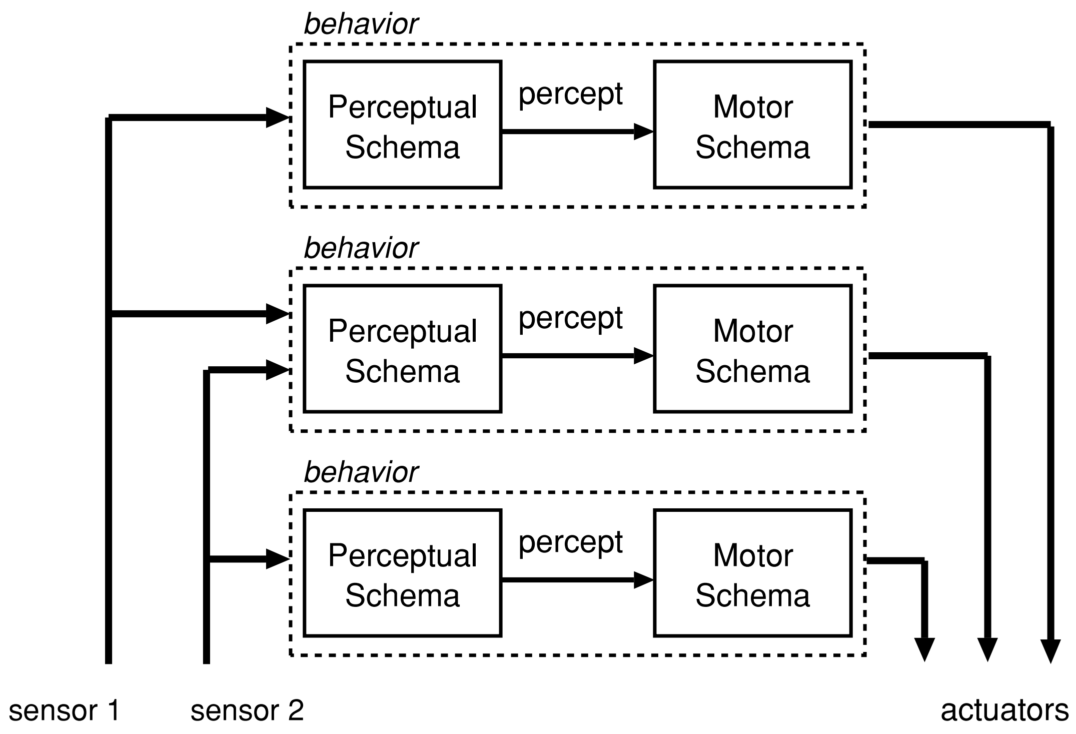

Figure 4.4 Behavior-specific sensing organization in the Reactive Paradigm: sensing is local, sensors can be shared, and sensors can be fused locally by a behavior.

4.2.1 Characteristics and connotations of reactive behaviors

As seen earlier, a reactive robotic system decomposes functionality into behaviors, which tightly couple perception to action without the use of intervening abstract (global) representations. This is a broad, vague definition. Over the years, the reactive paradigm has acquired several connotations and characteristics from the way practitioners have used the paradigm.

The primary connotation of a reactive robotic system is that it executes rapidly. The tight coupling of sensing and acting permits robots to operate in real-time, moving at speeds of 1-2 cm per second. Behaviors can be implemented directly in hardware as circuits, or with low computational complexity algorithms (O(n)). This means that behaviors execute quickly regardless of the processor. Behaviors execute not only fast in their own right, they are particularly fast when compared to the execution times of Shakey and the Stanford Cart. A secondary connotation is that reactive robotic systems have no memory, limiting reactive behaviors to what biologists would call pure stimulus-response reflexes. In practice, many behaviors exhibit a fixed-action pattern type of response, where the behavior persists for a short period of time without the direct presence of the stimulus. The main point is that behaviors are controlled by what is happening in the world, duplicating the spirit of innate releasing mechanisms, rather than by the program storing and remembering what the robot did last. The examples in the chapter emphasize this point.

The five characteristics of almost all architectures that follow the Reactive Paradigm are:

1. *Robots are situated agents operating in an ecological niche.* As seen earlier in Part I, *situated agent* means that the robot is an integral part of the world. A robot has its own goals and intentions. When a robot acts, it changes the world, and receives immediate feedback about the world through sensing. What the robot senses affects its goals and how it attempts to meet them, generating a new cycle of actions. Notice that situatedness is defined by Neisser's Action-Perception Cycle. Likewise, the goals of a robot, the world it operates in, and how it can perceive the world form the ecological niche of the robot. To emphasize this, many robotic researchers say they are working on *ecological robotics*.

SITUATED AGENT

ECOLOGICAL ROBOTICS

2. *Behaviors serve as the basic building blocks for robotic actions, and the overall behavior of the robot is emergent.* Behaviors are independent, computational entities and operate concurrently. The overall behavior is emergent: there is no explicit "controller" module which determines what will be done, or functions which call other functions. There may be a coordinated control program in the schema of a behavior, but there is no external controller of all behaviors for a task. As with animals, the "intelligence" of the robot is in the eye of the beholder, rather than in a specific section of code. Since the overall behavior of a reactive robot emerges from the way its individual behaviors interact, the major differences between reactive architectures is usually the specific mechanism for interaction. Recall from Chapter 3 that these mechanisms include combination, suppression, and cancellation.

3. *Only local, behavior-specific sensing is permitted.* The use of explicit abstract representational knowledge in perceptual processing, even though it is behavior-specific, is avoided. Any sensing which does require representation is expressed in *ego-centric* (robot-centric) coordinates. For example, consider obstacle avoidance. An ego-centric representation means that it does not matter that an obstacle is in the world at coordinates (x,y,z), only

EGO-CENTRIC

where it is relative to the robot. Sensor data, with the exception of GPS, is inherently ego-centric (e.g., a range finder returns a distance to the nearest object from the transducer), so this eliminates unnecessary processing to create a world model, then extract the position of obstacles relative to the robot.

4. *These systems inherently follow good software design principles.* The modularity of these behaviors supports the decomposition of a task into component behaviors. The behaviors are tested independently, and behaviors may be assembled from primitive behaviors.

5. *Animal models of behavior are often cited as a basis for these systems or a particular behavior.* Unlike in the early days of AI robotics, where there was a conscious effort to not mimic biological intelligence, it is very acceptable under the reactive paradigm to use animals as a motivation for a collection of behaviors.

4.2.2 Advantages of programming by behavior

Constructing a robotic system under the Reactive Paradigm is often referred to as programming by behavior, since the fundamental component of any implementation is a behavior. Programming by behavior has a number of advantages, most of them consistent with good software engineering principles. Behaviors are inherently modular and easy to test in isolation from the system (i.e., they support unit testing). Behaviors also support incremental expansion of the capabilities of a robot. A robot becomes more intelligent by having more behaviors. The behavioral decomposition results in an implementation that works in real-time and is usually computationally inexpensive. Although we'll see that sometimes duplicating specialized detectors (like optic flow) is slow. If the behaviors are implemented poorly, then a reactive implementation can be slow. But generally, the reaction speeds of a reactive robot are equivalent to stimulus-response times in animals.

LOW COUPLING
HIGH COHESION

Behaviors support good software engineering principles through decomposition, modularity and incremental testing. If programmed with as high a degree of independence (also called *low coupling*) as possible, and *high cohesion*, the designer can build up libraries of easy to understand, maintain, and reuse modules that minimize side effects. Low coupling means that the modules can function independently of each other with minimal connections or interfaces, promoting easy reuse. Cohesion means that the data and operations contained by a module relate only to the purpose of that module.