Chapter 4

Behavior-Based Architectures

One can expect the human race to continue attempting systems just within or just beyond our reach; and software systems are perhaps the most intricate and complex of man’s handiwork. The management of this complex craft will demand our best use of new languages and systems, our best adaptation of proven engineering management methods, liberal doses of common sense, and a God-given humility to recognize our fallibility and limitations.

—Frederick P. Brooks, Jr.

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

—C.A.R. Hoare

Chapter Objectives

1. To characterize what a robot architecture is.
2. To understand the requirements for the design of a behavior-based robotic architecture.
3. To understand, in depth, two different reactive robotic architectures: subsumption and motor schema.
4. To review many of the other behavior-based architectural choices available to the robot system builder.
5. To develop design principles for the construction of a behavior-based robotic architecture.
4.1 WHAT IS A ROBOTIC ARCHITECTURE?

In chapter 3, we learned about robotic behaviors, including methods for expressing, encoding, and coordinating them. To design and build behavior-based robotic systems, commitments need to be made to the actual, specific methods to be used during this process. This need leads us to the study of robotic architectures: software systems and specifications that provide languages and tools for the construction of behavior-based systems.

All of the architectures described in this chapter are concerned with behavioral control. As described in chapter 1, several non-behavior-based robotic architectures appeared before the advent of reactive control, for example, NASREM (section 1.3.1). Here, however, we focus on behavior-based systems. Though considerably varied, these architectures share many common features:

- emphasis on the importance of coupling sensing and action tightly
- avoidance of representational symbolic knowledge
- decomposition into contextually meaningful units (behaviors or situation-action pairs)

Although these architectures share a common philosophy on the surface, there are many deep distinctions between them, including

- the granularity of behavioral decomposition
- the basis for behavior specification (ethological, situated activity, or experimental)
- the response encoding method (e.g., discrete or continuous)
- the coordination methods used (e.g., competitive versus cooperative)
- the programming methods, language support available, and the extent of software reusability.

In this chapter we study several common robotic architectures used to build behavior-based systems. Tables appear throughout summarizing the characteristics for each of the behavior-based architectures discussed. Two of the architectures have been singled out for closer scrutiny than the others: the subsumption architecture using rule-based encodings and priority-based arbitration; and motor schemas using continuous encoding and cooperative combination of vectors.
4.1.1 Definitions

Perhaps a good place to begin searching for our definition of robotic architectures would be with the definition of computer architectures. Stone (1980, p. 5), one of the best-known computer architects, uses the following definition: "Computer architecture is the discipline devoted to the design of highly specific and individual computers from a collection of common building blocks."

Robotic architectures are essentially the same. In our robotic control context, however, architecture usually refers to a software architecture, rather than the hardware side of the system. So if we modify Stone's definition accordingly, we get:

Robotic architecture is the discipline devoted to the design of highly specific and individual robots from a collection of common software building blocks.

How does this definition coincide with other working definitions by practicing robotic architects? According to Hayes-Roth (1995, p. 330), an architecture refers to "... the abstract design of a class of agents: the set of structural components in which perception, reasoning, and action occur; the specific functionality and interface of each component, and the interconnection topology between components." Although her discussion of agent architectures is targeted for artificially intelligent systems in general, it also holds for the subclass with which we are concerned, namely, behavior-based robotic systems. Indeed, a surveillance mobile robot system (figure 4.1) has been developed that embodies her architectural design principles (Hayes-Roth et al. 1995). She argues that architectures must be produced to fit specific operating environments, a concept closely related to our earlier discussion of ecological niches in chapter 2 and related to the claim in McFarland and Bosser 1993 that robots should be tailored to fit particular niches.

Matarić (1992a) provides another definition, stating, "An architecture provides a principled way of organizing a control system. However, in addition to providing structure, it imposes constraints on the way the control problem can be solved." One final straightforward definition is from Dean and Wellman (1991, p. 462): "An architecture describes a set of architectural components and how they interact."
4.1.2 Computability

Existing robotic architectures are diverse, from the hierarchical NASREM architecture to purely reactive systems such as subsumption (section 4.3) to hybrid architectures (chapter 6). In what ways can instances chosen from the diversity of architectural solutions be said to differ from one another? In what ways can they be said to be the same?

The answer to these questions is related to the distinction between computability and organizing principles. Architectures are constructed from components, with each specific architecture having its own peculiar set of building blocks. The ways in which these building blocks can be connected facilitate certain types of robotic design in given circumstances. Organizing principles underlie a particular architecture's commitment to its component structure, granularity, and connectivity.

From a computational perspective, however, we may see that the various architectures are all equivalent in their computational expressiveness. Consider, for instance, the differences between programming languages. Different
choices are available to the programmer ranging from machine language to assembler to various high-level languages (such as Fortran, Cobol, C, Pascal, and LISP) to very high-level languages such as those used in visual programming. Is there any fundamental incompatibility in the idea that one language can do something that another cannot?

Consider the results that Bohm and Jacopini (1966) derived concerning computability in programming languages. They proved that if any language contains the three basic constructs of sequencing, conditional branching, and iteration, it can compute the entire class of computable functions (i.e., it is Turing equivalent). This essentially states that from a computational perspective the common programming languages have no differences.

The logical extension is that since all robotic architectures provide the capability to perform tasks sequentially, allow conditional branching, and provide the ability for iterative constructs, these architectures are computationally equivalent. All behavior-based robotic architectures are essentially software languages or frameworks for specifying and controlling robots. The level of abstraction they offer may differ, but not the computability.

This does not mean, of course that we will start writing AI programs in Cobol. It does mean that each current programming language has in turn found a niche in which it serves well and thus has survived (i.e., remained in usage) because it is well suited for that particular task. Some argue that the same holds for robotic architectures as well: each serves a particular domain (or niche) and will be subjected to the same environmental stresses for survival as are computer architectures or software languages.

Behavior-based robotic systems serve best when the real world cannot be accurately characterized or modeled. Whenever engineering can remove uncertainty from the environment, purely behavior-based systems may not necessarily afford the best solution for the task involved and hierarchical architectures (chapter 1) may prove more suitable, as, for example, in factory floor operations where the environment can be altered to fit the robot’s needs. More often than not, however, much as we try, we cannot remove uncertainty, unpredictability, and noise from the world. Behavior-based robotic architectures were developed in response to this difficulty and choose instead to deal with these issues from the onset, relying heavily on sensing without constructing potentially erroneous global world models. The more the world changes during execution, the more the resulting value of any plan generated a priori decreases, and the more unstable any representational knowledge stored ahead of time or gathered during execution and remembered becomes.
At a finer level, we will see that behavior-based architectures, because of their different means of expressing behaviors and the sets of coordination functions they afford, provide significant diversity to a robotic system's designer. Each approach has its own strengths and weaknesses in terms of what it is best at doing or where it is most appropriately applied. The remainder of this chapter discusses a variety of behavior-based robot architectural solutions. Not all are expected to withstand the test of time, and many will likely suffer a fate similar to that of early programming languages (e.g., ALGOL, SNOBOL) and fade off into obscurity. Ecological pressure from sources ranging from ease of use for the designer to generalizability to public opinion to exogenous factors (political, economic, etc.) will ultimately serve as the fundamental selection mechanism, not merely an academic's perspective on their elegance, simplicity, or utility.

As an aside, we note the recent controversy in 1989, 1994 stirred up by claiming that no computer program can ever exhibit intelligence as according to Penrose, intelligence must incorporate solutions to noncomputable problems as well as those that are computable. Interestingly, he does not dismiss the attainment of intelligence as utterly impossible in a device and presents a novel, but rather speculative, approach based on quantum mechanics (a micronucleus architecture if you will), rather than a computational approach, to achieve this goal. To say the least, his position has been strongly rebuked by many within the AI community and is often cursorily dismissed as rubbish. In the book's final chapter, we will revisit this issue of what intelligence means within the context of a robotic system and what we can or should expect from these systems. Suffice it to say, for now, that all the behavior-based architectures considered in this book are computational.

### 4.1.3 Evaluation Criteria

How can we measure an architecture’s utility for a particular problem? A list of desiderata for behavior-based architectures is compiled below.

- **Support for parallelism**: Behavior-based systems are inherently parallel in nature. What kind of support does the architecture provide for this capability?
- **Hardware targetability**: Hardware targetability really refers to two different things. The first regards how well an architecture can be mapped onto real robotic systems, that is, physical sensors and actuators. The second is concerned with the computational processing. Chip-level hardware implementations are often preferred over software from a performance perspective. What
type of support is available to realize the architectural design in silicon (e.g.,
compilers for programmable logic arrays [Brooks 1987b])?
• **Niche targetability:** How well can the robot be tailored to fit its operating
environment (Hayes-Roth 1995)? How can the relationships between robot and
environment be expressed to ensure successful niche occupation?
• **Support for modularity:** What methods does an architecture provide for
encompassing behavioral abstractions? Modularity can be found at a variety
of levels. By providing abstractions for use over a wide range of behavioral
levels (primitives, assemblages, agents), an architecture makes a developer's
task easier and facilitates software reuse (Mataric 1992a).
• **Robustness:** A strength of behavior-based systems is their ability to perform
in the face of failing components (e.g., sensors, actuators, etc.) (Payton et
al. 1992; Horswill 1993a; Ferrell 1994). What types of mechanisms does the
architecture provide for such fault tolerance?
• **Timeliness in development:** What types of tools and development environ-
ments are available to work within the architectural framework? Is the archi-
tecture more of a philosophical approach, or does it provide specific tools and
methods for generating real robotic systems?
• **Run time flexibility:** How can the control system be adjusted or reconfig-
ured during execution? How easily is adaptation and learning introduced?
• **Performance effectiveness:** How well does the constructed robot perform
its intended task(s)? This aspect also encompasses the notion of timeliness of
execution, or how well the system can meet established real-time deadlines.
In other instances, specific quantitative metrics can be applied for evaluation
purposes within a specific task context (Balch and Arkin 1994). These may
include such things as time to task completion, energy consumption, minimum
travel, and so forth, or combinations thereof.

These widely ranging criteria can be used for evaluating the relative merits
of many of the architectures described in the remainder of this chapter.

4.1.4 Organizing Principles

From the discussion in chapter 3, several different dimensions for distinguishing
robotic architectures become apparent, including
• Different coordination strategies, of particular note, competitive (e.g., arbit-
tration, action-selection, voting) versus cooperative (e.g., superpositioning)
• Granularity of behavior: microbehaviors such as those found in situated
activity-based systems (e.g., Pengi) or more general purpose task descriptions
(e.g., RAPs).
4.2 A FORAGING EXAMPLE

To ground the following architectural discussions, let us consider a well-studied problem in robotic navigation: foraging. This task consists of a robot's moving away from a home base area looking for attractor objects. Typical applications might include looking for something lost or gathering items of value. Upon detecting the attractor, the robot moves toward it, picks it up, and then returns it to the home base. It repeats this sequence of actions until it has returned all the attractors in the environment. This test domain has provided the basis for a wide range of results on both real robots (Balch et al. 1995; Mataric 1993a) and in simulation. Foraging also correlates well with ethological studies, especially in the case of ants (e.g., Gross et al. 1990).

Several high-level behavioral requirements to accomplish this task include:

1. Wander: move through the world in search of an attractor
2. Acquire: move toward the attractor when detected
3. Retrieve: return the attractor to the home base once acquired

Figure 4.2 represents these higher-level assemblages. Each assemblage shown is manifested with different primitive behaviors and coordinated in different ways as we move from one architectural example to the next.

4.3 SUBSUMPTION ARCHITECTURE

Rodney Brooks developed the subsumption architecture in the mid-1980s at the Massachusetts Institute of Technology. His approach, a purely reactive behavior-based method, flew in the face of traditional AI research at the time. Brooks argued that the sense-plan-act paradigm used in some of the first autonomous robots such as Shakey (Nilsson 1984) was in fact detrimental to
the construction of real working robots. He further argued that building world models and reasoning using explicit symbolic representations of knowledge at best was an impediment to timely robotic responses and at worst actually led robotics researchers in the wrong direction.

In his seminal paper, Brooks (1986) advocated the use of a layered control system, embodied by the subsumption architecture but layered along a different dimension than what traditional research was pursuing. Figure 4.2 shows the distinction, with the conventional sensor-plan-act vertical model illustrated in (A) and the new horizontal decomposition in (B). (The orientation of the lines that separate the components determines vertical and horizontal.)

Much of the presentation and style of the subsumption approach is dogmatic. Tenets of this viewpoint include:

- Complex behavior need not necessarily be the product of a complex control system.
- Intelligence is in the eye of the observer (Brooks 1991a).
- The world is its own best model (Brooks 1991a).
- Simplicity is a virtue.
- Robots should be cheap.
- Robustness in the presence of noisy or failing sensors is a design goal.
- Planning is just a way of avoiding figuring out what to do next (Brooks 1987a).
- All onboard computation is important.
- Systems should be built incrementally.
- No representation. No calibration. No complex computers. No high-bandwidth communication (Brooks 1989b).
This was hard to swallow for many in the AI community (and in many cases, still is). Brooks lobbied long and hard for rethinking the way intelligent robots in particular, and intelligent systems in general, should be constructed. This stance changed the direction of autonomous robotics research. Although currently many in the AI community take a more tempered position regarding the role of deliberation and symbolic reasoning (chapter 6), Brooks has not to date disavowed in print any of these principles (1991a).

Let us now move to the specifics of the subsumption architecture. Table 4.1 is the first of many tables throughout this chapter that provide a snapshot view of the design characteristics of a particular architecture in light of the material discussed in chapter 3.

4.3.1 Behaviors in Subsumption

Task-achieving behaviors in the subsumption architecture are represented as separate layers. Individual layers work on individual goals concurrently and asynchronously. At the lowest level, each behavior is represented using an augmented finite-state machine (AFSM) model (figure 4.4). The AFSM encapsulates a particular behavioral transformation function $\beta$. Stimulus or response signals can be suppressed or inhibited by other active behaviors. A reset input is also used to return the behavior to its start conditions. Each AFSM performs
<table>
<thead>
<tr>
<th>Name</th>
<th>Subsumption architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>Well-known early massive architecture</td>
</tr>
<tr>
<td>Precursors</td>
<td>Breinberg 1984; Walter 1953; Ashby 1952</td>
</tr>
<tr>
<td>Principal design method</td>
<td>Experimental</td>
</tr>
<tr>
<td>Developer</td>
<td>Rodney Brooks (MIT)</td>
</tr>
<tr>
<td>Response encoding</td>
<td>Predominantly discrete (rule based)</td>
</tr>
<tr>
<td>Coordination method</td>
<td>Competitive (priority-based arbitration via inhibition and suppression)</td>
</tr>
<tr>
<td>Programming method</td>
<td>Old method uses AFSMs; new method uses Behavior Language</td>
</tr>
<tr>
<td>Robots fielded</td>
<td>Allen, Gergits (hexapod), Squirt (very small), Tora, Seymour, Polly (four-legged), several others</td>
</tr>
<tr>
<td>References</td>
<td>Brooks 1986; Brooks 1990b; Horwitz 1993a</td>
</tr>
</tbody>
</table>

**Figure 4.4**  
Original AFSM as used within the subsumption architecture.
an action and is responsible for its own perception of the world. There is no global memory, bus, or clock. With this design, each behavioral layer can be mapped onto its own processor (Brooks 1987b). There are no central world models or global sensor representations. Required sensor inputs are channeled to the consuming behavior.

Figure 4.5 shows a simple robot with three behavioral layers. The system was implemented on a radio-controlled toy car (Brooks 1987b). The lowest behavior layer, **avoid-objects**, either halts or turns away from an obstacle, depending upon the input from the robot’s infrared proximity sensors. The **explore** layer permits the robot to move in the absence of obstacles and cover large areas. The highest layer, **back-out-of-tight-situations**, enables the robot to reverse direction in particularly tight quarters where simpler avoidance and exploration behaviors fail to extricate the robot.

As can be seen, the initial subsumption language, requiring the specification of low-level AFMSs, was unwieldy for those not thoroughly schooled in its usage. Recognizing this problem, Brooks (1990a) developed the Behavior Language, which provides a new abstraction independent of the AFMSs themselves using a single rule set to encode each behavior. This high-level language is then compiled to the intermediate AFSM representation, which can then be further compiled to run on a range of target processors.
4.3.2 Coordination in Subsumption

The name subsumption arises from the coordination process used between the layered behaviors within the architecture. Complex actions subsume simpler behaviors. A priority hierarchy fixes the topology. The lower levels in the architecture have no awareness of higher levels. This provides the basis for incremental design. Higher-level competencies are added on top of an already working control system without any modification of those lower levels.

The older version of subsumption specified the behavioral layers as collections of AFMs, whereas the newer version uses behavioral abstractions (in the form of rules) to encapsulate a robot's response to incoming sensor data. These abstractions are then compiled into the older AFM form, but this step is transparent to the developer.

Coordination in subsumption has two primary mechanisms:

- Inhibition: used to prevent a signal being transmitted along an AFM wire from reaching the actuators.
- Suppression: prevents the current signal from being transmitted and replaces that signal with the suppressing message.

Through these mechanisms, priority-based arbitration is enforced.

Subsumption permits communication between layers but restricts it heavily. The allowable mechanisms have the following characteristics:

- Low baud rate, no handshaking
- Message passing via machine registers
- Output of lower layer accessible for reading by higher level
- Inhibition prevents transmission
- Suppression replaces message with suppressing message
- Reset signal restores behavior to original state

The world itself serves as the primary medium of communication. Actions taken by one behavior result in changes within the world and the robot's relationship to it. New perceptions of those changes communicate those results to the other behaviors.

4.3.3 Design in Subsumption-Based Reactive Systems

The key aspects for design of subsumption-style robots are situatedness and embodiment (Brooks 1991b). Situatedness refers to the robot's ability to sense its current surroundings and avoid the use of abstract representations, and em-
bodiment insists that the robots be physical creatures and thus experience the world directly rather than through simulation. Mataric 1992a presents heuristics for the design and development of this type of robot for a specific task. The basic procedure outlined is as follows:

1. Qualitatively specify the behavior needed for the task, that is, describe the overall way the robot responds to the world.
2. Decompose and specify the robot's independent behaviors as a set of observable disjoint actions by decomposing the qualitative behavior specified in step 1.
3. Determine the behavioral granularity (i.e., bound the decomposition process) and ground the resulting low-level behaviors onto sensors and actuators.

An additional guideline regarding response encoding recommends the use of small motions rather than large ballistic ones by resorting to frequent sensing. Finally, coordination is imposed by initially establishing tentative priorities for the behaviors and then modifying and verifying them experimentally.

Let us now review the example of experimentally driven subamption-style design (Brooks 1989a) previously mentioned in section 3.1.3. The target robot is a six-legged walking machine named Genghis (figure 3.6). Its high-level behavioral performance is to be capable of walking over rough terrain and to have the ability to follow a human. This constitutes the qualitative description of task-level performance mentioned in step 1 above.

The next step, involving behavioral decomposition, must now be performed. Each of the following behavioral layers was implemented, tested, and debugged in turn:

1. Standup: Clearly, before the robot can walk, it needs to lift its body off of the ground. Further decomposition leads to the development of two FSMs, one to control the leg's swing position and the other its lift. When all six legs operate under the standup behavior, the robot assumes a stance from which it can begin walking.
2. Simple walk: This requires that the leg be lifted off the ground and swung forward (advance). A variety of sensor data is used to coordinate the motion between legs, including encoders returning the position of each leg's joints. When appropriately coordinated, a simple walk over smooth terrain (tripod gait) is achieved.
3. Force balancing: Now the issues concerning rough terrain are confronted. Force sensors are added to the legs, providing active compliance to changes in the ground's contour.
4. Leg lifting: This helps with stepping over obstacles. When required, the leg can lift itself much higher than normal to step over obstacles.

5. Whiskers: These sensors are added to anticipate the presence of an obstacle rather than waiting for a collision. This capability emerges as important through experiments with the previous behaviors.

6. Pitch stabilization: Further experiments show that the robot tends to bump into the ground either fore or aft (pitching). An inclinometer is added to measure the robot's pitch and use it to compensate and prevent bumping. Now the robot's walking capabilities are complete.

7. Prowling: The walking robot is now concerned with moving toward a detected human. The infrared sensors are tied in. When no person is present, walking is suppressed. As soon as someone steps in front of the robot, the suppression stops and walking begins.

8. Steered prowling: The final behavior allows the robot to turn toward the person in front of it and follow him. The difference in readings between two IR sensors is used to provide the stimulus, and the swing end points for the legs on each side of the robot are determined by the difference in strength.

The completed robot, satisfying the task criteria established for it, consists of fifty-seven AFMOSMs built in an incremental manner. Each layer has been tested experimentally before moving onto the next, and the results of those tests have established the need for additional layers (e.g., whiskers and pitch stabilization).

4.3.4 Foraging Example

The foraging example presented earlier also illustrates subsumption-based design. In particular, the robots Mataric (1993a) constructed for several tasks including foraging provide the basis for this discussion. The robots are programmed in the Behavior Language. The target hardware is an IS Robotics system (figure 4.6).

Each behavior in the system is encoded as a set of rules (standard for the Behavior Language). The overall system has actually been developed as a multiagent robotic system (chapter 9), but for now we will restrict this discussion to a single robot foraging. The following behaviors are involved:

- Wandering: move in a random direction for some time.
- Avoiding:
  - turn to the right if the obstacle is on the left, then go.
  - turn to the left if the obstacle is on the right, then go.
Figure 4.6
Subsumption-based foraging robot: R1. (Photograph courtesy of IS Robotics, Somerville, MA.)
4.3.5 Evaluation

When the criteria presented in section 4.1.3 are applied to evaluate the subsumption architecture, they identify the following strengths:

- Hardware retargetability: Subsumption can compile down directly onto programmable-array logic circuitry (Brooks 1987b).
- Support for parallelism: Each behavioral layer can run independently and asynchronously.
- Niche retargetability: Custom behaviors can be created for specific task-environment pairs.

The following characteristics emerge as neither strength nor weaknesses:
Chapter 4

- Robustness: This can be successfully engineered into these systems but is often hard-wired (Ferrell 1994) and hence hard to implement.
- Timeliness for development: Some support tools exist for these systems, but a significant learning curve is still associated with custom behavioral design. Experimental design, involving trial-and-error development, can slow development. Also, consistent with Brooks’ philosophy, simulators are not used to protest behavioral efficiency.

Under the criteria, the following show up as weaknesses:

- Run time flexibility: The priority-based coordination mechanism, the ad hoc flavor of behavior generation, and the architecture’s hard-wired aspects limit the ways the system can be adapted during execution.
- Support for modularity: Although behavioral reuse is possible through the Behavior Language, it is not widely evidenced in constructed robots. Subsumption has also been criticized on the basis that since upper layers interfere with lower ones, they cannot be designed completely independently (Hartley and Pipitone 1991). Also behaviors cannot always be prioritized (or should they be), leading to artificial arbitration schemes (Hartley and Pipitone 1991). Commitment to subsumption as the sole coordination mechanism is restrictive.

4.3.6 Subsumption Robots

Many different robots (figure 4.8) have been constructed using the subsumption architecture. Brooks (1990b) reviews many of them. They include

- Allen: the first subsumption-based robot, which used sonar for navigation based on the ideas in Brooks 1986.
- Tom and Jerry: two small toy cars equipped with infrared proximity sensors (Brooks 1990b).
- Grashe and Attlia: six-legged hexapods capable of autonomous walking (Brooks 1989a).
- Squirt: a two-ounce robot that responds to light (Flynn et al. 1989).
- Toto: the first map-constructing, subsumption-based robot and the first to use the Behavior Language (Mataric 1992b).
- Tito: a robot with stereo navigational capabilities (Saracik 1989).
- Polly: a robotic tour guide for the MIT AI lab (Hearst 1993b).
- Cog: a robot modeled as a humanoid from the waist up, and used to test theories of robot-human interaction and computer vision (Brooks and Stein 1994).
4.4 MOTOR SCHEMAS

Another approach, more strongly motivated by the biological sciences, appeared on the heels of the subsumption architecture. This behavior-based method used schema theory, which we reviewed in chapter 2. We recall from that chapter the great utility of schema theory; it provides the following capabilities for specifying and designing behavior-based systems (adapted from Brooks 1992):

- Schema theory explains motor behavior in terms of the concurrent control of many different activities.
- A schema stores both how to react and the way that reaction can be realized.
- Schema theory is a distributed model of computation.
- Schema theory provides a language for connecting action and perception.
- Activation levels are associated with schemas that determine their readiness or applicability for acting.