January 2001

Cognitive Modeling for Computer Animation: A Comparative Review

Liwei Zhao
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-01-02.

Cognitive Modeling for Computer Animation: A Comparative Review

Abstract

Cognitive modeling is a provocative new paradigm that paves the way towards intelligent graphical characters by providing them with logic and reasoning skills. Cognitively empowered self-animating characters will see in the near future a widespread use in the interactive game, multimedia, virtual reality and production animation industries. This review covers three recently-published papers from the field of cognitive modeling for computer animation. The approaches and techniques employed are very different. The cognition model in the first paper is built on top of Soar, which is intended as a general cognitive architecture for developing systems that exhibit intelligent behaviors. The second paper uses an active plan tree and a plan library to achieve the fast and robust reactivity to the environment changes. The third paper, based on an AI formalism known as the situation calculus, develops a cognitive modeling language called CML and uses it to specify a behavior outline or "sketch plan" to direct the characters in terms of goals. Instead of presenting each paper in isolation then comparatively analyzing them, we take a top-down approach by first classifying the field into three different categories and then attempting to put each paper into a proper category. Hopefully in this way it can provide a more cohesive, systematic view of cognitive modeling approaches employed in computer animation.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-01-02.

Cognitive Modeling for Computer Animation: A Comparative Review

Liwei Zhao
Center for Human Modeling and Simulation
University of Pennsylvania, PA 19104-6383, USA
lwzhao@graphics.cis.upenn.edu

Abstract

Cognitive modeling is a provocative new paradigm that paves the way towards intelligent graphical characters by providing them with logic and reasoning skills. Cognitively empowered self-animating characters will see in the near future a widespread use in the interactive game, multimedia, virtual reality and production animation industries. This review covers three recently-published papers from the field of cognitive modeling for computer animation. The approaches and techniques employed are very different. The cognition model in the first paper is built on top of Soar, which is intended as a general cognitive architecture for developing systems that exhibit intelligent behaviors. The second paper uses an active plan tree and a plan library to achieve the fast and robust reactivity to the environment changes. The third paper, based on an AI formalism known as the situation calculus, develops a cognitive modeling language called CML and uses it to specify a behavior outline or "sketch plan" to direct the characters in terms of goals. Instead of presenting each paper in isolation then comparatively analyzing them, we take a top-down approach by first classifying the field into three different categories and then attempting to put each paper into a proper category. Hopefully in this way it can provide a more cohensive, systematic view of coginitive modeling approaches employed in computer animation.

1 Introduction

Modeling for computer animation addresses the challenge of automating a variety of difficult and complex animation tasks. An early milestone was the combination of geometric models and inverse kinematics to simplify the laborious keyframing. The computer maintains a representation of how parts of model are linked together and the constraints are enforced as the objects are pulled around. This frees the animator from, necessarily, having to move every part of an articulated figure individually.

Similarly, using the laws of physics can free the animator from implicitly trying to emulate them when they generate motion. Physically-based models such as Witkin *et al.* (1987) [53] and Barzel & Barr (1988) [54] view the world as objects and constraints. Constraints connect objects together through desired geometric relationships or keep them in space. Otherwise, they float in space under the appropriate laws of physics (Badler *et al.* 1993) [3]. Passive objects, such as falling chains, colliding objects, rigid bodies, deformable solids, gases and fluids can be animated using physically-based models (Foster & Metaxas 1996; Metaxas 1996) [15, 35] . For animate objects biomechanically-based modeling can be employed. So far, it has been possible to use simplified biomechanical models to automate the process of locomotion learning in a variety of virtual creatures, such as fish, snakes, and some articulated figures (Tu 1996; Grzeszczuk & Terzopoulos 1996) [50, 21]. While the physically-based and biomechanically-based modeling can model and simulate characters accurately and realistically, they are usually too computationally expensive to be used in real-time animation without any preprocessing.

Research in behavioral modeling is making progress towards self-animating characters that react appropriately to perceived environment stimuli. The seminal work in this area was that of Reynolds (1987) [38]. His "boids" have found extensive applications throughout the games and animation industry. Recently the work of Badler *et al.* (1993) [3], Tu and Terzopoulous (1994) [49], Blumberg and Galyean (1995) [6], Hodgins *et al.* (1995) [22], Unuma *et al.* (1995) [51], and Thalmann *et al.* (1998) [48] has extended this approach to dealing with some complex behaviors for more sophisticated characters. This works well for low-level behaviors. For animations of specific high-level behaviors, things are more complicated because many of the high-level behaviors exhibited by the characters suffer from the problems of being hard-wired into the code thus are very hard to be reconfigured or extended.

1.1 Cognitive Modeling

Some researchers address these problems by introducing cognitive modeling (henceforth, CM) as the next logical step in the hierarchy of models that have been used for computer animation.

Building cognitive models is very much a research area at the forefront of artificial intelligence (AI) research and psychological research. The research in AI overlaps considerably with cognitive science. Many researchers in AI try to model their computer programs after human intelligence, and they derive inspiration from modeling human cognition (Stillings *et al* 1987, p.9) [47]. In psychological research, a cognitive model serves as a vehicle for understanding human behavior. If your model is successful at producing human-like behavior under certain assumptions, you can hypothesize that different behavior will emerge under different assumptions, change those assumptions in the model and see how it behaves. Explorations with models in this way can then be used to design experimental conditions that are likely to show measurable effects. So, CM is useful for AI scientists and psychologists, but why should computer animators care about CM?

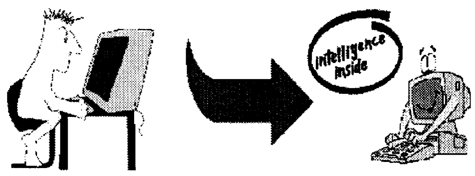


Figure 1: Shifting the Burden of the Work

Because computer animators are very interested in shifting the burden of the work to intelligent agents that have the following properties:

- **Autonomy**: agents operate without direct intervention and have some kind of control over their actions and internal states;
- **Reactivity**: agents perceive their environment and respond in a timely fashion to changes that occur in it;
- **Pro-activeness**: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative;
- **Interactivity**: agents interact with humans in real-time, which could include natural language understanding capabilities.

CM can play critical subsidiary roles in simulating these properties. Therefore, computer animators believe CM, by raising the level of abstraction at which the user can direct animated agents, makes it easier to produce animations. This level of functionality is obtained by enabling the agents themselves to do more of the work rather than the animator.

Hence, the goal of CM for animation is to have agents behave autonomously and intelligently in the virtual environment by emulating the perceptual, thinking, and/or motor processes a human goes through to complete a task. However, it does not necessarily imply that a cognitively empowered agent should get the job done with the least effort or in the least time. She may take the same amount of time that a human takes to perform a task. She may make the same kind of errors a human makes. She may require the same type of experience to learn to perform a task. She may do the same inefficient fumbling for a solution to a difficult problem.

1.2 Challenges

1.2.1 Fundamental Definitions

The task of developing a general cognitive model for compute animation or virtual reality (VR) games is a daunting task, greatly complicated by the lack of a consensus on some fundamental definitions such as intelligence and cognition.

Newell's definition of intelligence (Newell 1990) [36] implies that it is the knowledge within the system that makes it intelligent. From Newell's definition it can be inferred that in order to see if a system exhibits intelligence, it is important to study its content, and not its computational model (Jona & Schank 1993) [27]. However, it is important to note that it is the computational model which allows the implementation of the content. Also according to Newell (1990) [36], a system is intelligent to the extent that it uses the knowledge it possesses. Taking this notion to the extreme implies that a simple system operating in a small domain is more intelligent than a human operating in a large domain (Fehling 1993) [14]. Laird and Rosenbloom in their response (1993) [30] to this criticism, on behalf of Newell, seem to think that this is an issue of generality, and not intelligence. This debate suggests that intelligence is dependent upon the domain and subject in question. Maybe there is no universal definition of intelligence, as it really is a highly relative and subjective concept.

What cognition really is also is highly debated among proponents of two distinct approaches. One approach, the tradition upon which cognitive science was founded, is that of symbolic representation and processing (Vera & Simon 1993) [52]. The other more recent approach, emphasizing the role of the environment, the context, the social and cultural setting, and the situations in which characters find themselves, is variously called situated cognition (Greeno & Moore 1993; Agre 1993; Suchman 1993; and Clancey 1993) [20, 2, 46, 13]. The supporters of situated cognition tend to emphasize the importance of social interaction and environment and to minimize the importance of internal cognition; while the proponents of the traditional symbolic approach tend to downplay the importance of these social and external factors and to emphasize the importance of internal cognition. The dispute is really one of differing worldviews, but it defies attempts to produce a single universally accepted definition of cognition.

1.2.2 Fundamental Questions

Current computational cognitive models in computer animation demonstrate nothing like the intelligence and cognition and the efflorscence of adaptation evident in human or other agents' behaviors and such a result seems very far away. To make any progress, researchers must set goals much lower and hope to move gradually to the intelligent ideal.

However, this lowering of goals results in great deviations among the research community when answering some fundamental questions:

- What types of environments should be anticipated?
  - *Static Simulated Environments*, consist of unchanging surroundings in which an agent navigates and manipulates. The agent does not need to adapt to new situations, nor do its designers need to concern themselves with the issue of inconsistencies of the world model. An example of such an environment is a simulated office setting, where the doorways and halls never change, and there are no moving objects that populate the simulated space. Nothing changes in the static environment except through the action of the agent.
  - *Dynamic Simulated Environments*, change over time independent of the actions of the agents. They usually factor out the uninteresting variables and allow the agents to focus on the critical issues. However, due to the dynamic and unpredictable changing of the environment, agents that operate in the dynamic simulated enviornment require robust sensing/perception mechanisms and

high-level capabilities such as planning and learning. They may be even required to produce new plans rapidly based on updated sensory/perceptual information. They may have to reason about the temporal aspects of their plans.

- What capabilities are most important?
  - Capabilities taking place within the agent
    * Planning
    * Replanning
    * Learning: explanation-, abstraction-, or caching-based, etc.
    * Problem solving
    * Support for multiple simultaneous goals
    * Deductive and/or inductive reasoning
    * Self reflection
  - Capabilities related to interaction with the environment
    * Sensing and perception
    * Natural language understanding
    * Query answering and providing explanations
    * Prediction
    * Navigational strategies
  - Capabilities related to Execution
    * Real-time execution
    * Goal reconstruction
    * Focused behavior and selective attention
    * Responding intelligently to interrupts and failures
- What properties should be included?
  Agent properties identify and entail the techniques and methods that are used to realize a particular modeling component. For example most cognitive models include some sort of memory. Agent properties characterize the memory: Is the memory declarative, procedural, or episodic? Are there size limitations? Is memory uniformly accessed? Is it uniformly organized?

A research team's answers to these questions greatly influence its cognitive models. Unfortunately, no answer is completely satisfying or justified. For instance, most researchers consider the ability of planning fundamental to intelligence. However, the subsumption designers (Brooks 1986) [7] purposely ignore this capability. Both sides provide compelling arguments for their choices, so the final decision is highly subjective.

1.3 Overview

The remainder of the paper is organized as follows. Section 2 reviews the background of different CM strategies, taking into account the most important theoretical and practical issues. Here, our primary goal is to classify various CM approaches into three big categories: deliberative models, reactive models and layered models. The classification serves as a framework for the rest of the paper. Section 3 presents Steve, an animated pedagogical agent that helps students learn to perform physical and procedural tasks. Steve's cognition is based on the Soar model which is intended as a general deliberative cognitive architecture for developing systems that exhibit intelligent behaviors. Section 4 presents Hap, a reactive cognition model used in the Oz project to achieve fast and robust reactivity to world changes. Section 5 presents a layered model which, based on the situation calculus, develops a cognitive modeling lanaguage called CML and uses it to direct the characters in terms of goals. Section 6 presents the analytical conclusions we have drawn from our previous examinations, followed by a brief summary. This paper is an animation-centric review instead of an investigation on the AI side. We restrict our interests to real-time computer animation.

2 Background: Theory and Practice

With all those difficulties and challenges presented in the first section, how does one proceed? First we need to systematically review the approaches and strategies available. So far, there are three different strategies: deliberative models, the classical strategies; reactive models, the alternative strategies; and layered models, the intermediate strategies.

2.1 Deliberative Model

The foundation upon which the deliberative model, actually the whole symbolic AI paradigm, rests is the physical-symbol system, formulated by Newell and Simon (1976). A physical-symbol system is built from a set of elements, called *symbols*, which may be formed into symbol structures by means of a set of relations. A symbol system has a memory capable of storing and retaining symbols and symbol structures, and has a set of information processes that form symbol structures as a function of sensory stimuli, which produce symbol structures that cause motor actions and modify symbol structures in memory in a variety of ways. The processes that encode sensory stimuli into internal symbols are called perceptual processes, and the processes that decode motor symbols into muscular responses are called motor processes. Perceptual and motor processes connect the symbol system with its environment, providing it with its semantics, the operational definitions of its symbols.

The deliberative model is a model that contains an explicitly represented, symbolic model of world, and in which decisions are made through logical reasoning and planning, based on pattern matching and symbolic processing. In the early days – not very long ago, studies of this model dominated the AI and cognitive science field. Progress was made on many fronts, including planning, problem solving, reasoning, and language. A substantial number of symbol systems have been constructed and tested, partially successfully, for their ability to simulate human thinking and learning over a wide range of task domains. But after the initial flurry of activity, progress slowed due to the various difficulties mentioned earlier.

2.1.1 First Order Logic

First Order Logic (henceforth, FOL) is the logical foundation for symbolic AI. The basic building blocks are terms for referring to objects, and predicates for referring to relations. Terms and predicates can be combined to make atomic sentences to state facts. Then atomic sentences can be used with logical connectives to construct complex sentences. FOL also contains two standard quantifiers, called universal ($\forall$) and existential ($\exists$), to express properties of entire collections of objects, rather than having to enumerate the objects by name. FOL can express anything that can be programmed.

More detailed discussion of FOL can be found in a number of books including *Logic for Computer Science* by Gallier (1986) and *Logical Foundations of Artificial Intelligence* by Genesereth and Nilsson (1987).

Situation calculus is used for describing the changing world in FOL. It conceives of the world as consisting of a sequence of situations, each of which is a "snapshot" of the state of the world. Situations are generated from previous situations by actions. Three major problems have to be solved for practical use of situation calculus: the frame problem, the qualification problem and the ramification problem.

2.1.2 Planning

It has long been assumed that planning will be a central component of any deliberative model. Perhaps the best-known early planning system was STRIPS (Fikes & Nilsson 1971). This system takes a symbolic description of both the world and a desired goal state, and a set of action operators, which characterize the preconditions and postconditions associated with each action. It then attempts to find a sequence of actions that will achieve the goal, by using a simple means-ends analysis (MEA), which essentially involves matching the postconditions of actions against the desired goal. The STRIPS planning algorithm was very simple, and proved ineffective and impractical on problems of even moderate complexity. Much effort was subsequently devoted to developing more effective and practical planning techniques. Major innovations include hierarchical planning (Sacerdoti 1974) [43] and nonlinear planning (Sacerdoti 1975) [44].

Researchers in computer animation, trying to create autonomous intelligent agents, have attempted quite a few planning schemes by implicitly adopting a symbolic representation model of the simulated world. One of the papers we are going to analyze in this review employs a hierarchical decomposition planning mechanism. So it is a good time now for us to briefly review the practical planning being used in computer animation.

- **Regression Planning**
  This approach builds a plan by chaining backward from the desired goal through an action that will achieve the goal to the preconditions that must be achieved for this action to be effective. This is due to the empirical belief that the search space has a smaller branching factor near the goal state than that of near the initial state therefore a backward search from the goal is more tightly constrained and more effective. One of the drawbacks of this approach is the initial steps of the final plan are the last ones that are constructed. This is hardly acceptable in real-time computer animation. To satisfy the real-time constraints some researchers (Kurlander & Ling 1995) [28] propose a modified version that precompiles the plan scripts into a finite-state machine prior to the executions. In this specific case the operators are animation scripts, and the programmers declare preconditions and postconditions that explain how each of the scripts depend on and modify states. The price to pay is that all the possible situations have to be hard-coded into the finite state machines as a preprocessing step.

- **Interleaving Planning**

Given the planning delays and the requirement of adaptability to a changing unpredictable environment, interleaving planning suggests that do some planning, switch to perform some of the planned actions and then continue the planning. The open question is how much planning to do before the planned action takes over. Wood's AUTODRIVE system (Wood 1993) [55] has interleaving planning agents operating in a highly dynamic environment (a traffic simulation).

- **Hierarchical Decomposition**
  Most of planners we surveyed have adopted the idea of hierarchical decomposition: that an abstract operator can be decomposed into a group of steps that forms a plan that implements the operator. The composite steps of this plan can be further decomposed into even more specific plans until they are fully decomposed into hierarchically organized primitive actions that can be directly executed. These decompositions can be stored in a library of plans for retrieval as needed (Russell and Norvig 1995) [42]. This is a typical Divide-and-Conquer strategy, assuming the combination of implementations of subgoals can achieve the implementation of the goal. The action steps in the plan can be either partially ordered or totally ordered, partial ordering is preferred, though, due to the principle of least commitment.

  However the partial order planning has to solve three prominent problems: the selection of multiple applicable actions; subgoal interactions, also known as Sussman Anomaly; and constraint binding. Due to these problems, in practice hierarchical decomposition with partial ordering is rarely used directly without modification. ItPlanS (Geib 1995) [18] restricts a strong linearity (no subgoal interactions) and total ordering in its incremental hierarchical planner to satisfy the constraints of rapid response and limited knowledge.

  One of the papers we will examine adopts this method with some additional requirements. To construct a plan, Steve demands a task knowledge definition from the course author. Basically it is an augmented partial-ordering planner with substantial control knowledge to discourage unusual plans, and to guide the plan construction and revision. We will come to this point with more details when analyzing the planning model in the Steve paper.

However, in the mid 1980s, Chapman (1987) [12] established some theoretical results which indicate that even refined planning techniques will ultimately turn out to be unusable in any time-constrained system. These results have had a profound influence on subsequent planning research; perhaps more than any other, they have caused some researchers to question the whole symbolic AI paradigm, and thus have led to the work on alternative approaches that we will discuss in the next.

2.2 Reactive Model

As we mentioned above, there are many unsolved problems associated with symbolic AI. These problems have led some researchers to question the viability of the whole symbolic paradigm, and to the development of what are generally known as reactive models, which do not include any kind of central symbolic world model, and do not use any complex symbolic reasoning.

Probably the most vocal critic of symbolic AI has been Rodney Brooks. He has been arguing for over a decade that the road to intelligence consists of building situated agents which employ no explicit symbolic representation nor abstract reasoning (Brooks 1990; Brooks 1991a; Brooks 1991b) [8, 9, 10]. The "Creatures," as he called the humanoid robots he built at MIT, have a number of functionally distinct control layers that act independently. Sensors feed directly into distinct layers, each of which can react to the input with its own set of motor behaviors. Each layer has only the necessary information about the environment and the information is processed independently and in parallel. The lower layer represents more primitive behaviors (such as avoiding obstacles) and has precedence over layers further up the hierarchy. The higher layers subsume the roles of lower layers by suppressing their outputs. The system does not, at any point, have a centralized representation of its world.

Pattie Maes' competence modules loosely resemble the behaviors of Brooks' architecture (Maes 1991; Maes 1994) [33, 34]. Each module is specified by the designer in terms of preconditions and postconditions, and an activation level, which gives a real-valued indication of the relevance of the module in a particular situation. The higher the activation level, the more likely it is that the module will affect an agent's behavior. Once specified, a set of competence modules is compiled into a spreading activation network, in which the modules are linked to one another in ways defined by their preconditions and postconditions. For example, if module a has postcondition ϕ, and module b has precondition ϕ, then a and b are connected by a successor link. Other types of links include predecessor links and conflicter links. When an agent is executing, various modules may become more active in given situations, and may be executed. The result of execution may be a motor command.

Chapman and Agre (1986) [11] observed that most everyday activity is "routine", requiring little new abstract reasoning. Most tasks, once learned, can be accomplished in a routine way, with little variations. They proposed an idea that as most decisions are routine, they can be encoded into a low-level structure, which only need periodic updating, perhaps to handle new kinds of problems. Their approach was illustrated with the PENGI system (Agre and Chapman 1987) [1]. PENGI is a simulated computer game with characters controlled by "routines".

Rosenschein and Kaelbling (Rosenschein 1985; Rosenschein and Kaelbling 1986; Kaelbling and Rosenschein 1990; Kaelbling, 1991) proposed a situated automata in which agents are specified in declarative terms. This specification is then compiled down to a digital machine, which satisfies the declarative specification. This digital machine can operate in a provably time-bounded fashion; it does not do any symbol manipulation, and in fact no symbolic expressions are represented in the machine at all. An agent is specified in terms of two components: perception component RULER and action component GAPPS. RULER takes as input three components: a specification of the semantics inputs; a set of static facts; and a specification of state transitions. The programmer then specifies the desired semantics for the output and the compiler synthesizes a circuit whose output will have the correct semantics. All the declarative "knowledge" has been reduced to a very simple circuit (Kaelbling 1991). GAPPS takes as input a set of goal reduction rules (essentially rules that encode information about how goals can be achieved), a top level goal, and generates a program that can be translated into a digital circuit in order to realize the goal. The generated circuit does not represent symbolic expressions; all symbolic manipulation is done at compile time.

All above have a common idea: Compiling specifications to construct dedicated parallel cicuits to bypass the the symbolic processing and reasoning steps. This makes agents very responsive to their environment. Hap, which we will examine in details in Section 4, applies the same ideas of this situated actions and reactivity.

However, some researchers argue that although pure planning, with no situational feedback, is surely ineffective, yet it may be too radical to take the opposite extreme claiming planning and symbolic representation is irrevalent to cognition. So they advocate a more sophisticated intermediate approach — layered models, which we will discuss in the following.

2.3 Layered Model

Many researchers have suggested that neither a purely deliberative nor a purely reactive approach is suitable for CM. They have argued the case for a hybrid approach, which attempts to marry deliberative and reactive in a complementary fashion.

One of the best known layered models is the Procedural Reasoning System (PRS), developed by Georgeff and Lansky (1987) [19]. PRS is a belief-desire-intention architecture, which includes a plan library, as well as explicit symbolic representations of beliefs, desires, and intentions. Beliefs are facts, either about the external world or the internal states, and are expressed in classical FOL. Desires are represented as system behaviors. A PRS plan library contains a set of partially-elaborated plans, called knowledge areas (henceforth, KA), each of which is associated with an invocation condition. This condition determines when the KA is to be activated. KAs may also be reactive, allowing the PRS to respond rapidly to changes in its environment. The set of currently active KAs in a system represent its intentions. These various data structure are manipulated by a system interpreter, which is responsible for updating beliefs, invoking KAs, and executing actions.

The common characteristics of layered models that is the control must be both data-driven (in response to agent's current situation) and goal-driven (to satisfy one of agent's intention). In John Funge's Undersea World, which we will examine in Section 5, mermen can reason about their world based on acquired knowledge to achieve their goal; in case that reasoning can not be done in a timely fashion, the reactive system prevents mermen from doing anything stupid, such as bashing into rocks. Typical default reactive behaviors include "turn right", "avoid collision" and "swim for your life".

2.4 Discussion

Layered models, such as the PRS and RAP (Firby, 1989 & 1995), are currently a very active area of work, and arguably have some advantages over both purely deliberative and purely reactive models. One potential difficulty with such models, however, is that they tend to be ad hoc in that while their structures are well-motivated from a design point of view, it is not clear that they are motivated by any deep theory.

3 Steve's Cognitive Model

The first paper reviewed describes Steve, an animated agent that helps students learn to perform physical and procedural tasks. Steve, cohabiting with students in a three-dimensional simulated work environment, can demonstrate how to perform tasks and monitor students while they practise tasks, carrying on tutorial, task-oriented dialogs with students when needed.

Steve's architecture consists of three major modules: perception, cognition and motor control. The perception module monitors the state of the virtual world, maintains a coherent representation of it, and provides this information to the cognition and motor control modules. The cognition module interprets its perceptional input, chooses appropriate goals, constructs and executes plans to achieve those goals, and sends out motor commands. The motor control module decomposes these motor commands into a sequence of lower-level commands, controlling Steve's voice, locomotion, gaze, and gestures, and allowing Steve to manipulate objects in the virtual world. In this paper we focus on its cognition module, which is built on top of **Soar** (Laird, Newell, & Rosenbloom 1987; Newell 1990) [29, 36]. As much of Steve's design was influenced by features of Soar, it is necessary to briefly review Soar cognitive architecture.

3.1 Soar: A general cognitive architecture

Soar is a general cognitive architecture for developing systems that exhibit intelligent behaviors. In Soar, all tasks are represented as collections of problem spaces, which are made up of a set of states and operators that manipulate the states. Soar begins work on a task by choosing a problem space, then an initial state in the space. Soar represents the goal of the task as the final state in the problem space. Soar repeats its *Decision Cycle* as necessary to move from the initial state to the final state.

3.1.1 Knowledge in Soar

In order to act in a domain, Soar must have knowledge of that domain (either given to it or learned). The domain knowledge can be divided into two categories:

- Basic problem space knowledge: definitions of the state representation, the "legal move" operators, their applicability conditions and their effects.

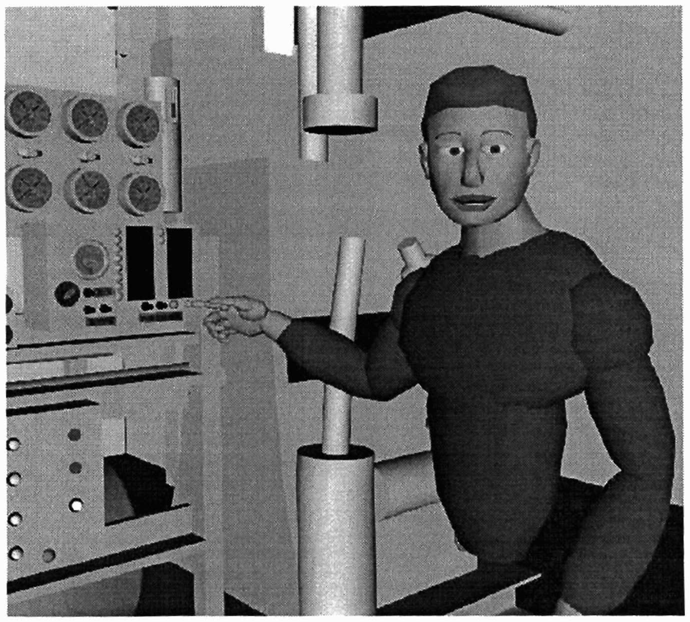


Figure 2: Steve

- Control knowledge, which gives guidance on choosing what to do, such as heuristics for solving problems in the domain.

Knowledge in Soar is encoded in production rules, which has the form of $C \Rightarrow A$, where C stands for conditions, and A for actions.

Two of Soar's memories are of relevance here: the production memory (or long-term memory), permanent knowledge in the form of production rules; and the working memory, temporary information about the situation being dealt with. Productions place preferences for working memory elements into preference memory. Types of preferences include acceptable, reject, require, prohibit, better, worse, reconsider, and indifferent. All perceptual and motor behavior is mediated through working memory. The motor modules accept commands from working memory and execute them. Their progress can then be monitored through sensors that are fed back into the system via perception.

3.1.2 Decision Cycle in Soar

Soar has a two-phase decision cycle, elaboration followed by decision. The two phases are repeated until the goal of the current task is reached. During the elaboration phase all productions which match the current working memory fire. All productions fire in parallel. The elaboration phase runs to Quiescence (until no more productions fire).

The decision phase examines any preferences put into preference memory (either in this phase, or previous ones), and chooses the next problem space, state, operator or goal to place in the context stack. The decision phase may change any current slot values, or any previous slot values in the context stack. If there is not enough information (or the information is contradictory) for the decision phase to choose the next slot value, then Soar reaches an impasse.

3.1.3 Impasses and Subgoaling in Soar

There are three possible types of impasses:

- Operators Zero Impasse: no candidate operators to apply.
- Operator Tie Impasse: too many, undifferentiable candidates.
- Conflict Impasse: Two or more operators are better than one another and they are not dominated by a third.

When Soar encounters an impasse in context level-1, it sets up a subcontext (or "subgoal") at level-2, which has associated with it a new state, with its own problem space and operators. Note that the operators at level-2 could well depend upon the context level-1. The goal of context level-2 is to find knowledge sufficient to resolve the higher impasse, allowing processing to resume there. For example, we may not have been able to choose between two operators, so the level-2 subgoal may simply try one operator to see if it solves the problem, and if not, tries the other operator. The processing at level-2 might itself encounter an impasse, set up a subgoal at level-3, and so on. So in general we have a stack of such levels, each generated by an impasse in the level above. Each level is referred to as a context (or goal), and each context can have its own state, problem space and operators.

Soar automatically creates subgoals in order to resolve impasses. This is the only way that subgoals get created.

3.1.4 Chunking in Soar

Soar includes a simple, uniform learning mechanism called chunking. Whenever a result is returned from an impasse, a new rule is learned connecting the relevant parts of the pre-impasse situation with the result. This means that next time a sufficiently similar situation occurs, the impasse is avoided. Notice in Figure 3 only *A*, *B*, and *C* are included in the conditions for the

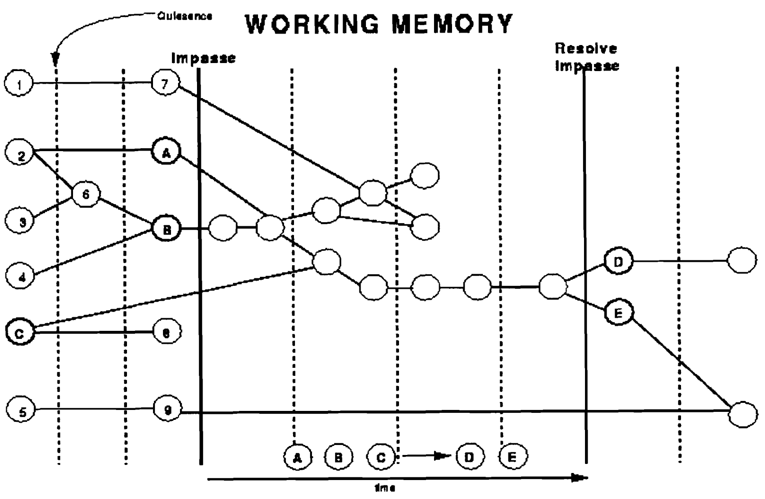


Figure 3: Chunking in Soar (from "*Unified Theories of Cognition*" [14])

production, working memory elements (WMEs) 7, 8, and 9 are not included since they are not along the path that leads to the goal even though they were created just before the impasse occurred; Similarly, WMEs 2, 4, and 6 are not included since they lead to WMEs which were used in impasse resolution but they were created before the impasse occurred. If in a future situation, condition elements *A*, *B*, and *C* occur, the resultant chunk shown at the bottom of the diagram will fire, regardless of the elements from which these conditions elements are created. And these new chunks are placed in the production memory immediately, and are available on the next elaboration phase, thus Soar's learning is intermixed with its problem solving.

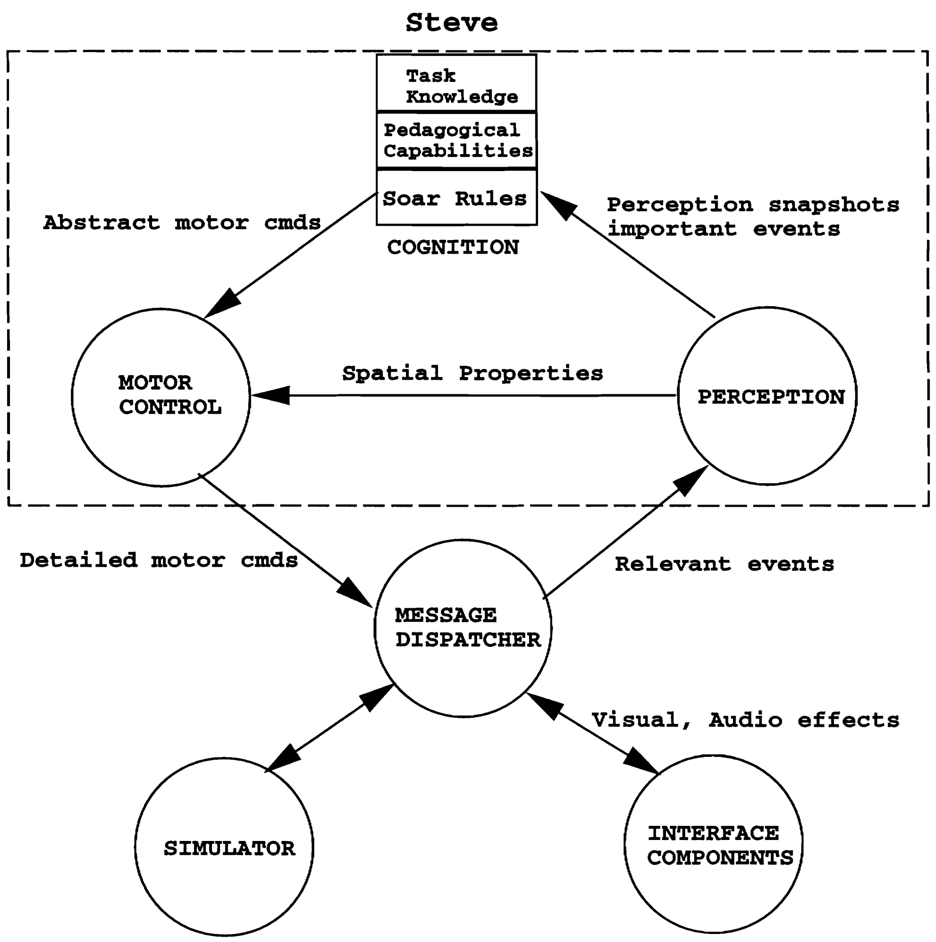


Figure 4: Steve's Cognitive Architecture

3.2 Steve's Cognition

Soar is a general cognitive architecture, but it does not provide built-in support for particular cognitive skills such as demonstration, explanation, monitoring and question answering. Steve, as an intelligent tutoring and training agent, requires such pedagogical capabilities. These capabilities are implemented as Soar production rules and layered on top of Soar. To demonstrate how to perform procedural task in a particular domain, Steve also needs a domain-specific task knowledge, which must be provided by the course author. Given appropriate task knowledge for a specific domain, Steve can employ these general pedagogical capabilities to teach students that knowledge. From top to bottom, the cognition is organized into three main layers:

- domain-specific task knowledge
- domain-independent pedagogical capabilities
- Soar production rules

This layered approach to Steve's cognition module (as shown in Figure 4) offers the flexibility of allowing Steve to be used in a variety of domains. Each new domain requires only new task knowledge. Steve's general pedagogical capabilities and underlying Soar production rules need not be modified.

3.2.1 Steve's Domain Task Knowledge

Steve represents domain tasks as hierarchical plans, using a relatively standard representation (Russell & Norvig 1995) [42]. First, each task consists of a set of steps, each of which is either a primitive action or a composite action. Composite actions give tasks a hierarchical structure. Second, there may be ordering constraints among the steps, each specifying that one step must precede another. These constraints define a partial order over the steps. Finally, the role of the steps in the task is represented by a set of causal links. Each causal link specifies that one step achieves a goal that is a precondition for another step. For example, putting on socks achieves the goal of SocksOn, which is a precondition for putting on shoes.

Figure 5 shows an example of a task definition: the task of performing a functional test of a high-pressure air compressor aboard a ship. It consists of three steps: press-function-test, check-alarm-light, and extinguish-alarm. In press-function-test step, the operator presses the test button on the control panel; then examines the light to make sure if it is still functional or burned

```
Task: functional-test
  Step: press-function-test, check-alarm-light, extinguish-alarm

  Causal Links:
    press-function-test achieves test-mode for check-alarm-light
    check-alarm-light achieves know-if-alarm-functional for end-task
    extinguish-alarm achieves alarm-off for end-task

  Ordering Constraints:
    press-function-test before check-alarm-light
    check-alarm-light before extinguish-alarm
```

Figure 5: An example task definition

out in the check-alarm-light step; finally operator turns off the light by pressing the reset button in the extinguish-alarm step. The task must be performed in this order. This is enforced by the ordering constraints in the task definition. In addition, several causal links exist among the steps. For example, press-function-test puts the devices in test mode, which is a precondition for check-alarm light. In order to complete the task, the operator must know whether the alarm light is functional, and make sure to turn off the alarm light. These are shown as the preconditions for end-task. Similarly, if the task is depended on conditions that must be established prior to starting the task, these conditions would be represented as effects of begin-task. Causal links serve to record the purpose(s) of steps in the task definition. They enable Steve to automatically generate explanations and to adopt procedures to unexpected circumstances.

The task definition only defines the structure of a task in terms of its steps and orderings. To complete the description, the course author must provide the *goals* and *primitive actions.* Each goal is defined by an attribute-value pair. Steve can represent two types of goals: goals that require putting the virtual world in some desired state (attributes of virtual world with specific values), which Steve can evaluate using perception, and goals that acquire information, which Steve can evaluate by checking his own mental state. Primitive actions are basic actions which can be directly executed by the motor. To simplify the course author's job, Steve has a general action library. Primitive action is an instance of one general action in the library with instantiated object name, motor command, and the perceptual attribute-value pair that will indicate that the primitive action has finished.

Steve uses the task knowledge to create the task model when being asked to demonstrate a specific task or to monitor the students performing the task. He creates the task model by adopting a top-down task decomposition proposed by Sacerdoti (1977) [45]. This approach is pretty simple: First, Steve initializes the task model to contain the name of the task. Next, he adds the task representation (steps, causal links and ordering constraints) for that task. Then he recursively repeats this process for any composite task until the task model has been fully decomposed into a hierarchically organized structure of primitive actions. The resulting task model is an important resource for Steve's plan construction.

3.3 Steve's Decision Cycle

Steve's cognition module operates by continually looping through a decision cycle. Each decision cycle goes through five phases:

1. Input Phase: Get the current perceptual information from the perception module.

2. Goal Assessment: Use the perceptual information to determine which goals of the current task are satisfied. This includes the end goals of the task as well as any intermediate goals (i.e., preconditions of task steps).

3. Plan Construction: Based on the results of goal assessment, construct a plan to complete the task.

4. Operator Selection: Select the next operator. Each operator is represented by a set of production rules that implement one of Steve's capabilities, such as answering a question or demonstrating an action. Steve's operators serve as the building blocks for his behavior.

5. Operator Execution: Execute the selected operator. In most cases, this will cause the cognition module to output one or more motor commands.

3.3.1 Input Phase

During the input phase, the cognition module receives three pieces of information from the perception module:

- The state of the simulator, represented as a set of attribute-value pairs.
- A set of important events that occurred since the last snapshot.
- The student's field of view, represented as the set of objects that lie within it.

3.3.2 Goal Assessment

Since each goal is associated with an attribute-value pair, Steve can assess each goal by simply determining whether its associated attribute-value pair is satisfied given the current perceptual input and mental state. This process is based on Soar's truth maintenance system. When the goal becomes satisfied, the rule fires, marking the goal satisfied. As long as the goal is satisfied, this result will remain, without any further processing required. If the goal becomes unsatisfied, Soar retracts the rule, along with its result. Thus Steve need not evaluate every goal on every decision cycle; each rule automatically fires or retracts when the status of its goal changes.

3.3.3 Plan Construction

Steve uses the task model to guide his plan construction and revision. The task model includes all the steps that might be required to complete the task (even if some are not necessary given the current state of world). Every decision cycle, after Steve gets a new perceptual snapshot and assesses the goals in the task model, he constructs a plan for completing the task. He first marks all the end goals as relevant for completing the task, then for each goal that is unsatisfied he finds the relevant steps in the task model that can achieve it and adds these steps to the plan. Each step added may further have unsatisfied preconditions, and each such precondition becomes a new goal that must likewise be achieved.

3.4 Steve's Domain-Independent Pedagogical Capabilities

3.4.1 Demonstration

To demonstrate a task to a student, Steve must perform the task himself, explaining what he is doing along the way. First, he creates the task model. Then, in each decision cycle, he updates his plan for completing the task and determines the next appropriate steps.

Choosing the Next Task Step to Demonstrate There may be multiple applicable steps to be executed next at any point of executing a task. In order to improve the communication between Steve and students, Steve maintains a focus stack to help to choose the next task step. Basically when executing a step (either primitive or composite), Steve pushes it onto the stack. Therefore, the bottom element of the stack is the main task on which Steve and the student are collaborating, and the topmost element is the one on which the demonstration is currently focused. When the step at the top of the focus stack is finished, Steve pops it off the stack. The main idea is to maintain the current focus or shift to a subtask of the current focus when multiple applicable steps are ready for execution.

Demonstrating a Task Step Steve always picks up the topmost step on the focus stack and demonstrates it to students. If the step is a composite step, Steve decomposes it into subtasks. If it is a primitive action, Steve simply performs the step. This is done by sending an appropriate motor command along with its associated text fragments and waiting for confirmation in his perception that command was executed.

Shifting to Monitoring Steve's demonstrations can end in one of two ways. Typically, he completes the task and announces his completion. Sometimes students may request to take over the rest of the demonstration. In such cases Steve shifts to monitoring the students.

3.4.2 Monitoring

When Steve plays the monitoring role, he still needs to maintain his own task completion plan and use it to evaluate the student's actions and answer their questions. In order to give students the opportunity to variety of situations, Steve should be able to adapt to various unexpected situations. Also, to allow students to deviate from the standard procedure, make mistakes and learn from recovering them, Steve needs to be able to repeatedly re-evaluate and revise his plan to support such flexibility. Steve can handle the following:

Evaluating the student's actions Steve evaluates the student's actions by trying to match the currently multiple applicable steps in the task model against the action the student is performing. If no match is found, Steve judges the student's action is incorrect. He acknowledges the students by text speech (i.e. simply says "no") and/or simple gestures (i.e. shakes his head). When student's action is correct, Steve nods his head in agreement.

Suggesting what to do next The student can always ask Steve "What should I do next?" Steve simply suggests the next step in his own plan. If there are multiple applicable next steps, Steve enumerates them. If Steve does not know either what to do next, he simply says he does not know.

Explaining the relevance of a step Sometime the student may ask questions about what the role of the action is in the whole task. Steve can answer such questions by generating explanation from the causal links in the plan. Causal links record the purpose(s) of steps. They are the connections between steps and goals, Steve can use these connections to rationalize his suggestions.

Shifting to demonstration The student may ask Steve "Show me what to do." In such case Steve takes over and shifts to demonstration. If there are multiple applicable next steps, Steve chooses one of them randomly.

3.4.3 After-Action Review

When Steve completes a demonstration, he asks the students whether they have any questions. They can ask him at this point to rationalize any one of his actions during the demonstration, and they can ask the follow-up "Why?" questions too. To answer such questions, Steve cannot rely on his current plan, since the task is already complete and the step in question is no longer relevant. Steve employs an episodic memory capability which can memorize Steve's actions and the situations in which the actions occurred.

3.5 Animation Results

3.6 Discussion

- **Augmented Partial-Order Planning**
  Planning is at the heart of Steve's cognition. No matter demonstrating a task to a student or monitoring the student's performance of the task, Steve must maintain a plan for completing the task. The plan enables Steve to identify the next appropriate action and, if asked, to explain the role of that action in completing the task (using causal links in the plan). In addition, Steve must be able to construct and revise plans quickly since he and students are collaborating on tasks in real time. Steve employs an augmented partial-order planner for his planning.

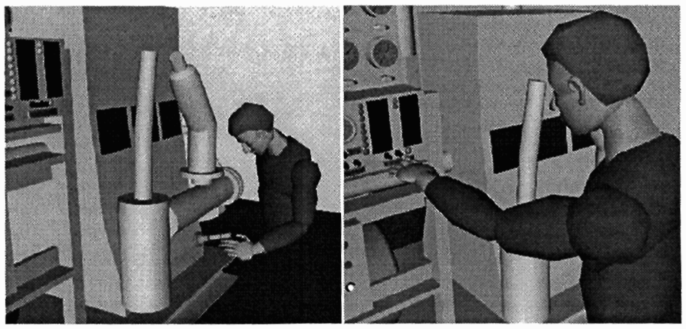


Figure 6: Steve is Demonstrating

As mentioned in Section 2, partial-order planners suffer from three major difficulties: the selection of multiple applicable actions; subgoal interactions; and the constraint bindings. A practical planning scheme must solve these problems to survive. We can ask whether Steve solves these problems by using a **task model** and whether the task model makes a good trade-off between plan expressiveness and execution efficiency.

- **Selection of multiple applicable actions**
  A partial-order planner may have multiple applicable actions that could achieve each goal, so it must go through all alternative plans, which often results in exponential searching. In constrast, Steve uses the task model as a guideline for choosing the appropriate next action to achieve each relevant, unsatisfied goal, so the searching is dramatically reduced. But this ease is bought at a cost. First, recall the task model is built based on the task definition given by the course author. The course author who provides such task definitions must have a skill knowledge about completing the tasks and can write down such knowledge in a clear and procedural way. A simple example is bicycle-riding. It would be very hard for Steve to demonstrate or monitor bicycle-riding because it is hard for course authors to articulate the procedure to perform the riding. Obviously the simplified task model employed greatly restricts the applications of Steve to a very limited domains such as operation and equipment maintainance where procedural knowledge is available. So far there has been no report that Steve works well in other domains.

Secondly, using procedural knowledge instead of declarative knowledge to specify the task definition means Steve can only perform tasks in a way that the course author can forsee. This greatly reduces Steve's autonomy. Declarative knowledge specification is desirable in that knowledge can be manipulated, decomposed and analyzed by the reasoning engine and can be used in a way the designer can not forsee, however, procedural knowledge specification is possibly faster usage in performance. Steve's plan construction is predictably fast. In this sense we may say Steve's rapid response is achieved by sacrificing of his autonomy.

- **Subgoal interaction**
  A partial-order planner must identify the possible subgoal interactions (clobber) and add appropriate ordering constraints. In constrast, Steve simply uses the ordering constraints provided in the task model; if two steps in the plan have an ordering constraint in the task model, that ordering constraint is added to the plan. As long as there are no conflicting subgoal interactions in the task model, there will be no conflicting subgoal interactions in the plan. Again the elimination of subgoal interactions totally depends on the involvement of the course author to debug. The course author is required to detect all possible subgoal interactions and resolve them by either demotion or promotion.
- **Variable binding constraint**
  To perform a task, a partial-order planner must break plan steps down into fully specified motion directives. Therefore, it must maintain a set of variable binding constraints, and it may have to search through when there are alternatives. In contrast, the steps in the task model of Steve are instances of actions in the action library, so they have no variables. Hence, Steve need not reason about binding constraints. However, the price to pay for this is: the course author must provide for each action instance with the object name, motor commands, and the perceptual attribute-value that will indicate the action is finished. The work of variable reasoning is simply shifted to the course author. Even worse, all these action parameters are hard-wired for a given situation, making it very inflexible to be reused in other situations.

In summary, the planning model Steve employs is simple and provides efficient execution and fast responses, but it sacrifices flexibility and expressiveness. It requires the course author do all the hard work providing not only the task definition but also the goals and instantiated primitive actions. Such instantiated structures make it work predictably fast, but virtually eliminate all the nondeterminism.

- **Miscellaneous**
  My techinical complain about this paper is that it lacks description about how the general pedagogical capabilities are implemented using Soar production rules. Although the connection is a little implementation oriented, this layer is important to convey the information of how to implement a particular set of cognitive skills using the general built-in cognitive capabilities of Soar, which is essentially the point that they advocate.

4 Woggle's Cognitive Model

The next paper reviewed presents a reactive cognition model, called `Hap`, designed as part of the `Oz` project at CMU. `Oz` project is aimed at building a dramatically interesting simulated world which includes intelligent, emotional and believable agents. In order to foster the illusion of reality, the Oz designers claim that agents (woggles) must have broad, though shallow, capabilities. Therefore they have attempted to build a broad architecture, called `Tok`, which contains several components: an emotion component called `Em`, a natural language system called `Glinda`, and a cognition component called `Hap` (the architecture is shown in Figure 7).

4.1 Hap: A Reactive Cognition Model

Initially Oz designers intended to provide reactivity by using a planner in the background feeding a reactive frontend that would execute plans. However, as they pursued this idea, the reactivity seemed to be spread through the whole system, so that they regard now reactivity as fundamental to the entire paradigm and abandon the background planning.

Due to this historical reason Hap designers call sequences of actions as plans, however, Hap does no explicit planning. There is no global internal symbolic world model, and no global planner/reasoner from a traditional AI planning pointview. All plans are simply chosen from a static plan library prepared in advance by the designers. These plans are either ordered or unordered collections of subgoals and primitive physical actions that can be used to accomplish a goal. Subgoals are statically provided by the designer, primitive physical actions are given by the domain.

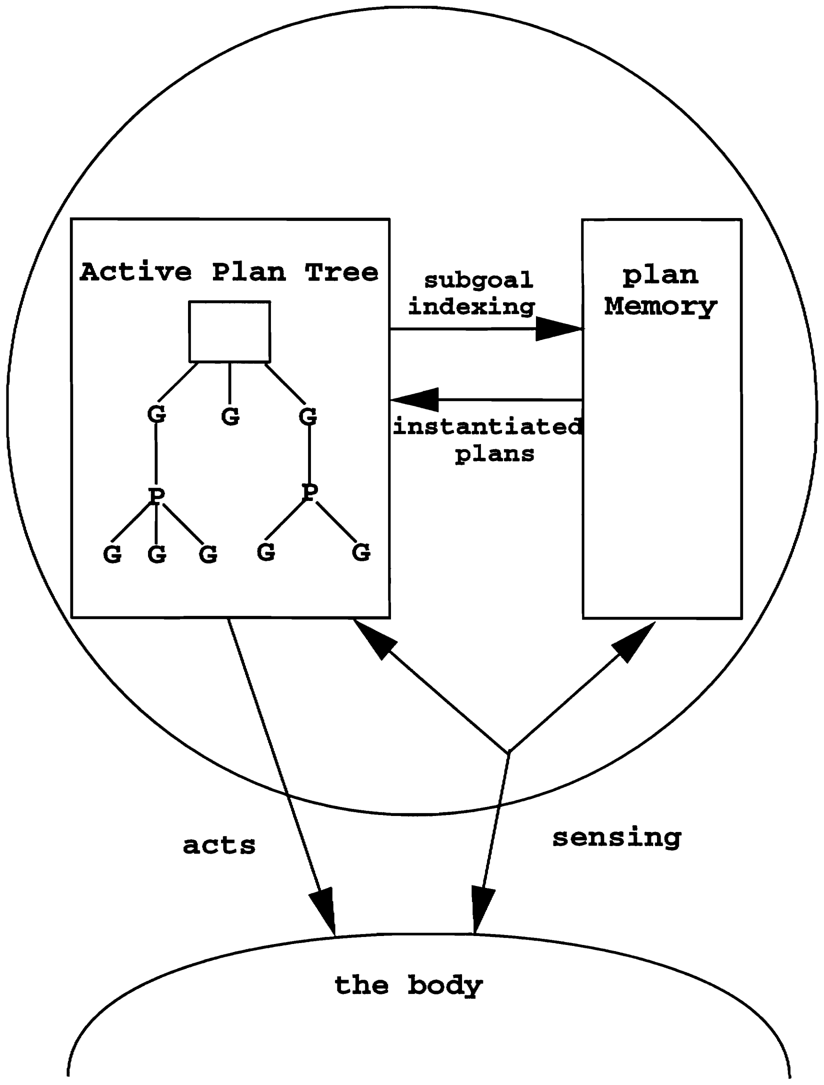


Figure 7: The Hap Cognitive Architecture

Hap supports goal-directed actions and allows the encoding of cognitive tasks. It continuously chooses the agent's next action based on perception, current goals, and emotional state. Perceptual information is provided by sensors which are automatically controlled by Hap to turn on or off when appropriate. Goals contain an atomic name and a set of parameters which are instantiated when the goal becomes active, for example, `(open <door>)`. Multiple plans can be written for a given goal, with Hap choosing among them at execution time. If a plan fails, Hap will attempt any alternate plans for the given goal.

The explicit representations of goal and emotional states may be a little controversial. Vera and Simon argued (1993) [52] that such representations are good examples of orthodox symbol system. But in Hap the goals and emotional states are represented functionally and aim at interacting with the world in a direct and unmediated stimulus-response manner. Thus Hap can be thought of as a kind of "soft form" reactive cognition model.

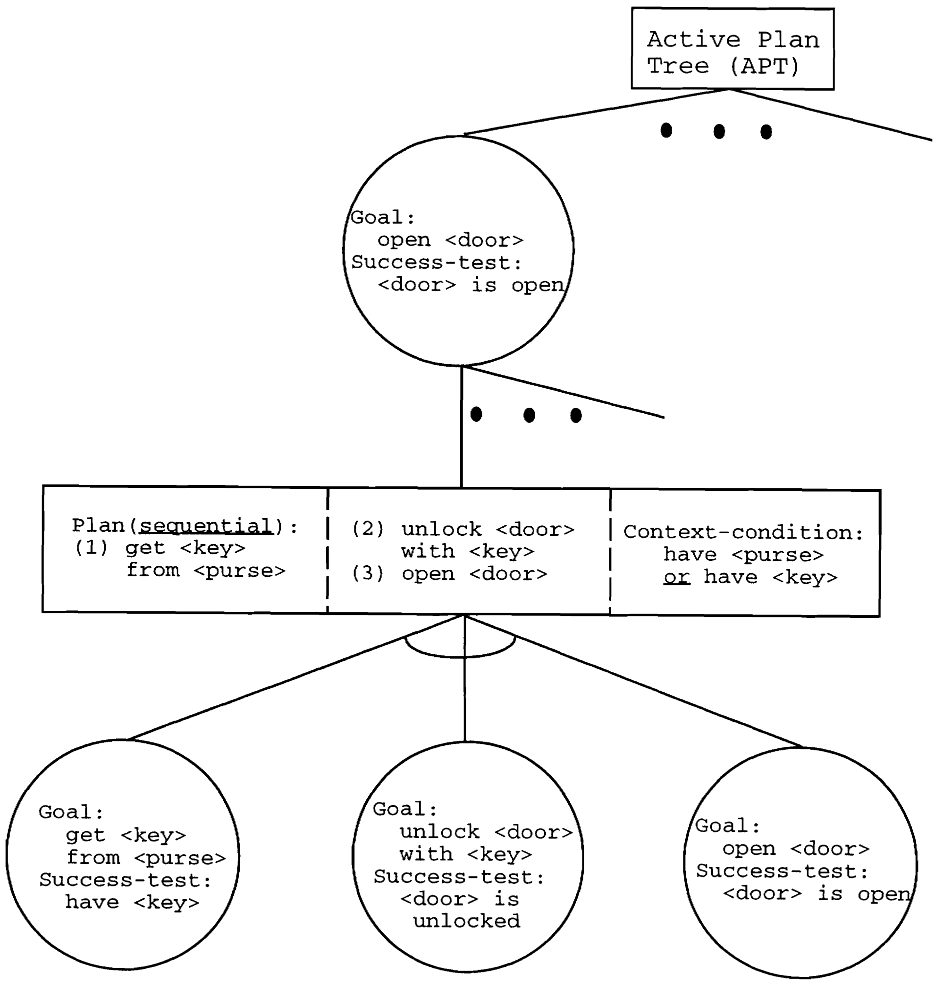
To illustrate a Hap agent's direct reaction to changing events in the world to achieve a goal, consider the following situation:

> An agent has the goal of opening a locked door. She has two applicable plans: get a key from her purse, unlock the door, and open it; or knock and wait. If, while looking for a key, someone opens the door for her, she should notice that her goal was satisfied and not keep working to accomplish it. In the same scenario, if she were searching in her purse for the key when her mischievous nephew snatched the purse, she should abandon that plan and try knocking (and perhaps deal with the nephew later.) (Loyall & Bates 1991) [32]

This example shows the two types of reactivity which Hap explicitly supports: recognizing the spontaneous achievement of a goal or subgoal, and realizing when changes in the domain make the pursuit of an active plan meaningless.

4.1.1 Active Plan Tree

Hap stores all active goals and plans in a structure called the active plan tree (henceforth, APT). This is a tree composed of alternating layers of goals and plans that represent Hap's current execution state. Each plan node in the tree has some number of goal nodes as its children. These goal nodes are the subgoals which must be satisfied for the plan to succeed. Each goal node either has no children (and thus it is a leaf goal node) or its child is the



Active Plan Tree (APT)
Goal: open <door> Success-test: <door> is open
Plan(sequential): (1) get <key> from <purse>
(2) unlock <door> with <key> (3) open <door>
Context-condition: have <purse> or have <key>
Goal: get <key> from <purse> Success-test: have <key>
Goal: unlock <door> with <key> Success-test: <door> is unlocked
Goal: open <door> Success-test: <door> is open

Figure 8: An Example of Active Plan Tree

current plan for the goal. The APT is constantly changing: expanding as plans with their component subgoals are chosen for goals and contracting as goals and plans succeed or fail. Physical actions succeed or fail depending on their realization in the domain. Goals succeed when a plan for the goal succeeds, and fails if all of the applicable plans have failed. Plans succeed if all of the component steps succeed and fail if any of the step fails. Therefore, the APT may be thought of as an AND-OR tree, where the goals are OR nodes and the plans are AND nodes. (The APT for the open-door case is shown in Figure 8 as an example.)

At the root of the APT is a special node which is parent to all of the top-level goals. These top-level goals are classified according to whether they are persistent. The persistent goals are continuous goals and thus are never removed from the tree. They can be reset so that they are again available to be pursued. Other goals are removed from the tree once they have succeeded or failed.

Associated with each goal node is a success-test and a priority. The success-test is a method for recognizing when a goal's purpose in the enclosing plan is spontaneously achieved. If it ever becomes true its associated goal is deemed to have been completed and no longer needs to be pursued and is thus removed from the tree. The subtree rooted at the goal node is also removed since its only purpose was working toward the goal. For example, the first subgoal of the `(open <door>)` described above has a success-test associated with it to determine if the agent is already `(have <key>)`. If this test is true when the plan begins, the step `(get <key> from <purse>)` would be skipped. Also, if the agent is in the process of `(get <key> from <purse>)` when some external factor causes the test to be true, the success-test would enable Hap to recognize that the goal has succeeded and stop further pursuing it. The priority is a number representing how important the goal is to the agent. Preferring higher priority goals is one of the methods to arbitrate between multiple goals.

Associated with each plan node is a context-condition and a specificity. When a context-condition becomes false its associated plan is deemed no longer applicable. In the example above if context-conditions `(have <purse>)` and `(have <key>)` are both false, the plan fails, making that plan node contracted, and a new alternative plan must be chosen at the `(open <door>)` goal level. Thus the context-condition for a plan must be continuously yielding true to have any chance of making sense and achieving its goal. The context-condition is necessary, but not sufficient, for the plan to work. A very dumb agent might use true for most context-conditions. This corresponds to not being very aware of world changing. Specificity is a measure of how specific each plan is. More specific plans are preferred over less specific plans by the plan arbiter.

4.1.2 Plan Memory

Plan memory is a set of production rules. The left hand side of each production is composed of a goal-expression and a precondition-expression. The goal-expression contains the goal name and zero or more variables. For a subgoal to match a production, the goal names must match and the number of values given in the subgoal must match the number of variables in the goal-expression. In addition, the precondition-expression must be true in the current state for the production to be applicable. The precondition-expression can reference the values from the goal-expression variables in its tests.

The right hand side of the production contains the context-condition, specificity and plan expression which are instantiated to create the plan and subgoal nodes for the APT. The plan node is created using all three of these, and the plan expression is used to create the subgoal nodes. The plan expression can be either a sequential or parallel arrangement of steps. Each step contains a goal expression, a priority modifier and a success-test expression. The goal expression and success-test are used directly to construct the appropriate goal node, and the priority modifier is added to the priority of the parent goal node to create the priority of this node. In this way a subgoal can be more, less or the same level of importance as the goal node which spawned it.

By the time the paper was written there are about 250 goal types and 500 plans available for a complete woggle. Keep in mind all these must be prepared by the woggle designers in advance. During the animation there is no dynamic plan construction, nor replanning under unexpected circumstances. What a woggle does given a current situation is to simply follow the circuit-switched action sequences. There is no deliberation in between.

4.1.3 Execution Loop

The execution loop consists of three steps: update the APT based on changes in the world, pick a leaf goal to execute, and execute the goal. Executing the goal can take the form of performing a primitive physical action, or expanding a subgoal.

The first step of the execution loop is to update the APT based on changes in the perceived world state: goals whose success-tests are true and plans whose context-conditions are false are removed along with any subordinate subgoals or plans; the parents to the removed subgoals or plans are notified that either the goal has been achieved or that the plan has failed. This effect can then propagate up the tree causing enclosing plans and goals to either succeed or fail. After the tree is adjusted, computation continues with the rest of the tree.

In the next step, the next leaf goal to execute is chosen by the goal arbiter. It chooses a leaf goal in the following fashion: first it refers to higher priority goals. If there are multiple goals with the same priority, it prefers to remain on the same higher-level goals that it was last working on rather than switching to new higher-level goals. Finally it will randomly choose among remaining leaf goals.

A primitive physical action is executed by sending the action to the body of the agent. A subgoal is expanded in following manner: first the subgoal expression is used to index the plan memory, then from the resulting set of applicable plans the more specific one is chosen and instantiated, created plans and subgoals are placed in the APT for consideration in the next loop.

4.1.4 Selective Sensing

Sensing in a real-time animated world must be efficient. Agents in Hap employ task-specific sensors which can be automatically turned on or off as needed. Each sensor observes the changes in the world and notifies the cognition module of such changes. Such sensory information is very important to evaluate agent's preconditions, success-tests and context-conditions. Hap automatically manages to turn on and off as appropriate. When a leaf subgoal is chosen for execution, sensors needed to evaluate the preconditions for that goal's plans are automatically turned on, and then turned off after a plan is chosen. Similarly, when a particular goal or a plan is present in the APT, all sensors related to its success-tests or condition-tests remain on. When a goal or a plan is removed from the tree, the corresponding sensors are automatically turned off.

4.1.5 Parallel Execution

In Hap all of an agent's active goals can be attended to, potentially generating multiple actions in parallel. Hap uses a greedy approach by attending to the most critical goals first and mixing in others as time allows. In each decision cycle Hap chooses the most critical applicable leaf goal and executes it until it is interrupted or suspended. In such a case Hap allocates the available time to threads running other, perhaps unrelated, critical goals selected by the goal arbiter.

During the parallel execution Hap will not allow any two incompatible goals to be pursued at the same time. Two actions are considered as incompatible if they use the same body resources.

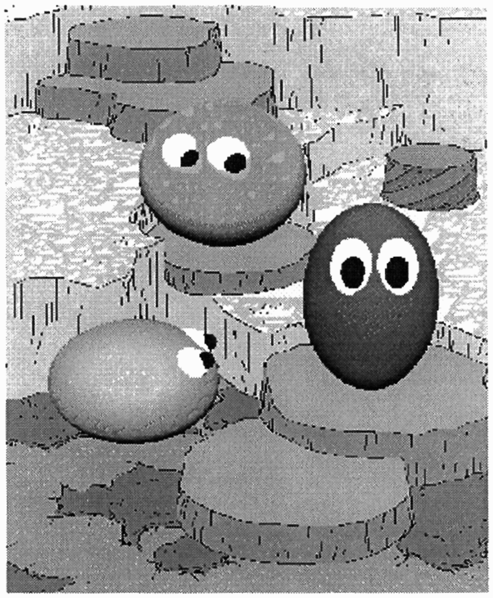


Figure 9: Woggle's World

4.2 Animation Results

The woggle's world is shown in Figure 9. It is modeled as a terrain field with simple physics. It is inhabited by four woggles: Bear, Wolf, Shrimp and a user-controlled woggle. Each woggle's body is an ellipsoid with eyes. They can play, sleep, fight, relax, explore the world, etc. The three woggles are designed to have somewhat stereotypical personalities: Wolf is aggressive; Shrimp is friendly and meek; Bear is a protector. These characterizations are reflected in their behavior. Shrimp is often timid and almost never does anything bold. Bear will often try to put out any strife in the world, for example, if he sees the user-controlled woggle is intimidating Shrimp, he becomes sad and tries to protect Shrimp. But Wolf sees the fight as an opportunity to amuse himself.

4.3 Discussions

In the deliberative model as we have seen in Section 2, agents tend to operate according to a sense-plan-act cycle. During sensing, the symbolic representation of the state of the world is updated by making inferences from sensory information. The agent then constructs a plan to accomplish its current goal in the symbolically represented world by composing a set of operators (primitive actions the agent can perform). Finally, the plan is executed. After the plan completes (or is interrupted due to some unplanned-for contingencies), the cycle repeats. When something unexpected happens, a replanning is necessary. However, rather than employing the sense-plan-act cycle, Hap is reactive. Its execution is a sense-indexing-react loop. The difference is in the middle: a deliberative model such as Soar explicitly considers alternatives and rejects all but one, while a reactive model such as Hap has no explicit deliberation except plan memory indexing. This does not necessarily mean that a reactive model is *less creative* than a deliberative model, because a real planner also does what it is programmed to do. By carefully considering the possible situations of sensing-acting sequences and providing appropriate production rules in the plan memory, a reactive model can still produce a lot of different patterns because of many possibilities of dynamic situations occuring environmentally and internally. It is arguable that a reactive system is doing less things itself, shifting the hard work such as plan construction to the designers. However, a traditional planner would also be given a lot of domain knowledge if it has to work efficiently, as we have seen previously in the Steve paper.

Replacing the planning (similar to on-line interpretation) with the indexing (similar to off-line compilation), Hap is predictably faster, and more fault-tolerant. In addition, it increases the possibility of parallel implementation. Different production rules in the plan memory could be active at the same time, allowing compatible actions to take place simultaneously.

4.3.1 Burden of Proof

My technical comments to Hap are two questions: How will it scale? How to implement a higher level of cognition?

- Scalability
  The capability of woggles responding to emergencies depends on whether or not there exists an appropriate sequence of actions in the plan

memory for woggles to execute. Woggles would fail to respond appropriately when there are no rules available in the plan memory for unexpected situations. In such case new rules must be constructed and added to the plan memory so that when a similar situation happens later woggles won't fail again. That's the scalability problems of the plan memory: Whether or not it degrades the reactivity when there are too many rules clustered in the plan memory? Whether or not the old rule components can be reused when constructing new rules? Whether or not more efficient indexing is required as the plan memory scales? Because some situations happen frequently, it may be helpful to cache the most recently used rules on the top level of the plan memory. What kind of caching strategy is better to achieve a high rule hit-ratio? All these questions are not addressed in the paper.

- Higher Level of Cognition
  To jump toward a hill, the woggle in the current implementation of Hap does not make use of a representation of the location of terrain in relation to its goal. It deals with each obstacle as it comes to it and does not remember whether the path it took last time was longer or shorter. However, creatures, like a real wolf or a real bear, appear to perform actions based on a more robust representation of the world. As the example in the paper mentioned that Bear will try to protect Shrimp when he detects the user is intimidating Shrimp. In such a case it makes sense that Bear should take the most direct route to the location where the fight happens. This requires a significantly more complex and more permanent representation. Currently they do not have such a higher-level cognition and probably they will not have unless a new underlying representation is adopted.

Finally, choosing `Hap` as a representative of the reactive models is somewhat questionable, since Hap is not a purely reactive model. According to my survey, most of the models employed in computer animation so far that are generally regarded as reactive are actually not purely reactive (Reynolds 1987; Agre & Chapman 1987; Tu & Terzopoulous 1994; Blumberg 1995) [38, 1, 49, 6]. In Hap, there are some production rules which need to be selected through pattern matching, there are some variables that need instantiation during the pattern matching. There is some alternative plans to be chosen when current plan fails. There are some low-level sensory information which needs to be further processed using production rules in order to recognize the abstract composites of that sensory information such as woggles fighting, relaxing or playing games.

Some deliberative models proponents (Vera and Simon 1993) [52] argue that such representations are good demonstrations that symbolic processing is at the heart of the intelligence and therefore the reactive models are not at all antithetical, but complementary, to the deliberative models. Nevertheless, some researchers, as we shall see in the next section, believe these are good indications of the necessities to combine the two models together to take advantages of a high-level control (reasoning/planning) and a low-level sensing-reacting. This results in the third approach in cognitive modeling — layered modeling.

5 Mermen's Cognitive Model

The third paper reviewed simulates a physics-based undersea world, which is inhabited by mermen, fabled creatures of the sea with the head and upper body of a man and the tail of a fish. Other inhabitants are predator sharks. An artificial life simulator implements these creatures as fully functional autonomous agents. The modeling includes a graphical display model that captures the form and appearance of these creatures, a biomechanical model captures their anatomical structures and simulates the deformation of their body. A behavioral control model implements shark's brain and is responsible for motor, perception and low-level behavior control. The behavioral control model is very similar to that in (Tu & Terzopoulos 1994) [49]. The mermen are equipped with a cognitive model which includes a reasoning engine and a reactive system. The reasoning engine enables them to reason about their world based on the domain knowledge either provided by the animator or acquired through sensing. The reactive system mediates between the reasoning engine and the physics-based environment. The reactive system translates the low-level commands generated by the reasoning engine down to the necessary detailed muscle actions. It also acts as a fail-safe should the reasoning engine temporarily fall through. When the reasoning engine cannot decide upon an intelligent course of actions in a reasonable amount of time, the reactive system continually tries to prevent the mermen from doing anything stupid, such as bashing into obstacles. Typical default reactive actions are "*turn right*", "*avoid collision*", and "*swim for your life*".

5.1 Domain Knowledge and Character Directions

A character's knowledge of its world is referred to as the background domain knowledge. It is a set of animator supplied information which can provide the character with understanding of when actions are possible, and

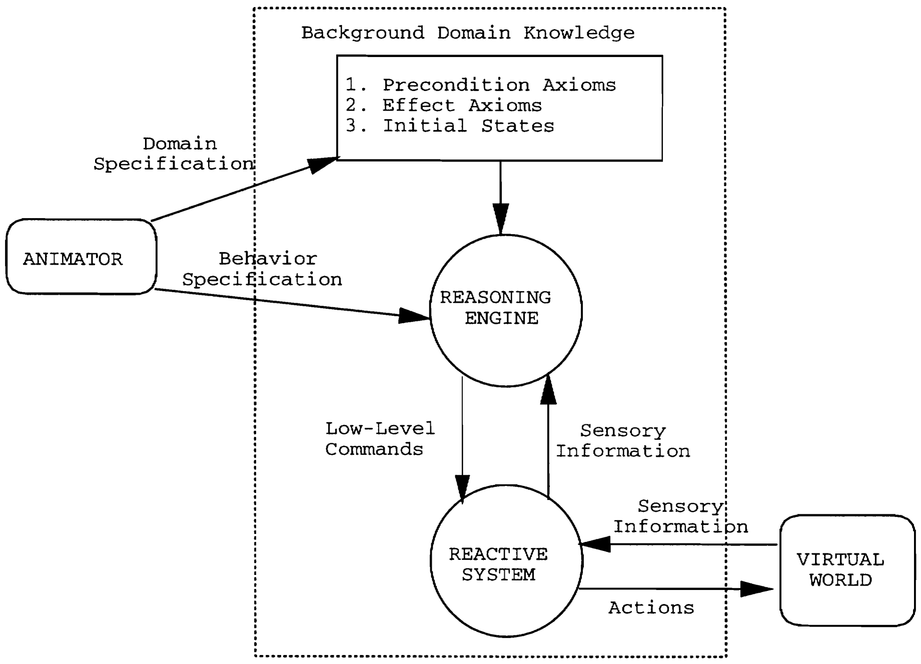


Figure 10: Mermen's Cognitive Model

what the effect of actions are on the virtual world. The domain knowledge specification in this paper heavily rests on the use of situation calculus.

5.1.1 Situation Calculus

As mentioned in the introduction, situation calculus is used for describing a changing world in First Order Logic (FOL). It conceives of the world as consisting of a sequence of situations, each of which is a "snapshot" of the state of the world. Situations are generated from previous situations by actions. A domain independent constant s_0 denotes the initial situation. Any property of the world that can change over time is known as a fluent. A fluent is a function, or relation, with a situation term as its last argument. For example, $\texttt{Alive}(x, s)$ is a fluent to keeps track of whether Jack x is alive in a situation s. The situation S' resulting from doing action a in situation s is given by distinguished function do: $action \times situation \to situation$. For example, $\texttt{Aimed}(do(\text{aim}, s))$ means the gun is aimed at Jack after aiming it. The possibility of performing action a in situation s is denoted by a distinguished predicate Poss: $action \times situation$. Sentences that specify what the state of the world must be before performing some action are known as precondition axioms. For example, $\texttt{Poss}(\text{shoot}, s) \Rightarrow \texttt{Loaded}(s)$ means the gun can only be shot if it's loaded. Effect axioms give necessary conditions for a fluent to take on a given value after performing an action. We can use effect axioms to state the effects of the actions. For example, $\texttt{Poss}(\text{shoot}, s) \land \texttt{Aimed}(s) \Rightarrow \neg\, \texttt{Alive}(do(\text{shoot}, s))$ means if the gun is aimed at Jack and it can be shot then he is dead after shooting it.

As mentioned earlier, situation calculus is perplexed by three problems: the qualification problem, the frame problem, and the ramification problem.

- The Qualification Problem
  The *qualification problem* is that of trying to infer when an action is possible. In the gun-shooting example, only certain necessary condition is given, not all the things that may prevent shooting the gun are enumerated. For instance, we cannot shoot if the trigger is too stiff, or if the gun is encased in concrete, etc. By employing a closed-world assumption, we may obviate this problem and assume that the set of necessary conditions is also a sufficient set. For instance, under this assumption the precondition axiom for shoot now becomes:

$$\texttt{Poss}(shoot, s) \Leftrightarrow \texttt{Loaded}(s)$$

- The Frame Problem
  The *frame problem* is of trying to infer what remains unchanged by an

action. In previous examples we only wrote down what changed after an action; we did not write down all the things that stayed the same. The frame problem can be solved provided characters represent their knowledge with the assumption that effect axioms enumerate all the possible ways that the world can change. This closed world assumption provides the justification for replacing all the effect axioms with `successor state axioms`. For instance, the successor state axiom for $\texttt{Alive}(s)$ states that Jack is alive, if and only if, he was alive in the previous state and he was not just shot:

$$\texttt{Poss}(a,s) \Rightarrow [\texttt{Alive}(do(a,s)) \Leftrightarrow \texttt{Alive}(s) \wedge \neg(a = \mathit{shoot} \wedge \texttt{Aimed}(s))]\,.$$

- The Ramification Problem
  The *ramification problem* is that of trying to determine all the implicit side-effects of actions caused by the presence of state constraints. In general, state constraints may give rise to intractable problems (Lin & Reiter) [31], but in some cases, the ramification problem can be dealt with by making all the side-effects explicit. This can be done by compiling the state constraints into the successor state axioms. For example, we can modify the successor state axiom for $\texttt{Alive}(s)$ as follows:

$$\texttt{Poss}(a,s) \Rightarrow (\texttt{Alive}(do(a,s)) \Leftrightarrow \texttt{Alive}(s) \wedge \neg(a = \mathit{shoot} \wedge \texttt{Aimed}(s))$$
$$\wedge\neg(a = \mathit{detonate} \wedge \texttt{NearBomb}(s))).$$

5.1.2 Primitive and Complex Actions

The actions discussed previously, defined by corresponding preconditions and successor state axioms, are referred to as a primitive actions. We have explained how they can be used by characters to keep track of its changing world. Next we show how to define new complex actions in terms of the previously defined primitive actions.

Complex actions consist of a set of primitive actions connected by operators. The complete list of operators for defining the complex actions is defined recursively and given below.

Type	Operator	Meaning
Sequence	$\alpha\ ;\ \beta$	do action α, followed by action β
Test	p?	do nothing if p is true, otherwise fails
Conditional	if p then α else β	do α if p is true, otherwise do β
Nondeterm. iteration	$\alpha*$	do α zero or more times.
Iteration	while p do α od	do α while p is true
Nondeterm. actions	$\alpha \mid \beta$	do action α, or action β
Nondeterm. arguments	$(\pi x)\alpha(x)$	pick some argument x and perform the action $\alpha(x)$
Procedures	proc $P(x_1, \cdots, x_n)$ α end	declares a procedure $P(x_1, \cdots, x_n)$

Complex actions are also called sketch plans. The effect of a complex action α is defined by the macro Do(α, s, s'), where s' is a state that results from doing α in state s. These *macros* expand out into a situation calculus expression, thus ensuring complex actions inherit the solution to the frame problem for primitive actions. Given some advice, represented as a complex action `program`, the underlying reasoning engine (actually a theorem prover) attempts to prove:

$$\texttt{AXIOMS} \models \exists Do(\texttt{program}, s_0, s).$$

The resulting (constructive) proof results in a term $s = do(a_n, \cdots, do(a_1, s_0))$, such that $[a_1, \cdots, a_n]$ is the sequence of primitive actions that the character should perform in order to follow the advice `program`.

5.1.3 CML

Cognitive Modeling Language (henceforth, CML) is an implementation of complex actions within the situation calculus. It forms a middle ground between the regular logic programming and traditional imperative programming, therefore, it provides animators with the ability to specify a "sketch plan" on a higher-level without the need for logical connectives. CML's control constructs are closely related to the logical constructs used in the situation calculus. The comparison between situation calculus syntax (SC syntax) and CML syntax is given below:

Type	SC Syntax	CML Syntax
Sequence	α ; β	<action> ; <action>
Test	p?	test <expr>
Conditional	if p then α else β	if (<expr>) <action> else <action>
Nondeterm. iteration	$\alpha*$	star <action>
Iteration	while p do α od	while (<expr>) <action>
Nondeterm. actions	$\alpha \mid \beta$	choose <action> or <action>
Nondeterm. arguments	$(\pi x)\alpha(x)$	pick (<expr>) <action>
Procedures	proc $P(x_1, \cdots, x_n)$ α end	void P(<arglist>) <action>

Following is an example of a CML procedure, the one to the left below, with its corresponding complex action in situation calculus. They define a depth-bounded (to n steps) depth-first planner:

```
proc planner(n) {
    choose test(goal);
    or {
        test(n> 0);
        pick(a) {
            primitiveAction(a);
            do(a); }
        planner(n−1); } }
```

```
proc planner(n)
    goal? |
    [ (n > 0)?;
      (π a)(primitiveAction(a)?; a);
      planner(n−1) ]
end
```

A CML program can consist of multiple CML procedures and control structures. The CML program can be compiled by a CML compiler (Funge 1998a) [16] into the equivalent Prolog code. Running the resulting Prolog code will return a list of all possible sequences of primitive actions that meet the specification represented by the original CML program.

5.1.4 Domain Knowledge and Character Directions

Any property of the domain that can change over time can be represented as a *fluent*. For example, the prey's position can be represented as `PreyPos`, where `PreyPos`(p, s) means that the prey is in region p in situation s. The predator's position, `PredPos`, and the prey's desired position, `PreyGoalPos` can be similarly defined. A property that does not change over time can just be represented as a relation without a situation argument. For example, `Occluded`(p, q, o) states that region p is *always* hidden from region q by obstacle o, and `Occupied`(p, o) states region p is *always* occupied by o. Predicates can also be used to specify other domain knowledge. For example, `Type`(jaws, predator), and `Type`(merman, prey) specify the type of predator and prey. The truth of predicates such as `Occupied`(p, o) can be quickly calculated at run-time; it need not be stored in a precomputed database.

Animators can use various structures of CML to encode high-level control for issuing advice or directions to a character from the character's point of view. For example, consider the problem of a merman trying to come up with a plan to hide from the predator. A traditional planning approach will be able to perform a search of various locations according to criteria such as whether the location is hidden, or far from the predator, etc. Unfortunately, this kind of brute-force planning is prohibitively expensive. In contrast, the control structures of CML allow the animator to encode heuristic knowledge to help overcome this limitation. For example, a CML procedure can encode the following heuristics: If the current position is good enough then stay there, else search the hidden locations around you (the expensive part); otherwise try to run away from the predator. The searching can call another subprocedure using another simple heuristics: look for cover near obstacles because hidden positions are usually near the obstacles.

```
proc evade() {
    while predatorApproaching do
        sense() ;
        update() ;
        choose testCurrPosn() ;
        or searchflee() ;
    od
end
```

```
proc searchflee()
  if ∃r NearRock(r)
  then
    (πr) (NearRock(r))?;hideBehind(r)
  else
    (πr) (AwayPredDir(r))?;setGoal(r)
  end
```

A key feature of the CML is the ability to use **nondeterminism**. Nondeterminism is not randomness. The above CML specification may admit more than one candidate as a hidden position for a merman to hide behind. At the design time they are not determined. At the run time only one is chosen by the reasoning engine based on the current sensory information. This nondeterminism greatly reduces the amount of work required from the animator and is very useful for rapidly developing new characters.

5.2 Sensing

Due to the unpredictability of the changing environment, some actions (events) are generated by the environment and not the character. Such actions are referred to as exogenous actions. While the cause of an exogenous action is difficult to state its effect need not be. For example, `movePred`(x) simply moves the predator to a new position x. To avoid inefficient and unnatural behavior, characters need to take these effects into consideration. For example, when a merperson tries to evade, she is supposed to know where the predator currently is. But the problem is she is uncertain about this because the predator is moved by mysterious external forces and outside the ability of her control. The paper reviewed uses a special fluent called interval-valued epistemic (henceforth, IVE) fluent to represent the character's uncertainty about aspects of the world.

For each sensory fluent f, there is a corresponding IVE fluent $\mathcal{I}_f$, which is a set of pairs $<u, v>$, where $u, v \in \mathcal{R}^{*+}$. The IVE fluent $\mathcal{I}_f$ is used to represent an agent's uncertainty about the value of f. For example, suppose an exogenous action `setSpeed` changes the predator's speed, and a fluent speed keeps track of the predator's speed. An IVE fluent $\mathcal{I}_{speed}(s_0) = <10, 20>$ states that the predator's speed is initially known to be between 10 and 20 m/sec. The character's uncertainty about the predator's speed usually increases over time until a sensing action causes the interval to collapse to its actual value.

Both exogenous actions and sensing actions can be incorporated into the situation calculus by modifying the definition of macro expansion for complex actions to allow for the possible occurence of exogenous actions and sensing actions. In this review we shall skip it due to the lengthy technical problems involved. More details can be found in Funge's Ph.D thesis (Funge 1998b) [17].

5.3 Reasoning Engine

The reasoning engine takes three different inputs: the background domain knowledge, sketch plans, and the sensory information.

The background domain knowledge is a set of animator provided axioms that collectively constitute a causal theory providing the character with an understanding of when actions are possible and how they affect the world. The sketch plans specified by the animator in CML language can be compiled using a CML compiler [16] into equivalent Prolog programs. At run-time the Prolog code will generate a list of all possible primitive actions, which are further selected by the reasoning engine based on the sensory information.

5.4 Reactive System

At every animation frame, the reasoning engine commits to the actions it has decided by sending the primitive actions to the underlying reactive system. The underlying reactive system then executes the primitive actions and returns some sensory information. This new sensory information is used to update all the IVE fluents.

More importantly, the reactive system also implements some primitive actions that are common to all characters. It autonomously executes and arbitrates among these primitive actions. These primitive actions include reactive ones such as "avoiding collisions", and basic locomotion such as "go to a particular position".

The drawback of separating these primitive actions out from the high-level reasoning system is that they become less flexible and can not be reconfigured through logical reasoning. However, there are a number of benefits from this separation:

- Primitive behaviors are usually character-independent, once they are operational the need to change them is minimal.
- Primitive behaviors are intended to be components of high-level behaviors, they are supposed to be as efficient as possible as they will be executed frequently.
- An independent reactive system can function as a fail-free, in case the high-level reasoning engine temporarily falls through, the reactive system can still prevent the character from doing anything stupid, such as bashing into obstacles.

5.5 Undersea Animations

The undersea animations revolve around pursuit and evasion behaviors. The hungry sharks try to catch and eat the mermen and the mermen try to use their superior cognitive power to avoid this grisly fate. For the most part, the sharks are instructed to chase the mermen they see. If they cannot see any mermen, they go to where they last saw one. If all else fails, they start to forage systematically. Because the shark is a larger and faster swimmer, it has little trouble catching the mermen in open water. However, if there are rocks in the underwater scene, the cognitively empowered mermen can take advantage of the rocks to try and avoid being eaten. They can hide behind rocks and hug them closely so that sharks have difficulty seeing or reaching them. To cope with fast moving environments, the mermen base their decisions on where to go on the positions that they predict the sharks will be in when they get to its goal. So long as it is safe to do so, the mermen will try to visit other obstacles. To avoid being caught, the mermen can also swim through the cracks which are too narrow for the shark to pass through without risking injury.

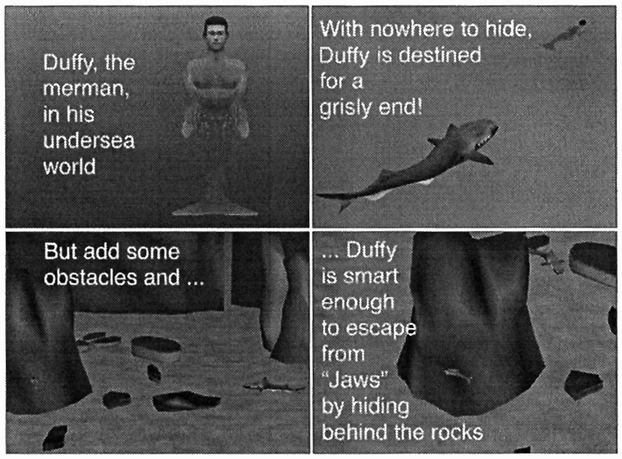

Duffy, the merman, in his undersea world
With nowhere to hide, Duffy is destined for a grisly end!
But add some obstacles and ...
... Duffy is smart enough to escape from "Jaws" by hiding behind the rocks

Figure 11: Undersea Animations

5.6 Discussions

The merits of this paper are as noticeable as its deficiencies. It applies a theory of action that allows animated characters to perceive, reason and act in a dynamic virtual world. It implements a remarkably useful framework for high-level control and low-level reaction that combines the advantages of a reasoning system and a reactive system. However, many of the shortcomings or limitations can be easily spotted:

- Scalability
  In the reactive model, the scalability problem is concerned with how to extend the hardwired connections (or plan memory). In this model, the scalability problem is concerned with the reasoning engine. When there are many reasoning characters active in the virtual undersea

world, things would start to get slow dramatically because the underlying reasoning engine is working on a more declarative style of knowledge. Even though the conversion from sketch plans to Prolog code can be done as a pre-processing step, the speed problem is still prominent. However, when there are only a few characters in the simulated world, the very reasoning engine can offer the animator great ease and flexibility by placing heavy reliance on the character's reasoning abilities. This reduces the amount of work required from the animator and is hence extremely useful for rapid prototyping of new characters. One way to relieve the scalability problem is to gradually reduce the non-determinism in the reasoning process, but this subsequently results in less flexibility of the high-level control.

- Stability and Robustness
  A key problem with the logical approach to control is that once an inconsistency arises the whole system comes crashing to a halt. It makes the system extremely unstable.

- Planning
  Compared with the highly sophisticated and effective planners such as a SOAR planner, the planning mechanism employed in this paper is very simple. It depends on its underlying Prolog theorem prover to provide a list of all possible sequences of primitive actions which are further selected by its reasoning engine. It is something like a simple action selection mechanism.

- Learning
  Strictly speaking, there is no learning mechanism involved in this system. The reasoning engine makes decisions based on the background domain knowledge, character directions and sensory information. The decision making process does not really generate any new knowledge. There is no new knowledge flowing from the reasoning engine to the background domain knowledge base for future use. Later on when an exactly similar situation arises, the whole process has to repeat to generate the primitive actions.

- Natural Language Control
  Instruction and interaction is made relatively easier by using logical representation that correspond to the animator's way of thinking about the character's world. Logical representation, as a mean of communication, is clear and precise. However, it would be a worthwhile

enhancement to allow for an interaction language more skin to natural language (Badler 1998) [4]. This would provide a version more suited to direct the characters by non-technical people.

6 Summary

The reader will recall that, after the introduction, the paper began with a broad overview of cognitive modeling approaches, taking into account the most important theoretical and practical issues. A framework is outlined by classifying all the cognitive modeling approaches into three categories. We moved on to discuss the applications of these modeling techniques in compute animation. Steve uses a deliberative model, based on Soar. A woggle has a reactive model to achieve fast responses and reactivities. Merpeople are equipped with a high-level reasoning engine and a low-level reactive system to evade from the sharks in the dynamic undersea world.

Following is the table containing the comparisons among these three cognitively empowered characters.

Character	Planning	Learning	Behavior Design Complexity
Steve	Yes(Complex)	Yes	High
Woggle	No	No	Medium
Merman	Yes(Simple)	No	Low

Character	Behavior Direction Efficiency	Flexibility	Performance
Steve	Medium	High	Low
Woggle	Low	Low	High
Merman	High	High	Medium

Character	Robustness	Applications
Steve	Medium	Tutorial, Training (in Procedural Domains)
Woggle	High	Interactive Drama, Entertainment
Merman	Low	Interactive Computer Games

A key note is that, in any of the papers reviewed, computer animation does not play a role as a test-bed, but as a driving force, for picking up appropriate cognitive modeling techniques. Each paper has a different animation domain and therefore different requirements and expected animation results. On the face of these differences, the models do not seem to be contradictory: They emphasize different agent behaviors in different animated environments. Nonetheless, the social structure of research is such that individual researchers will justify their approaches by emphasizing the weakness of others and the strength of their own. It is this very human set of orientations and responses that leads to these compelling but conflicting research approaches. Putting each approach into a bigger framework and comparatively analyzing its virtues and goodness as well as its shortcomings and flaws, this paper gives a more objective, systematic overview of these different approaches.

References

[1] P.E. Agre and D. Chapman. PENGI: An implementation of a theory of activity. Proceedings AAAI-87, pp. 268-272, Seattle, WA, 1987

[2] P.E. Agre. The Symbolic Worldview: Reply to Vera and Simon. Cognitive Science 17, pp. 61-69 (1993).

[3] N.I. Badler, C. Phillips, and B.L Webber. Simulating Humans: Computer Graphics, Animation, and Control. Oxford University Press, New York, 1993.

[4] N. Badler, R. Bindiganavale, J. Bourne, M. Palmer, J. Shi and W. Schuler. "A parameterized action representation for virtual human agents," Workshop on Embodied Conversational Characters, North Tahoe, CA, October 1998.

[5] J. Bates, L. Byran and R. Scott. An Architecture for Action, Emotion, and Social Behavior. In Proceeding of Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Springer-Verlag, Berlin, 1992.

[6] B.M. Blumberg and T.A. Galyean. Multi-level Direction of Autonomous Creatures for Real-Time Virtual Environment. In Proceedings of SIGGRAPH'95, Los Angeles, CA, August 1995, pp. 47-54.

[7] R.A. Brooks. A Robust Layered Control System for A Mobile Robot, IEEE Journal of Robotics and Automation., RA-2, pp. 14-23, 1986.

[8] R.A Brooks. Elephants Don't Play Chess, *Robotics and Autonomous Systems* Vol. 6, 1990, pp. 3-15.

[9] R.A. Brooks. Intelligence without reason. In Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), pp. 569-595, Sydney, Australia, 1991.

[10] R.A. Brooks. Intelligence without representation. Artificial Intelligence(47):139-159, 1991

[11] D. Chapman and P. Agre. Abstract reasoning as emergent from concrete activity. In Georgeff and Lansky (ed.), Reasoning About Actions and Plans. pp. 411-424. Morgan Kaufmann Publishers: San Mateo, CA, 1986.

[12] D. Chapman. Planning for conjunctive goals. Artificial Intelligence (32):333-378, 1987.

[13] W.J. Clancey. Situated Action: A Neuropsychological Interpretation. Cognitive Science 17, pp. 87-116 (1993).

[14] M.R. Fehling. Book Review: Unified Theories of Cognition: Modeling Cognitive Competence, *Artificial Intelligence* 59, pp. 295-328.

[15] N. Foster and D. Metaxas, Realistic Animation of Liquids, Graphical Models and Image Processing:58(5), 1996, pp. 471-483.

[16] J. Funge. CML Compiler Document. University of Toronto, http://www.cs.toronto.edu/~funge/cml/index.html, 1998.

[17] J. Funge. Making Them Behave: Cognitive Models for Computer Animation. Ph.D. Thesis, University of Toronto, 1998.

[18] C.W. Geib. The Intentional Planning System (ItPlanS). University of Pennsylvania, PhD Thesis, 1995.

[19] M.P. Georgeff and A.L. Lansky. Reactive reasoning and planning. In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pp. 677-682, Seattle, WA.

[20] J.G. Greeno and J.L. Moore. Situativity and Symbols: Response to Vera and Simon. Cognitive Science 17, pp. 49-59 (1993).

[21] R. Grzeszczuk and D. Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction, In Proceeding of SIGGRAPH'95, Los Angeles, CA, August, 1995, pp. 63-70.

[22] J.K. Hodgins, W.L. Wooten, D.C. Brogan, and J.F. O'Brien. Animating Human Athletics. In Proceedings of SIGGRAPH'95, Los Angeles, CA, August 1995, pp. 71-78.

[23] W.L. Johnson, J. Rickel, R. Stiles, & A. Munro. Integrating Pedagogical Agents into Virtual Environments, Presence: Teleoperators and Virtual Environments 7(6):523-546, December 1998.

[24] W.L. Johnson. Pedagogical Agents, invited paper at the International Conference on Computers in Education. Also to appear in the Italian AI Society Magazine, 1999.

[25] W.L. Johnson, J.W. Rickel, and J.C. Lester. Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. To appear in Intl. Journal of Artificial Intelligence in Education, 2000.

[26] W.L. Johnson, J.W. Rickel. Intelligent Tutoring in Virtual Environment Simulations, ITS'96 Workshop on Simulation-Based Learning Technology.

[27] M.Y. Jona and R.C. Schank. Book Review: Issues for Psychology, Artificial Intelligence and Education: A Review of Newell's Unified Theories of Cognition. Artificial Intelligence 59, pp. 375-388, 1993.

[28] D. Kurlander and D.T. Ling. Planning-Based Control of Interface Animation. In the Proceeding of CHI'95, Denver, pp. 472-479. Also Microsoft Research Technical Report MSR-TR-95-21, 1995.

[29] J.E. Laird, A. Nowell, and P.S. Rosenbloom. Soar: an architecture for general intelligence. Artificial Intelligence, vol. 33, 1-64, 1987

[30] J.E. Laird and P.S. Rosenbloom. On Unified Theories of Cognition: A Response to the Reviews, *Artificial Intelligence* 59, pp. 389-413, 1993.

[31] F. Lin and R. Reiter. State Constraints Revisited. Journal of Logic and Computation, Special Issue on Actions and Processes. 4(5), pp. 655-678, 1994.

[32] A. Loyall and J. Bates. Hap: A Reactive Adaptive Architecture for Agents. Technical Report CMU-CS-91-147, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1991.

[33] P. Maes. The agent network architecture (ANA). SIGART Bulletin, 2(4):115-120, 1991.

[34] P. Maes. Agents that reduce work and information overload. Communications of the ACM, 37(7):31-40, 1994.

[35] D.N. Metaxas. Physics-Based Deformable Models. Kluwer Academic Publishers, Boston, 1996, ISBN 0-7923-9840-8.

[36] A. Newell. Unified Thories of Cognition. Harvard University Press, Cambridge, MA, 1990.

[37] A. Newell. Unified Theories of Cognition and the Role of Soar in *Soar: A Cognitive Architecture in Perspective* J.A. Michon and A. Akyurek (Editors), Kluwer Academic Publishers, Dordrecht, The Netherlands. 1992

[38] C.W. Reynolds. Flocks, Herds and Schools: A Distributed Behavioral Model. In M.C.Stone (Ed.) Computer Graphics (SIGGRAPH'87 Proceedings), volume 21, pp. 25-34, July, 1987.

[39] J.W. Rickel, W.L. Johnson. STEVE: A Pedagogical Agent for Virtual Reality (video), in Proceedings of the Second International Conference on Autonomous Agents, May 1998.

[40] J.W. Rickel, W.L. Johnson. Task-Oriented Dialogs with Animated Agents in Virtual Reality. In Proceedings of the First Workshop on Embodied Conversational Characters, Tahoe City, CA, October 1998.

[41] J.W. Rickel, W.L. Johnson. Animated Pedagogical Agents for Team Training. In ITS '98 Workshop on Pedagogical Agents, San Antonio, TX, pp. 75-77, August, 1998.

[42] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Englewood Cliffs, NJ: Prentice Hall, 1995.

[43] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. Artificial Intelligence(5):115-135, 1974.

[44] E. Sacerdoti. The non-linear nature of plans. Proceedings of IJCAI-75, pp. 206-214, Stanford, CA, 1975.

[45] E. Sacerdoti. A Structure for Plans and Behaviors, New York: Elsevier North-Holland, 1977.

[46] L. Suchman. Response to Vera and Simon's Situated Action: A Symbolic Interpretation. Cognitive Science 17, pp. 71-75 (1993).

[47] N.A. Stillings *et al.* Cognitive Science: An Introduction. The MIT Press, Cambridge, MA. ISBN 0-262-19257-8, 1987.

[48] D.Thalmann and P. Becheiraz. A Behavioral Animation System for Autonomous Actors Personified by Emotions, Proc. First Workshop on Embodied Conversational Characters (WECC '98), Lake Tahoe, CA.

[49] X. Tu and D. Terzopoulos. Artificial Fishes: Physics, Locomotion, Perception, and Behavior. In A. Glassner (Ed.) Computer Graphics (SIGGRAPH'94 Proceedings) Orlando, Florida, pp. 43-50, July, 1994.

[50] X. Tu. Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior, Ph.D Dissertation , Department of Computer Science, University of Toronto, January, 1996.

[51] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier Principles for Emotion-based Human Figure Animation. Proceedings of SIGGRAPH 95, pp. 91-96. LA, August 1995.

[52] A.H Vera and H.A. Simon. Situated Action: A Symbolic Interpretation. Cognitive Science 17, 7-48 (1993).

[53] A. Witkin, K. Fleisher, and A. Barr. Energy Constraints on Parameterized Models. Computer Graphics, 21(3):225-232, 1987.

[54] A. Witkin and M. Kass. Spacetime Constraints, Computer Graphics, 22(4):159-168, 1988.

[55] S. Wood. Planning and Decision Making in Dynamic Domains. Ellis Horwood: Chichester, England, 1993.