Scaling Intelligent Robotics

Joanna Bryson
Laboratory for Cognitive Neuroscience and Intelligent Systems
University of Edinburgh
Level 8, Appleton Tower
Edinburgh EH8 9JZ, UK
joannab@dai.ed.ac.uk

Abstract

This paper describes an approach, architecture and methodology for programming intelligent robots. We argue that due to practicality and scaling issues, behavior based robotics is the most promising current approach. We describe four stumbling blocks to building better behavior based robots: deficiencies in learning, goal structures, control state, and programmability. In answer, we present our own architecture, Edmund, and describe experiments in blocks world and on a mobile robot.

1 Introduction

This paper addresses the problem of making a robot do what you want. Experience shows this requires not just precision in executing commands, but autonomous intelligence. We want instructions such as "grasp this object then move there" without worrying about the precise location of the object or minor impediments in its intended path.

Abstract instructions require generality and robustness best achieved by behavior based robotics. In this approach, intelligence is decomposed into perception/action units. For example, one behavior may be grasping, which could rely on an interaction between various motors and strain gauges. Another might be retrieving a part, which might combine vision and feedback from the grasp behavior with signals to motion behaviors.

Segmenting robot design this way is a successful approach to solving specific problems such as uncalibrated grasping and moving robots in particular directions. However, recombining these segments into complex and useful programs has proven challenging. So far, most successful programming of intelligent robots has been conducted using hybrid systems, where a behavior based substrate is linked to a planning system. Examples are Firby's RAP architecture for mobile robots [Firby, 1996], Malcolm's SOMASS assembly robotics system [Malcolm *et al.*, 1989], and Benson's TRAIL [Benson, 1996] which learns to fly a flight simulator. But reliance on classical planning still exposes these systems to issues of scaling, rigidity and combinatorics as their domains and tasks become more complex.

Such systems assume only lower, reflexive levels of action can be successfully modeled as behavior based — intelligence requires a controlling, rational level. We disagree. Under the architecture we have developed, the modularity and parallelism of behavior based systems are dominant. Elements such as learning and planning are, if needed, themselves parts of tightly designed behavior modules. This paper describes the motivations for our approach, the approach itself, and applications of the approach in both real and simulated domains.

2 Artificial Intelligence and Behavior Based Robotics

Consider the archetypical task of developing a household robot. Conventional programming methodology cannot cope with the complexity of controlling such a robot. Programmers cannot anticipate the number of situations encountered by a robot in this task, the level of detail required to control the motors, or the ambiguity of the appearance of the environment from the perceptual information a robot receives.

Tasks like picking a path across an actively used room involve large numbers of unknowns which generate combinatorially large spaces of possible actions. To address such problems, the AI community has developed automated planning. These systems involve characterizing the current environment and goal states as predicates, and the robot's actions as operations on these predicates. Plans are created by finding sequences of actions that convert the current state into the goal.

Unfortunately, this approach is expensive both computationally and in preparation. The number of attributes needed to describe the environment, of side effects and potential side effects of actions, and of rules or preconditions governing an operation are easily underestimated. "Common sense" inhibits our appreciation of the number of potential actions and situations which we learn to favor or ignore. The size of the search space is also increased by action outcomes being probabilistic, and by the environment changing independently over the execution period of the plan. These problems are encountered even in simulated systems; planners seldom address the issues of perception and actuation in a real robot. Yet in animals, perception and motor control take far more of the brain's mass than planning and motivation. The behavior based approach to AI assumes that sensing and control are in fact the harder problems of intelligence.

Behavior based robotics may be thought of as the customized design of a system of miniature plans. Each behavior knows how to behave for a particular situation. Each behavior directly perceives which situation exists, at least insofar as it concerns that behavior. Brooks' subsumption architecture, the approach that first defined the field, has the programmer fully specifying the robots' actions. Each "carefully engineered" behavior [Brooks, 1991b] is the programmer saying "I may not know how to direct the robot all the way to the kitchen, but I know what it should do when it bumps into something."

Behavior based AI emphasizes perception and actuation as central to each unit of its intelligence. This and a design methodology of incremental development on real robots result in behavior based robots being characterized by some of the fastest and most natural appearing activity of any approach. For example, Polly [Horswill, 1993] navigated in a cluttered, populated office environment at speeds of 1 m/sec with almost no impacts, using vision for its primary perception techniques.

The problems of behavior based design are generally considered to be *behavior decomposition* and *behavior arbitration.* Decomposition is the problem of deciding how to modularize the units of intelligence. Arbitration is a problem of parallel modules sharing finite resources — multiple behaviors active simultaneously conflicting over the use of a single actuator. The two problems are codependent. Reducing the number of behaviors or constraining modularization to minimize conflict can significantly simplify arbitration. Many papers in the literature display fewer behaviors than the original MIT archetypes, but this solution defeats the purpose of the behavior based approach and has obvious difficulty in scaling complexity.

3 Problems and Solutions for Behavior Based Programming

To some extent, problems of behavior decomposition and arbitration are fundamental to the behavior based paradigm, equivalent to issues of function decomposition and control flow in conventional software design. Such problems can be exaggerated or alleviated by convention and approach. We have identified four issues as being key stumbling blocks in standard behavior based design. We have also developed an architecture to address them. This section identifies the issues, and introduces our approach in their context. The next section describes our architecture.

3.1 Learning

The lack of learning in early behavior based robots has already been well criticized, but is still worth examining. The early lack of learning was a natural consequence of one of the fundamental principles of the approach. As Brooks said in one of the field's seminal papers, "There should be no representation. No variables instantiated, no rules matched, and no choices made." [Brooks, 1991a] With no representation there can be no learning, since representation is definitionally the mechanism by which information is stored and recalled. This stance was critical to the programming revolution Brooks helped initiate; adhering to it created the environment for the perceptual innovations mentioned earlier. However, as a strict principle it was recognized as impractical well before the AI Journal published that article. At the IJCAI in the same year, Brooks presented a paper describing behavior based learning. Learning should be part of the behavioral modules and encapsulated within them, and there should be no distinction between process and data [Brooks, 1991b].[1]

Most learning in behavior based systems is in fact confined to a single module, for example [Mataric, 1990] builds a map, or [Pebody, 1995] calibrates a laser. An exception is [Bryson, 1995] where adaptive networks are in several of the modules, and in fact the paper advocates the use of adaptive requirements as a guideline to behavior decomposition. In general though the community favors monolithic learning solutions such as reinforcement learning, genetic algorithms and single neural networks which are used in an attempt to learn the control system as a whole. These resembles the hybrid systems controlled by planners structurally, but are not as successful behaviorally.

We advocate learning as a simple part of the behavior modules. In natural intelligent systems, memory is an inseparable part of perception. Learning in our architecture is storing and recalling information as an integral part of perceptual acts. In the blocks world example below, there is no learning beyond changes in physical and attentional state. But in the mobile robot, adaptation is used for sensor smoothing, bump avoidance and route learning.[2]

3.2 Multiple Goals

The problem of managing multiple goals simultaneously is fundamental to action selection; and a major area of research for automated planning. Two goals may catastrophically interfere, possibly depending on their ordering, possibly necessarily. On the other hand, one goal may be achieved simply during the pursuit of another. Recognizing this situation is *opportunistic.* Unfortunately, the pursuit of opportunism can result in *dithering*

[1] This progression echos that of the pioneering ethologist Conrad Lorenz, who initially maintained that no animal behavior was learned. When he later came to believe in adaptive behavior, he emphasized that the individual's learning must be small and highly constrained in order not to "break" the intricacies of evolved, instinctive behavior.

[2] Even the basic robotic task of avoiding an object just struck requires *some* state; the robot must engage in a specialized behavior *after* its bumpers register a hit.

— being trapped in a local minima between solving two goals, rather than completing either one.

Within behavior based AI, the solution to coping with multiple goals is very much architecture dependent. Brooks' *subsumption architecture* [Brooks, 1991a] handles interference and dithering through design; it is the responsibility of the programmer. The behaviors are grouped into *layers*, each of which is dedicated to achieving one goal. Brooks' methodology develops layers one at a time, testing only the interface of the newest to the one previous.

Brooks lists opportunism as one of the main attributes of the subsumption approach. Since each behavior runs continuously in parallel, if a goal needs to be met its behaviors will immediately recognize the situation. But what if more than one goal needs to be met? Under subsumption, the goals are linearly ordered – newer, higher layers are said to "subsume" the lower. Thus lower layers have priority since higher layers assume them. This strict linearity is restrictive for complex tasks. In reality, different situations leading to the same goals may have different priorities. Often, different situations lead to different goals. It would be difficult to design a subsumption architecture robot that could navigate to a room with a pile of blocks and then proceed to solve a stacking problem.

The linear goal stack also means that adding new layers directly increases the number of active goals at any time. This makes for a kind of dithering. The highest layers, being lowest priority, tend to have very little time dedicated to them.

Maes describes the importance of more powerful and flexible goal handling in her own article in [Maes, 1990]. She presents an alternative behavior based architecture called *spreading activation* in which multiple goals are fully connected to all possible behaviors. Goals have activation values corresponding to their urgency, and sensing behaviors have activation levels depending on the current world state. The behaviors between them are connected by links specifying weighted relationships. Nodes spread their activation to their neighbors either positively or negatively as determined by their links' weights. The first node/behavior over a threshold activation fires. This solution provides for multiple goals and opportunism, but is unworkable on large scales. Its indeterminism leads to dithering of behaviors activated by different goals. Also, the computation cost per cycle grows exponentially on the number of behaviors.

Our solution derives from both these approaches. Goals are ultimately rooted in drives which determine priority based on sensor preconditions and on the goal's activation. The goal's activation is determined both by programmer provided values and internal state. Each goal is associated with a set of behaviors which may themselves be goals. Thus the goal structure is something like a multi-rooted tree. However, when a behavior associated with one goal activates a second, that second goal replaces the first in receiving its activation from the drive. Consequently, a drive may result in cycles of behavior or some other complicated structure of subgoals, but the number of goals actively being pursued at any time is fixed. This reduces the time slicing form of dithering.

If a goal is met, it terminates its behavior. No record is kept of the previous goals that activated it. However, the drive that started the chain will restart it unless the drive is satiated. Thus control may return to the goal's predecessor, or pass to another behavior now applicable if the environment has changed. The system is thus still opportunistic on termination of a goal, the duration of which can be determined by the programmer. In addition, multiple drives are pursued in parallel, so that urgent conditions can come to the attention of the robot, interrupting less important tasks.

3.3 Control State

The tradeoffs between opportunism and dithering in pursuing multiple goals are closely related to a more general problem known as the "external Markov assumption:" the assumption that the correct next behavior can be determined from the external environment. This assumption can be trivially disproved. If a robot is asked to deliver a message to an office on a hallway, nothing in that hallway will indicate which office it was asked to enter. Standard behavior based theory solves this with a hack, such as specially coded envelopes for the robot to carry and refer to.

A similar problem arose with the first PhD thesis robot under subsumption architecture, Herbert [Connell, 1990]. Herbert searched for empty cans, then stopped to grasp them. However, early in the grasping motion, Herbert's arm blocked its camera. The robot would then move on without the can, since its grasping behavior was no longer stimulated. Connell and Brooks overcame this problem by augmenting subsumption behaviors with timers, so they could be inhibited from claiming control for timed periods.

These solutions treat perceptual ambiguity as an exception, with work arounds. In fact, the external Markov assumption is often false. These are fundamental problems, not exceptions, and deserve to be addressed directly in the design of the architecture. Robots with automated planning do not share these problems. Planning requires systems to keep track of current goals and subgoals. Also, the plans themselves serve as a record of what the robot has already done, which together with external state may be enough to deduce the next relevant action, even without attending to a goal. This view of plans is in keeping with "What are Plans For?" [Agre and Chapman, 1988]. This article is often used to support the behavior based approach because it advocates action selection on the basis of a current optimum rather than to a plan. But Agre and Chapman point out in answer to their title that the greatest use for a plan is internal communication.

Our architecture addresses this problem in two ways. As mentioned in the previous section, it attends to both high-level drives and current goals. The current goal should disambiguate any perceptually identical situations: that is, within the context of the select behaviors associated with it, the programmer should ensure that the hidden Markov assumption holds. Our architecture has an additional control state structure, the *behavior sequence*. Modeled after action patterns and chains observed in animals, behavior sequences consist of primitive senses and actions. This is another tool for persistence through ambiguous states, and for reducing the combinatoric cost of action selection.

3.4 Programming

Because behavior based AI places the responsibility for performance almost entirely on the programmer, addressing issues of programmability is crucial to the approach. Our project addresses this problem in several ways. We reduce the difficulties with external Markov assumption and multiple goal pursuit, as explained above. Also, high level goal and sequence constructs are specified in scripts which may be edited with a GUI. This simplifies task programming and facilitates debugging. We are also attending to and documenting methodology.

One design contribution of this architecture is simplifying the task of establishing priorities and atomicity within an essentially parallel structure. Many behavior based programmers find their most difficult task to be removing interference between simultaneously active behaviors. Most architectures require programmers to specify extensive systems of mutual inhibition. Alternatives are the VUB "sub-cognitive" architecture, in which all the behaviors run continuously [Steels, 1994][3], and Blumberg's architecture, [Blumberg, 1994], in which the architecture itself provides locking mechanisms at action bottlenecks to guarantee no interference. Our architecture provides a way to conceptualize and execute essentially parallel behavior in linear ways, whether strictly linear as in the behavior sequences or merely roughly prioritized as in the goal competences.

4 Details of the Architecture

The previous section describes our architecture, which is named Edmund, in terms of motivations and attributes. This section describes it in terms of structure. The following section describes the two domains in which we have implemented our architecture, simulated blocks world and actual mobile robotics, and briefly describes some of the experiments we have performed.

4.1 Program Description

As described above, Edmund consists of behaviors associated with goals. The goals are themselves also behaviors, the sole outcome of which is recognition of success. These clusters of behaviors are referred to as *competences*. A behavior may be another competence, or it may be a *behavior sequence*. Behavior sequences consist of *primitives*, basic behaviors that are classified as *senses* and *actions*. Primitives in themselves are still reasonably high-level behaviors and may consist of a considerable body of code, or exploit such provided on the robot platform. Senses may return complex values, therefor within a behavior sequence they are coupled with a function that determines whether they have "failed", meaning the sequence aborts. Actions can also force abortion in extreme circumstances, but ultimately the success or failure of an action is determined from the environment, as indicated by which behavior triggers next.

```
;;;;; green-on-red – put a red block on a green block
;– the top goal / competence
(green-on-red ()
     ((attn-frame-home t) (attn-frame-shape block) (attn-frame-color red)
       action-trace-up-from-attn (action-frame-shape block)
       (action-frame-color green))
     ((green-on-red 100) (place-green-on-red 80) (place-red 80)
       (grab-green 70) (grab-red 70) (clear-home 30) (check-hand 20)
       (check-home 20)))
;– some of its behaviors
(check-hand () (move-attn-frame-to-hand))
(clear-home ((attn-frame-home t) (attn-frame-shape block))
     (move-action-frame-to-attn (frames-synched-p t) sweep-action-frame))
(grab-green
     ((attn-frame-home t) (attn-frame-shape block) (attn-frame-color red)
       move-action-frame-to-attn action-trace-up-from-attn
       (frames-synched-p t))
     ((action-frame-in-hand t) (action-frame-color green))
     ((grab-green 100) (pick-up-green 90) (lose-block 50)
       (synch-on-green 50) (unbury-green 50) (find-green 40)))
```

Figure 1: A sample script from the blocks world simulation. Competence format: (name trigger goal behavior-list), where "goal" is a behavior sequence with the same name as the competence. Behavior sequence format: (name trigger sequence)

The program system consist of three main code elements.

- Short, high level scripts describing the organization of behaviors. These determine the personality and purpose of the robot for a particular run.
- The architecture itself, which runs the behaviors.
- The code for the primitives. These may consist of calls to the underlying platforms.

There is also a short executive section which configures the robot (or initializes the simulation,) reads the "brain" script into the architecture, and then forces looping of the top level competence. In the blocks world, this competence is just a task, such as "grasp a green block" or "run the copy demo." In the mobile robot, this top competence is a set of drives. Its name is "life", and it has no goal.

4.2 Architecture Specifics

The behavior scripts and primitive code described above are task and hardware specific. Here we describe the specifics of the architecture itself. Code and documentation is available from [website].

The main implementation feature of the architecture is the mock-parallelism needed to maintain reactivity.

[3] This has now been abandoned for a more hierarchical approach because too much programming of each behavior was dedicated to modeling the others to prevent interference.

The competences and behavior sequences are read from scripts into objects that serve essentially as templates. When a competence or behavior is active, a new instance is formed with extra state to hold place while other elements are active.

Behavior sequences are fairly simple; they are implemented with an array and need only keep track of their current index. On receiving control, a sequence executes the behavior at its index. If the behavior succeeds, the index is incremented and the sequence returns `continue`, otherwise it wipes its state and returns `failure`. If the sequence completes it returns `success`. As stated earlier with respect to actions, returning `success` does not indicate that the behavior sequence had its intended effect, only that it completed execution without incident.

A *trigger* is a special case of a behavior sequence that does not time slice, it is assumed to run atomicly. All behaviors in it should therefor be quick, such as senses that return information on already processed perceptual information, or perception-based actions such as shifts in attention.

Competences have two tasks — they select between behaviors and pass control to already selected behaviors. Behavior selection consists of sorting possible behaviors by priority, then selecting the highest priority behavior that triggers. There are two types of competences, interruptible and uninterruptible. An uninterruptible competence does not perform behavior selection until a currently selected behavior has completed. An interruptible competence reprioritizes and tests on each execution. If an interruptible competence is exclusive, the selecting a new behavior flushes the state of a previously executing behavior.

Drives are a special case of a competence designed to run high-level checks in autonomous agents. They are non-exclusive, but also uninterruptible. Consequently, each drive executes its behavior as if in parallel with the others. Drives maintain reference to their initial behavior. If a drive is executing a competence that terminates, the next time the drive is sufficiently activated it will restart its original process. Drives are thus the source of goal persistence in Edmund, though they support the reactivity of the competences they call.

Behavior priority within any competence may be dynamic, because behaviors are allowed to habituate. A habituating behavior lowers its priority as a function of the number of times it is executed, while it recovers as a function of time. Habituation allows for periodic firing of drives, and for guaranteeing termination in other competences by limiting the number of retries. Habituation can be used to enable alternate strategies.

If none of its behaviors have positive priority or trigger, the a competence terminates, wiping state, and returns `failure`. If the goal behavior succeeds, the competence terminates returning `success`. If a competence is not a drive and triggers a new competence, it returns that new competence, which then replaces the first competence in the architecture's execution ordering. As explained in section 3.2, no record is kept of the parent, there is no "goal stack". A competence not returning one of these values returns `continue.`

Primitives, whether actions or sense / value function pairs, only return `success` or `failure.` If an action is essentially durative, like "advance" in the "move" competence of the mobile robot script below, it should be sliced in some reasonable way, as control needs to be returned reasonably quickly. Often this is externally transparent. For example, the move competence sets velocities for both steering and translation; incremental changes in these appear smooth to an external observer. Some primitive behaviors may simply take time, during which the robot will be blind. This is why habituation reduces as a function of real time: the robot may detect periods of sensor deprivation and compensate.

4.3 Decomposition and Methodology

It may seem that an architecture of this complexity would increase the difficulty of behavior decomposition, but in our experience the problem is fairly straightforward. Competences are characterized by the task of attaining a goal. Another characteristic is that any element of behavior towards achieving that goal may be executed any number of times, including not at all. In contrast, behavior sequence is stereotypical in its execution.

The difference between a behavior sequence and a primitive action is more arbitrary. Sometimes theory guides in choosing primitives, as in the blocks-world example below where primitives are provided by a theory of human visual systems. As in conventional programming, reuse can be an indicator. Elements of a behavior can be broken out as separate primitives if the programmer wants one part for another purpose. As just stated, time slicing is also a consideration since actions are atomic, while behavior sequences give up control between each element.

Edmund is designed for rapid prototyping and ease of testing. In general one would follow [Brooks, 1991a] and begin with basic behaviors and competences. However, one can also easily insert new intermediate competences. In the mobile robot example below, the bumpers were added on relatively late because they were acquired after the basic robot. Incorporating them involved creating a new competence and adding it into the drives. It also involved altering the sensor fusion routine already used in the **move** competence. Retesting established behavior was a trivial matter of running earlier scripts and comparing them to their previous behavior.

4.4 Related Architectures

This architecture was developed primarily after experience with subsumption architecture, informed by research into spreading activation nets, neural networks, and the ethological concepts fixed action patterns and reaction chains. We were also inspired by [McGonigle, 1991] and [Chapman, 1989], who each pointed out the utility of control state over purely reactive control.

Two other recent architectures are similar to Edmund. Nilsson's TR [Nilsson, 1994] has units very like competences for structuring reactive behavior. However, Nilsson's approach is much closer to the planning community; his actions and predicates are atomic and logic based. Correia [Correia and Steiger-Garção, 1995] has a hierarchical architecture containing both reactive and sequential elements, also inspired by fixed action patterns. However, his competence-like units are not organized into a further hierarchy, but rather negotiate for behavioral precedence. We expect our approach to be easier to scale and easier to develop under than either of these systems. However, the fact both of these approaches have achieved interesting behavior on real robots is encouraging.

5 Experiments and Sample Scripts

5.1 Blocks World

Our first prototype was written in Common Lisp Object System on a Macintosh IIx computer. This included both the controller and the simulation. The simulation and primitives were based on Whitehead's PhD thesis [Whitehead, 1992] due to our interests in both reinforcement learning and visual routines theory. We first implemented the task his system learned, picking a green block out of several stacks of blocks. We then implemented a task his thesis mentions early as his goal, placing a green block on a red block (see figure 1). Finally we implemented a simple version of the copy demo, where the robot has to build a tower like one it sees in the environment.

This nature of task was chosen to answer a challenge: Malcolm once stated [Malcolm, 1992] that behavior based robotics could never perform a task as complicated as assembly. Interesting results besides the proof of concept were:

- Placing a red block on a green block would require more attention state than Whitehead was allowing himself in a flat architecture like a reinforcement learning look-up table. By focusing our *process* attention using subgoals, we reduced the requirements of *perceptual* attention.
- The copy demo appears a more complicated task due to its generality, but in fact requires less code because the primary goal is external and does not need to be represented in the process.

5.2 Mobile Robot

Our current system (a second prototype) is written in Object Oriented Perl (5.002+) on a Nomad 2000 mobile robot running linux on an Intel 486 processor. Our initial goal was to show we hadn't thrown out the baby with the bathwater, that our architecture could still perform archetypical mobile robot tasks as well as traditional planning routines. The Nomad operates in a normal laboratory environment. It is cylindrical, approximately one meter high and nearly half a meter in diameter at the base. It has 16 infrared and sonar distance sensors, 2 rings of 10 each collision sensors, and approximate Cartesian odometric information generated from its motions.

```
;;;;;proto-nav – ask directions at decision points until recognize the way
;– the top drive competence
(life () (nil) DRIVE ((talk 90 30 1) (cog-route 80 5 6) (walk 50) (wait 30)))
;– the drive's behaviors
(wait () (snore sleep))
(talk () (talk-speak))
;;; the "goal" here is to get blocked... in other words, wander til you do.
(walk () ((has-direction t) (move-view 'blocked) lose-direction)
   ((walk 100) (bump 70) (pick-dir 60 10 5) (start 50 10 5) (continue 50)))
;– walk's behaviors
(start ((has-direction nil)) (ask-direction synch-directions mark))
(pick-dir ((has-direction nil) pick-dir habit-dir) (synch-directions mark))
(continue ((has-direction t)) (move narrow (move-view 'clear) correct-dir))
(bump ((bumped t)) (yelp back-off reg-bump lose-direction))
```

Figure 2: A sample script for the Nomad robot, route learning. Formats the same as figure 1

Our first task was simple obstacle avoidance, allowing the robot to move freely throughout its environment, including passing through narrow doorways. Even this task requires a level of adaptation beyond habituation: we need perceptual memory to sanitize the sonar readings. Our next task was route learning, in which the robot asks directions if in an unknown territory, but rapidly learns to complete a requested route (see figure 2). We are currently working on consolidating such route memory into a representation with which the robot can perform path planning such as in [Mataric, 1990]; this should be done by IJCAI's April deadline.

6 Conclusion

The extent to which a robot's behavior is determined by programming rather than adapted by planning or learning, and the way in which these two interact, is analogous to the nature/nurture debate in the behavioral sciences. Of course, the determined/adapted quandary is one of design rather than analysis. This paper has presented a system for a very Lorenzian version of robot programming, where behavior is seen as carefully designed, and learning and planning occur only in well-defined, tightly constrained ways. Nevertheless, the behavior is not fragile with respect to the environment or precision of the agent, rather it has all the robustness of a behavior based system. We have presented an analysis of the constraints to date on behavior based programming and designed an architecture to overcome them. Our goal in the coming year is to demonstrate this system as a platform for scaling useful robot behavior in a simple, intelligent way.

References

[Agre and Chapman, 1988] Philip E. Agre and David Chapman. What are plans for? AI memo 1050, MIT, Cambridge, MA, September 1988.

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

[Blumberg, 1994] Bruce Blumberg. Action-selection in hamsterdam: Lessons from ethology. In Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1994. MIT Press.

[Brooks, 1991a] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[Brooks, 1991b] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the 1991 International Joint Conference on Artificial Intelligence*, pages 569–595, August 1991.

[Bryson, 1995] Joanna Bryson. The reactive accompanist: Adaptation and behavior decomposition in a music system. In Luc Steels, editor, *The Biology and Technology of Intelligent Autonomous Agents.* Springer-Verlag, 1995.

[Chapman, 1989] David Chapman. Penguins can make cake. *AI Magazine*, 10(4):51–60, 1989.

[Connell, 1990] Jonathan H. Connell. *Minimalist Mobile Robotics: A Colony-style Architecture for a Mobile Robot.* Academic Press, Cambridge, MA, 1990. also MIT TR-1151.

[Correia and Steiger-Garção, 1995] 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, Germany, 1995. Springer-Verlag.

[Firby, 1996] James Firby. Modularity issues in reactive planning. In *Proceedings of the Third International Conference on AI Planning Systems*, pages 78–85, 1996.

[Horswill, 1993] Ian D. Horswill. *Specialization of Perceptual Processes.* PhD thesis, MIT, Department of EECS, Cambridge, MA, May 1993.

[Maes, 1990] P. Maes. *Designing Autonomous Agents : Theory and Practice from Biology to Engineering and back.* MIT Press, Cambridge, MA, 1990.

[Malcolm *et al.*, 1989] C. Malcolm, T. Smithers, and J. Hallam. An emerging paradigm in robot architecture. In *Proceedings of IAS*, volume 2, Amsterdam, Netherlands, 1989.

[Malcolm, 1992] Chris Malcolm. Personal communication, November 1992.

[Mataric, 1990] Maja J. Mataric. A distributed model for mobile robot environment-learning and navigation. AI memo 1228, MIT, Cambridge, MA, May 1990.

[McGonigle, 1991] Brendan McGonigle. Incrementing intelligent systems by design. In Jean-Arcady Meyer and Stuart Wilson, editors, *From Animals to Animats*, pages 478–485, Cambridge, MA, 1991. MIT Press.

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

[Pebody, 1995] Miles Pebody. Learning and adaptivity: Enhancing reactive behaviour architectures in real-world interaction systems. In F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, *Advances in Artificial Life (Third European Conference on Artificial Life)*, pages 679–690, Berlin, Germany, 1995. Springer-Verlag.

[Steels, 1994] Luc Steels. Building agents with autonomous behavior systems. In Luc Steels and Rodney Brooks, editors, *The 'artificial life' route to 'artificial intelligence'. Building situated embodied agents.* Lawrence Erlbaum Associates, New Haven, 1994.

[Whitehead, 1992] Steven D. Whitehead. Reinforcement learning for the adaptive control of perception and action. Technical Report 406, University of Rochester Computer Science, Rochester, NY, Feb 1992.