Animation From Instructions

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

Bonnie L. Webber
University of Pennsylvania, bonnie@inf.ed.ac.uk

Jeffrey Esakov
University of Pennsylvania

Jugal Kalita
University of Pennsylvania

Animation From Instructions

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 paper, we discuss the relationship between instructions and behavior that underlie our work and the overall structure of our system. We then describe in some what 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.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-17.

Animation From Instructions

MS-CIS-90-17
GRAPHICS LAB 33
LINC LAB 165

Norman I. Badler
Bonnie L. Webber
Jeffrey Esakov
Jugal Kalita

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

March 1990

Animation from Instructions

Norman I. Badler Bonnie L. Webber Jugal Kalita
Jeffrey Esakov
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389*

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 paper, 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.

1 Introduction

The enterprise we have embarked upon 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 emerging technology of *virtual reality* [29], 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.

*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 paper.

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 videotapes, 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 [19]. 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.

While red-slashed icons on warning signs may be effective in *reminding* people of what behavior is forbidden ("no smoking", "no wearing high-heeled shoes", even "no haunting"), 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 language.

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 (cf. Section 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 (i.e., 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/her capabilities and features of the given workplace. Veridical narration must reflect both the agent's actions and the conditions motivating them. This can be provided by an outside commentator, but he or 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 otherwise contingent relations between actions assumed by language), the *reasons* are separate for animation and narration, but the *conclusion* is the same. On the 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 which 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 whats and whys 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 and higher level programming languages and 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-a-vis the

[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 enable a widening of the user community beyond current manually skilled (or programming-wise) animators, to include for example, human factors engineers with tasks to design or evaluate and trainers with new personnel to instruct.

production of the narrative that accompanies animation (or even live instructional material) to clarify and explain the action.

This paper describes some of our work to date on producing narrated animations from Natural Language instructions. Its structure is as follows:

- Section 2 briefly describes previous efforts to connect Natutal Language instructions with computer graphics animations.
- Section 3 discusses instructions as given in Natural Language and characterizes computational models for understanding them.
- Section 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) agents.
- Section 5 elaborates the portion of the system between the planned actions and semantically valid primitives that may be characterized computationally.
- Section 6 describes the simulator and temporal constraint planner that organizes the primitives into a sequential stream of executable motions.
- Section 7 outlines the available motion generators.
- Section 8 summarizes agent performance issues that arise at various places in the system.
- Section 9 offers some observations and conclusions.

2 Background

Because there has been so little substantive work published (or, to our knowledge, done) on controlling and augmenting animation with Natural Language (but cf. [24, 32, 36, 71, 76]), 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 described by Zeltzer in his insightful analysis [91]. He recognized the need for planning and the requirement that tasks be executable as *skills* known to the animated agent. In 1983 we outlined a system designed to translate task descriptions into animation [5] that has evolved into the structure presented here.

Previous attempts to animate Natural Language have been weakened by a limited or arbitrary set of motion control schemes available to implement task semantics. Prior to 1978, Badler *et al* [10, 81] tried to design a suitable architecture but were stymied during implementation by large numbers of particular animation process problems requiring clarification and solution [9]. Later, problems of locomotion control, essential to action of a mobile agent, were addressed by others [18, 34, 90] yielding managable models. Inverse kinematics for end-effector goal positioning were adopted from robotics or invented for biomechanical models [8, 34, 50, 49, 52, 93].

More recently, dynamics simulation have allowed objects to move in physically correct motions [3, 40, 43, 44, 55, 86, 87], and geometric constraints resolved by kinematic, force, or energy considerations have freed animators from having to specify the details of object trajectories [13, 17, 33, 88, 89].

Zeltzer [90], Reynolds [67], and Ridsdale [68] have explored behavioral models which tried to control the complexity of interaction between many individuals or parts of the same object. By putting together enough animation processes, the Thalmanns have even used "synthetic actors" to animate the personalities, expressions, and actions of life-like figures of Marilyn Monroe and Humphrey Bogart [78].

Coming even closer to Natural Language-level instructions, high level action descriptions have been compiled into simple animations based on a small number of pre-defined actions [22, 30, 76] or motor skills [56, 92], and Berk *et al* [15] have used English to specify colors for animation, and Becket [14] has used English to select and modify texture map parameters, to make more realistic and "imperfect" pictures.

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 [65, 66], Suchman [75], and Agre and Chapman [2]. What this view of plans gives us is a simple and uniform way to talk about instructions: it does not, by itself, solve the problem of fully linking understood 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 to be described in Section 3.3.

3.1 Plan as Data Structure

In [65], 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 [69] are a clear example of this ***plan as data structure*** view. Given a goal to achieve (i.e., 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 labelled 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.

To a great extent, this is the view of plans that has been taken by Mellish and Evans [53], by Dale [21] and by Feiner and McKeown [28] in their respective work on generating Natural Language instructions from plans. Feiner and McKeown's system COMET uses schemata hand-encoded from radio repair instructions and produces coordinated graphics images and Natural Language texts to convey instructions that are demonstrably clearer and more understandable than either Natural Language or images alone ever could be. Although Mellish and Evans produce text based on (non-linear) plans, the text structures they produce diverge from the structure of the corresponding plans, in order to cleanly separate a specification of what needs doing from a justification of why it needs doing in a particular way or a particular order. We will have more to say about Dale's system EPICURE in Section 3.3.

3.2 Plans as Mental Phenomena

The alternative view of plans presented by Pollack [65, 66] is the ***plan as mental phenomenon*** view, which builds upon earlier ideas about plans put forth by Bratman [16]. In this view, having a plan to do some action β 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 β 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 β.

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(β), where β 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 s/he 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 α_1 must be performed before some other action α_2 in order to do action β_1 or that α_2 must be performed before α_1 in order to do β_2. 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. S/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 [77], plan inference, and 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 s/he develops appropriate beliefs,[3] which the agent may then draw upon in attempting to "do β". 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 upon in deciding what to do next.

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,

$$\textbf{Instruction} \Rightarrow \textbf{Behavior}$$

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 [24]. Key frames from an animated simulation of two agents (John and Jane) at a control panel following an instruction sequence that begins

John, look at switch twf-1.
John, turn twf-1 to state 4.
Jane, look at twf-3.
Jane, look at tglJ-1.
Jane, turn tglJ-1 on.

[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 depreciate X using revised schedule Y"), one might also assume they are meant to instill beliefs about particular occurences as well.

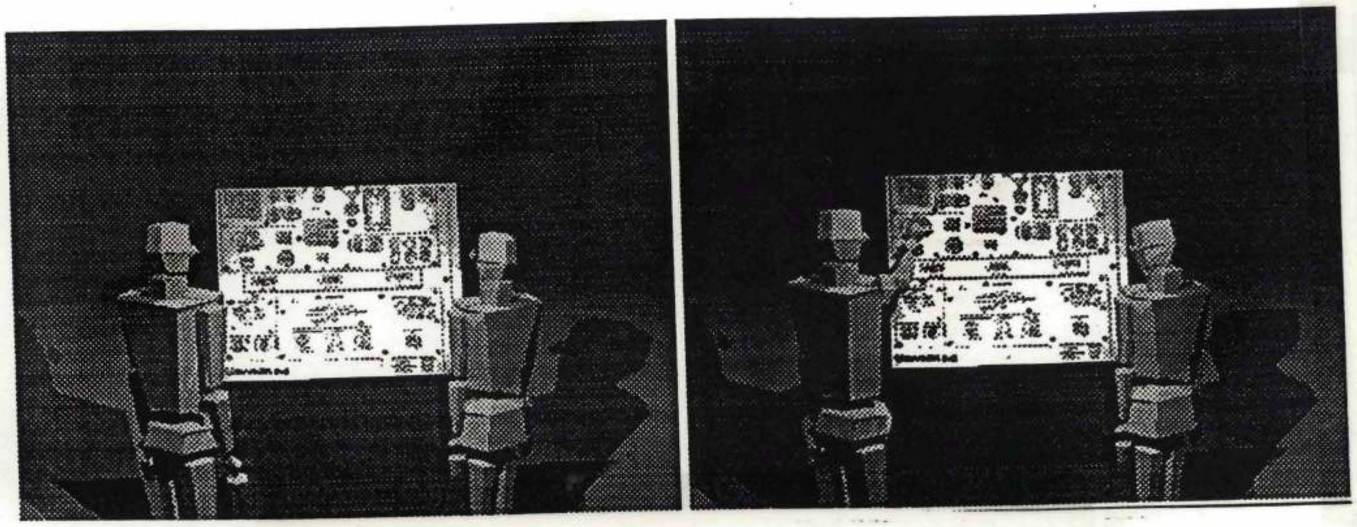

Figure 1: Control Panel Animation

are shown in Figure 1.

In contrast, other common instruction sets (especially ones composed as texts rather than given incrementally "on-line") depart from this "Simon Says" model in many ways, including:

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.[4] A simple example of this appears in the instructions that Agre and Chapman [2] 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] left up the hill" may have an intrinsic end point (i.e., 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 s/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.

[4]This is not the case in "Simon Says" type instructions, where each action description contains an intrinsic culmination [54].

2. The instructions may describe a range of behavior appropriate under different circumstances. The agent is only meant to do that which s/he 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 1/2" pipe threads. If supply pipe is larger (3/4"), 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 branchpoint 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 s/he 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 of instructions for filling holes in plaster where the lath has disintegrated as well as the plaster:

> 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). It has often been observed that a referring term in an instruction 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 co-reference and other relationships between referring expressions is necessary for understanding instructions (or producing understandable ones [21]). Thus, there is a level at which action-describing utterances in instructions must be *modelled*, 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" [21].

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* relation between instructions and behavior through the intermediary of an agent's beliefs and evolving plan. That is,

$$\textbf{Instructions} \Rightarrow \textbf{Beliefs} \Leftrightarrow \textbf{Plan} \Leftrightarrow \textbf{Behavior}$$

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, we must have an account of the behavior that follows from an agent's beliefs and the intention of performing some action β. The latter is the goal of much research in the *plans as mental phenomenon* paradigm. For the most part here, we hope to draw upon 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 [35], from questions of the form

How do I do β. I want to do α.

At the Natural Language end of our *animation from instructions* work, we have begun to analyze several constructions commonly found in Natural Language instructions [84], 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, which 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 [39] and the unforeseeable contingencies that can arise because of mis-communication, mis-understanding, and mis-coordination among multiple intentional agents.

3.5 Summary

In this section, 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* that are embodied in our system design. In the next section, we will present this design, stressing the several architectural features that reflect this *plans as mental phenomena* 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 upon 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.

4 System Design

We have been working on the development of an *animation from instructions* system in the framework of the architecture shown in outline in Figure 2. Here we briefly describe the various components, with the representation produced by the *Semantic Mapper* discussed further in Section 5, the *Simulator*, in Section 6, and the *Motion Generators* in Section 7. General human performance issues that cut across several of these components are highlighted in Section 8.

4.1 System Overview

In Figure 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 independently 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 on-going algorithm development and refinement.

Various sorts of world and agent information is 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 together 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*.

4.2 Natural Language Processor

When the Natural Language Processor is complete, it will consist of a parser, semantic interpreter, and discourse processor (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 [32]. 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) [1, 70] or Combinatory Categorial Grammars (CCGs) [59, 72].

The output of Natural Language Processing will be a set of descriptions of actions involved in performing the task, relationships among those actions, and conditions/constraints on their performance. The descriptions can be viewed as either the beliefs of the animated agent or the

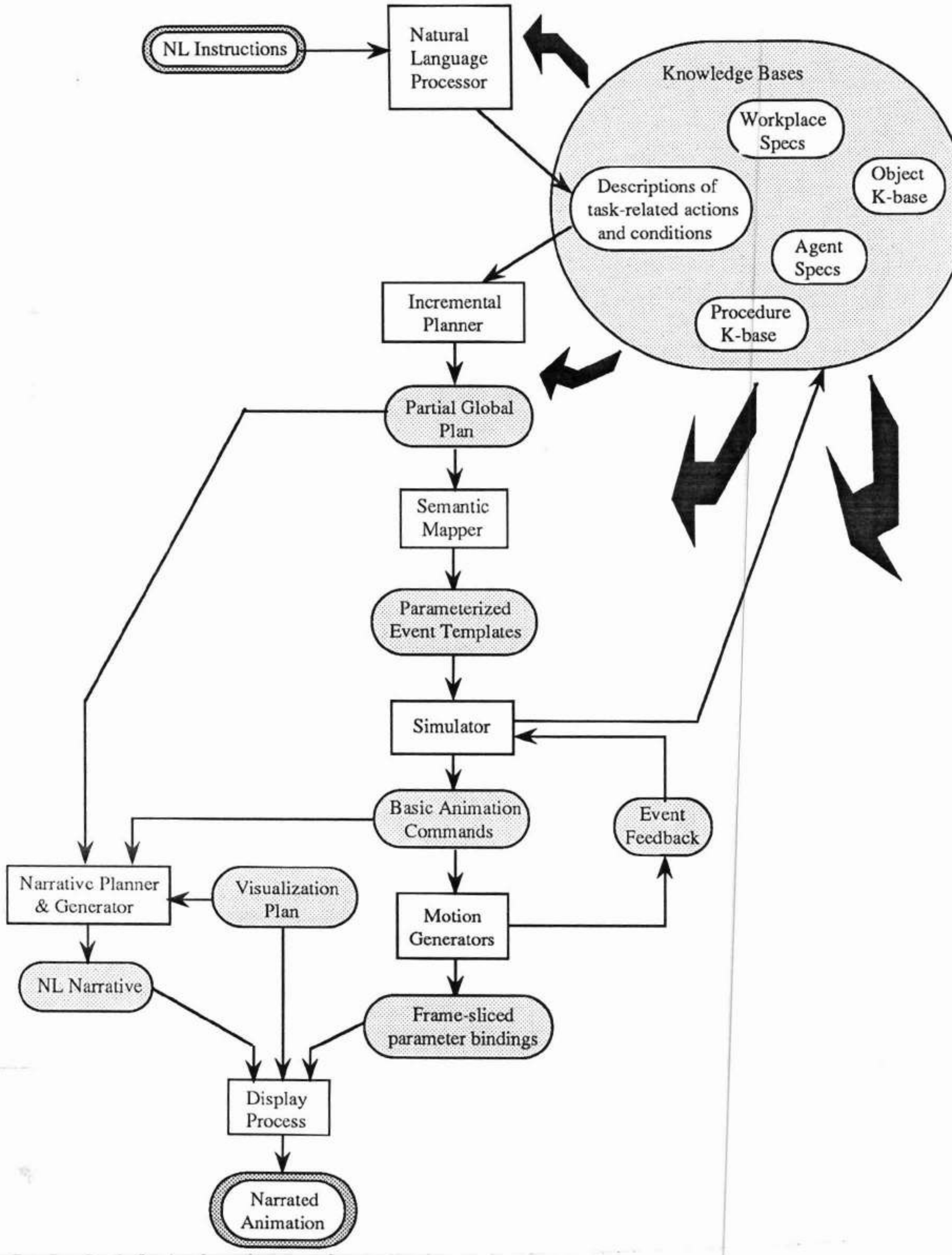

Figure 2: Design Framework

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 [54, 60, 83], as well as information contained in the system's several knowledge bases.

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* (cf. Figure 2) to develop a plan over time much like an Earley parser [23] develops a parse tree over time: it focusses its analysis in an initial window that gradually moves rightward. 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 analysed 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 have 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* back to the *Simulator* and from there, back 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 "actual" performance features to percolate back 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* since they are tied to particular instantiated behaviors and coincidental relationships discovered during motion execution. For example, later in Section 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[6]; 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.

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 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 [46]. 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 comprise a unique lexicon which enables the meaning of an instruction to be expressed by animatable event primitives.

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.*

The temporal CSP involves both imposing consistency on the elements of the *temporal constraint set* (which may reference any nodes of the *partial global plan*) and "percolating" consistent relations down to the leaf nodes. The solution then is a partial linear ordering of the events that make up the leaves of the *partial global plan.* Our current TCSP solution methodology uses spring-like constraints to model the temporal relations and an iterative numerical process to obtain a candidate partial ordering [7].

The *simulator* itself runs a conventional discrete simulation cycle by maintaining an event queue, incrementing a discrete clock, and interpreting the currently active *parameterized event templates.* During event simulation, changes to the world model (geometry, resources, agent capability, etc.) may result in the activation of other event templates leading to contextually-dependent future actions. Feedback signals from the *Motion Generators* may also affect current events: for example, spatial difficulties (such as collisions) or agent capability failures (such as

[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.

insufficient strength). Such features are mostly missing from other knowledge base simulation schemes [41].

4.6 The Motion Generators

A ***basic animation command*** provides sufficient data to execute one of the available *Motion Generators* (Section 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 [20]. 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 due 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),
- external events in the world that bring about change independently of the agent, thereby undoing the needed effect of some action, eliminating the need for some action, or necessitating 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 8).

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

4.7 Display Process

The *Display Process* is based on **Jack**, a powerful interactive graphics system for the manipulation and display of articulated figures [62]. **Jack** is used to define and execute a ***visualization plan*** (see Figure 2), which is the sequence of scenes, cuts, and camera views used to show the agent action. Presently 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 [68]. Establishing an effective *visualization plan* is non-trivial future effort, requiring cinematic knowledge, artistic design, and (ultimately) understanding of visual communication. Presently, default views may be manually defined based on an observer-agent who is in the scene but who may or may not be visible[7]. 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 [27]. 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.

4.8 Narrative Planner and Generator

Because of the important role that narration plays in comlementing 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:

- Without text explaining the *reasons* for observable actions, as researchers in plan recognition have noted [19], such actions may be incomprehensible to observers. Exaggerating behavior to make it more communicative may have the adverse effect of making it less veridical[8]. Sharing the burden of communication between Natural Language and graphics, as Feiner and McKeown have noted [28], takes advantage of the best of both possible worlds.

- A frequent criticism levelled at automatic text generation is that it requires hand-crafted representations as input: even the increased customizability offered by text generation systems has not been accepted as an adequate counter-argument to this criticism. Here, narration and animation are driven from common representations – a combination of the representations produced in the context of understanding the original instructions, plus the *Visualization Plan* used in producing the animation.

To satisfy the joint goals of providing an overall context in which to view the events on the "screen" and explaining the reasons for particular events that are happening, the *Narrative Planner and Generator* require information from the *partial global plan* (for the "whys"), the *basic animation commands* (for particulars of what's happening "now") and the *visualization plan* (for what can the viewer see – in particular, what is centrally visible as opposed to being off-center or invisible, as the latter may have to be brought to the viewer's attention verbally, through the narration).

[7]When the observer is visible, the resulting views are of great importance in task analysis as 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.

[8]A situation inversely turned advantageous by the skilled cartoon animator [79].

4.9 Summary

This overview has shown the connections and scope of each of the components of the system. In particular, we described the major "pipeline" of information from the language instructions through an incremental planner to a representational level semantically interpreted and then simulated. The output of the simulator is a time-ordered sequence of explicit animation commands to be executed by various motion generators. Additional issues of information feedback to guide and control both the incremental planner and simulator were mentioned. A visualization plan and a narrative generator format the final presentation to the viewer.

5 Between Action Descriptions and Actions

Of fundamental importance to driving animation from Natural Language is an appropriate interface between Natural Language descriptions of actions and the 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 [46].

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 ones we consider are: geometric constraints, aspectual features, and kinematic/dynamic distinctions.

5.1.1 Geometric constraints

Geometric constraints provide information regarding how one or more objects or sub-parts 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.

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 [60]. These features include repetition and termination.

Repetition: Some verbs denote underlying motions that require repetitions of one or more sub-motions – 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, lace, load.

Termination conditions: Some verbs denote underlying motions which have intrinsic terminal conditions that are 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 or by context-dependent, task-dependent criteria such as default resulting states of affected object(s), or obtained through reasoning from commonsense knowledge, or knowledge of the goal to achieve, or defined by explicit feedback from simulation. Examples of such verbs are: hold, press, scrub, shake.

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 which causes it. Examples include: push, shove, pull, drag, wring, hit, strike, punch, press.

3. Both kinematic and dynamic: These are verbs which have strong path as well as force components: swing, grip/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 8.

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[9].

5.2.1 Geometric relations and geometric constraints

We specify geometric relations in terms of the following frame structure.

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* and *orientational.* The two slots called *source-constraint-space* and *destination-constraint-space* refer to one or more objects, or parts or features thereof which 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. They are distinguished by the *execution-type* component. The execution class or type of a constraint may be *achieve, break, maintain* or *modify.*

5.2.2 Kinematics

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

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

[9]The dynamics case is not discussed here as it is presently being elaborated both in terms of classical physical forces and as agent strength requirements [52].

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, its 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.

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 complexities involved in the execution of an action. The constructs we presently use are: *repeat-arbitrary-times* and *concurrent*. The keyword *concurrent* is specified when two or more components need to be satisfied or achieved at the same time. The keyword *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 sub-motion 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 fifty times.* Other temporal information is carried along from the *partial global plan.*

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.

```
Verbal-representation: agent:
                       object:
                       kernel-actions:
                       selectional-restrictions:
```

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

5.3 An Example

Here we show how individual representations of the verb *put* and a prepositional phrase headed by *on* combine to link the action description *put the block on the table* with executable actions.

5.3.1 *Put*: establishing positional constraints

Webster's dictionary [85] defines one sense of the meaning of the verb *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

```
put (l-agent, l-object, l-locative) ⟵ agent: l-agent
                                      object: l-object
                                      kernel-action:
                                          geometric-constraint:
                                              execution-type:achieve
                                              spatial-type:   positional
                                              geometric-relation: l-locative
```

This representation tells us that *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 *in, on* or *across.*

5.3.2 *On*: Support by a physical object

In our attempt to provide precise, implementable meanings of prepositions, we have been influenced by Badler [4], Gangel [32] and Herskovits [42]. They are currently limited in that they work with simple, solid, non-deformable geometric objects.

Among the senses of *on* defined by Herskovits [42] is the one we are interested in here: *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 which 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 *outer* surface (otherwise *in* is preferred). We describe this meaning of *on* as

```
on (X,Y) ⟵ geometric-relation:
                spatial-type:                  positional
                source-constraint-space:       any-of (self-supporting-spaces-of (X))
                destination-constraint-space:  any-of (supporter-surfaces-of (Y)))
            selectional-restrictions:
                horizontal (destination-constraint-space)
                equal ((direction-of (normal-to destination-constraint-space) "global-up")
                area-of (source-constraint-space) ≤ area-of (destination-constraint-space)
                free-p (destination-constraint-space)
```

Given a geometric object, the geometrical function *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 *supporting-surfaces-of* finds out the surfaces on an object on which other objects can be supported. The functions *direction-of, normal-to, horizontal-p* and *area-of* are self-evident. The two directional constants *global-up* and *global-down* are defined with respect to a global reference frame.

5.3.3 Composing descriptions: *Put the block on the table*

The 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:
                         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) ≤ 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[10]. 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.

5.4 Semantic Mapper Output

The *Semantic Mapper* produces *parameterized event templates*, which are instances of *event classes* in a pre-defined hierarchy known to the *Semantic Mapper* and the *Simulator* via the shared *Knowledge Base*. Thus the *Semantic Mapper* outputs instances of the event classes with certain kinematic, temporal, etc., constraints. We have seen that some constraints come from the *Knowledge Base* workplace model, some aspectuals provide temporal constraints from the *partial global plan*, and other slots are dependent on the class definition of the verb.

[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.

6 The Simulator

Input to the *simulator* consists of instantiated *parameterized event templates*, which represent the tasks known to it [24, 26]. *Instantiation* involves the creation of a specific template from the class definition, which may be done either by the *Semantic Mapper* or by pre-definition 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 which 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 – cf. Section 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 focusses temporal control and exception handling in the simulator.

6.1 Event Scheduler[11]

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 desired), or as a quoted LISP form: *'(n id)*, where *n* is a number and, *id* is **hr**, **min**, **sec** representing hours, minutes and seconds, respectively.

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)))
```

[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.

```
                                  (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:

Event	Start	Duration	End
A	09:56:16.121	00:04:55.522	10:01:11.641
B	10:07:27.762	00:56:16.119	11:03:43.879

Example 2:

```
(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 (exactly "11:00"))))
```

The result of temporal constraint satisfaction is now:

Event	Start	Duration	End
A	09:55:02.184	00:04:54.044	09:59:56.227
B	10:04:58.410	00:55:02.184	11:00:00.594

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.

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 [26]. 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))))
  (on-deletion
   (cancel-event-constraints))
  (body
    (let ((minutehand (send-message *clock* :minute-hand-orientation))
          (hourhand (send-message *clock* :hour-hand-orientation)))
      (if (= minutehand 360) (setf minutehand 0))
      (if (= hourhand 360) (setf hourhand 0))
      (achieve-orientation (send-message *clock* :second-hand-referent)
                           360 :startvalue 0)
      (achieve-orientation (send-message *clock* :minute-hand-referent)
                           (+ minutehand 6) :startvalue minutehand)
      (achieve-orientation (send-message *clock* :hour-hand-referent)
                           (+ hourhand .5) :startvalue hourhand))))
```

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 which 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-interruption
    ;; 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))))

(deftemplate 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-interruption
    (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[12] a **preen-daemon** event template which monitors the motion database for a particular condition: **front-of figure reflector**. If that condition occurs, the daemon will interrupt this event instance and delete itself so that it will not continually report the same condition.

The template **walk** is a member of the event class called **:locomotion**. When an interrupt (as generated by an instance of **preen-daemon**) is received by an instance of this **walk** template, the associated animation commands are cancelled and the event blocks. The **:block** message causes the interrupt to be applied to each of the parent events of a sub-event. Conversely, when an event sends a continuation signal and unblocks, the message travels up to the parent events. When an event is continued, the body is re-evaluated; however, it is not always necessary to re-execute an entire event hierarchy. The hierarchy tree is traversed (up toward the parents) and re-executed from the event highest in the tree with a false precondition. Thus our agent may continue to walk, but not necessarily exactly on the same cycle, at the same pace, or even to the same target that was active on interruption.

The input to the simulator must fully specify how to handle interrupts. That is, each event within a simulation must explicitly define its interrupts and the corresponding handler. There are many different types of daemons which can be defined. Daemons such as *in front of*, *next to*, *to the left of*, *to the right of* are typical, and new daemons are being defined as needed. Presently our simulator restricts a given daemon to monitor only a single type of event with a single set of parameters. For example, while it is not possible to define a single daemon to monitor whether *any* agent passes in front of a mirror (as in the previous section), it is possible to define multiple daemons, each monitoring a single agent's relationship with a mirror.

When a condition occurs, the motion generator triggers an interrupt within the simulator. The interrupt is specific to the event in which the corresponding daemon was generated. One parameter to the interrupt is the interrupt name (as specified in the `(generate-daemon)` construct) which allows the event to distinguish between multiple types of interrupts and act accordingly. In particular, this event template should be modified so that an interruption will cause a graceful shut-down of the walk event and not an immediate freeze.

6.3 Summary

This section has described the process of converting *parameterized event templates* into sequences of *basic animation commands*. First a temporal constraint satisfaction method takes the current set of events and schedules them into a appropriate partial ordering. Then a discrete event

[12]In this case the event templates are being defined manually with the Knowledge Base. Eventually such 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 stops to preen himself.*

simulator steps through the active event queue and outputs animation commands contained in the instantiated event templates. Exceptional conditions may be monitored by *daemons* whose outputs feed back to the simulator, potentially altering its active event queue and hence its execution trace.

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 [73].

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 [93]. 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 which the end effector will take to achieve a goal position [52]. This algorithm is used whenever the end-effector (hand) is moving objects with mass. It makes use of parameters which define the "comfort" of a movement, mass of the object, and the strength of the end-effector chain (arm) [38].

Forward dynamics applies the specified forces or torques to any object, the agent, or any of their parts [87, 58]. Dynamics effects are primarily used to affect the workplace and agent as a

[13]The walk generator is still under development.

whole, since the strength-guided motion is more controllable and efficient for task achievement. Typically task instructions do not specify forces directly; rather, motions result from internal *strength* and *joint limit* capabilities of the agent. The motions are much less like a marionette and much more like a real agent working with muscles against gravity. The dynamics simulation process is primarily used to move passive objects (for example, a thrown ball) or the entire agent, especially when such forces are the consequence of actions initiated by the agent. For example, executing a rapid turn while driving a car will cause centrifugal force to act on the body, altering its posture and, quite importantly, the direction of external forces which its strength model must resist.

The walk generator is based on Bruderlin and Calvert's dynamics and constraint model [18]. In our present applications it suffices to have limited point-to-point agent mobility. More complex locomotion tasks or sporting events have not been undertaken.

Facial expressions are executed based on a muscle model of the face derived from an extensive analysis of facial regions. Motion control is effected by the selection, modification, or removal of Action Units based on the Ekman and Friesen model [64].

These five generators yield a surprisingly versatile collection of ***basic animatable tasks***:

- reach, touch, place, position [via inverse kinematics]
- orient, align [via inverse kinematics]
- attach, unattach, grasp, ungrasp [via topological connections in the underlying geometric database of objects]
- put, take [like reach, but with attached object]
- look at [via inverse kinematics using orientation]
- lift, push, pull [via strength-guided motion]
- stand up, sit down [via strength-guided motion]
- follow path [via forward kinematics]
- exert force or torque [via forward dynamics]
- walk along a path, walk to a goal point [via walk generator]
- produce facial expressions from speech intonation and emotion [implementation in progress [61]
- walk [implementation in progress based on [18]]

All of these constructs can be interrupted at any point throughout the motion. If that is done, the object maintains its current state and the motion generator essentially "forgets" the animation command.

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 [68]. This is because different events may need to be planned during the traversal of the path which in effect would divide the single event of navigation into a series of separable (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 pre-conditions (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 [11, 6] has led to the present set of motion generators and the belief that they form a relatively complete set for instruction-directed movement synthesis.

8 Agent Performance Issues

Natural Language instructions for a task rarely specify how long it takes. Rather than "pick up the box in 5 seconds"[14], 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 variations among different agents. We have already noted the roles of inverse kinematics, strength, and constraints in formulating a complete animation interface. 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.

[14] which, if heard, is usually interpreted inchoatively as "In 5 seconds, [begin to] pick up the box"

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 strength, 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 1 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).

Task timing model: Fitts' Law gives some timing estimates for very smple reach and view events [24]. 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 pre-determined duration [16]. Such an event is finished when the goal has been achieved; a daemon can report the sucess (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.

[15] Imagine someone grasping your hand and then twisting it a full turn!

[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' Law data.

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 planner 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 animation of a responsive human-like agent has been very encouraging. Connections between these disparate fields are being made which promise to deeply affect them both.

References

[1] Anne Abeille and Yves Schabes. Parsing idioms in lexicalized TAGs. *Proceedings of the 4th Meeting of the European Chapter of the Association for Computational Linguistics.* Manchester, England, April 1989.

[2] Phillip Agre and David Chapman. What are Plans For? A.I. Memo 1050a, Artificial Intelligence Laboratory, MIT, October 1989.

[3] W. Armstrong, M. Green, and R. Lake. Near-real-time control of human figure models. *IEEE Computer Graphics and Applications*, 7(6):52–61, June 1987.

[4] Norman I. Badler. *Temporal scene analysis: Conceptual descriptions of object movements.* University of Pennsylvania, Computer and Information Science, Technical Report MS-CIS-76-4. (Also PhD dissertation, Department of Computer Science, University of Toronto, 1975).

[5] Norman I. Badler, Bonnie L. Webber, James U. Korein, and Jonathan D. Korein. TEMPUS, A system for the design and simulation of mobile agents in a workstation and task environment. *Proc. IEEE Trends and Applications Conference* 1983: 263-269.

[6] Norman I. Badler. A representation for natural human movement. In *Dance Technology I*, J. Gray, (ed.), AAHPERD Publications, Reston, VA: 23-44, 1989.

[7] Norman I. Badler, Scott Kushner, and Jugal Kalita. *Constraint-based temporal planning.* Technical Report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1988.

[8] Norman I. Badler, Kamran Manoochehri, and Graham Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, 7(6):28–38, June 1987.

[9] Norman I. Badler, Joseph O'Rourke, and Bruce Kaufman. Special problems in human movement simulation. *Computer Graphics*, 14(3):189-197, July 1980.

[10] Norman I. Badler, Joseph O'Rourke, Stephen Smoliar, and Lynne Weber. *The simulation of human movement by computer.* Technical Report, Department of Computer and Information Science, University of Pennsylvania, PA, August 1978.

[11] Norman I. Badler, Joseph O'Rourke, and Hasida Toltzis. A spherical representation of a human body for visualizing movement. *IEEE Proceedings*, 67(10):1397-1403, Oct. 1979.

[12] Norman I. Badler, and Stephen W. Smoliar. Digital representations of human movement. *ACM Computing Surveys*, 11(1):19-38, March 1979.

[13] Ronan Barzel and Alan Barr. A modeling system based on dynamic constraints. *Computer Graphics* 22(4), 1988.

[14] Welton Becket and Norman I. Badler. Imperfection for Realistic Image Synthesis. Technical Report, Computer and Information Science Dept., University of Pennsylvania, Philadelphia, PA, January 1990.

[15] Toby Berk, Lee Brownston, and Arie Kaufman. A new color-naming system for graphics languages. *IEEE Computer Graphics and Applications*, 2(3):37–44, May 1982.

[16] Bratman, M. Taking Plans Seriously. *Social Theory and Practice*, 9:271-287, 1983.

[17] Lynne S. Brotman and Arun N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, 1988.

[18] Bruderlin, A. and T. W. Calvert. Goal-Directed, Dynamic Animation of Human Walking. *Computer Graphics* 23(3), 233–242, 1989.

[19] Phil Cohen. The Need for Referent Identification as a Planned Action. *Proc. of International Joint Conference on Artificial Intelligence*, August 1981, pp. 31-36,

[20] Diana T. Dadamo. Effective control of human motion animation, Technical Report MS-CIS-88-52, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1988.

[21] Robert Dale. *Generating Referring Expressions in a Domain of Objects and Processes.* PhD Thesis, University of Edinburgh, Edinburgh, Scotland, 1988.

[22] Karin Drewery and John Tsotsos. Goal-directed animation using English motion commands. *Proc. Graphics Interface '86*, Vancouver, 131–135, 1986.

[23] Earley, J. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* 13(2), 1970, 94-102.

[24] Jeffrey Esakov and Norman I. Badler. An Architecture for Human Task Animation Control. In *Knowledge-Based Simulation: Methodology and Applications* P.A. Fishwick and R.S. Modjeski (eds.), Springer Verlag, New York, 1989.

[25] Jeffrey Esakov, Norman I. Badler, and Moon Jung. An investigation of language input and performance timing for task animation. *Proc. Graphics Interface '89*, Waterloo, Canada, 1989, 86-93.

[26] Jeffrey Esakov. An Architecture for Human Task Performance Analysis. PhD Thesis, Dept. of Computer and Information Science, University of Pennsylvania, August 1990 (expected).

[27] S. Feiner. APEX: An Experiment in the Automated Creation of Pictorial Explanations. *IEEE Computer Graphics and Applications*, 5(11):29–37, November 1985.

[28] Feiner, S. and McKeown, K. Coordinating Text and Graphics in Explanation Generation. *Proc. ARPA Speech and Natural Language Workshop*, October 1989, Los Altos CA: Morgan Kaufmann, pp. 424-433.

[29] S.S. Fisher, M. McGreevy, J. Humphries and W. Robinett. Virtual Environment Display System. *Proc. 1986 Workshop on Interactive 3D Graphics*, Chapel Hill NC, ACM Publications, 1986.

[30] Paul A. Fishwick. *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction.* PhD thesis, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1986.

[31] Paul A. Fishwick. The role of process abstraction in simulation. *IEEE Trans. Systems, Man, and Cybernetics*, 18(1):18–39, Jan/Feb. 1988.

[32] Jeffrey S. Gangel. *A motion verb interface to a task animation system.* Master's thesis, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, August 1985.

[33] Francesco Gardin and Bernard Meltzer. Analogical representations of naive physics. *Artificial Intelligence*, 38(2):139–159, March 1989.

[34] Michael Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures, *Computer Graphics*, 19(3):263–270, 1985.

[35] A.I. Goldman. *A Theory of Human Action.* Princeton University Press, Princeton NJ, 1970.

[36] Mark Green and Hanqui Sun. A Language and System for Procedural Modeling and Motion. *IEEE Computer Graphics and Applications*, 8(6):52–64, November 1988.

[37] H. Paul Grice. Logic and Conversation. In *Syntax and Semantics 3: Speech Acts*, New York: Academic Press, 1975.

[38] Marc R. Grosso, Richard D. Quach, Ernest Otani, Jianmin Zhao, Susanna Wei, Pei-Hwa Ho, Jiahe Lu and Norman I. Badler. Anthropometry for computer graphics human figures. Technical Report, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1989.

[39] Barbara Grosz and Candace Sidner. Plans for Discourse. In P.R. Cohen, J.L. Morgan and M.E. Pollack (eds), *Intentions in Communication*. Cambridge MA: MIT Press, 1990.

[40] James K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, 1988.

[41] Gary Hendrix. Modeling simultaneous actions and continuous processes. *Artificial Intelligence*, 4:145–180, 1973.

[42] Annette Herskovits. *Language and Spatial Cognition* Cambridge University Press, 1986.

[43] C. Hoffmann and J. Hopcroft. Simulation of physical systems from geometric models. *IEEE Journal of Robotics and Automation*, RA-3(3), June 1987.

[44] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, *Computer Graphics*, 21(4):215–224, 1987.

[45] J.K. Kalita and N. I. Badler. Semantic analysis of action verbs based on animatable primitives. Technical Report, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1989.

[46] J.K. Kalita. Analysis of Some Actions Verbs and Synthesis of Underlying Tasks in an Animation Environment. Forthcoming Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania.

[47] Robin Karlin. *SEAFACT: A semantic analysis system for task animation of cooking operations.* Master's thesis, Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, December 1987.

[48] Robin Karlin. Defining the semantics of verbal modifiers in the domain of cooking tasks. In *Proceedings of the 26st Annual Meeting of ACL*, pages 61–67, 1988.

[49] James U. Korein. *A Geometric Investigation of Reach.* MIT Press, Cambridge, MA, 1985.

[50] James U. Korein and Norman I. Badler. Techniques for goal directed motion. *IEEE Computer Graphics and Applications*, 2(9):71-81, November 1982.

[51] Philip Lee. Kinematic paths from a strength and comfort model. PhD proposal, Dept. of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, 1989.

[52] Philip Lee, Susanna Wei, Jianmin Zhao and Norman I. Badler. Strength guided motion. To appear, *Computer Graphics*, 1990.

[53] C. Mellish and R. Evans. Natural Language Generation from Plans. *Computational Linguistics* 15(4), December 1989, pp.233-250.

[54] Marc Moens and Mark Steedman. Temporal Ontology and Temporal Reference. *Computational Linguistics.* 14(2), 1988, pp. 15-28.

[55] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 23(4):289–298, 1988.

[56] P. Morasso and V. Tagliasco (Eds.). *Understanding Human Movement.* North-Holland, New York, 1986.

[57] Joseph O'Rourke and Norman I. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Trans. PAMI*, 2(6):522-536, November 1980.

[58] Otani, Ernest. Software Tools for Dynamic and Kinematic Modeling of Human Motion. Master's thesis, Department of Mechanical Engineering, University of Pennsylvania, Technical Report, MS-CIS-89-43, Philadephia, PA, 1989.

[59] Remo Pareschi and Mark Steedman. A lazy way to chart-parse with categorial grammars. *Proc. 25th Annual Meeting of the Association for Computational Linguistics*, Stanford CA, pp.81-88, July 1987.

[60] Rebecca Passonneau A Computational Model of the Semantics of Tense and Aspect. *Computational Linguistics.* 14(2), 1988.

[61] Catherine Pelachaud. *A 3D animation system for facial expression, emotion, and intonation.* Forthcoming PhD Dissertation, Department of Computer and Information Science, University of Pennsylvania, 1990.

[62] Cary Phillips and Norman I. Badler. Jack: A toolkit for manipulating articulated figures. In *ACM/SIGGRAPH Symposium on User Interface Software*, Banff, Canada, 1988.

[63] Cary Phillips, Jianmin Zhao and Norman I. Badler. Interactive real-time articulated figure manipulation using multiple kinematic constraints. To appear, *Symposium on Interactive 3D Graphics*, March 1990.

[64] Stephen Platt. *A structural model of the human face.* PhD Dissertation, Department of Computer and Information Science, University of Pennsylvania, 1985.

[65] Martha Pollack. *Inferring Domain Plans in Question-Answering.* PhD Thesis, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia PA. (Available as Technical Report MS-CIS-86-40, University of Pennsylvania, May 1986.)

[66] Martha Pollack. Plans as complex mental attitudes. In *Intentions in Communication*, J. M. P. Cohen and M. Pollack, Eds., MIT Press, 1990.

[67] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model, *Computer Graphics*, 21(4):25–34, 1987.

[68] Gary Ridsdale, S. Hewitt and T. W. Calvert. The interactive specification of human animation, *Proc. Graphics Interface '86*, Vancouver, Canada, 1986: 121-130.

[69] Earl Sacerdoti. *A Structure for Plans and Behavior* Elsevier, New York, 1977.

[70] Yves Schabes & Aravind Joshi. An Earley-type parsing algorithm for tree adjunction grammars. *Proc. of the 26th Annual Meeting of the Association for Compuational Linguistics*, Buffalo NY, June 1988, pp.258-269.

[71] Robert F. Simmons and Gordon Bennett-Novak. Semantically analyzing an English subset for the clowns microworld. *American J. of Computational Linguistics*, Microfiche 18, 1975.

[72] Mark J. Steedman. Grammar, Interpretation, and Processing from the Lexicon. In W. Marslen-Wilson (ed.), *Lexical Representation and Process.* Cambridge MA: MIT Press, 1989.

[73] Scott Steketee and Norman I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control, *Computer Graphics* 19(3):255–262, 1985.

[74] P.F. Strawson. On Referring. *Mind* 59:320–344, 1950.

[75] Lucy Suchman. *Plans and Situated Actions: The problem of human machine communication.* Cambridge University Press, 1987.

[76] Yosuke Takashima, Hideo Shimazu, and Masahiro Tomono. Story driven animation. *CHI + GI '87 Proceedings*, 149-153, ACM SIGCHI, 1987.

[77] Josh Tenenberg. Inheritance in Automated Planning. *Proc. First Int'l Workshop on Knowledge Representation and Reasoning*, Toronto Canada, May 1989, pp.475-485.

[78] Nadia Magnenat-Thalmann and Daniel Thalmann. The direction of synthetic actors in the film *Rendez-vous a Montreal IEEE Computer Graphics and Applications*, 7(12):9–19, December 1987.

[79] Frank Thomas and Ollie Johnston. *Disney Animation: The Illusion of Life.* Abbeville Press, New York, 1981.

[80] Zeno Vendler. *Linguistics in Philosophy.* Cornell University Press, Ithaca NY, 1967.

[81] Lynne Weber, Stephen Smoliar, and Norman I. Badler. An architecture for the simulation of human movement. *Proc. ACM National Conf.*, 737-745, Washington, DC, Dec. 1978.

[82] Bonnie Webber. The Interpretation of Tense in Discourse. *Proc. 25th Annual Meeting of the ACL.* Stanford University, Stanford CA, July 1987. pp. 147-154.

[83] Bonnie Webber. Tense as discourse anaphor. *Computational Linguistics.* 14(2):61-73, 1988.

[84] Bonnie Webber and Barbara Di Eugenio. Free Adjuncts in Natural Language Instructions. *Proc. of COLING-90.* University of Helsinki, Finland. August 1990.

[85] Henry Woolfe (ed.). *Webster's New Collegiate Dictionary.* G. & C. Merriam Company, Springfield MA, 1981.

[86] Jane Wilhelms. Virya - a motion editor for kinematic and dynamic animation. *Proceedings Graphics Interface '86*, 141–146, Vancouver, 1986.

[87] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, June 1987.

[88] Andrew Witkin, Kurt Fleisher and Alan Barr. Energy constraints on parameterized models, *Computer Graphics*, 21(3):225–232, 1987.

[89] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, 1988.

[90] David Zeltzer. Motor control techniques for figure animation, *IEEE Computer Graphics and Applications* 2(9):53–59, Nov. 1982.

[91] David Zeltzer. Toward an integrated view of 3-D computer animation. *The Visual Computer: The International Journal of Computer Graphics*, 1(4):249-259, 1985.

[92] David Zeltzer. Task-level Graphical Simulation: Abstraction, Representation and Control. *Making Them Move: Mechanics, Control and Animation of Articulated Figures.* Morgan-Kaufmann, 1990.

[93] Jianmin Zhao and Norman I. Badler. Real time inverse kinematics with joint limits and spatial constraints. Technical Report MS-CIS-89-09, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1989.