Specialized Learning and the Design of Intelligent Agents

Joanna Bryson

September 5, 1997

1 Introduction

In some ways agents appear significantly more intelligent now than they did a few decades ago. We have learned to create rapid, robust behaviors by replacing extensive planning and modeling with carefully engineered behaviors and continuous sensing of the environment. On the other hand, in some ways our new agents look stupider than the robots of the 70's. We no longer have demonstrations involving assembly and manipulation as well as map building, as Shakey and FREDDY did. Essentially, we have robots with a very limited set of skills that are capable identifying when these skills should be applied. They cannot generate their own goals or learn new tasks; in short, they cannot perform complex behavior.

The issue of scaling the complexity of reactive robotics is not so much one of having reached a theoretical ceiling beyond which planning and symbolic thought are necessary. Rather the current problems have to do with technique and methodology. Developing intelligent agents requires not only modularized behavior, but modularized learning. Successful learning requires heavy bias; for complete, autonomous agents this bias must be individually specialized to many different adaptive tasks. This involves identifying the proper roles of state and learning and reducing them individually, though only as far as is reasonable.

The primary contribution of this thesis is an architecture for supporting the development of intelligent agents with learning in multiple representations and memory localized to behaviors. We provide a control system that

allows for a more intuitive programming of these behaviors into high level tasks and overall robot personae. We also provide a methodology for determining appropriate decompositions of behaviors and state.

1.1 The Theory

In struggling to minimize state, the architects of current reactive systems have wound up complicating both learning and control unnecessarily. Successfully exploiting the reactive intelligence strategy requires dividing the issues of *when* a behavior is expressed from *how* it is expressed. How a behavior is expressed is dependent on perception, not simply sensing. Perception is constructed from memory and experience as much environment. If we follow the behavior-based, society of mind paradigm for programming intelligence, then this implies memory itself and consequently learning should be modular and specialized. By "modular" I do not mean to imply "independent", merely localized by behavior and representation. This is consistent with animal intelligence, where learning has been shown to be highly specialized and constrained.

Following the lessons of biology and ethology, we have established three different sources of intelligent behavior:

- **Provided Learning**: In animals, learning that is performed at the species level. In robots, learning that is performed by the developer.
- **Required Learning**: Learning that is performed reliably by the individual agent. Provided any reasonable environment and opportunity for experience, the agent will necessarily acquire this sort of information.
- **Open Learning**: Intuitively, reasoned learning. Learning that comes about by insight rather than by reliable, predictable means.

In general, the agent should have the most restricted form of learning possible in order to ensure correctness, although notice the tradeoff is lack of adaptability. Required Learning is necessary for anything that is agent or experience dependent, essentially it is calibration. Open Learning is necessary for anything that will not necessarily provide sufficient consistent experience to be learned reliably.

Learning and knowledge are expressed in state. State involved in intelligence I divide into the following four categories:

- **Control State**: State in the form of the currently active behavior or behaviors, and the paths that activation can follow. This is used to record the immediate past in terms of action selection and, in combination with sensor information, determines the agents next act.
- **Deictic Variables**: Also thought of as attention. Small numbers of fixed variables which reference some particular object of attention thus generalizing the cases over which particular plans embodied in control state may operate.
- **Perceptual Memory**: Perception requires memory, because very little information is available in an instant of sensory information. These are specialized systems and representations where information can accumulate for anywhere from a few cycles to the lifetime of the agent.
- **Meta-State**: state about state. Room for monitoring or manipulating control patterns, activation levels, or perception.

This is ordered in descending reactivity, again a more reactive solution should be favored over a less except when necessary. Deictic variables should be used when control structures would otherwise be replicated, since replication is unreliable. Perception also replaces excessive control state, this time in terms of steps needed to detect the appropriate context for selecting a control state. Meta-state is necessary for an agent to learn from (or redescribe) its own experience or alter its own processes.

1.2 The Methodology

This dissertation presents a methodology based on the principles above, an architecture that divides control state from information flow, and the principle of rapid prototyping. Control determines *when* a behavior is expressed, information or perception determines *how.* The rapid-prototyping adapts and augments principles from object oriented design, particularly the rules for determining object hierarchies.

The control architecture allows for more intuitive programming than standard parallel behavior-based systems. The structure is roughly hierarchical,

although it limits stack depth to ensure reactivity while simultaneously supporting arbitrarily long chaining or even looping structures. The two main component parts are *competences*, which are like standard reactive plans or STRIPS triangle tables, and *action patterns*, which are sequential. Special competences, called *drives*, support coarse-grained parallelism, and a system of prioritization and habituation allows for allocation of attention resources via best-effort scheduling. Different controls for different tasks or behavioral personae are programmed in a scripting language.

Behaviors and control scripts are initially co-evolved by the developer, though as the behavior library matures, additional scripts may often be added with few or no modifications to the library. Behaviors are based on objects constructed around the minimal state needs determined by the perceptual requirements to disambiguate control situations. A behavior may be thought of as a chain of events that results from the activation of a particular primitive. An apparently continuous behavior, such as moving across a floor, is in fact a continuous sequence of activations, the lack of which would result in the eventual termination of the apparent behavior. Behaviors may read and/or affect multiple memory objects.

1.3 Structure of the Thesis Proposal

This proposal presents fairly mature work, although more remains to be done in terms of demonstrating the technique. The next section describes some of the high level details of the thesis. The following section presents the context for this research. There follows a fairly complete description of the architecture and methodology, and finally a section relating the theoretical discussion above to the methodology, including illustrations from applications.

2 Details of the Thesis

2.1 Evaluation

Evaluation of my thesis claims will be primarily by comparison of the complexity, robustness and diversity of developed applications with those of other systems. Such comparisons are notoriously difficult, however the issue of standards and scientific methodology has received a great deal of attention

recently — there have been three special workshops on the issue in the last nine months in the UK alone. The remainder of this section addresses such concerns. It is expected that some evaluation will also be done by reason, appealing to programmers' knowledge and intuitions as to the pragmaticism of my methodology. This approach will also be supported by literature review.

There are three methods of comparison between architectures I hope to explore. First, qualitative and quantitative differences based on descriptions in the literature. Examples of these for the issue of complexity are: the number of goals simultaneously pursued, whether high level goals can be generated or reordered by the agent, whether learning exists, whether meta-learning exists (learning of behavior, possibly including learning behavior.) Robustness criteria include length of a single run, transportability of behavior between environments, percentage successful completions, and variety of appropriate behaviors expressed successfully. Diversity criteria include the number of different platforms and tasks the architecture is evaluated on, and the qualitative difference between these platforms and the requisite behaviors of the tasks.

The next method for comparison is direct competition. Until recently autonomous agents lacked the standard testbeds that problems such as neural networks / machine learning, language acquisition and even assembly robotics have. Now however there are a few competitions available. The most interesting is probably Robo-cup, a competition based on the game of soccer. Of course such testbeds often lead to specialized solutions or are shown, like simulations, to neglect important problems. Also, for special competitions there are issues as to the number of programmers and other resources devoted by different teams. However, competitive strategies give some standard for comparison and have been used to advance the state of the art in other areas of AI.

Another form of competition is on the development level — if a developer is given a choice of platforms, which do they choose? If the same developer attempts multiple platforms, which do they find most useful and most expedient? Unfortunately, both forms of comparison are expensive in terms of time, so this proposal does not promise the inclusion of either, but we may seek them opportunistically.

The final method of evaluation which I am interested in is whether the system is capable of creating useful models of natural intelligence. So far, despite the efforts of the autonomous agent community, reactive, behavior-

based robotics has had very limited success in attracting the attention of the ethologists and experimentalists who's theories it attempts to model. Part of the reason for this has been the inability to encode the complex behaviors these scientists document, and this is at least partly due to the limits of learning methods. The criteria of success in this domain would be peer review conducted in a conference or journal of a biological intelligence community.

2.2 Related Work

Below is a list of related work, esp. that is not mentioned elsewhere in this thesis proposal (so see also the bibliography.) It is clustered by area.

2.3 Mainstream Agent Architectures

These are widely used architectures:

Rosenshein and Kaelbling ... 3T / RAPS (Firby) ... SOAR ... PRS / BDI ... RS ... TR / TRAIL – Nilsson, Benson...

These are well-known or very recent architectures, though mostly used by their creators:

DAMN – Roitblat ... CIRCA – Atkins ... GLAIR – Hexmoor ... ATLANTIS – Gat ... Aldus, Arkin, everyone in the current JETAI issue ...

2.4 Vaguely Biological Architectures

Horswill ... Ballard, Chapman, deictic stuff ... Brooks subsumption ... Steels, Pfieffer, McGonigle task grammars, McFarland ... Maes / Rhodes ... Minsky? Push? ... (From Bruce) Tyrrell, Payton ...

2.5 Specifically Modeling Animals

Blumberg (dogs)... Terzopolous (fish, dolphins, mermen)... Webb (crickets)... Cog (babies)... Ballard vision work (people)... General alife fox and hounds type stuff?...

2.6 Neural Modeling of Rats and Navigation

There is loads and loads of this stuff, but its not specifically what I'm targeting. Will especially mention: O'Keefe/Donnett, Sharkey's lab / Prescott, Meyer's review paper. Desert Ant in Zurich, wasp stuff.

2.7 Thesis Outline

These items don't necessarily represent chapters – there could one-to-one, one-to-several or several-to-one mappings.

- Introduction Section : similar to the thesis proposal so far (excluding this and the literature review list) and also the next section on thesis context.
- Description of my architecture along with methodology for analyzing tasks. This will be done by construction with respect to types of state usage. This will be

  1. purely control state,
  2. deictic representation (still done in control state),
  3. perception and learning (the library state), and
  4. meta state — being able to perceive and modify internal state and therefor behavior.

  This analysis should be more general / useful than simply for my specific architecture, as explained later in the agent architecture section.

  A first cut of this is also present in the proposal

- Example Section, probably broken into:

  - breadth — blocks world, learning to grasp (with Robert Ringrose if we get this working soon), Nomad robot work;
  - comparison — soccer; and
  - memory modeling — rat hippocampal learning and exploration behavior;

Each of these examples also have to have a small lit. review section included, but that certainly won't be an important part of this thesis; there are plenty of other review articles on these tasks.

- Review of Architectures — this will be a somewhat detailed evaluation of to what extent you could do the above things in the dominant architectures of the moment. Most thesis I read have brief mentions of many architectures, I'd like to do at least a page or two on the ones that are most developed instead. I'd also like to reduce my system to one of the architectures that's been formally described, probably RS or PRS.
- Conclusions and further work. Further work will probably include some speculations about meta-learning and cognition.

3 Time line

Work on the control architecture and methodology has going on for several years now. Also, the behavior library for the rat simulation is already being prototyped. Because Prof. Stein is going on sabbatical in the first term of 1998, I propose to return in either July or September of 1998 with the rat library and hopefully the soccer library (depending on the nature of my employment in early 1998) in advanced stages. Writing of the thesis would commence immediately, with time expected to be dedicated to further experiments and paper writing, as well as meeting remaining MIT requirements (I need a minor and an area exam. I met the core requirements in place in 1995 before I left, and I've also passed my OQE.) The review of current architectures (and the formal description of my own) in particular will take some time. I expect this process to be completed 9 to 12 months from my return, thus finishing by summer term of 1999 at the latest.

4 Context, Definitions and Approach

4.1 Intelligent Agents

This thesis work is about developing cognitive architectures for autonomous agents. The field of autonomous agents is active for two sets of reasons, one

theoretical and the other practical. The theoretical issues are driven by epistemological approaches to addressing AI. The suggestion is that specializing on particular problems has tended to result in unique solutions not comparable to natural intelligence, and not integrable with other elements of AI. Natural intelligence is based on complete, interacting systems which leverage solutions developed for one set of problems and apply them to others. Practical issues of autonomous agent research are driven by a need for encapsulation to reduce the complexity of control. A system that can solve fundamental problems such as navigation for itself is useful whether for exploring Mars or the Internet.

Notice for either set of motivations, full autonomy is not necessarily an aim. Animals are dependent on their environment, and cognitively advanced ones dependent on learning from conspecifics. People using agents as tools usually want to be able to direct and change priorities for their agents. Users may appreciate even minimal extra intelligence that smoothes and simplifies their task, even if full-time supervision of the agent is still necessary. Thus when I speak of autonomous agents, I imply only that the agent is situated to a particular environment, and can perform some non-trivial level of independent behavior within that environment.

4.2 Robotics and Simulation

Autonomous agent research focuses particularly on the problems of robotic agents. The first attraction of the robotic platform is that it to some extent replicates the environment and therefor the problems of animal intelligence. This hopefully biases solutions to also being in common with animals. Animal intelligence is of interest not only intrinsically, but as the underpinnings of human intelligence. Also, it is well studied — some consider in particular neuroscience to be the fastest advancing area of intelligence research.

The second attraction of robotics is that the platform provides some relatively clear indications of success and failure. Although the field has been plagued with disagreement about metrics of relative validity for different strategies, on a gross level there can be no doubt as to whether a robot succeeds or fails at a particular task, so long as that task is well defined. A robot designed to provide tours that becomes lost, collides with obstacles, or fails to navigate its entire route clearly fails.

One other claimed advantage of robotics is that the alternative, a simula-

tion, is also an artifact, and therefor represents knowledge and assumptions being made by the researcher. Historically, AI has often guessed incorrectly what will be the hard problems to solve, and of course it is the assumed problems that are worked into the simulation. As the adagé says, once you understand the problem you have the better part of the solution.

On the other side of the argument, many of these complaints are addressed by conducting research in simulations provided by third parties, or by working with agents on problems for which "cyberspace" is the intended final domain. Many researchers find some experience with real robotics gives them sufficient understanding of its problems that they are able to develop code in simulation that does transfer fairly directly into robotics. Also, robotics itself has proved not immune to defining a problem which may not be complete or relevant. Research robots can well be considered a poor simulation of actual industrial or even household pet requirements.

4.3 Agent and Cognitive Architectures

The accepted definition of an agent architecture was phrased by Pattie Maes as "[a] particular methodology for building [agents]" (Wooldridge & Jennings 1995). This has come to mean a description of the functional elements of a generic agent's software system, including specification of the composition and interaction of these elements. The first well-known agent architecture was Rod Brooks' subsumption architecture (Brooks 1986). It is composed of modules called *behaviors* which handle direct interactions between sensors and actuators for a robot. The architecture specifies the nature of these modules (finite state machines augmented with timers) how they are interconnected (by *wires* which can inhibit or suppress each others' input or output) and how they are organized (into *layers*, each accomplishing a particular goal and subsuming the operation of the previous goal.)

The current archetypical robot architecture is a three layered approach, such as GLAIR (Hexmoor 1995) with a bottom *sensori-actuator* layer which is essentially primitive directions and inferences from the underlying hardware, a *perecptuo-motor* middle layer which specifies with higher level representation and control for reflexive behavior, and a *knowledge* level, which operates with traditional AI knowledge representation and planning. In these systems, each level has both a theoretical purpose and an actual implementation. Research topics include what elements should be in which layers,

learning within each layer, learning between layers (eg compiling knowledge into reflexes, or rediscribing reflex into knowledge), the correct interaction of the layers, and the correct interaction of elements within the layers.

A cognitive architecture implies a psychological perspective. Psychologists have been attempting to describe the processes controlling (largely human) thought and behavior for the better part of this century, their high-level models are often called "cognitive architectures." In an agent architecture, the term tends to indicate the apparent ability to deliberate, such as testing alternative solutions to a problem.

Psychologists proposing cognitive architectures must answer the question "how do they develop?" To what extent is the architecture innate in a child, to what extent is it learned? Is it innate as in genetically programmed, but developed in phases rather than present at birth? To what extent is it present in animals? An agent architecture must also answer this question: how does an agent come to be functional? Development for an artifact is normally performed by its designer, but even so, the process is not simple.

4.4 Reactive and Behavior-based AI

The architecture in this thesis is built on the reactive, behavior-based approach to artificial intelligence. The reactive approach specifies that an agent responds more or less directly to a situation, without serious deliberation or planning occuring between sensing and action. Although counter to many intuitive notions of rationality and intelligence, this approach has been demonstrated effective for problems from navigation-based robot tasks (Connell 1990) to playing video games (Agre & Chapman 1987) and modeling human perception and problem solving (Ballard, Hayhoe, Pook & Rao forthcoming).

Behavior-based AI refers to an approach inspired by Minsky (1986), where numerous small, relatively simple elements of intelligence act in parallel, each handling their own area of expertise (Brooks 1991*b*, Matarić 1997). In theory, these simpler elements are both easier to design and more likely to have evolved. The apparent complexity of intelligence arises from two sources: the interaction between multiple units running in parallel, and the complexity of the environment the units are reacting to. Behavior-based systems are not necessarily reactive; certainly Minsky (1986) intended planning and complex representation as parts of the units. Reactive systems are likely to

be behavior-based, because clustering intelligence into such units reduces the number of potential actions that may need to be selected.

The central design problem for behavior-based systems is behavior arbitration — determining which parallel module controls actual physical behavior at any one time. This is a problem not only for design but also for development: it can be very difficult to determine if faulty behavior is the result of one or more behaviors operating simultaneously. The central problem of reactive planning is the external Markov assumption — the notion that the next action can be entirely determined by external state. In fact, most intelligent agents will find themselves by turns passing through identical environments pursuing differing goals, and should thus select different behaviors.

4.5 Programming and Situated Planning

Current research in agent architectures is generally said to involve hybrid systems, that is, systems with traditional planners operating on reactive "primitives" which provide a simpler, more idealized world in which planners can operate. In fact, the problem of classical planning scales exponentially; any amount of simplification still raises the ceiling of behavioral complexity only slightly (Horswill 1997). Many researchers, even those who intend to use planning eventually, in fact are creating systems that integrate hand-coded planning (a hallmark of the reactive approach) with their behavior-based primitive substrate. (Rosenschein & Kaelbling 1995, *Special Issue: Software Architectures for Hardware Agents* 1997, Singh in press).

Hand-coded plans avoid the problems and combinatorics of general purpose planning, while still providing a mechanism for solving the external Markov conundrum. Which plan or goal is active provides internal state to disambiguate behavior, even if the plan isn't particularly proscriptive. (Agre & Chapman (1988) argues that plans serve better as internal communication than as detailed recipes.) Of course, plans themselves generally reduce ambiguity to one or a few behaviors to be selected next. These may at least provide bias for behavior selection, even if they are not deterministically in control in many reactive system.

Levison (1996) refers to the current paradigm of agent programming as *situated planning.* On-line planning may still exist, but be divided among multiple planners (Levison 1996). Planners may also be subservient to more

reactive layers that call them only when encountering specialized situations (Gat 1991). The important lesson is that as early as 1993, most of the planning and mainstream robotic and software agent communities had accepted that reactive elements are necessary for intelligent behavior, but further structure needs to be provided to support complex behavior. The current research issue is how these elements are to be developed and combined.

4.6 Ethology and Learning

Autonomous agent research more than most branches of artificial intelligence has always acknowledged the extent to which it leverages the solutions of natural intelligence, particularly Ethology, the study of behavior as an evolutionary derivative. One of the main conferences of the field, the Simulation of Adaptive Behavior, is explicitly about replicating animal learning. Animals are our primary working example of what we consider intelligence (Brooks 1991*a*, McGonigle 1991).

At the turn of the century, behaviorists (the predecessors of ethologists who concentrated on laboratory experiments), proposed that animals learn only through a general process of being able to create associations.

> The [general process assumption] position is that all learning is based on the capacity to form associations; there are general laws of learning that apply equally to all domains of stimuli, responses, and reinforcers; the more frequent the pairings between the elements to be associated, the stronger the associative strength; the more proximate the members of an association pair, the more likely the learning(Gallistel, Brown, Carey, Gelman & Keil 1991).

This learning by association is also called "conditioning", and it does appear to be a general learning mechanism that follows various parameter-based rules across species, presumably indicating common underlying mechanism. However, the behaviorist research eventually demonstrated animals *cannot* learn to associate any stimulus with any response. Pigeons can learn to peck for food, but cannot learn to peck to avoid a shock. They can, however, learn to flap their wings to avoid a shock, but not for food (Hineline & Rachlin 1969). Originally interpreted as constraints placed on open learning that might be dangerous, research has since indicated the opposite. Special systems exist to form important associations.

As an example, the amygdala (a brain organ in the limbic system) has been shown to explain a similar double dissociation to the above in rats. Rats presented with "bad" water learn different cues for its badness depending on the consequence of drinking it – if drinking it leads to shocks, they learn visual or auditory cues, if drinking it leads to poisoning they learn taste or smell cues (Garcia & Koelling 1966). The rapid learning to associate poison (and potability) with a smell has since been credited to learning in a an organ in the brain's limbic system specialized to emotional reactions, the amygdala (Carlson 1994). Both the rat and the pigeon example demonstrate useful adaptivity in nature. Poison is often indicated by smell or taste, while acute pain is often the consequence of something that can be seen or heard.

Gallistel et al. (1991) hypothesize that learning by an individual organism serves as a last resort for evolution. It is only relied on when behavior cannot be fully predetermined, because the competence involved requires flexibility on a less than evolutionary time scale [1]. Such learning is also not associative. Examples include the precise acoustic location of prey by barn owls (necessary because it is dependent on the shape of the individual bird's head) and bees learning the ephemeris of the sun for navigation (dependent on season and latitude). These are not cases of classical conditioning. The animals seem to be born with a limited number of variables which are instantiated at a certain stage of development by simple observation of the world.

Such variable filling can also take the form of perceptual learning and or categorization. Vervet monkeys have three distinctive warning cries for predators which require different defensive action. These cries are dedicated to pythons, martial eagles, and leopards. Baby vervets make cries from a very early age, but across overly general objects. For example, they may give the "eagle" cry for anything in the sky, the "leopard" cry for any animal, the "python" cry for a stick on the ground. Thus they are born attending to the sorts of stimuli they need to be aware of, but they learn fine discrimination experientially (Seyfarth, Cheney & Marler 1980).

Gallistel et al. (1991) also support the notion that although learning is specialized, it may be leveraged by multiple processes once established. For example, foraging animals allocate foraging time between sites proportional

[1] Though notice researchers into the Baldwin Effect have demonstrated there is little evolutionary pressure to genetically hardcode rules that are consistently learned (Hinton & Nowlan 1987).

to the relative amount of fodder available at those sites. Gallistel et al. (1991) suggest that the same mechanism an animal uses to localizing itself within its environment may be used for effectively modeling the forage availability.

5 Details of Architecture

5.1 Approach

> Because, in extreme cases, agent synthesis can be intractable, it is not a reasonable objective to try to develop universal solution methods. Rather, it is preferable to develop a methodology for specifying particular action strategies, and to develop an inventory of solved special cases that can expedite agent construction in practical situations. (Rosenschein & Kaelbling 1995)

One of the goals of this thesis is to make it easier to express knowledge or intuitions about intelligent behavior into control structures for real agents. In other words, we want to help people program robots directly. This research took as a starting point the reactive, behavior-based approach which had already been developed to this purpose and was being widely researched. The next step was to identify and address problems that were inhibiting scaling of development under this approach.

The first problem, as emphasized in the introduction, is simply programmability. Methodology is crucial. Behavior-based architectures are essentially hand coded, so their complexity is highly dependent on the methodological support for the software designers. Developing a methodology for programming agents under an architecture is consequently as important as developing the architecture itself. The section on methodology below is adapted from a paper emphasizing the leverage of object-oriented design into designing agent architectures. In the thesis, I intend to have more formal and explicit descriptions of how to decompose an intuitive plan into an Edmund script, including which elements need to be a part of control, and which a part of the behavior library.

The next area of concern is control structure. Strict parallelism provides insufficient bias for action selection and goal persistence. Animal intelligence appears to provide sequencing primitives for both representation and action

(McGonigle & Chalmers 1996, Tanji 1996). At the same time, rigidly connected structures as in Brooks' subsumption architecture (Brooks 1986) or even as constructed by Firby's RAPs (Firby 1987) overly constrain behavioral flexibility. Subsumption fixes goal prioritization in a linear order, while RAPs constructs tree structures not suited the continuous, non-terminating processes of life-like agents, such as language (Firby 1995). The control section below describes the current version of my architecture, Edmund. Edmund addresses the selection bias problem by providing structures for both parallel and sequential control components. It addresses the flexibility issue by providing a parallel reactive drive structure, each element of which allows for chaining subhierarchies.

The final and main problem area addressed in this thesis is the need for specialized, modularized learning. In behavior-based AI, learning tends to be either nonexistent, as in strictly reactive systems (Brooks 1986), or fairly homogeneous, with each behavior having a generic learning capability (Pebody 1995, Albus 1997). Often it is monolithic, as in systems that employ reinforcement learning, genetic algorithms, or neural networks to supervise control (Kaelbling 1993, Mondada & Floreano 1995, Whitehead 1992, Koza 1991, Sims 1994, Harvey 1995) Vertebrate brains have specialization both by organs and regions within organs for storing relevant state, whether that state is fairly transient computations of perceptual or motor information or longer term skills and information (Carlson 1994). BBAI systems with specialized learning or state elements (such as (Matarić 1992, Horswill 1997) have been successful, but no principled system for supporting multiple state elements with varying representations has been developed. As explained in the introduction, our approach is to separate control from dataflow, and create an object-oriented library as the foundation for the behaviors. Objects consist of state associated with methods, which helps encapsulate different representations. Behaviors are built of these methods.

It might be argued that a fourth, and fundamental, problem with the reactive, behavior-based approach is the lack of high-level planning, or of language. I am not proposing to research the inclusion of planning for several reasons. Planning in the classical sense has been shown intractable on resource bounded systems (Chapman 1987) and consequently seems inappropriate as a top or center piece of an architecture designed for a situated system (Horswill 1997). Also, plan-based systems are likely biological, especially for non-linguistic animals. Though there is no question that language

Sense	A perceptual primitive returning a value.
Act	A primitive affecting the external world or internal state, including perception.
Trigger	A sequence of primitives executed atomically, for selecting actions. Should have no impact external to agent.
Lap	An action pattern, a sequence of primitives which may have external impact. The L is for "learnable", not yet implemented.
Competence	A prioritized set of Laps or other Competences and their triggers. Includes a goal lap that, if successfully triggered, indicates completion.
Drive	A type of competence (may also contain Drives) that runs its components in parallel. Generally has no goal.

Figure 1: The classes composing the Edmund architecture.

itself exists and affects individual human behavior, my research interests are more focussed on the systems that must support language than language itself.

The following subsections describe the control architecture, the concept of a behavior library, and the methodology for co-developing these units. They focus on a particular example, developing simple navigation for an off-the-shelf laboratory mobile robot.

5.2 Control

5.2.1 Elements

The Edmund architecture classes are as specified in figure 1. The classes may be divided into two sets, primitives and collections. Primitive actions and senses are the interface to the library code, which will be discussed in the next section. The ordering of primitives into control structures via collective classes is specified in script files. Libraries are specific to a platform, while scripts are specific to task or persona running on that platform.

Edmund was written under object-oriented design primarily to exploit inheritance and polymorphism: a collective class does not need to know the type of an element it contains, merely when to execute it. Edmund also extends the class-object idiom to the instances of its classes. There may, for example, be multiple occurances of a single primitive occuring in different

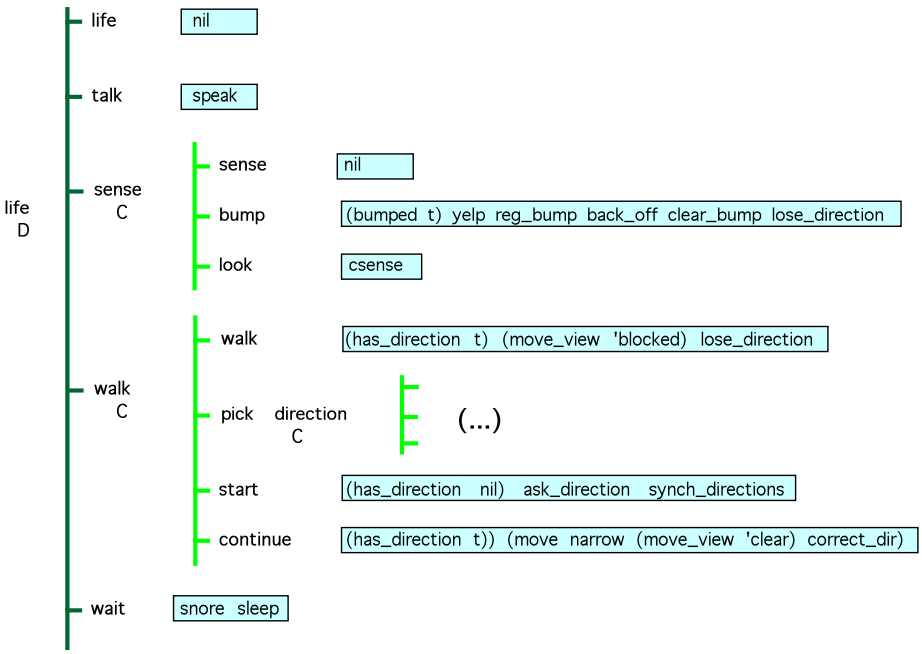

Figure 2: Competences and action patterns relying on ENRL primitives. This script enables the robot to do rudimentary map building while asking for and following instructions in unfamiliar areas.

control contexts, that is, in different LAPs. Within an action sequence, a sense primitive serves as a conditional to check whether the sequence should continue. Consequently, scripts always associate senses with tests, which are in turn stored in the sense instance when it is incorporated into one of the sequence-based classes.

Similarly, the collective classes have template information stored as class data and create instances when invoked. These instances point to the class data and have instance variables for maintaining run-time state. For example, an instance of an action sequence needs state for the index of the element currently executing, an instance of a particular competence needs state for the current priorities (with habituation) of its elements.

5.2.2 Reactivity

Edmund's reactiveness comes from two sources. First, a competence on execution checks the trigger of each element to find the highest priority element that can run. No assumptions are made about the outcome of an action sequence; the environment is always resampled for selecting the next element. Second, drives allow the execution of multiple competences or actions in parallel. Although the program only attends to a single drive component at a time, each component maintains state while the other components are being attended to. Consequently, their execution proceeds as if in parallel, via coarse-grained time slicing. In the library described below, primitive acts often give commands to parallel hardware systems and do not wait for a result, so true parallelism from the perspective of the robot is achieved.

The continuous sampling of the environment through triggers results in a form of active perception. Competences that are currently active and have priority will be the ones executing their components' triggers, thus extra time and attention is given to seeking information relevant in the current context.

Which drive component receives attention is determined by priority level. The priorities can be "habituated"; scripts can specify a penalty per execution and a recovery rate per second. This allows for allocation of attention resources via best-effort scheduling. Competence elements may also habituate and be scheduled. This allows for alternative solutions of similar priority to be tried, and for competences to guarantee failure after a certain number of attempts. Competences fail if none of their elements can be triggered; elements with 0 priority or less are not tested for triggering.

When the top drive or competence has determined where its attention lies, that element is executed. Since drives and competences can be nested, an architecture like this might become highly unreactive due to a long chain, or worse, a loop of components calling each other. Edmund solves this problem simply. Competences don't attend to other competences, but rather shift their own attention. Assume a drive $D = (C_a, C_b, L_c, \ldots)$ where competence $C_a = (L_1, L_2, C_3, \ldots)$. If C_a's arbitration selects C_3 as the highest priority triggerable element, then C_a flags itself for replacement. The next time D fires, it contains $(C_3, C_b, L_c, \ldots)$. Thus the the activation tree is never deeper than the deepest drive plus three (a competence, a lap, and a primitive). In practice, we have yet to nest drives, and doubt that they should need to be nested more than two layers.

This chaining of attention guarantees reactivity, but might fail to provide behavior persistence. A competence that is an element of another's list might reasonably be described as a subgoal. Once a subgoal has completed, shouldn't its parent continue? Edmund addresses this issue in the drives. A drive not only keeps track of the processes to which it is currently attending, but also has a default process for each element that starts if that element is triggered with no other active process. Thus if a competence completes, the drive returns attention to the root of the tree that brought it to that competence in the first place. If the context is the same, then attention will return to the parent competence. If the context has changed, some other behavior may now be more appropriate. A competence can terminate by failing to execute any of its elements or by succeeding to trigger its goal element.

5.2.3 Biological Plausibility

Edmund is designed to be a platform for running ethologically based models of intelligence. We have attempted to make it neurologically plausible. Behavior sequences are supported in the natural intelligence literature as described in the beginning of this section. Competences are not directly supported, but could exist as sets skills that are collectively stimulated to varying degrees in a particular context, similar to priming in spreading activation networks (Maes 1989, Neely 1991). Similar action selection mechanisms have been postulated by (Glasspool 1995).

Drives allow the simulation of parallel activity in non-cortical parts of the brain. For example areas of the forebrain such as the amygdala operate on separate channels of sensor input and can interrupt cortical activity by providing independent activation, while elements of the hindbrain maintain metabolic event scheduling independent of cortical sensing and activation.

The perpetuation of activation and its chaining through competences as described above is related to the problems of binding and attention, issues which are receiving a great deal of attention in neuroscience at the moment. See for example (von der Malsburg 1995, Phillips & Singer forthcoming).

5.3 Perceptual Learning

5.3.1 Structure

Direction	For moving and detecting motion-relevant information, knows current and preferred directions. Instances correspond to faces of the robot.
PMem	Perceptual Memory, a short list of recent combined sensor readings. The main source of sensing. No significant instances.
Bump	Provide information about local "invisible" obstacles. Instances represent collision points.
Wave	Detect people waving at sonar (a source of communication).
DpLand	Decision point landmarks, the basis for navigating (vs. wandering.) Instances represent choice points and the decisions made there.

Figure 3: The classes composing the Edmund's Nomad Robot Library (ENRL).

The functions and state on which the sensing and action primitives depend are located in platform-dependent behavior libraries. So far two libraries have been constructed, one for using visual routines theory in a blocks world simulating (as in (Whitehead 1992, Chapman 1989)), and the other for a Nomad 200 mobile robot. The examples that follow focus on Edmund's Nomad Robot Library (ENRL), which is the more recent work.

The control architecture's only interface to its libraries are function pointers that are passed to its primitives on loading. A library routine might consist entirely of calls to underlying hardware or software routines provided by the agent's platform. This was the case for the visual routines library, which was nearly entirely reactive (see Section 7.2.2.) More often, behaviors cluster around shared function and resource, particularly memory, or state. This makes object-oriented design a natural programming strategy for such libraries. State in these objects can be anything from very short term (< 1 second) perceptual memory, for things like "sanitizing" sonar readings, to long term memory, for tasks like map learning. Often different but related forms of state are stored as class and instance variables for a single class.

An example of the above technique is class Bump. When an obstacle is encountered via the robot's bumpers, an instance of Bump will record its approximate coordinates, while a Bump class variable refers to the list of all currently remembered Bumps. The primitives for detecting collisions (using bumper state information provided on the Nomad platform), storing instances (using similarly provided odometric information), recalling and

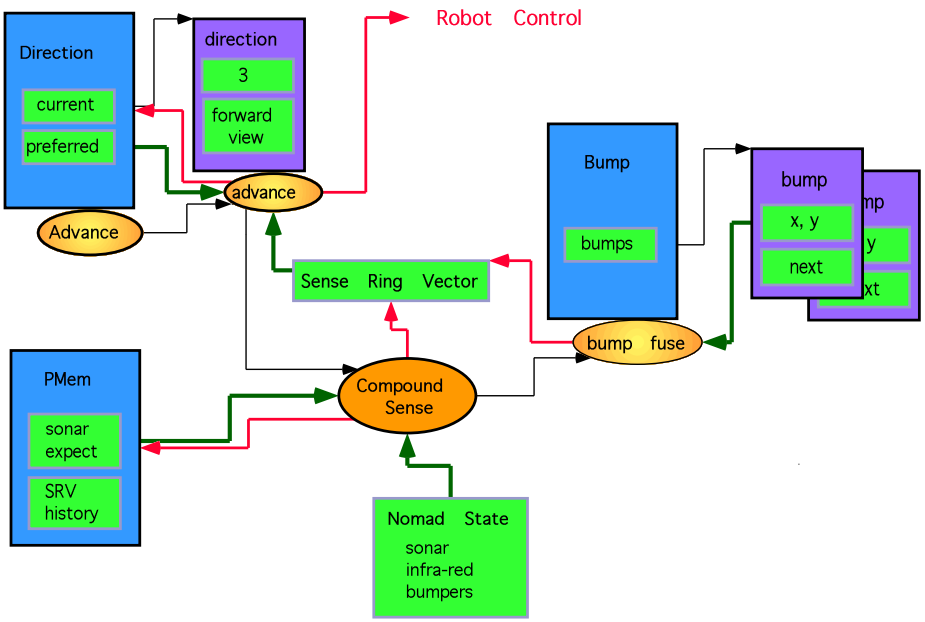

Figure 4: The objects in ENRL that are involved in the primitive behavior **move**.

pruning information from memory are all based on methods in the Bump class. The information (except for that provided directly by the Nomad) is not computed automatically, but only on demand when relevant primitives are called from the control structure.

5.3.2 A Sample Behavior

This section describes one of the elements of ENRL, Direction, as an example of the nature the library. This example will also be referred to in the following section on methodology. Direction is largely responsible for the familiar task of low-level navigation and obstacle avoidance. The behaviors necessary for motion are based on its methods, though these in turn refer to information and methods from the perceptual classes.

There are 16 instances of Direction correlating to the 16 faces of the Nomad, each with a sonar and an infra-red sensor. (see figure 5). An Act primitive, **move**, calls the Direction member function *advance* on whichever instance of Direction the robot is currently pursuing. There were initially

two subclasses of Direction to include an additional 16 directions on the axis between two faces, but in experiments these proved unnecessary for smooth trajectories and were removed to simplify the code.

Figure 5: The Nomad 200 robot on which ENRL has been developed. In this picture it is encountering a desk — the surface is too low for the sonar sensors to detect when proximate, and the leg too dark for the infra-red sensors, so the robot must learn about the desk with its bumpers.

Advance works as follows. The current Direction instance computes a new velocity for translation, based on the room it has to maneuver. It also computes a velocity for rotation, based on the difference between the robot's current steering angle θ_r, and the Direction's own orientation, θ_d. The formulae are based on power curves, making the robot's trajectories fluid and natural[2]. *Advance* directs the Nomad to move at these velocities, then finishes by choosing a new current direction if obstacle avoidance is called for.

[2]The steering rate is $k_2 e^{-k_1 \frac{1}{\sqrt{\theta_c}}}$ where the k_1 (26) adjusts the slope of the exponential, k_2 (450) adjusts the scale of the result, and θ_c is the difference between θ_r and θ_d, adjusted

Advance is an example of how an apparently continuous action can be sliced appropriately for an architecture based on discrete steps. Operating on the Nomad robot, ENRL results in smooth, continuous motion. In fact, high level drive attention is being switched between motion, sensing the external world, and checking and reporting battery level. Motion is also a competence, and routinely checks the triggers for starting and terminating motion before choosing the case of continuing motion, executing **move**. In Nomad's commercial simulation, the calls to *advance* do appear as discrete events, because the commands cannot be performed by separate hardware systems as they are on the real Nomad.

5.3.3 The Use of State

Advance requires several types of state to go with its sensing capabilities. These are reused in other primitive behaviors. The choice of speed and direction are determined by the perceived space surrounding the robot. This is represented by an array of 16 values representing the distance in inches from the robots faces to the nearest obstacle. This perceptual array is a construction regularly updated with information from sonar, ir and bump sensors. Information from the sonars is not taken directly, because sonars often give false readings. Sonar values are based on a perceptual memory of previous values and rates of change – sudden changes may not be believed for nearly half a second, then they just as suddenly replace the old value in the perceptual memory. This approach is consistent with human perception (Dennett & Kinsbourne 1992). Bumps require a quite different, more persistent form of perceptual memory since, in order to be useful, they need to influence behavior by being avoided for some time after impact. Consequently PMem sonar memory is stored in a fixed size array, while Bumps are stored in instances indexed off a class variable and trimmed as part of the perceptual process.

In addition to these perceptual classes, Direction itself stores state as class variables recording which direction is currently being pursued, and which direction was actually intended or preferred. Another Act primitive, **correct-dir**, tries to move the current direction closer to the preferred one. This is

for the nearest direction of rotation. The translation velocity is $k_2 e^{-k_1 \frac{1}{\sqrt{dist}}}$ where k_1 (5) is as above, k_2 (100) is the maximum velocity in tenths of an inch per second, and $dist$ is the nearest obstacle in the direction of motion also in tenths of an inch.

called in scripts where the robot is to display persistence in an intended direction as part of an action sequence with **move**. Other, permanent, state in Direction includes a vector in the instances that specifies a distorted window on the perceptual array mentioned above. The window indicates a scaled relevance of elements in the perceptual array to forward motion in that direction. The Nomad can pass through doorways only a few inches wider than itself, but will stop four inches in front of most obstacles.

5.4 Methodology

Much of the variety of approach in autonomous agent architectures reflects variety in the underlying theories of intelligence. Ideally, agents should provide us with platforms for testing and developing such theories. However, our only perspective for viewing and comparing the behavior of our agents superimposes our ability to program on top of our underlying theories. As a result, methodology is an inextricable part of intelligent agent research.

5.4.1 Edmund and Object-Oriented Design Methodology

A good architecture should encourage and facilitate a methodological approach. The previous section describes the implementation of Edmund, which exploits some aspects of object oriented design directly. We recommend using object-oriented design to organize state and behavior into logical units. We suggest maintaining a separate, well commented interface file between the architecture primitives and the library methods. This is to some extent necessitated by the architecture, since every primitive named in a script must be added to either the Sense or Act class with an associated pointer to a function. Keeping those functions simple and in a single file with canonical documentation can only be recommended.

One of the key problems of behavior-based design is decomposition. How does one determine what the elements of behavior are? Should they be encoded as competences, sequences, or primitives supported in the behavior library? We suggest the correct way to address this question is analogous to the solution to developing a large object hierarchy in advanced object-oriented design.

One of the main principles of OOD is rapid prototyping. Although analysis bootstraps the system, the principle is that specification is worked out

through simplified objects design. If the hierarchy appears to work, the objects are fleshed out with full code. At any time during development, a set of rules can determine that the hierarchy should be adjusted. Replication of code indicates the need for a common super class. Needing half the functionality of a class indicates a need for dividing a class into a parent and child. A parent with a single child is an opportunity to collapse structure unless instances of the parent actively use methods overwritten in the child. Too much multiple inheritance may indicate a bad basic hierarchy decomposition.

Similarly we support a system of rapid decomposition and continuous design, development and testing. The libraries and scripts should be co-evolved by the programmer into greater complexity. Libraries and the related primitives are organized along the lines of function and their required state[3]. There are rules similar to those for object hierarchy decomposition for control structure element selection. Sequences that require cycles are actually Competences. Competences that reliably fire only a few elements in order may be Laps. If sections of what was previously considered a single behavior primitive are required in another script, the behavior can be broken down with elements to run either in sequence in a Lap, or in parallel in a Competence.

An example of these techniques can be found in the development of the motion primitives **move** and **correct-dir** mentioned in section 5.3.2. **Move** initially behaved as it does now, but as part of developing a behavior for reliable legs for a simple map-learning routine was extended to correct the direction. This invalidated an old script "wander," which changed directions without concern and was useful for herding the robot into desired areas. The new **move** was therefor broken into elements which included a cleaner version of its old self, and **correct-dir**. These elements are now run as a Lap in the map-building scripts.

As suggested in Brooks' subsumption architecture (Brooks 1986) development is probably best done beginning with basic behaviors such as motion. But unlike Brooks, we do not expect layers will necessarily be fully or correctly implemented before moving to the next one. If as the project progresses problems are discovered in early levels, earlier scripts exercising only those levels can be used to test that redevelopment does not damage

[3](Bryson 1995) also suggests adaptivity requirements as a good indicator for behavior decomposition.

earlier behavior The scripting facility also allows for easily switching elements between competences and drives. Elements in drives can be scheduled the most reliably. However, an element which only is required in a particular context may waste cycles if it is scheduled as a drive. A Lap for handling bumps was initially added into the competence for motion, "walk", but this overlooked the fact some Laps in walk were long and the bumpers wouldn't be checked between their elements once they were triggered. Thus "bump" was moved to the top level drive. On the other hand, a perceptual routine for watching for landmarks was successfully moved into "walk" since it was only needed in the context of motion, and could rely on perceptual memory if it was called less frequently.

5.4.2 Other Concerns and Future Work

Scheduling in Edmund is inexact. Too many things may be scheduled per second with no direct indication they are failing to execute. This is to some extent an artifact of the reactive nature of the system — we expect some events and behaviors to arrest the attention of the agent. To some extent scheduling can be reasoned about with simple benchmarking, facilities for which are built into the control program. These help determine the constraints on the number of elements run per second, which allows estimates for rate at which varying numbers of drives can be executed.

On the Nomad, sensing is fairly expensive and limits the number of cycles for the architecture to about 340Hz. However, if the robot stops moving and reduces its sense sample rate, the cycle rate increases by an order of magnitude. This suggests that the robot should engage in phased periods of activity, for example switching between sense-intensive work like exploration and compute-intensive work like map learning. This strategy is found in most mammals (e.g. (Wilson & McNaughton 1994)), and is something the Edmund drive system should facilitate. We intend to model this in the near future.

In ENRL the object metaphor, particularly encapsulation, does break down in relation to sensing. This is not surprising given that external to the robot, information is flowing in streams. Senses filter and, in the case of digital systems, segment this information. Many areas of animal brains, both cortical and subcortical, fuse information from multiple senses. In fact, very few animal behaviors are independent of sensory context. Pitch detec-

tion, time sense, and intentional learning and recall all show strong context dependency (Carlson 1994). Encapsulation violations in the object library amount to methods or even functions calling for information from multiple classes. OOD methodologies address the problems of interdependencies by limiting and documenting the points of interface.

These OOD methodologies have been directly transferable to ENRL. It is possible, however, that the control structure itself may need to be fused with the perception and action units. We are currently exploring the use of episodic memory for tasks involving search of multiple contingencies, such as principled (directed) navigation. This could well involve treating learned patterns something like drives or competences, with associated patterns of habituation or inhibition.

It should also be noted that even with the help of Edmund, programming robots can be very slow work, particularly developing the fundamental motion and sensing routines. More work is needed in establishing the role of robotics versus simulation in research, and standards of comparison for programmer productivity within various systems. We intend to continue exploring the Edmund approach both by extending the complexity of the systems written in it and by applying it to more different platforms. We are also reviewing more established architectures with customized development tools. If we can model the essential elements of our approach in a system that provides better tools and wider user support, we feel we should exploit these programming benefits.

Finally, one challenge facing scaling behavior-based robotics is to move to multi-programmer projects. Even in single-programmer projects, ascribing performance and bugs to a particular behavior or interaction of behaviors is often difficult. We hope the encapsulation of both the libraries and the control structure will help facilitate development for multiple programmers as well as for individuals. We intend to work on extending our methodology to such a project within the next two years.

6 State and Learning

6.1 Definitions

State, memory, learning and representation are inextricably linked. *State* is essentially information. Within computer science it connotes the current value for a predetermined variable, but in fact everything that varies, along every axis of variation, can be considered state. From the agent's perspective, this includes external factors such as what objects are in its environment, what their color and orientation are, the nature of the current ambient light and temperature; and internal factors such as the angles of joints and motors, energy levels, the focus of visual apparatus, the focus of processing attention, previous experiences recorded in memory, and current expectations presumably also so recorded.

The most intuitive definition of *memory* is a persistent change in the agent's internal state, particularly processing state. This isn't complete. In understanding intelligent behavior, we need to take into account even tiny latencies in activation in systems from sensory to cortical. However, persistent change such as brain damage or the loss of a limb is not normally considered memory, even though it results in changes of behavior. On the other hand, many agents construct a short or long term context in their external environment to help them think (Agre 1988). In humans, this strategy becomes more and more important during the process of aging, so that many very elderly people if moved to a different apartment go from virtual autonomy to virtual helplessness. This implies that we want our definition to be something like "a persistent, *meaningful* change in the agent or its controlled environment." Unfortunately, a full discussion of 'meaningful' is beyond the scope (or even the area) of this thesis.

We will start with a simple, if not fully grounded, definition of *learning* as a change in memory (where memory is understood to be 'meaningful'). Similarly, *representation* is the mechanism by which that learning is stored and recalled. There will be considerably more discussion of these two terms below. This discussion begins with the fundamental though: state.

7 Designs of State

This section discusses four different types of state. They form a continuum from supporting what is generally not thought to be learning to what might be considered meta-learning.

7.1 Control State

One of the myths of artificial intelligence is that the early experimenters in reactive planning championed stateless systems. In fact, the fundamental units of subsumption architecture are (augmented) finite state machines. Thus each behavior at any moment records a particular state of execution, and this state helps determine what that behavior will do next.

The most reactive behaviors do respond directly and continuously to factors in the environment. For example, the first few Braitenburg Vehicles (Braitenberg 1984), where motor speed was directly correlated to light or heat intensity, or Polly's vision-based obstacle avoidance algorithm (Horswill 1993), where Polly chose its speed and steering angle (left, straight or right) based on the largest extent of ground coming up from the bottom of the camera frame.

However, even simple behaviors require some kind of stored internal state, and often sequential control. For example, one of the famous subsumption robots, Genghis (Angle 1989), was designed to look like an insect, and would back off and turn away from an obstacle detected by bumping its antennae. Notice that the environmental state to which the robot reacts is only detected while the antenna is bent, but the behavior determined by that state must be perpetuated across a period of backing up persisting after the antenna is disengaged, and this behavior must be followed by the turning behavior that avoids coming back to the same location.

In a conventional AI system, the initial collision would have resulted in an internal representation of the collision point as an assumed obstacle, and the registration of this knowledge would have resulted in a planner selecting a new behavior (such as exploring the obstacle) or a new trajectory. In Genghis, the collision is registered only temporarily in control state. One behavior is forced into a different state, which results in the different actions. Once normal behavior is resumed, the event will be forgotten.

This disposability of information may sound wasteful, but in practice

such information is difficult to record accurately, and is often transient. The superiority of the reactive approach to local navigation has been well proven empirically (Maes 1990, Bonasso, Firby, Gat, Kortenkamp, Miller & Slack 1997). A more significant criticism however, is that this approach exchanges complexity of representing the environment for complexity in designing the control.

7.1.1 Programming Control State

Programming under the original reactive architectures is notoriously difficult (Wooldridge & Jennings 1995) which has been part of the motivation for the two and three layer architectures (Bonasso et al. 1997) and situated planning approach in general. I believe there are two main reasons for this problem. v First, subsumption architecture and approaches similar have both sensor information flow and control structure represented in the same connections between behaviors. Ideally in subsumption architecture, each behavior should receive sensor information directly, so that each behavior is itself reactive. In fact, many of the behaviors take their inputs from reading the outputs of other behaviors, and similarly many express their own behavior by inhibiting or suppressing (overwriting) the inputs or outputs of other behaviors. (See (Brooks 1989) for example.)

Trying to effect control and data flow through the same mechanisms leads to complications. One of the principles of subsumption architecture is that once a layer of competence is developed, it should never be altered for higher layers. In Herbert (Connell 1990), this principle broke down when a behavior was added to reach and grasp a soda can. The robot's arm broke its line of sight to the can, so the lower behavior that moved unless it saw a soda can would start moving in mid reach. Brooks and Connell solved this problem by adding timers to the behaviors (thus the *augmented* finite state machine) to allow for their short-term global suppression if one behavior needed to "take over".

This example also illustrates the second reason why programming is difficult in such systems — many behaviors are best described as elements of a sequence. Many systems based on ethological motivation have systems of competing behaviors, where each behavior determines its own activation level (Blumberg 1996, Norman & Shallice 1986, Correia & Steiger-Garção 1995). However, such systems can be difficult to program and to scale, even

in situations where some explicit sequencing is set in terms of activation from precondition to dependency (Maes 1989).

7.1.2 The Current Approach

This thesis proposes that control and dataflow should in fact be separated. Control is in effect lifted from the typical behavior-based system, which still consists of modules connected through flows of information. However the timing for when a behavior is expressed is determined separately in a scripted set of drives, competences and action patterns (see Section **??**). Sequences and competences express full and partial orderings of behaviors respectively. They also guarantee that only one behavior within the set is expressed at a time. Drives execute their components in course-grained parallel, together with competences they ensure the system is still reactive.

To resolve Herbert's dilemma, the grasp of the can would have been either an action pattern or a competence[4] that was an element of the competence that searched for and retrieved cans. By making these all part of a single drive element we can ensure they would never be operated at the same time. Essentially each drive element can be thought of as an attention resource. By having a control structure (whether a sequence or a competence) for the grasp, we remove the ambiguity of what not seeing the can means. If the robot is attending to the grasp process, then it has found the can it is grasping, not searching. If the can is lost (say, knocked away by a clumsy grasp) then when the grasping terminates unsuccessfully the search would resume. (See figure **??**.)

Although this control structure is capable of expressing loops or infinite chains, I am not prepared to demonstrate that it could encode every subsumption architecture program (though perhaps I could show one way or another with a little work.) However, it could almost certainly express any published *algorithm*, or intended behavior, that has been implemented in subsumption architecture. My claim is that this architecture intuitively expresses the intentions of the programmer and the interrelations of the behaviors. We tend to think about behavior in terms of causality and preconditions, priorities, sequences, and in goal-oriented clusters. These things are easily expressed.

[4] An action pattern would be appropriate for the exact behavior in (Connell 1990).

This architecture cannot be used for some of the applications of the more ethologically oriented architectures. For example, Cooper, Shallice & Farringdon (1995) demonstrates the Norman & Shallice (1986) model breaks down similarly to humans given certain parameter values which model appropriate neurological diseases. These parameters are inaccessible in my model, the tradeoff for programmability and realistic execution speeds. Consequently, the types of ethological research that can be conducted are limited to a higher level of description.

In theory, a reactive system such as a subsumption architecture program should be implementable purely in terms of a control system with the primitives grounded to simple, detached (rather than interconnected) primitive behaviors with no variables or shared memory. I have not done any experiments in this modality; my most stateless system also uses deictic variables.

7.2 Deictic Representation

One of the problems of traditional representation has been simply the number of items a system needs to represent, if it represents a complete world model. A solution to this that is gaining increased acceptance has been the deictic variable (Ballard et al. forthcoming, Horswill 1997). A deictic variable is a permanent memory structure that can be incorporated into plans, but that's reference may be applied to many targets. For example, and old style representation of a blocks world problem might require individual labeling every block available. Deictic variables would provide a restricted set of labels such as the-block-I'm-holding or even the-green-block-I-most-recently-saw.

Deictic variables are gaining increasing support in the AI community. The idea was first popularized in the work of Agre and Chapman (Agre & Chapman 1988, Chapman 1990) who used the approach to write systems that successfully played computer games with essentially reactive architectures. The concept behind deictic variables can also be found in Minsky's pronemes (**?**) in visual attention Ullman (1984), and apparently even in philosophical work of Heidegger (Dreyfus 1992). Whitehead (1992) uses deictic variables to simplify reinforcement learning. Horswill (1995) uses them in combination with a live visual image to produce a system that answers prolog-like queries ("Is there a green block on a red block?") about the scene with no formal database other than the visual signal itself.

The benefits deictic representation bring to reactive architectures are sim-

ilar to those brought traditional representation, though more substantial. As mentioned earlier, one of the common criticisms of reactive planning is that it transfers the complexity of data representation into control complexity. Thus the reduction of complexity for a reactive system is not in terms of amount of data, but in the size of the control architecture, the number of behaviors, or the number of reactive plans (Horswill 1997, Rhodes 1996). Case based reasoning can also be similar to reactive planning with deictic variables, except that a separate process is responsible for recognizing a context, selecting an appropriate plan, and filling in the few fixed variables (Levison 1996).

7.2.1 The Current Approach

Even pure reactive planning is closely related to deictic representations. In the example given earlier of Genghis avoiding an obstacle, the obstacle might be perceived externally as the deictic variable the-thing-I-just-hit. In this case the representation is not encoded explicitly, it exists only in the state of the behaviors responsible for the avoidance. However, making deictic representations explicit requires little modification to the system described in Section 7.1.2.

Deictic representation requires two things. First some state to be allocated for the variable. This does not necessarily have to be added at the architecture level. The reference of the variable may be whatever is in the hand, where the camera is centered, or what the vision system's tracking system is fixated on. Thus in a real robot, the deictic variable state may be situated either in the external world (the place right in front of the robot), within the robot's mechanism, or refer to state in a peripheral system such as a visual routine processor. Next, there need to be primitives for setting (Acts) and sampling (Senses) that variable.

7.2.2 Trials

The first implementation of the LAP architecture was a blocks world simulation called Braniff. The simulation was based on Whitehead (1992), a reinforcement learning thesis based on visual routine theory. Braniff was intended to be transfered to a real robot arm and vision system, but this transfer has not been completed. The primitive behaviors, though their names were taken

from Whitehead, were assumed to be grounded completely[5] in commands of the Visual Routine Processor described in (Horswill 1995). Thus the behaviors themselves had no internal state beyond control. The state of the deictic variables was recorded in the simulated environment; in the robotic implementation that state would have been maintained by the VRP.

Braniff was conducted in blocksworld to address a challenge to behavior-based AI by Chris Malcolm. Malcolm's own research uses reactive, behavior-based primitives to support a planning system for assembly (Malcolm & Smithers 1990, Wilson 1996); he considers assembly too complex of a task for a strictly behavior-based system(Malcolm 1992). Whitehead's thesis was chosen in particular because of its status as a well-known advance of reinforcement learning. Reinforcement learning develops a lookup table for behavior selection based on observed state. Our intention was to compare our own system with behavior arbitration as results from reinforcement learning.

Whitehead's thesis learned to pick up a green block when presented with up two four stacks of up to four red, green and blue blocks. His system used only two visual markers (deictic variables); keeping this number constrained was necessary for the reinforcement learning, since it is highly susceptible to combinatoric explosion. The target green block is selected by exploiting a known human visual primitive for "popping up" elements of a visual scene with a simple descriptive criteria, such as color or shape. No attempt was made to optimize which green block was targeted for grasping. The system learned to first select a green block, then to move visual attention to the top of its stack to grasp and remove the top block, until the target block (or another, higher green block) was unburied and held in hand.

Scripting this solution as a competence under Braniff was a fairly straight forward piece of coding, once the simulation and architecture were debugged. Every element of the competence was habituated with no recovery, to guarantee failure in the case where no green blocks were available. Essentially, one marker is used to keep track of the selected green block, thus providing persistence. The other is used to find and manipulate the top of the stack, and to observe what is located in the hand. Observing the hand is necessary to detect goal completion. One critical programming strategy emerged from this experience: elements of the same competence must not swap state on

[5]With the exception of the robot arm commands, which were quite unrealistically considered primitives both in Whitehead (1992) and our simulation.

trigger information. For example, one element should not move attention to the hand as its final task, while another moves it to the block. Triggers need to have canonical states, and elements that move triggers *out* of these states must move it back before completing. Of course, elements to move triggers *into* a canonical state are often a necessary part of goal completion.

The next task we attempted was also taken from Whitehead (1992): at one point he states his initial goal for the simulation is to stack a red block on a green block. However, no results were presented beyond grasping red and green blocks. One reason for this became obvious in coding the task: another significant location is needed, the location of the stack. The state of the stack indicates the next task for the robot, whether to pick up a green block, a red block, or whether the stack is complete. In addition, without the mechanism of a subgoal, a fourth marker would be necessary. In order to gauge the state of the stack, a marker must check its contents. If that marker detects that the stack contains a green block, this can trigger the competence to grasp a green block, which uses both spare markers. If there were no separate competence, another marker would be needed to make constant comparisons to the top of the stack. This demonstrates a swapping of control state for deictic state, with considerable combinatorial ramifications.

As a final experiment that took about three hours, I constructed a version of the copy demo very similar to (Chapman 1989). The copy demo requires making a stack to match a goal stack existing in the environment. Coding this more general task actually took no more steps than coding the red on green task. The extra complexity of tracing two stacks was compensated for by not having to represent the goal entirely in the control structure.

7.3 Perception

Perception is a psychological abstraction for the level at which processed sensory information is used for cognition. The abstraction is clearest in humans: what we perceive is what we report, whereas what we sense is presumably more directly a consequence of the environment. Perception is dependent both on learning and on context (Carlson 1994). We can learn to discriminate differences without having any explicit knowledge of what those differences are. In some cases, this learning requires no feedback, simply exposure to the appropriate stimulus (Sundareswaran & Vaina 1994).

The advance of reactive, behavior-based AI has been a change in the

source of action selection. Plans relying on complex internal models of the world have been replaced by distributed actions relying directly on fast, cleverly specialized perceptual routines. In the process, however, the complexity of task that can be managed has been seriously reduced. The previous sections proposed that a better structured control state allows for greater behavior complexity, while still allowing for reactivity. Perceptual state definitionally reduces reactivity, since it relies on "old" information. However, it plays the same role as structured control state in that it preserves information already experienced to disambiguate control decisions.

This same effect might be created with an enormous action-selection lookup table that included events experienced at varying times before the choice as part of the indexing. Taken from this perspective, the use of perceptual learning can be seen as similar to the use of deictic variables. In fact, the more like a deictic variable perceptual learning can become, the better. The thesis of this dissertation is that intelligent agents can best be developed through the use of multiple, possibly interacting, special-purpose units of learning. This thesis is really a restatement of the established result that learning is dependent on bias (**?**, **?**). Our thesis requires developers to maximize bias by minimizing learning and constraining each necessary adaptive element individually.

7.3.1 Implementation and Trials

The fundamental question is when should a programmer resort to perceptual learning? The answer is iterative, in keeping with this dissertation's emphasis on development. Learning is applied when programming control seems too complex otherwise. For example, if the triggers discriminating a set of elements of a competence become too convoluted, either because of large numbers of dependencies or large numbers of steps to find those dependencies, it would probably be easier to reduce the triggers to one or two statements based on new perceptual state. This in turns implies that a new behavior computes this state apparently continuously, so that a value is available on demand. Besides the general assertion that learning should be used as little as possible, the obvious indication that it has been overused is if the agent cannot learn required perceptual information reliably. In this case, there probably needs to be further constraint, at least on this element. Though notice that the additional information needed may also take the form

of more perceptual state rather than more control.

Perceptual learning was not required for Braniff and the assembly work described above, but was very necessary for implementing the next version of that system, Edmund, on a mobile robot. The first use of behaviors centered around state was simply to manage the information load from 32 range sensors and 3 drive velocities (translation, angle of drive, and turret rotation.) The two types of range sensors were consolidated into single values, indicating the nearest hazard in 16 directions. The intended directions of motion were limited to these 16. As explained in Section 5.3, each direction was given an object with an appropriate window into the sensor array, and deictic variables were set up to attend to two directions, current and preferred.

Running the robot provided the first demands for more persistent perceptual memory. The sonar readings are frequently very noisy, so they were buffered with a behavior that guesses the next value from the previous few. This information obviously has a very short useful lifespan, on the order of half a second maximum. Another problem is obstacles that cannot be detected by either range device (see Figure 5.) Bump data is stored on individual incident, but both buffered sonar and bumper information are fused into the original sensor array, so that none of the motion behaviors that depended on it needed to be altered. Bump data is aged not by time, but buy distance. It is both most relevant and most accurate while the robot is still within ten feet of where the impact occurred. Since the stored information is purely odometric, long term representation of bump events would need to have another representation.

Our next tasks were to provide the robot with more cognitive navigation abilities. First, we want the robot to be able to repeat a path it had learned by direction. This involves storing information on the contexts in which decisions (rather than default behavior) were made and record the direction chosen. This memory was stored indexed only by rough location, and is open to revision and consolidation. Secondly, we want the robot to be able to successfully select a direction from a number of possible strategies. This requires an episodic memory of recent events. The events themselves are the same sorts of elements as the mapping information, but their access structure is time rather than location dependent.

These examples illustrate several reasons why one element of memory may be distinct from another.

- The original signal is very different.
- The memory needs to record events that occur at different rates.
- The memory needs to decay at different rates.
- Patterns from the memory emerge at different rates.
- Different representations are most simple or minimal.

7.4 Meta Learning

The final form of state is state about other state. Within the context of the architecture developed for this thesis, this can essentially be reduced to two forms of state — state about control state, and state about perceptual memory. This research to date reflects almost no work in meta learning, but I anticipate some of the following avenues being explored before the end of the dissertation. Implementing them will take a fairly subtle shift, rather than the addition of an entire layer. Essentially, anything can be stored in the same manner as perceptual memory, including control structures.

Control state in Edmund divides into two pieces: activation levels and control structure. A great deal of work has been done on varying activation levels between behaviors. In particular, the ethological literature describes modules competing for attention by evaluating their own activation level, and several ethologically inspired architectures do exactly this (Maes 1989, Correia & Steiger-Garção 1995, Blumberg 1996, Cooper et al. 1995). Blumberg (1996) in particular has achieved perhaps the most coherent behavior in this genre by combining a strategy of resource-based intelligent constraints and mutual inhibition between competing behaviors.

Edmund already allows the controlled variation of activation level through habituation and recovery. I have resisted unconstrained competition and manipulation of activation levels because of both realtime and programmability issues. However, it may well become necessary to allow some behaviors to reprioritize others.

The action patterns and competences of the architecture were designed to support another kind of meta-learning, possibly referred to as thought. The idea is to allow evolutionary-type processes to derive new behaviors or solutions when old ones cannot be produced. (See also (Calvin 1996).) The

simple problems behind this plan is how to generate new thoughts and how to test and store a winning solution. The hard problems are how to tell when a new plan is needed, and how to recognize which solution wins.

Another application of meta-level control would be to implement Norman & Shallice (1986)'s notion of deliberate action. Their theory holds that deliberate control is essentially triggered by special circumstances. Deliberation then monitors and somehow modifies routine processes, possibly by slowing them or providing additional sensory information.

The research in which I'm current engaged is somewhat related, but involves perceptual memory rather than control. The notion is that the learning of sequential tasks involves the monitoring of memory, or a memory of memory. My starting hypothesis is that episodic memory is not stored with chronological ordering. Recent events are clearer simply because of decay caused by interference, which is aggravated by change in context. We know that changing context changes the indexing of the place cells in the hippocampus (Wilson & McNaughton 1994), a memory organ implicated in storing episodic memory (McClelland, McNaughton & O'Reilly 1995). Recent research indicates that human sequence learning can best be modeled by associating elements of the sequence graded relation to its start and end (Henson 1996). Consolidated motor sequence memory seems to have particular cells dedicated to units of behavior and to transitioning (Tanji 1996). Presumably, then, some element of memory allows for this transduction through observing episodic memory.

Implementing this final form of meta-memory requires no structural changes to the Edmund architecture. Additional behaviors and memories are simply added into the behavioral library. Meta-state involving control would require some architectural changes as the representations for control are currently isolated from the behavioral library. However, the overall theoretical structure would again need very little change.

8 Designs of Learning

The previous two sections have given an overview of the architecture provided to support my thesis research, and a decomposition of state and memory with illustrations of how this is supported by the architecture. In summary, this final section relates these tools back to the principles of learning set forward

in the introduction.

My thesis is that developing complex intelligent behavior requires not only modularized behavior, but also modularized learning. Agent programs should constrain and specialize each learning task as much as possible. The previous sections discuss the technical aspects of a methodology to support this approach, and a framework for describing memory. The architecture provides separate components for control and perception. Flow of control is separated from flow of information. Control determines when a behavior is expressed, perception determines how.

State is decomposed into control state: sequences and associated competences that help remove the ambiguity for selecting the next behavior and focus persistence by recording in their state the current and recent courses of action. Control state records both static knowledge of procedure and dynamic decisions on current approach. Deictic variables simplify control state by providing a point of reference through which procedures can be generalized across contexts. Perceptual memory allows control to be determine by factors that appear only across time rather than in the immediate environment. Meta-state allows the agent to learn from and change its own behavior.

Although the progress through these elements reflects an increase of power, it also reflects a decrease not only in reactivity but in constraint or bias. As explained in Section 4.6, animal behavior studies indicate that evolution tends to hard code as much information as possible. In the decomposition of learning I made in the introduction, such hard coded information is called provided knowledge. In robots, the designer is the provider of the knowledge. In my system, this provision takes place both in design of the behaviors and of the control scripts.

Required learning is learning performed by the agent which provisions by the designer guarantee will occur given sufficient experience in a reasonably informative environment. Here learning tends to be highly constrained, not simply associative. In the system described, this is supported by deictic and perceptual learning.

Some of the perceptual learning discussed seems to meet the criteria more of the third form of knowledge, open learning. Map building, although highly constrained and routinely occurring, is not guaranteed to complete or be accurate. There is considerable question as to whether any animal learning is completely open, including human. Language, for example, certainly seems

to be acquired at least partially like other forms of required learning. There are sensitive developmental stages after which language cannot be fully acquired, but language develops in all humans exposed to it during the that time. Similarly, there is evidence that human reasoning is acquired gradually by exposure through uninformed methods and only approaches rationality due to selectionist pressures (**?**). Interestingly, language and reasoning might both be considered examples of learning provided by the species, since they are derivative of culture, in which case they might be considered provided learning as well.

References

Agre, P. E. (1988), The Dynamic Structure of Everyday Life, Technical Report 1085, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, Massachusetts.

Agre, P. E. & Chapman, D. (1987), Pengi: An Implementation of a Theory of Activity, *in* 'Proceedings of the Sixth National Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Inc., Seattle, Washington, pp. 196–201.

Agre, P. E. & Chapman, D. (1988), What are Plans For?, AI memo 1050, MIT, Cambridge, MA.

Albus, J. S. (1997), 'The NIST Real-time control system (RCS): an approach to intelligent systems research', *Journal of Experimental & Theoretical Artificial Intelligence* **9**(2/3), 147–156.

Angle, C. M. (1989), Genghis, a Six Legged Walking Robot, Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts. Bachelor's thesis.

Ballard, D. H., Hayhoe, M. M., Pook, P. K. & Rao, R. P. N. (forthcoming), 'Deictic codes for the embodiment of cognition', *Brain and Behavioral Sciences*.

Blumberg, B. M. (1996), Old Tricks, New Dogs: Ethology and Interactive Creatures, PhD thesis, Learning and Common Sense Section.

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D. P. & Slack, M. G. (1997), 'Experiences with an architecture for intelligent, reactive agents', *Journal of Experimental & Theoretical Artificial Intelligence* **9**(2/3), 237–256.

Braitenberg, V. (1984), *Vehicles: Experiments in Synthetic Psychology*, MIT Press, Cambridge, Massachusetts.

Brooks, R. A. (1986), 'A Robust Layered Control System for a Mobile Robot', *IEEE Journal of Robotics and Automation* **RA-2**, 14–23.

Brooks, R. A. (1989), 'A Robot That Walks: Emergent Behavior from a Carefully Evolved Network', *Neural Computation* **1**(2), 253–262.

Brooks, R. A. (1991*a*), Intelligence Without Reason, *in* 'Proceedings of the 1991 International Joint Conference on Artificial Intelligence', pp. 569–595.

Brooks, R. A. (1991*b*), 'Intelligence without Representation', *Artificial Intelligence* **47**, 139–159.

Bryson, J. (1995), The Reactive Accompanist: Adaptation and Behavior Decomposition in a Music System, *in* L. Steels, ed., 'The Biology and Technology of Intelligent Autonomous Agents', Springer-Verlag.

Calvin, W. H. (1996), *The Cerebral Code*, MIT Press.

Carlson, N. R. (1994), *Physiology of Behavior*, 5 edn, Allyn and Bacon, Boston.

Chapman, D. (1987), 'Planning for conjunctive goals', *Artificial Intelligence* **32**, 333–378.

Chapman, D. (1989), 'Penguins Can Make Cake', *AI Magazine* **10**(4), 51–60.

Chapman, D. (1990), Vision, Instruction, and Action, Technical Report 1204, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, Massachusetts.

Connell, J. (1990), *Minimalist Mobile Robotics: A Colony-style Architecture for a Mobile Robot*, Academic Press, Cambridge, Massachusetts. also MIT TR-1151.

Cooper, R., Shallice, T. & Farringdon, J. (1995), Symbolic and continuous processes in the automatic selection of actions, *in* J. Hallam, ed., 'Hybrid Problems, Hybrid Solutions, Frontiers in Artificial Intelligence and Applications', IOS Press, Amsterdam, pp. 27–37.

Correia, L. & Steiger-Garção, A. (1995), A Useful Autonomous Vehicle With a Hierarchical Behavior Control, *in* F. Moran, A. Moreno, J. Merelo & P. Chacon, eds, 'Advances in Artificial Life (Third European Conference on Artificial Life)', Springer-Verlag, Berlin, Germany, pp. 625–639.

Dennett, D. C. & Kinsbourne, M. (1992), 'Time and the Observer: The Where and When of Consciousness in the Brain', *Brain and Behavioral Sciences* **15**, 183–247.

Dreyfus, H. L. (1992), *What Computers Still Can't Do*, MIT Press, Cambridge, Massachusetts.

Firby, J. (1987), An investigation into reactive planning in complex domains, *in* 'Proceedings of the National Conference on Artificial Intelligence (AAAI)', pp. 202–207.

Firby, J. (1995), Personal communication.

Gallistel, C., Brown, A. L., Carey, S., Gelman, R. & Keil, F. C. (1991), Lessons From Animal Learning for the Study of Cognitive Development, *in* S. Carey & R. Gelman, eds, 'The Epigenesis of Mind', Lawrence Erlbaum, Hillsdale, NJ.

Garcia, J. & Koelling, R. A. (1966), 'The relation of cue to consequence in avoidance learning', *Psychonomic Science* **4**, 123–124.

Gat, E. (1991), Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots, PhD thesis, Virginia Polytechnic Institute and State University.

Glasspool, D. W. (1995), Competitive Queuing and the Articulatory Loop, *in* J. Levy, D. Bairaktaris, J. Bullinaria & P. Cairns, eds, 'Connectionist Models of Memory and Language', UCL Press.

Harvey, I. (1995), The Artificial Evolution of Adaptive Behaviour, PhD thesis, The University of Sussex.

Henson, R. N. A. (1996), Short-term Memory for Serial Order, PhD thesis, St. John's College.

Hexmoor, H. H. (1995), Representing and Learning Routine Activities, PhD thesis.

Hineline, P. & Rachlin, H. (1969), 'Escape and avoidance of shock by pigeons pecking a key', *Journal of Experimental Analysis of Behavior* **12**, 533–538.

Hinton, G. E. & Nowlan, S. J. (1987), 'How Learning Can Guide Evolution', *Complex Systems* **1**, 495–502.

Horswill, I. (1997), 'Visual architecture and cognitive architecture', *Journal of Experimental & Theoretical Artificial Intelligence* **9**(2/3), 277–293.

Horswill, I. D. (1993), Specialization of Perceptual Processes, PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Horswill, I. D. (1995), Visual routines and visual search, *in* 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Montreal.

Kaelbling, L. P. (1993), *Learning in Embedded Systems*, MIT Press, Cambridge, MA. Book form of 1990 Stanford University Thesis.

Koza, J. R. (1991), Evolving Emergent Wall Following Robotic Behavior Using the Genetic Programming Paradigm, *in* 'European Conference on Artificial Life', Paris, France.

Levison, L. (1996), Connecting Planning and Action via Object-Specific Reasoning, PhD thesis, School of Engineering and Applied Science, Computer and Information Science Department.

Maes, P. (1989), How To Do the Right Thing, A.I. Memo 1180, MIT, Cambridge, MA.

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

Malcolm, C. (1992), Personal communication.

Malcolm, C. & Smithers, T. (1990), Symbol Grounding via a Hybrid Architecture in an Autonomous Assembly System, *in* P. Maes, ed., 'Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back', MIT Press, Cambridge, MA, pp. 123–144.

Matarić, M. J. (1992), 'Integration of Representation Into Goal-Driven Behavior-Based Robots', *IEEE Journal of Robotics and Automation* **8**(3), 304–312.

Matarić, M. J. (1997), 'Behaviour-based control: examples from navigation, learning, and group behaviour', *Journal of Experimental & Theoretical Artificial Intelligence* **9**(2/3), 323–336.

McClelland, J. L., McNaughton, B. L. & O'Reilly, R. C. (1995), 'Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionst models of learning and memory', *Psychological Review* **102**(3), 419–457.

McGonigle, B. (1991), Incrementing Intelligent Systems by Design, *in* J.-A. Meyer & S. Wilson, eds, 'From Animals to Animats', MIT Press, Cambridge, MA, pp. 478–485.

McGonigle, B. & Chalmers, M. (1996), The Ontology of Order, *in* L. Smith, ed., 'Piaget: A Critical Assessment', Routledge.

Minsky, M. (1986), *The Society of Mind*, Simon and Schuster, New York, New York.

Mondada, F. & Floreano, D. (1995), 'Evolution of neural control structures: Some experiments on mobile robots', *Robotics and Autonomous Systems* **16**, 183–195.

Neely, J. H. (1991), Semantic priming effects in visual word recognition: A selective review of current findings and theories, *in* D. Besner & G. W. Humphreys, eds, 'Basic Processes in Reading: Visual Word Recognition', Lawrence Erlbaum Associates, chapter 9.

Norman, D. A. & Shallice, T. (1986), Attention to Action: Willed and Automatic Control of Behavior, *in* R.Davidson, G. Schwartz & D. Shapiro, eds, 'Consciousness and Self Regulation: Advances in Research and Theory', Vol. 4, Plenum, New York, pp. 1–18.

Pebody, M. (1995), Learning and Adaptivity: Enhancing Reactive Behaviour Architectures in Real-World Interaction Systems, *in* F. Moran, A. Moreno, J. Merelo & P. Chacon, eds, 'Advances in Artificial Life (Third European Conference on Artificial Life)', Springer-Verlag, Berlin, Germany, pp. 679–690.

Phillips, W. A. & Singer, W. (forthcoming), 'In search of common cortical foundations', *Brain and Behavioral Sciences.*

Rhodes, B. (1996), PHISH-Nets: Planning Heuristically in Situated Hybrid Networks, Master's thesis, Learning and Common Sense.

Rosenschein, S. J. & Kaelbling, L. P. (1995), 'A Situated View of Representation and Control', *Artificial Intelligence.*

Seyfarth, R. M., Cheney, D. L. & Marler, P. (1980), 'Monkey responses to three different alarm calls: Evidence of predator classification and semantic communication', *Science* **14**, 801–803.

Sims, K. (1994), Evolving 3D Morphology and Behavior by Competition, *in* R. A. Brooks & P. Maes, eds, 'Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems', MIT Press, Cambridge, Massachusetts, pp. 28–39.

Singh, M. P., ed. (in press), *The Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL97)*, Springer-Verlag.

Special Issue: Software Architectures for Hardware Agents (1997).

Sundareswaran, V. & Vaina, L. M. (1994), Learning Direction in Global Motion: Two Classes of Psychophysically-Motivated Models, *in* G. Tesauro, D. Touretzky & T. Leen, eds, 'Advances in Neural Information Processing Systems 7', Denver, CO.

Tanji, J. (1996), Involvement of motor areas in the medial frontal cortex of primates in temporal sequencing of multiple movements, *in* R. Caminiti, K. Hoffmann, F. Lacquaniti & J. Altman, eds, 'Vision and Movement: Mechanisms in the Cerebral Cortex', Vol. 2, Human Frontier Science Program, Strasbourg, pp. 126–133.

Ullman, S. (1984), 'Visual Routines', *Cognition* **18**, 97–159.

von der Malsburg, C. (1995), 'Binding in models of perception and brain function', *Current Opinion in Neurobiology* **5**, 520–526.

Whitehead, S. D. (1992), Reinforcement Learning for the Adaptive Control of Perception and Action, Technical Report 406, University of Rochester Computer Science, Rochester, NY.

Wilson, M. & McNaughton, B. (1994), 'Reactivation of Hippocampal Ensemble Memories During Sleep', *Science* **261**, 1227–1232.

Wilson, M. S. (1996), 'Reliability and Flexibility — A Mutually Exclusive Proble m for Robotic Assembly?', *IEEE Transactions on Robotics and Automation*.

Wooldridge, M. & Jennings, N. R. (1995), 'Intelligent Agents: Theory and Practice', *Knowledge Engineering Review*.