of the hill), it is not the intended termination of the action in the context of these instructions. Its intended termination is the point at which the action of "taking the first right" commences—that is, when the agent recognizes that he has reached the first right.

The previous example illustrated the scope of an action provided by subsequent utterances. Manner can be specified as well—for example, where cautions or warnings in instructions provide information on how not to do an action, or what to avoid while doing it. For example, the following are part of instructions for installing a diverter spout on a bath tub faucet:

Install new spout. When doing so, DO NOT use lift knob or hose connection for leverage. Damage may result! Tighten by hand only.

2. The instructions may describe a range of behavior appropriate under different circumstances. The agent is only meant to do that which she recognizes the situation as demanding during its performance. For example, the following are part of the same instructions for installing a diverter spout:

Diverter spout is provided with insert for ½" pipe threads. If supply pipe is larger (¾"), unscrew insert and use spout without it.

In the above case, the relevant situational features can be determined prior to executing the instructions. In other cases, they may only be evident at the branch-point itself. For example, the following are part of instructions for filling holes in plaster over wood lath:

If a third coat is necessary, use prepared joint compound from a hardware store.

Here, the agent will not know if a third coat is necessary until she sees whether the first two coats have produced a smooth level surface.

3. As in the plan as data structure model, instructions may delineate actions at several levels of detail or in several ways. For example, the following are part

---

4. This is not the case in "Simon Says" type instructions, where each action description contains an intrinsic culmination (Moes and Steedman, 1988).
has disintegrated:

Clear away loose plaster. Make a new lath backing with metal lath, hardware cloth, or, for small holes, screen. Cut the mesh in a rectangle or square larger than the hole. Thread a 4- to 5-inch length of heavy twine through the center of the mesh. Knot the ends together. Slip the new lath patch into the hole...

Here the second utterance prescribes an action at a gross level, with subsequent utterances specifying it in more detail.

4. Some actions change significant features of objects; others result in their creation (or destruction). A referring term in an instruction often reflects the state of its referent at the point at which the instruction would be carried out. For example, in the following, the effect of the prescribed mixing action is to convert a powder into a paste.

Mix plaster compound according to package directions. With a flexible putty knife or scraper, force the thick creamy plaster into the opening.

Recognizing coreference and other relationships between referring expressions is necessary for understanding instructions (or producing understandable ones (Dale, 1988)). Thus, there is a level at which action-describing utterances in instructions must be modeled, in order to understand them, prior to behaving in accordance with them. Dale’s EPICURE system for generating natural language instructions contains such a “world model” (Dale, 1988).

5. Instructions may only provide circumstantial constraints on behavior but not specify when those circumstances will arise. For example, the following comes from instructions for installing wood paneling:

When you have to cut a sheet [of paneling], try to produce as smooth an edge as possible. If you’re using a handsaw, saw from the face side; if you’re using a power saw, saw from the back side. Otherwise you’ll produce ragged edges on the face because a handsaw cuts down, and a power saw cuts up.

Such cases as these illustrate an indirect relationship between instructions and behavior through the intermediary of an agent’s beliefs and evolving plan. That is,

Instructions \(\Rightarrow\) Beliefs \(\Rightarrow\) Plan \(\Rightarrow\) Behavior

### 3.3.4 Implementing These Ideas

If we adopt this plan as mental phenomenon view in our animation from instructions work, then not only must we be able to derive from the natural language instructions an appropriate set of beliefs, but we must also have an account of the behavior that follows from an agent’s beliefs and the intention of performing some action \(\beta\). The latter is the goal of much research in the plan as mental phenomenon paradigm. For the most part here, we hope to draw on advances made by others—for example, Pollack’s work on inferring what she has called simple plans, ones whose actions are only linked by Goldman’s generation relation (Goldman, 1970), from questions of the form

How do I do \(\beta\). I want to do \(\alpha\).

At the natural language end of our animation from instructions work, we have begun to analyze several constructions commonly found in natural language instructions (Webber and Di Eugenio, 1990), in order to expand the range of instruction texts from which we can accurately derive appropriate agent beliefs about the actions involved and their relationships. These beliefs can then be used to drive our (incremental) simulator. Changes in the (simulated) environment resulting from (simulated) actions on the agent’s part will then feed back through the agent’s (simulated) perceptions, thereby possibly changing the set of beliefs informing the agent’s decisions about subsequent actions.

In this way, we hope to produce veridical simulations of tasks performed in response to instructions, thereby enabling animation from instructions to be used more and more as a tool in task design and instruction. To this end, we will assume (at least initially) that the instructions cover all contingencies: If the simulator has to simulate an event that is impossible in the environment (because of unforeseen circumstances), the simulation will just stop, indicating why things cannot proceed. Not limiting ourselves in this way would require us to address the full scope of the AI planning problem, taking us away from a useful enterprise even with this limitation. For this reason, we will also be assuming there is only one intentional agent\(^5\) to avoid both the issues involved in achieving successful coordination (Grosz and Sidner, 1990) and the unforeseeable contingencies that can arise because of miscommunication, misunderstanding, and miscoordination among multiple, intentional agents.

### 3.3.5 Summary

So far, we have tried to make clear precisely what we mean when we use the term instructions and what we take to be the relationship between instructions, plans, and behavior embodied in our system design. In the next section, we will present this design, stressing the several architectural features that reflect this plan as mental phenomenon approach.

---

5. We distinguish three kinds of agents: intentional agents, mechanistic agents, and the world. Mechanistic agents, like washing machines and cars, can act and change the world, but only when acted on by an intentional agent or the world. The system we are developing assumes a single intentional agent, an independently changing world, and any number of mechanistic agents.
We have been working on the development of an animation from instructions system in the framework of the architecture shown in outline in Figure 3.2. Here we briefly describe the various components, with the representation produced by the semantic mapper discussed further in Section 3.5, the simulator, in Section 3.6, and the motion generators in Section 3.7. General human performance issues that cut across several of these components are highlighted in Section 3.8.

3.4.1 System Overview

In Figure 3.2, filled ovals represent data structures and boxes represent processes. The overall structure of the system is much like a pipeline in that natural language instructions enter at the top and complete animations emerge at the bottom. Unlike a pipeline, however, the system may be entered at any level and must be so designed for modularity, testing, and expansion. Portions of the system are used independent of the levels above. For example, the display process is a software base for direct graphical manipulation needed for real-time interactive task evaluation and geometric design, and the motion generators are the subject of ongoing algorithm development and refinement.

Various sorts of world and agent information are stored in knowledge bases; for diagrammatic simplicity we have omitted most of the specific connections and focused instead on general content categories such as the object knowledge base, the (geometric) workplace specification, the agent specifications, and so on. One exception is the explicit connection between natural language instructions and an incremental planning level, which involves the specific creation of task-related actions and conditions in the knowledge base. Note, however, that the diagram is not meant to imply that all knowledge is embedded in a single representation; rather, we have collected the different types to emphasize their presence and availability to all stages of the system.

The remainder of this section outlines the principal components of the system: the natural language processor, the incremental planner, the semantic mapper, the simulator, the motion generators, the display process, and the narrative planner and generator.

3.4.2 Natural Language Processor

When the natural language processor is complete, it will consist of a parser, semantic interpreter, and discourse process (the latter to resolve—with respect to a discourse model of the instructions rather than the actual environment—the referring expressions used in the instructions). Up to now, we have been using a rather simple bottom-up parser called BUP in our language-to-animation research (Gangal, 1985). There seems to be value now in going to a much simpler, lexicalized system (one in which the lexicon essentially is the grammar) such as one based on lexicalized tree adjunction grammars (TAGs) (Abelie and Schabes, 1989; Schabes and Joshi, 1988) or combinatory categorial grammars (CCGs) (Pareschi and Steedman, 1987; Steedman, 1989).

The output of natural language processing will be a set of descriptions of actions involved in performing the task, relationships among these actions, and conditions/constraints on their performance. The descriptions can be viewed as
either the beliefs of the animated agent or the beliefs of the system, which controls the animated agent much like a marionette: The effect is the same. The language of these descriptions must combine features of procedural programming languages (including conditional, iterative, and while constructions), features of first-order logic representations, and features of frame/schema representations.

In developing appropriate descriptions of the intended relationships between actions, the discourse processor will make use of ideas about tense, aspect, and temporal and contingent modifiers presented in (Moens and Steedman, 1988; Passonneau, 1988; Webber, 1988), as well as information contained in the system’s several knowledge bases.

3.4.3 Incremental Planner

At this point, the system must begin to make a connection between the action descriptions computed by the natural language processor and the behavior of its animated agent. Because, as has so often been noted, the world (here, the workplace) and even the agent’s capabilities can change over time, independent of the agent’s behavior and “intentions,” we have chosen to use an incremental planner (see Figure 3.2) to develop a plan over time much like an Earley parser (Earley, 1970) develops a parse tree over time: It focuses its analysis on an initial window that gradually moves to the right. Just as, at an intermediate step, the analyses computed by an Earley parser are global in assuming an eventual full analysis as a sentence but only partial in having completely analyzed only an initial substring, so the output of the incremental planner should be taken to be a partial global plan: It is global in reflecting the overall goal of “doing β,” but only partial in having elaborated in detail the actions related to its upcoming behavior.

To this end, the incremental planner must be able to augment the set of beliefs about the actions whose performance is being considered next with:

- beliefs about other actions that may not have been made explicit in the instructions (because they were in some way “obvious”), and
- the expansion of all the action descriptions to a level of detail and vocabulary appropriate for the semantic mapper.

The information required to elaborate beliefs about actions in this way comes from the system’s knowledge base, which includes information about actions, objects, the current state of the workspace, and the current state of the agent.

Before going on to describe briefly the properties and responsibilities of the semantic mapper, we must say a bit more about the way that incremental planning takes account of the current state of the “world” (here, the current state of the workplace and the agent). There is a two-step feedback loop from the motion generators to the simulator and from there, to the workplace specifications and agent specifications in the system’s knowledge base, both of which are consulted during the planner’s decisions about the next actions to consider. This enables

66 Badler et al.

“actual” performance features to percolate up through higher levels of symbolic interpretation and affect decisions at more than one level.

For example, some events and situations can only be detected by the motion generators because they are tied to particular instantiated behaviors and coincidental relationships discovered during motion execution. For example, later in Section 3.6 we will show a “preening” event, which is precipitated solely by the agent’s passing before a mirror. While not part of some overall plan, the interruption is as much a part of the animated behavior as the task that caused the agent to accidentally pass the mirror in the first place. Likewise, conditions that depend on changing workplace characteristics can affect the selection or completion of events. For example, “apply a quantity of patching compound” will affect both the decisions of the simulator and the incremental planner: The simulator must decide whether or not to send the motion generators another “apply a quantity of patching compound” action; the incremental planner must monitor changes to the workplace specifications to determine the need for a third coat of patching, depending on whether the second coat has resulted in a smooth, level surface.

3.4.4 The Semantic Mapper

At any point, output from the incremental planner to the semantic mapper consists of the next elaborated actions from the partial global plan. As described in Section 3.5, the output of the semantic mapper is a set of parameterized event templates, which combine the temporal constraints, kinematic and dynamic features, and geometric constraints related to individual simulatable events. The mapper operates compositionally, building specifications of the features and constraints associated with the event description as a whole from those associated with its various parts (Kalita, 1990). A parameterized event template is therefore a specific instance of an event class (already known to the semantic mapper) where as much information as possible is filled in from the knowledge base and the given instructions in the partial global plan. The event classes known to the semantic mapper therefore define a unique lexicon that enables the meaning of an instruction to be expressed by animatable event primitives.

3.4.5 The Simulator

The simulator does several things: It solves a temporal constraint satisfaction problem (TCSP) among the current parameterized event templates, maintains an event queue, schedules the active events, and outputs basic animation commands, which are then interpreted by the motion generators.

6. Actions are of course specified to the motion generators as temporally ordered sequences of geometric, kinematic, and dynamic constraints that must be met—not as natural language utterances.
3.4.6 The Motion Generators

A basic animation command provides sufficient data to execute one of the available motion generators (Section 3.7). The current animation command set includes motion paths (as key parameters), end effector goals, forces and torques, geometric constraints, simple locomotion, and facial muscle actions. Each command is executed by a corresponding motion generator: for example, forward kinematics by parametric interpolation; reach goals by inverse kinematics or a strength-guided reach; forces and torques by dynamics; geometric constraints by inverse kinematics; and facial muscles by a facial action model.

Since multiple animation commands may affect the same body part we use a merging method to maintain physical consistency in the object location database (Dadamo, 1988). The selection of the weighting parameters for this model has not been extensively studied, and more robust methods will be examined. We expect that the modifiers used in the original natural language instructions will be realized in the most appropriate terms in the semantic mapper, thereby limiting conflict between alternative specifications.

The actual performance (motion, timing, and success) of basic animation commands will vary according to:

- different physical characteristics and anthropometry of the agents (e.g., strength, speed, size, workload capacity, etc.)
- different environmental characteristics (e.g., actual scene geometry, object placement)
- prior event processing (posture resulting from previous task),
- immediate event contingencies (e.g., collision avoidance).

68 Badler et al.

- external events in the world that bring about change independent of the agent, thereby undoing the needed effect of some action, eliminating the need for some action, or necessitating that a goal be achieved by some other procedure. If that procedure is contingent on some state holding in the world over its course.

The information for some of the performance models are stored within various knowledge bases (Section 3.8).

While executing a basic animation command, any of the unexpected occurrences will feed back information to be considered by the simulator. The result may change the evaluation of the current parameterized event templates during subsequent simulation cycles.

3.4.7 Display Process

The display process is based on Jack, a powerful interactive graphics system for the manipulation and display of articulated figures (Phillips and Badler, 1988). Jack is used to define and execute a visualization plan (see Figure 3.2), which is the sequence of scenes, cuts, and camera views used to show the agent action. Currently this is totally under manual control of the animator, although we intend to (semi)automate its production based on the agent plan augmented with the intention and focus of the action (Ridsdale et al., 1986). Establishing an effective visualization plan is nontrivial, future effort, requiring cinematic knowledge, artistic design, and (ultimately) understanding of visual communication. Currently, default views may be manually defined based on an observer agent who is in the scene but who may or may not be visible. Establishing the line of view and the size of the viewed scene can be accomplished through spatial constraints. Knowledge of the intentions of the agents is necessary to form an appropriate view (Feiner, 1985). The observer may need to be positioned by a language-based description or manual methods. Preliminary study of cinematography terms indicates that language-based control of the camera is feasible.

3.4.8 Narrative Planner and Generator

Because of the important role that narration plays in complementing animation, the system as a whole is being designed with a narrative planner and generator component firmly in mind. There are two reasons for this:

7. When the observer is visible, the resulting views are of great importance in task analysis—they show the world as a real observer would see it; i.e., with self perception of other body parts. The typical movie, however, uses an "invisible" camera disembodied from any part of the environment. The camera may be attached to something moving, but is itself unseen.
primitive action elements that a simulator can animate. In English, action descriptions are not conveyed solely by the main verb but are distributed over the verb, its arguments, and its modifiers. The work here assumes that a description of the action can be built compositionally from features associated with its component elements. A more detailed description of this work is given in (Kalita, 1990).

3.5.1 Relevant Features

For a significant class of natural language verbs, it appears that their link to simulatable action elements can be based on an analysis of the movements they represent, taking into consideration physical attributes alone. The attributes we consider are: geometric constraints, aspectual features, and kinematic-dynamic distinctions.

3.5.1.1 Geometric Constraints Geometric constraints provide information regarding how one or more objects or subparts of objects relate to one another in terms of physical contact, absolute or relative location, inter-object distance, absolute and relative orientation, or path of motion. Verbs dealing with geometric constraints may denote their establishment and maintenance (e.g., attach, hold, fix, grab, grasp, hook), their elimination (e.g., detach, disconnect, disengage, release) or their modification (e.g., loosen, tighten). These constraints may be positional or orientational.

1. Positional constraints: This refers to situations in which a 0-, 1-, 2- or 3-dimensional object is constrained to a 0-, 1-, 2- or 3-dimensional region of space. For example, in order to execute the command “Put the ball on the table,” a point on the surface of the ball has to be brought in contact with (or constrained to) a point on the surface of the table. To execute the command “Put the block in the box,” the volume occupied by the block must be constrained to the interior volume of the box.

2. Orientational constraints: The meaning of the preposition across in the sentence “Place the ruler across the table” requires, inter alia, that the longitudinal axis of the ruler and the longitudinal axis of the table top be perpendicular to each other.

3.5.1.2 Aspectual Components Aspect involves inherent semantic features of a lexical item pertaining to the temporal structure of the situation denoted by the lexical item, independent of context (Passonneau, 1988). These features include repetition and termination.

1. Repetition: Some verbs denote underlying motions that require repetitions of one or more submotions—for example, roll, screw, scrub, shake, rock. For other verbs, repetitions may or may not be performed (i.e., their performance depends on the object(s) involved, information gathered from linguistic input, etc.). Such verbs include cut, fill, face, load.
intrinsic terminal conditions reached in the normal course of events and beyond which the processes cannot continue. Some examples are: align, assemble, attach, close, detach, drop, engage, fix, fill, place, release.

Other verbs do not denote motions with intrinsic end conditions. The termination point may be determined by accompanying linguistic expressions, obtained through reasoning from commonsense knowledge or knowledge of the goal to be achieved, or defined by explicit feedback from simulation. Examples of such verbs are: hold, press, scrub, shake.

3.5.1.3 Kinematic–Dynamic Characterization of Actions Dynamics describes the force or effort influencing motion. Kinematics deals with direct path or goals, and motion specification. Often movements along the same spatial path and toward the same spatial goal may be represented by different verbs such as touch, press, and punch. These distinctions can be formulated in terms of dynamic specification.

1. Kinematic: These are verbs whose underlying motions can be expressed as a movement along an arbitrary path or at an arbitrary velocity. Some examples are turn, roll, rotate.

2. Dynamic: For a verb in this category, its underlying motion can be characterized by describing the force that causes it. Examples include: push, shove, pull, drag, wring, hit, strike, punch, press.

3. Both kinematic and dynamic: These are verbs that have strong path as well as force components: swing, grasp (vs. touch), twist.

In practice, more control over the execution phase of the motion is obtained if the need for dynamics is converted into a strength requirement for the agent. This replaces the problem of absolute force specification with a relative specification as a fraction of the maximum performance rate (Section 3.8).

3.5.2 Obtaining a Representation for the Verbs: Primitives

The semantic mapper must output sets of parameterized event templates describing the actions to be simulated. These templates have slots for geometric relations and constraints, kinematics, and dynamics information.

3.5.2.1 Geometric Relations and Geometric Constraints We specify geometric relations in terms of the following frame structure.

72 Badler et al.

Geometric-relation: spatial-type:
source-constraint-space:
destination-constraint-space:
selectional-restrictions:

Spatial-type refers to the type of the geometric relation specified. It may have one of two values: positional or orientational. The two slots called source-constraint-space and destination-constraint-space refer to one or more objects, parts or features thereof that need to be related. For example, in order to execute the command "Put the cup on the table," one normally brings the bottom surface of the cup into contact with the top surface of the table. The command "Put the ball on the table" requires bringing an arbitrary point on the surface of the ball in contact with the surface of the table top. Since the items being related may be arbitrary geometric entities (i.e., points, surfaces, volumes, etc.), we call them spaces; the first space is called the source space and the second the destination space. The slot selectional-restrictions refers to conditions (static, dynamic, global or object-specific) that need to be satisfied before the constraint can be executed.

Geometric constraints are geometric goals; they are specified as follows:

Geometric-constraint: execution-type:
geometric-relation:

Geometric constraints are of four types, distinguished by the execution-type component. The execution class or type of a constraint may be achieve, break maintain, or modify.

3.5.2.2 Kinematics The frame used for specifying the kinematic aspect of a motion is the following:

Kinematics: motion-type:
source:
destination:
path-geometry:
velocity:
axis:

Motions are mainly of two types: translational and rotational. In order to describe a translational motion, we need to specify the source of the motion, destination, the trajectory of the path, and the velocity of the motion. In the case of rotational motion, the path-geometry is always circular. The velocity, if specified, is angular. An axis of rotation should be specified; otherwise, it is inferred by consulting geometric knowledge about the object concerned.

3.5.2.3 Temporals and Aspectuals The central part of an object's motion consists of one or more components: dynamics, kinematics, and geometric constraints—along with control structures stating aspectual or other complexi
The keyword \textit{concurrent} is specified when two or more components need to be satisfied or achieved at the same time. The keyword \textit{repeat-arbitrary-times} provides a means for specifying the repetitiveness property of certain verbs. A verb’s semantic representation need not specify how many times a motion or submotion may need to be repeated. However, since every motion is presumed to end, the number of repetitions will have to be computed during the simulation (based on tests for some suitable termination conditions) or by inference, unless specified linguistically as in “Shake the block about 50 times.” Other temporal information is carried along from the partial global plan.

3.5.2.4 Representation of Verbal and Sentential Meaning Since our meaning representation is verb based, the template for the representation of the meaning of a verb is also the frame for representation of meanings of sentences. The representation for a sentence has the following slots:

\begin{verbatim}
Verbal-representation: agent:
  object:
  kernel-actions:
  selectional-restrictions:
\end{verbatim}

Selectional restrictions may refer to dynamic or static properties of objects or the environment.

3.5.3 An Example

Here we show how individual representations of the verb \textit{put} and a prepositional phrase headed by \textit{on} combine to link the action description “Put the block on the table” with executable actions.

3.5.3.1 Put: Establishing Positional Constraints Webster’s dictionary (Woollf, 1981) defines one sense of the meaning of the verb \textit{put} as “to place in a specified position or relationship.” We consider only the positional aspect of the meaning to obtain a lexical definition. The lexical entry is

\begin{verbatim}
put (l-agent, l-object, 1-locative)
\end{verbatim}

This representation tells us that \textit{put} requires us to achieve a positional constraint between two objects, parts or features thereof. It does not indicate the type of positional relation to be achieved. The details of the geometric relation to be achieved have to be provided by the locative expression used, which may be in terms of prepositions such as \textit{in}, \textit{on}, or \textit{across}.

3.5.3.2 On: Support by a Physical Object In our attempt to provide precise, implementable meanings of prepositions, we have been influenced by the definitions proposed by Badler (1975), Gangel (1983), and Herskovits (1986). They are currently limited in that they work with simple, solid, nondeformable geometric objects.

Among the senses of \textit{on} defined by Herskovits (1986) is the one we are interested in here: \textit{spatial entity supported by physical object}. Examples of its use are seen in sentences such as “Put the block on the table” and “Put the block on the box.” One support situation that is commonplace is where the located object rests on a free, horizontal, upward-facing surface of the reference object. This need not be a top surface of the reference object, though it almost always is an \textit{outer} surface (otherwise \textit{in} is preferred). We describe this meaning of \textit{on} as on \((X, Y)\)

\begin{verbatim}
  \textbf{geometric-relation:}
  \textbf{destination-location: positional}
  \textbf{destination-constraint-space:}
    \textbf{any-of} (supporter-surfaces-of \((Y)\))
  \textbf{spatial-type: positional}
  \textbf{source-constraint-space:}
    \textbf{any-of} (self-supporting-spaces-of \((X)\))
  \textbf{destination-constraint-space:}
    \textbf{any-of} (supporter-surfaces-of \((Y)\))
  \textbf{selectional-restrictions:}
    horizontal (destination-constraint-space)
    equal ((direction-of
      (normal-to destination-constraint-space) "global-up")
    area-of (source-constraint-space) \leq area-of
      (destination-constraint-space)
    free-p (destination-constraint-space)
\end{verbatim}

Given a geometric object, the geometrical function \textit{self-supporting-spaces-of} obtains a list containing surfaces, lines or points on the object on which it can support itself. For example, a cube can support itself on any of its six faces, and a sphere on any point on its surface. The function \textit{supporting-surfaces-of} finds out the surfaces of an object on which other objects can be supported. The functions \textit{direction-of}, \textit{normal-to}, and \textit{area-of} are self-evident. The two directional constants \textit{global-up} and \textit{global-down} are defined with respect to a global reference frame.
sentence consists of the action verb put, a prepositional phrase headed by on and a referent noun phrase. The meaning of the whole sentence, obtained by composing the meanings of its constituent parts, is as follows:

agent: "you"
object: block-1
kernel-action: geometric-constraint:
  execution-type: achieve
  spatial-type: positional
  geometric-relation:
    destination-constraint-space: (area, area)
      spatial-type: positional
      source-constraint-space: any-of
        (self-supporting-spaces-of (block-1))
      destination-constraint-space: any-of
        (supporter-spaces-of (table-1))
  selectional-restrictions:
    horizontal-p (destination-constraint-space)
    equal
      (direction-of
        (normal-to destination-constraint-space)
        "global-up")
    area-of (source-constraint-space) \leq
    area-of (destination-constraint-space)
    free-p (destination-constraint-space)

In order to execute the action dictated by this sentence, the system looks at the knowledge stored about the block to find a part or feature of the block on which it can support itself. It can be supported on any one of its faces and no face is more salient than any other for supporting purposes. A cube (the shape of the block) has six faces, and one is chosen randomly as the support area. The program searches the knowledge stored about the table for a part or feature which can be used to support other objects. It gathers that the table's function is to support "small" objects on its top, which is also horizontal as required by a selectional restriction. Finally, the system concludes that one of the sides of the cube has to be brought in contact with the top of the table.

3.6 The Simulator

Input to the simulator consists of instantiated parameterized event templates, which represent the tasks known to it (Esakov and Badler, 1990; Esakov, 1990). Instantiation involves the creation of a specific template from the class definition, which may be done either by the semantic mapper or by predefinition in the knowledge base.

The simulator has two basic parts: an event scheduler and a discrete event simulator. The event scheduler is a temporal constraint satisfier that binds any constraints on start, stop, or duration times given in the templates to suitable clock times. Our implementation allows constraint specifications to include "fuzziness" terms such as about, around, exactly, and preferred, which places a weight on the relative importance of the constraint. This binding then results in a partial ordering of the instantiated events. The discrete event simulation employs symbolic and quantitative world knowledge to "execute" the partially ordered events by mapping them to "basic animation commands" that actually drive the motion generation process.

An important feature of the simulator is that the progress of the motion generators (which are invoked "outside" the simulator—see Section 3.7) can in fact affect the course of the simulation. This is done through the use of daemons, which examine the motion database, and through the support of interrupts and continuations within an event template. This separation of motion from event simulation permits more modular and efficient construction of the motion generators and focuses temporal control and exception handling in the simulator.

Event Scheduler The event scheduler is governed by a clock, which represents the time within the simulated domain. Currently, the syntax used to represent time is based upon the 24-hour clock metaphor where a time is represented as a string of the form: HH:MM:SS.DDD (with as many decimal places as

10. Of course, the program could select the face which is already in the most appropriate orientation. It is therefore easy to see how modifiers such as sideways or upside-down could be interpreted as variations from this default.

11. For the sake of brevity, the examples do not represent the full functionality of the simulator, nor do they represent a semantically complete input set.
It is possible to indicate constraints on start, stop, and duration times of an event. These constraints can be to particular “clock” times or to the stop/start/duration times of another event. Furthermore, temporal adverbials (noted above) can be used to indicate the importance associated with achieving a particular temporal constraint.

The following two examples illustrate how temporal constraints (:time-constraints) in parameterized event templates (of some unspecified event class) are resolved to specific times and partially ordered by their start times. Only the relevant slots of the templates are shown here.

Example 1

```
(instantiate EVENT ()
  :instancename 'A
  :time-constraints ((start "10:00")
    (duration (about (5 min)))
    (end ((10 min) before (start b))))

(instantiate EVENT ()
  :instancename 'B
  :time-constraints ((duration (1 hour)))
    (end "11:00")))
```

The result of temporal constraint satisfaction here is:

<table>
<thead>
<tr>
<th>Event</th>
<th>Start</th>
<th>Duration</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>09:55:02.184</td>
<td>00:04:54.044</td>
<td>09:59:56.227</td>
</tr>
<tr>
<td>B</td>
<td>10:04:58.410</td>
<td>00:55:02.184</td>
<td>11:00:00.594</td>
</tr>
</tbody>
</table>

As can be seen, the event scheduler resolves the constraints to specific times at which the events should begin and end. In the second example, the qualifier *exactly* caused a shifting so that the constraint that task B should end at 11:00 is given a higher priority.

### 3.6.2 Discrete Event Simulation

Once temporal constraints are satisfied, the simulator may begin the actual advancement of its time clock to execute scheduled events. The discrete event simulation algorithm is based on an event queue from which the current (active) events are selected at each clock tick, then converted into basic animation commands (Esakov, 1990). The event queue is “dynamic” in that (1) as events are finished, they are deleted, and (2) as new events are instantiated and scheduled (as output of the semantic mapper, based on the evolving partial global plan), they are added to the queue in the correct partial order. Interruptions (by message-passing) from an event or a feedback signal from a motion generator may also add (or delete) events. In addition, the simulator is responsible for managing available resources so that, for example, the agent’s right arm is not given more than one task at a time.

Example 3 shows a single-direction information flow from the discrete event scheduler to basic animation commands. The task consists of generating the motion of the hands of a clock by changing orientation goals:

Example 3

```
(deftemplate show-time (start end)
  (default-duration (= end start))
  (precondition t)
  (runcondition t)
  (successcondition (= (send-message *clock* :real-clock) end))
  (on-success
   (progn
    (cancel-event-constraints)
    (instantiate yaps:show-time
      (send-message *clock* :real-clock)
      (+ (send-message *clock* :real-clock) 60)
      :instancename *clock
      :step '(60 sec)))))
```
This event template show-time has two parameters: start time and stop time. The default duration of an event is the difference between the two times. There is no precondition to firing this event and once fired, there is no condition that must be true during its duration (the "runcondition"). The definition of success of this event is that its simulation time equals its end time. Upon successful completion, the current animation command (achieve-orientation) constraints are cancelled and a new instantiation of the show-time template is created which will generate the motion for the next 60 seconds. The given time values are bound to the parameters upon instantiation, and temporal constraint satisfaction is trivially able to schedule this new instance of show-time for the next cycle.

The body of the template shows the achieve-orientation commands that are used to generate the motions for the second, minute, and hour hands of the clock. In a minute, the second hand will move 360 degrees, the minute hand 6 degrees and the hour hand a half of a degree. Were a digital clock to be used, a different set of animation commands would be used.

Example 4 demonstrates the interruption and continuation capabilities of the simulator, as well as a command for a more complex motion generator. Here a "walk" generator provides locomotion for an agent, and a daemon keeps track of when the agent passes in front of a mirror. When this occurs, an interrupt is generated temporarily blocking the "walk" event in favor of the "look in the mirror" event.

Example 4

deftemplate preen-daemon (figure reflector)
"Generate a series of preening commands for figure."
(variables daemon-name)
(precondition (not (in-front figure reflector))
(default-duration t)
(on-interuption
   ; An interrupt, must be because figure is in front of a reflector
   (progn
      (send-to-event-class :locomotion :interrupt)
      (instantiate preen (figure reflector)
         :on-success
            (progn
               (send-to-event-class :locomotion :continue)
               (send-message self :continue)))
      (on-continuation
        (setf daemon-name (generate-daemon :front-of figure reflector
                                      :interrupt-name :front))
      (on-deletion (delete-daemon daemon-name)
        (body
         (setf daemon-name (generate-daemon :front-of figure reflector
                                      :interrupt-name :front)))
      (deffunction walk (figure destination)
       "Generate commands for figure to walk to destination. The hard work is done in the walk motion generator."
       (event-class :locomotion)
       (precondition (not (at figure destination))
       (on-interuption
         (progn
            (cancel-event-constraints)
            (send-message self :block))
       (on-continuation (send-message self :unblock))
       (success-condition (at figure destination))
       (on-failure
        (send-to-parent self :failure :walk))
       (body
        (locomote figure (send-message destination
                         :full-geometric-reference))))

This example starts by defining a preen-daemon event template which monitors the motion database for a particular condition: front-of figure.

12. In this case the event templates are being defined manually with the knowledge base. Event templates will be generated via natural language and the semantic mapper from instructions such as "When John passes in front of a reflecting surface, he steps back to mirror himself."
3.7 Motion Generators

As the simulator schedules and executes the active events (in ignorance of the actual graphical frame times), it creates basic animation commands that invoke specific motion generators. Specifications that originally appeared as part of the body of the event template are passed through as a series of parameter values (for frame-to-frame interpolation), a force or torque, or a target location/orientation for an object. This information is processed by several motion generators. These tools can become quite sophisticated, generating and solving, for example, dynamics equations or constraint equations. Also, complex generation tools, such as a "walk" or a facial expression generator, could be used. In any case, however, the output of a motion generator is a complete binding of all the necessary object position, orientation, and shape parameters on a frame-by-frame basis. We have implemented six types of motion generators:

1. Forward kinematic-guided motion (key parameter interpolation),
2. Inverse kinematic-guided motion (constraint satisfaction),
3. Strength-guided motion,
4. Forward dynamics,
5. Walk,¹³
6. Facial expressions.

Forward kinematic-guided motion is used for the (create-dof-motion) simulator construct. This special case construct allows one to specify the end-time values to achieve for a parameter through the use of linear interpolation. The more general technique of specifying multiple value/time pairs is available as is the use of other interpolation algorithms (Steetee and Badler, 1985).

Inverse kinematic-guided motion uses a constraint solver to determine the configuration of a jointed chain of rigid segments given a goal position or orientation of the end-effector (Zhao and Badler, 1989). Positions may be specified as (sets of) points, lines, or planes, or surfaces of another object.

Strength-guided motion uses a strength model to determine the path that the end-effector will take to achieve a goal position (Lee et al., 1990). This algorithm is used whenever the end-effector (hand) is moving objects with mass. It makes use of parameters that define the "comfort" of a movement, mass of the object, and the strength of the end-effector chain (arm) (Gross et al., 1989).

Forward dynamics applies the specified forces or torques to any object, the agent, or any of their parts (Wilhelm, 1987; Oiani, 1989). Dynamics effects are primarily used to affect the workplace and agent as a whole, since the strength-

---

¹³ The walk generator is still under development.
Each motion generator performs appropriate path planning for the basic animation commands it interprets. Of course, some path planning must occur at higher levels in the global planning process. The criteria for the division of path planning is that the work done at the discrete event simulation stage should not impact the planning of other events. For example, planning the navigation of a complex path is more appropriately performed within the global planner (Ridsdale et al., 1986). This is because different events may need to be planned during the traversal of the path, which is divided into the single event of navigation into a series of separate (yet linked) events. When the simulator executes the instantiation of some event template, the expected result is the normal completion of the subtask within the larger global plan.

The motion generators produce the required agent or object motion paths for each basic animation command for each frame. For example, lifting a weight onto a shelf may be given as one of the event templates within some globally defined task. The strength-guided motion generator is responsible for computing the arm motion and the actual path: It is not supposed to establish preconditions (holding the weight), but it is entrusted with computing reasonable arm and torso motions to actually move the end-effector holding the weight along a spatial path to the goal position. If something happens (such as failure or confounding obstacles) to interrupt this motion, then control is returned to the simulator to determine alternative strategies within the active event template. The interruptions are determined by explicit feedback from resident daemons watching for changes in the knowledge base or conveying error signals from the motion generators.

The collection of motion generators is meant to allow convenient simulation of the scope and variety of human movement. Our own efforts have focused on task execution rather than locomotion-dependent activities. Examination of numerous motion representation systems (Badler et al., 1979; Badler, 1989) has led to the current set of motion generators and the belief that they form a relatively complete set for instruction-directed movement synthesis.

### 3.8 Agent Performance Issues

Natural language instructions for a task rarely specify how long the task takes. Rather than “pick up the box in five seconds,” one is more apt to hear “pick up the box [carefully, quickly].” That is, duration is left to the agent as a consequence of the agent’s abilities. One of the most fundamental concepts underlying the entire simulation approach to executing the parameterized event templates is that agent structure, capability, and behavior dictate most of the actual motion parameters and should therefore yield natural movements and individual varia-

---

14. which, if heard, is usually interpreted incoherently as “In five seconds, [begin to] pick up the box.”
These specifications alone, however, still require temporal duration information. The problem is that all of these specifications provide only some of the possible parameters of the agent motion. Performance models supply reasonable values for the remainder.

Several different kinds of performance models are accessible through various components of the knowledge bases:

**Body segment size:** Different length limbs and different sized bodies clearly affect the manner in which goals are reached, whether goals can be achieved without locomotion, whether an agent can fit into a space, etc.

**Joint limits:** Physical structure and clothing restrict motion, likewise affecting the manner of goal achievement and the postures available. In particular, joint limits force orientation goals to propagate rotations along the body linkage.\(^{15}\)

**Strength model:** Different agents have remarkably different strengths, which clearly affects what tasks each can do. Strength also affects how tasks are done. Any force requirements imposed by a task must affect the positioning of the body joints for best use of available strength (subject to the **comfort model**). For example, the task "pick up the box" will result in radically different body postures and motions if the box weighs one pound or 100 pounds.

**Comfort model:** Just because an agent has sufficient strength to perform some task does not mean that it will be accomplished in a fashion requiring maximum exertion. The comfort model tries to distribute loads on body joints so that the overall force load minimizes the ratio of required to available torque at each joint.

**Fatigue model:** There is an energy penalty associated with any motion. By accumulating the energy and allowing for reasonable recovery times, a fatigue factor can be computed, which scales the available strength.

**Task knowledge model:** Part of the higher level agent capability model is a list of the parameterized events the agent "knows" how to execute. We have seen that this knowledge effects the extent of plan elaboration. Also, different agents can have different task models (one agent's "carry" may allow only one object, while another's allows any stable stack of objects) and different methods for executing tasks in the global plan (one agent might "paint" in careful linear strips while another uses randomly oriented strokes).

\(^{15}\) Imagine someone grasping your hand and then twisting it a full turn

---

**Task timing model:** Fitts's Law (Fitts, 1954) gives some timing estimates for very simple reach and view events (Esakov and Badler, 1989). The strength-guided motion animation tool moves the end-effector at a rate consistent with the torque availability at the affected joints and therefore does not need a predetermined duration.\(^{16}\) Such an event is finished when the goal has been achieved; a daemon can report the success (or failure) back to the simulator.

**Resource allocation model:** Each human agent has two hands, two feet, one direction of view, etc. During event execution, resources must be monitored and allocated to avoid contention, overload, or deadlock. Resource requirements must be noted for each task. Different agents can have different resources available, for example, a robot arm and a human differ in the number of end-effectors and their capacity.

Each of these models has been implemented in our animation system. Although animations to date have been primarily of the "Simon Says" variety, the methodology has shown the usefulness and even necessity of performance models in filling in essential information linking instructions to action.

## 3.9 Conclusions

We have presented a comprehensive framework for the interpretation of natural language instructions and their subsequent execution by a synthetic agent. The entire process is viewed as a pipeline with feedback, including a variety of knowledge bases to support planning, semantic mapping, agent capabilities, and workplace geometry.

Proceeding from natural language processing through an incremental planer that would produce partial global plans to carry out the required tasks, we saw that an incremental planning approach is necessary to handle the requisite task variability and environmental conditions. Components of the global plan are then expressed as parameterized event templates by a semantic mapper, which essentially builds operational (executable) definitions of motion verbs and their modifiers. These templates are simulated to produce basic animation commands which in turn are directly executable by extant motion generators. The final frame-by-frame animation is viewed through a manually constructed visualization plan for manipulating the virtual camera so that animated images are displayed to the user on a Silicon Graphics 4D workstation.

We have many experiments to undertake and much more code to write, but our progress so far in integrating natural language instructions with the anima

\(^{16}\) An interesting experiment, as yet untried, is to compute the action times of the simulated agent with a reasonable strength model and determine how close the results mirror known Fitts's Law data.
Acknowledgments

This research is partially supported by Lockheed Engineering and Management Services (NASA Johnson Space Center), NASA Ames Grant NAG-2-426, FMC Corporation, Martin-Marietta Denver Aerospace, NSF CISE Grant CDA88-22719, and ARO Grant DAAL03-89-C-0031 including participation by the U.S. Army Human Engineering Laboratory. The authors would like to thank Mark Steedman for his comments and advice on earlier versions of this chapter.

References


Zeltzer, D. 1990. Task-level graphical simulation: abstraction, representation and

Zhao, J. and N.I. Badler. 1989. Real time inverse kinematics with joint limits and
spatial constraints. Technical Report MS-CIS-89-09, Department of Com-
puter and Information Science, University of Pennsylvania, Philadelphia, PA.