Modularity and Design in Reactive Intelligence

Joanna J. Bryson and Lynn Andrea Stein
MIT Artificial Intelligence Laboratory, NE43-833
Cambridge, MA 02139, USA
joanna@ai.mit.edu, lynn.stein@olin.edu

Abstract

Software design is the hardest part of creating intelligent agents. Therefore agent architectures should be optimized as design tools. This paper presents an architectural synthesis between the three-layer architectures which dominate autonomous robotics and virtual reality, and a more agent-oriented approach to viewing behavior modules. We provide an approach, Behavior Oriented Design (BOD), for rapid, maintainable development. We demonstrate our approach by modeling primate learning.

1 Introduction

The last decade of research has shown impressive convergence on the gross characteristics of software architectures for complete agents such as autonomous robots or virtual reality (VR) characters [Kortenkamp *et al.*, 1998; Sengers, 1999; Bryson, 2000a]. The field is now dominated by 'hybrid', three-layer architectures [Gat, 1998]. The hybrids combine: (1) *behavior-based* AI, the decomposition of intelligence into simple, robust, reliable modules, (2) *reactive planning*, the ordering of expressed actions via carefully specified program structures, and (optionally) (3) *deliberative planning*, which may inform or create new plans or behaviors.

In this paper, we take the view that software design and methodology are critical to the advances that have been made in complete agents. We contribute architectural features that further facilitate the human engineering of these agents, including the effort to incorporate reliable learning and planning into the agent. We do this by enhancing the programmability of reactive plans and reintroducing modularity to the organization of the 'plan primitives'. In our system, plan primitives are not themselves modules, but *interfaces* to semi-autonomous behavior modules encapsulating specialized state for learning and control. This brings hybrid architectures closer to multi-agent systems (MAS) and behaviors closer to active objects [van Eijk *et al.*, 2001]. We also present a methodology for constructing complete agents in the architecture we describe, and some example agents.

2 Intelligence by Design

One of the most important aspects of the reactive revolution of the late 1980's is often overlooked. The break-throughs in robotics associated with reactive and behavior-based systems are usually attributed to the loss of deliberate planning and/or explicit representations. The real contribution of the reactive paradigm was explained nearly a decade earlier: you can't learn something you don't practically already know [Winston, 1975], nor, by extension, plan something you can't nearly already do. The reason is simple combinatorics [Chapman, 1987; Wolpert, 1996]. As evolutionary linguists and case-based reasoning researchers often tell us, what makes humans so intelligent are our exceptional ability to store and transmit solutions we manage to find [e.g. Hammond, 1990; Knight *et al.*, 2000].

Reactive and behavior-based AI thus facilitate the advance of AI in two ways. First, by severely deprecating both planning and state (and consequently learning), the reactive approach increased by default the emphasis on one of the largest problems of AI and software in general: design. Second, the behavior-based approach made fashionable a proven software design methodology: modularity.

The primary contributions of this paper are methodological. We provide more productive ways of creating hybrid systems. This is not to say that developing good software is ever easy, or that learning or productive planning should not be used to the fullest extent practical. In fact, we emphasize the role of learning in our methodology. What we *are* saying is that we favor approaches to hybrid systems that facilitate human design, because humans designers do most of the hard work in artificial intelligence.

3 Fundamentals of Reactive Plans

The terms 'reactive intelligence', 'reactive planning' and 'reactive plan' appear to be closely related, but actually signify the development of several different ideas. *Reactive intelligence* controls a reactive agent — one that can respond very quickly to changes in its situation. Reactive intelligence has sometimes been equated with statelessness, but that association is exaggerated. Reactive intelligence *is* associated with minimal representations and the lack of deliberation.

Reactive planning is something of an oxymoron. It describes the way reactive systems handle the problem traditionally addressed by conventional planning: action selection. Action selection is the ongoing problem (for an autonomous agent) of deciding what to do next. Conventional deliberate planning assumes the segmentation of intelligent behavior into the achievement of discrete goals. A deliberate planner constructs a sequence of steps guaranteed to move an agent from its present state toward a goal state. Reactive planning, in contrast, chooses only the immediate next action, and bases this choice on the current context. In most architectures utilizing this technique, reactive planning is facilitated by the presence of *reactive plans.* Reactive plans are stored structures which, given the current context (both internal and environmental), specify the next act.

We will quickly address the concerns some researchers have with reactive planning. Hierarchical plans and centralized behavior arbitration *are* biologically plausible [Byrne and Russon, 1998; Prescott *et al.*, to appear]. They are sufficiently reactive to control robots in complex dynamic domains [e.g. Kortenkamp *et al.*, 1998] and have been shown experimentally to be as reactive as non-hierarchical, decentralized systems [Bryson, 2000b]. Although they do provide a single failure point, this can be addressed either by standard MAS techniques [e.g. Bansal *et al.*, 1998], or accepted as a critical system, like a power supply or a brain. Finally, as demonstrated by coordinated MAS and in Section 4 below, they do not preclude semi-autonomous behavior modules operating in parallel.

3.1 Basic Elements of Reactive Plans

Reactive plans support action selection. At any given time step, most agents have a number of actions which could potentially be expressed, at least some of which cannot be expressed simultaneously, for example sitting and walking. In architectures without centralized action selection, [e.g. Arkin, 1990; Maes, 1990], the designer must fully characterize *for each action* how to determine when it might be expressed. For engineers, it is generally easier to describe the desired behavior in terms of sequences of events. This strategy is complicated by the non-determinism of environments. Several types of events may interrupt the completion of an intended action sequence. These events fall into two categories: (1) some combination of alarms, requests or opportunities may make pursuing a different plan more relevant, and (2) some combination of opportunities or difficulties may require the current 'sequence' to be reordered. We have determined 3 element types for reactive plans which, when combined, support both of these situations.

Simple Sequences

The first element type is a simple sequence of primitive actions: $\iota_1, \iota_2, \ldots \iota_n$. In our own architecture, we call this element an *action pattern*. Including the sequence as an element type is useful for two reasons. First, it allows an agent designer to keep the system as simple as possible, which both makes it more likely to succeed, and communicates more clearly to a subsequent designer the expected behavior of that plan segment. Second, it allows for speed optimization of elements that are reliably run in order, which can be particularly useful in sequences of preconditions or in fine motor control.

Executing a sequential plan involves priming or activating the sequence, then releasing for execution the first primitive act ι_1. The completion of any ι_i releases the following ι_{i+1} until no active elements remain. Notice that this is *not* equivalent to the process of *chaining*, where each element is essentially an independent production, with a precondition set to the firing of the prior element. A sequence is an additional piece of control state; its elements may also occur in different orders in other sequences.

Basic Reactive Plans

The next element type supports the case when changes in circumstance can affect the order in which a plan is executed. We developed this idiom independently, and called it a *competence*. However, it occurs in a number of other architectures [e.g. Fikes *et al.*, 1972; Nilsson, 1994; Correia and Steiger-Garção, 1995] and is so characteristic of reactive planning, that we refer to the generic idiom as a *Basic Reactive Plan* or BRP.

A *BRP step* is a tuple $\langle \pi, \rho, \alpha \rangle$, where π is a priority, ρ is a releaser, and α is an action. A *BRP* is a small set (typically 3–7) of plan steps $\{\langle \pi_i, \rho_i, \alpha_i \rangle *\}$ associated with achieving a particular goal condition. The releaser ρ_i is a conjunction of boolean perceptual primitives which determine whether the step can execute. Each priority π_i is drawn from a total order. Each action α_i may be either another BRP or a sequence as described above.

The order in which plan steps are expressed is determined by two means: the releaser and the priority. If more than one step is operable, then the priority determines which step's α is executed. If no step can fire, then the BRP terminates. The top priority step of a BRP is often, though not necessarily a goal condition. In that case, its releaser, ρ_1, recognizes that the BRP has succeeded, and its action, α_1 terminates the BRP.

The details of the operation of a BRP are best explained through an example. BRPs have been used to control such complex systems as mobile robots and flight simulators [Correia and Steiger-Garção, 1995; Benson, 1996; Bryson and McGonigle, 1998]. However, for clarity we draw this example from blocks world. Assume that the world consists of stacks of colored blocks, and that an agent wants to hold a blue block.

$$\uparrow \left\langle \begin{array}{crl} \underline{\text{Priority}} & \underline{\text{Releaser}} \Rightarrow & \underline{\text{Action}} \\ 4 & \text{(holding) (held 'blue)} \Rightarrow & \textit{goal} \\ 3 & \text{(holding)} \Rightarrow & \text{drop-held, lose-fix} \\ 2 & \text{(fixed-on 'blue)} \Rightarrow & \text{grasp-top-of-stack} \\ 1 & \text{(blue-in-scene)} \Rightarrow & \text{fixate-blue} \end{array} \right\rangle \tag{1}$$

In this case priority is strictly ordered and represented by position, with the highest priority step at the top. We refer to steps by priority.

This single reactive plan can generate a large number of expressed sequential plans. In the initial context of a red block stacked on a blue block, we might expect the plan 1–2–3–1–2–4 to execute. But if the agent is already fixated on blue and fails to grasp the red block successfully on first attempt, the expressed plan would look like 2–1–2–3–1–2–4. If the unsuccessful grasp knocked the red block off the blue, the expressed plan might be 2–1–2–4. This BRP is identically robust and opportunistic to changes caused by another agent.

If an action fails repeatedly, then the above construction might lead to an indefinite behavior cycle. This can be prevented through several means. Our competences allow a retry limit to be set at the step level. Thus a *competence step* is really a quadruple $\langle\pi,\rho,\alpha,\eta\rangle$, where η is an optional maximum number of retries. Other systems often have generic rules for absence of progress or change.

The most significant feature of a BRP is that it is relatively easy to engineer. To build a BRP, the developer imagines a worst-case scenario for solving a particular goal. The priorities are then set in the inverse order that the steps might have to be executed. Next, preconditions are set, starting from the highest priority step, to determine whether it can fire. For each step, the preconditions are simplified by the assurance that the agent is already in the context of the current BRP, and that no higher priority step can fire.

Plan Manipulation

Finally, a reactive system must be able arbitrate *between* plans. We do this with a third element type called a *drive collection*, also based on the BRP. A 'step', or in this case, *drive element*, now has five elements $\langle\pi,\rho,\alpha,A,\nu\rangle$. For a drive, the priority and releaser π and ρ are as in a BRP, but the actions are different. A is the *root* of a BRP hierarchy, while α is the *currently active* element of the drive. If a drive element is selected for action, but its α is null because a BRP or sequence has just terminated, then α is set to the A for that drive. The drive element begins action selection again from the root of its hierarchy.

This system improves reaction time by eliminating the stack that might be produced when traversing a plan hierarchy. On every program cycle, the agent checks only the drive-collection priorities, and at most one other set of priorities, if α is currently a BRP rather than a sequence. It also allows the agent to periodically re-traverse its decision tree and notice any context change. This approach also allows the hierarchy of BRPs to contain cycles or oscillations, which are frequently useful patterns of behavior. Since there is no stack, there is no *obligation* for a competence chain to terminate.

The fifth member of a drive element, ν, is an optional maximum *frequency* at which this element is visited. This is a convenience for clarity, like the retry limit η on the competence steps — either could also be controlled through preconditions. The frequency in a real-time system sets a temporal limit on how frequently a drive element may be executed. For example, on a mobile robot [Bryson and McGonigle, 1998] we had the highest priority drive-element check robot's battery level, but this was only executed every two minutes. The next highest priority was checking the robot's sensors, which happened at 7Hz. Other, lower-priority processes then used the remaining interspersed cycles.

One further characteristic discriminates drive collections from competences / BRPs. Only one element of a competence is expected to be operating at any one time, but for a drive collection, multiple drives may be effectively active simultaneously. If a high-priority drive takes the attention of the action-selection mechanism, the program state of any active lower drive is preserved. In the case of our robot, if the navigation drive is in the process of selecting a destination when the battery needs to be checked, attention returns to the selection process exactly where it left off once the battery drive is finished. Further, action primitives in our system are not stand-alone, consumatory acts, but are interfaces to semi-autonomous behaviors which may be operating in parallel (see Section 4 below.) Thus the action 'move' in the robot's script merely confirms or transmits current target velocities to already active controllers. A moving robot does not stop rolling while its executive attends to its batteries or its sensors.

3.2 Discussion — Other Reactive Architectures

We refer to reactive plan structures as described above as Parallel-rooted, Ordered Slip-stack Hierarchical (POSH) action selection. Although we freely distribute implementations of this architecture in both C++ and Lisp / CLOS, we have also implemented versions of POSH action selection in other architectures [Bryson and Stein, 2001].

The functionality of the BRP, which in our experience is a critical element of reactive planning, is surprisingly missing from several popular architectures. In effect, architectures using middle layers like PRS [Georgeff and Lansky, 1987] seem to expect that most of behavior can be sequenced in advance, and that being reactive is only necessary for dealing with external interruptions by switching plans. On the other hand, architectures such as subsumption [Brooks, 1991] or ANA [Maes, 1990] expect that there is so *little* regularity in the arbitration of behavior that all actions must be considered for execution at all times. We have found the most expedient solution to the design problem of reactive planning is to categorize selection into things that need to be checked regularly, things that only need to be checked in a particular context, and things that one can get by not checking at all. These categories correspond to our three types of plan elements: drive collections, competences, and action patterns.

4 Semi-Autonomous Behavior Modules and the Role of Perceptual State

Besides emphasizing the use of modularity, the behavior-based movement also made an important engineering contribution by emphasizing specialized learning [e.g Brooks, 1991, pp. 158–9]. Specializing learning increases its probability of success, thus increasing its utility in a reliable agent. Similarly, modularity simplifies program design, at least locally, thus increasing the probability of correctness. Governing the interaction of multiple independent behavioral modules can be a difficulty, but we have already addressed this issue in the previous section.

Consider this description of standard hybrid systems:

> The three-layer architecture arises from the empirical observation that effective algorithms for controlling mobile robots tend to fall into three distinct categories: (1) reactive control algorithms which map sensors directly onto actuators with little or no internal state; (2) algorithms for governing routine sequences of activity which rely extensively on internal state but perform no search; and (3) time-consuming search-based algorithms such as planners. [Gat, 1998, p. 209]

Gat's view of three-layer architectures is particularly close to our own view of agent intelligence, because it puts control firmly in the middle, reactive-plan layer. The deliberative 'layer' operates when prompted by requests. However, we differ on the notion that there are many actions which can really map sensors to actuators with little internal state or consideration for the past.

Nearly all perception is ambiguous, and requires expectations rooted in experience to discriminate. For a mobile robot, this 'experience' may be from the last half a second, for discriminating sonar 'ghosts', half a minute, to move around a bumped object invisible to sonar, or days, as in remembering a local map. Primitive actions governed by the reactive plans may depend on any of this information. We do not believe this information should be passed between 'layers' either by micro-management or as parameters. Rather, in our model, the primitives of a reactive plan interface directly to semi-autonomous behavior modules. Each module maintains its own state and may possibly perform its own 'time-consuming' processes (such as memory consolidation or search) in parallel to the main activity of the complete agent. Thus our view of agent control is very similar to Gat's, except that (1) we increase the number, specificity and potential simplicity of the modules composing his top layer, and (2) we replace the notion of a bottom layer with the that of an interface between the action selection module of the robot and its (other) behavior modules.

5 Developing an Agent

The process of developing an agent with these two attributes, POSH action selection and a behavior library, we call Behavior Oriented Design. The analogy between BOD and OOD is not limited to the metaphor of the behavior and the object, nor to the use of methods on the behavior objects as primitives to the reactive plans. The most critical aspect of BOD is its emphasis on the design process itself. The old problem of behavior decomposition (and new, analogous one for MAS) is solved by using state requirements, as in modern object decomposition. Also as in OOD, BOD emphasizes cyclic design with rapid prototyping. The process of developing an agent alternates between developing libraries of behaviors, and developing reactive plans to control the expression of those behaviors.

5.1 The Initial Decomposition

The steps of initial decomposition are as follows. (1) Specify at a high level what the agent is intended to do. (2) Describe likely activities in terms of sequences of actions (prototype reactive plans.) (3) Identify sensory and action primitives from these sequences. (4) Identify the state necessary for these primitives, clustering them by shared state (prototype behaviors). (5) Identify and prioritize goals or drives that the agent may need to attend to (prototype POSH drive roots). (6) Select a first behavior to implement.

5.2 The Development Process

The remainder of the development process is not linear. It consists of the following elements, applied repeatedly as appropriate: coding behaviors, coding reactive plans, testing and debugging code, and revising the specifications made in the initial phase.

Usually only one behavior and one reactive plan will be actively developed at a time. We strongly suggest maintaining the lists developed in the initial phase as documentation. Where possible, such documentation should be part of active code. For example, the primitive list should be a file of code specifying the interface calls. Similarly, old reactive plans should be preserved with their development history and used as a test suite as modifications are made to the behavior libraries.

5.3 Revising the Specifications

The most interesting part of the BOD methodology is the set of rules for revising the specifications. The main design principle of BOD is *when in doubt, favor simplicity.* A primitive is preferred to an action sequence, a sequence to a competence. Heuristics then indicate when the simple element must be decomposed into a more complex one. One guiding principle is to reduce redundancy. If a particular plan or behavior can be reused, it should be. If only part of a plan or a primitive action can be used, then a change in decomposition is called for. In the case of an action primitive, the primitive should be decomposed into two or more primitives, and the original action replaced by a sequence. If a sequence sometimes needs to contain a cycle, or often does not need some of its elements to fire, then it should really be a competence. A new plan element should have the same name and functionality as the primitive or sequence it replaces. This allows established plans to continue operating without change.

Reactive plan elements should not require long or complex triggers. Perception should be handled at the behavior level; it should be a skill. A large number of triggers should be converted into a single perceptual primitive. Another problem that crops up in competence design can be the presence of too many elements. More than seven elements in a competence, or difficulty in appropriately prioritizing or setting triggers, indicates that a competence needs to be decomposed into two. If several of the elements can be seen as working to complete a subgoal, they may be moved to a child competence. If two or more of the elements always follow each other in sequence, they should be converted into an action pattern. If the competence is actually trying to achieve its goal by two different means, then it should be broken into two sibling competences which are both inserted into the original competence's parent.

6 Example: Modeling Transitivity in a Non-Human Primate

Although we have used our methodology on a mobile robot [Bryson and McGonigle, 1998] and on virtual reality characters [Bryson and Thórisson, 2001], we find artificial life (ALife) more conducive for quantitative comparisons [e.g. Bryson, 2000b] and for communicating with researchers. We thus illustrate BOD by building an ALife model of primate intelligence. Unfortunately, this shows only degenerate examples of drive collections, but space is limited.

We begin by modeling the ability of a monkey to perform transitive inference [McGonigle and Chalmers, 1977]. This task is interesting, because it was initially believed to require reasoning, which was in turn considered to require language. Squirrel monkeys (*saimiri scuireus*) were trained on four pairs of colored pucks; AB, BC, CD, and DE; to favor the earlier element of the pair. Transitivity is the ability to generalize this pairing spontaneously (without further training) when first presented with the novel pairs such as AC, AD, BD and so on. The behavior of the monkeys on this task has already been modeled by Harris and McGonigle [1994], who modeled the transitive capability as a prioritized stack of production rules of the type 'avoid E' or 'select A'. Based on the nature of the errors the monkeys made, and their separate performance on three-item sets, different stacks were built representing the rules learned by each monkey.

We begin by replicating a simplified version of Harris's system modeling a skilled monkey. To simplify the simulation task, we model both the monkey and its testing environment as a single intelligent agent with two behaviors. The 'monkey' has two pieces of variable state — its hand and its visual attention, the test box has only a test-board for holding two or three items.

$$
\begin{aligned}
&\textit{life} \Rightarrow \left\langle\left\langle \begin{array}{r} \text{(no-test)} \Rightarrow \text{new-test} \\ \text{(grasping)} \Rightarrow \text{finish-test} \\ \Rightarrow \textbf{elvis-choice} \end{array} \right\rangle\right\rangle \\
&\textbf{elvis-choice} \Rightarrow \left\langle \begin{array}{r} \text{(see-red)} \Rightarrow \textbf{noisy-grasp} \\ \text{(see-white)} \Rightarrow \text{grasp-seen} \\ \text{(see-blue)} \Rightarrow \text{grasp-seen} \\ \text{(see-green)} \Rightarrow \text{grasp-seen} \end{array} \right\rangle \\
&\textbf{noisy-grasp} \Rightarrow \langle \text{screech} \rightarrow \text{grasp-seen} \rangle
\end{aligned}
\tag{2}
$$

This plan gives an example of each element type described in Section 3, though the action pattern is gratuitous. Priority is again listed from the top. The drive collection, life, has no goal, so it never ends. New-test is a call to the test behavior which randomly resets the test board; finish-test clears it. The seeing primitives all map to a single method on the monkey behavior, which performs a 'visual' scan of the test board and leaves visual attention on an appropriately colored object, if it exists. Grasp-seen, also a method on the monkey behavior, is thus able to use deictic reference without variable passing.

We now enhance the model by forcing the monkey to learn the ordering of the pucks. This also requires augmenting the test-behavior to reward the monkey appropriately.

$$
\begin{aligned}
&\textit{life} \Rightarrow \left\langle\left\langle \begin{array}{r} \text{(no-test)} \Rightarrow \text{new-test} \\ \text{(rewarded)} \Rightarrow \textbf{end-of-test} \\ \text{(grasping)} \Rightarrow \textbf{elvis-reward} \\ \Rightarrow \textbf{educated-grasp} \end{array} \right\rangle\right\rangle \\
&\textbf{elvis-reward} \Rightarrow \left\langle \begin{array}{r} \text{(find-red)} \Rightarrow \text{reward-found} \\ \text{(find-white)} \Rightarrow \text{reward-found} \\ \text{(find-blue)} \Rightarrow \text{reward-found} \\ \text{(find-green)} \Rightarrow \text{reward-found} \end{array} \right\rangle \\
&\textbf{educated-grasp} \Rightarrow \langle \text{adaptive-choice} \rightarrow \text{grasp-seen} \rangle \\
&\textbf{end-of-test} \Rightarrow \langle \text{consider-reward} \rightarrow \text{finish-test} \rangle
\end{aligned}
\tag{3}
$$

We add a new behavior, sequence learning, to the system, which though ascribable to the monkey, is separate from the existing monkey-body behavior. The sequence-learner contains a list of known objects with weights, a 'significant difference' and a 'weight shift'. If the monkey is correct in its discrimination, but its certainty was less than significant-difference, then consider-reward adds weight-shift to the weight of the winner, and renormalizes the weights in the list. If it is wrong, consider-reward shifts the weight to the loser.

The test machine is now in charge of both setting and rewarding the behavior. The new primitive 'find' searches the world for a colored puck, then if it is found, a reward (or lack of reward) is given based on whether the machine is attending to the monkey's hand or the test-board. The end-of-test action-pattern calls actions in sequence from two different behaviors — the monkey's sequence learner learns, then the test box records and resets. Educated-grasp is now a method on sequence-learner; it does visual examination, evaluation, and the grasp.

The above is just the first two steps of the development of a learning version of Harris' model. This example demonstrates how functionality can be easily added, and shifted between plans and behaviors. The second model above converges to a correct solution within 150 trials, with significant-difference set to .08, and weight-shift to .06. We have additional forthcoming results showing that adding another layer of learning (the select-avoid rules) results in further characteristics typical of primate learning.

The sequence-learning task is interesting because it illustrates not only the interaction between reactive plans and modular behaviors, but also begins to model how a behavior might be designed to learn a reactive plan. However, the BOD methodology is not constrained to modeling only a single agent. We are currently modeling conflict resolution in primate colony social interactions in collaboration with Jessica Flack of Emory University. In this work, we use different processes to model different agents. Each agent has its own instance of the behavior objects and its local copy of a POSH plan.

7 Conclusions

Software engineering is the key problem of developing complete agents. The advances made due to the reactive and behavior-based movements come primarily because they trade off slow or unreliable on-line processes of search and learning for the one-time cost of development, and by emphasizing the use of modularity. These are not reasons to fully abandon learning and search, but they are reasons to use it only in constrained ways likely to be successful.

It is easiest to design action selection if one exploits the sequencing and prioritization skills of programmers. We have shown how sequences can be adapted into powerful reactive plans. One of the components we consider critical to successful reactive planning, the BRP, is present to our knowledge in only one architecture with a reasonably large user base, TR [Nilsson, 1994]. Most other architectures either only allow for reactivity *between* plans, or don't allow for structured plan elements at all. We have also created a mapping between the three layer architectures that dominate complete agent research today, and the original behavior-based architectures, which are more like today's MAS architectures. We suggest that the primitive elements typical of the lowest layer of a three-layer architecture should interface to semi-autonomous behavior modules, which are comparable to high-level processes in some three-layer architectures. This allows the basic actions coordinated by the reactive plans direct access to appropriately processed state necessary for informing perception and determining the parameters of actuation.

References

Ronald Arkin. Integrating behavioral, perceptual and world knowledge in reactive navigation. *Robotics and Automation*, 6(1):105–122, 1990.

Arvind K. Bansal, Kotagiri Ramohanarao, and Anand Rao. Distributed storage of replicated beliefs to facilitate recovery of distributed intelligent agents. In Munindar P. Singh, Anand S. Rao, and Michael J. Wooldridge, editors, *Intelligent Agents IV (ATAL97)*, pages 77–92, Providence, RI, 1998. Springer.

Scott Benson. *Learning Action Models for Reactive Autonomous Agents*. PhD thesis, Stanford University, December 1996. Department of Computer Science.

Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

Joanna Bryson and Brendan McGonigle. Agent architecture as object oriented design. In Munindar P. Singh, Anand S. Rao, and Michael J. Wooldridge, editors, *The Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL97)*, pages 15–30, Providence, RI, 1998. Springer.

Joanna Bryson and Lynn Andrea Stein. Architectures and idioms: Making progress in agent design. In C. Castelfranchi and Y. Lespérance, editors, *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL2000)*. Springer, 2001. *in press*.

Joanna Bryson and Kristinn R. Thórisson. Dragons, bats & evil knights: A three-layer design approach to character based creative play. *Virtual Reality*, 2001. *in press*.

Joanna Bryson. Cross-paradigm analysis of autonomous agent architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):165–190, 2000.

Joanna Bryson. Hierarchy and sequence vs. full parallelism in reactive action selection architectures. In *From Animals to Animats 6 (SAB00)*, pages 147–156, Cambridge, MA, 2000. MIT Press.

Richard W. Byrne and Anne E. Russon. Learning by imitation: a hierarchical approach. *Brain and Behavioral Sciences*, 21(5):667–721, 1998.

David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

Luis Correia and A. Steiger-Garção. A useful autonomous vehicle with a hierarchical behavior control. In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, *Advances in Artificial Life (Third European Conference on Artificial Life)*, pages 625–639, Berlin, 1995. Springer.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

Erran Gat. Three-layer architectures. In David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, chapter 8, pages 195–210. MIT Press, Cambridge, MA, 1998.

M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.

Kristian J. Hammond. Case-based planning: A framework for planning from experience. *The Journal of Cognitive Science*, 14(3), September 1990.

Mitch R. Harris and Brendan O. McGonigle. A modle of transitive choice. *The Quarterly Journal of Experimental Psychology*, 47B(3):319–348, 1994.

Chris Knight, Michael Studdert-Kennedy, and James R. Hurford, editors. *The Evolutionary Emergence of Language: Social function and the origins of linguistic form*. Cambridge University Press, 2000.

David Kortenkamp, R. Peter Bonasso, and Robin Murphy, editors. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. MIT Press, Cambridge, MA, 1998.

Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents : Theory and Practice from Biology to Engineering and back*, pages 49–70. MIT Press, Cambridge, MA, 1990.

Brendan McGonigle and Margaret Chalmers. Are monkeys logical? *Nature*, 267, 1977.

Nils Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

Tony J. Prescott, Kevin Gurney, F. Montes Gonzalez, and Peter Redgrave. The evolution of action selection. In David McFarland and O. Holland, editors, *Towards the Whole Iguana*. MIT Press, Cambridge, MA, to appear.

Phoebe Sengers. *Anti-Boxology: Agent Design in Cultural Context*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1999.

Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Generalised object-oriented concepts for inter-agent communication. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII (ATAL2000)*. Springer, 2001.

Patrick Winston. Learning structural descriptions from examples. In Patrick Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill Book Company, New York, 1975.

David H. Wolpert. The lack of A priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.