Journal of Experimental and Theoretical Artificial Intelligence (JETAI) 9, 1997, 215-235.
Special issue on Architectures for Physical Agents.

The Saphira Architecture: A Design for Autonomy

Kurt Konolige
Karen Myers
Enrique Ruspini
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
{konolige,myers,ruspini}@ai.sri.com

Alessandro Saffiotti*
IRIDIA, Universitè Libre de Bruxelles
50 av. F. Roosevelt, CP 194/6
1050 Brussels, Belgium
asaffio@ulb.ac.be

April 2, 1996

Abstract

Mobile robots, if they are to perform useful tasks and become accepted in open environments, must be fully autonomous. Autonomy has many different aspects; here we concentrate on three central ones: the ability to attend to another agent, to take advice about the environment, and to carry out assigned tasks. All three involve complex sensing and planning operations on the part of the robot, including the use of visual tracking of humans, coordination of motor controls, and planning. We show how these capabilities are integrated in the Saphira architecture, using the concepts of coordination of behavior, coherence of modeling, and communication with other agents.

*This paper reports work done while this author was at SRI International.

1 Autonomous Mobile Agents

What are the minimal capabilities for an autonomous mobile agent? Posed in this way, the question is obviously too broad; we would like to know more about the task and the environment: What is the agent supposed to do — will it just have a limited repertory of simple routines, or will it have to figure out how to perform complex assignments? Will there be special engineering present in the environment, or will the agent have to deal with an unmodified space? How will its performance be judged? Will it have to interact with people, and in what manner?

As has become clear from the mobile robot competitions at the last three NCAI conferences [31], the more restricted the environment and task (the less *open-ended*, the better mobile agents can perform. Designers are adept at noticing regularities, and taking advantage of them in architectural shortcuts. As a consequence, contest creators have become more subtle in how they define the rules, striving to reward mobile agents that exhibit more autonomy. To do so, they have had to grapple with refinements of the questions just posed. Although there may be no definitive answers, we can try to address these questions, like the contest creators, by articulating a scenario in which the autonomous agent must perform. To push our research, we have tried to make the environment and task as natural and open-ended as possible, given current limitations on the robot's abilities.

Fortunately, in designing a scenario we had outside help: in March 1994 we were approached by the producers of the science show "Scientific American Frontiers," who were interested in showcasing the future of robotics. After some discussion, we decided on a scenario in which our robot, Flakey, would be introduced to the office environment as a new employee, and then asked to perform a delivery task. To realize the scenario, Flakey would need at least the following capabilities:

Attending and Following. A supervisor would introduce Flakey to the office by leading it around and pointing out who inhabited each office. Flakey would have to locate and follow a human being. It would also have to know if a human was present by speech, e.g., going to an office door and inquiring if anyone was present.

Taking advice. Advice from the teacher would include map-making information such as office assignments and information about potential hazards ("There's a possible water leak in this corridor."). It would also include information about how to find people ("John usually knows where Karen is").

Tasking. Flakey would have to perform delivery tasks using its learned knowledge. The task was chosen to illustrate the different types of knowledge Flakey had: maps, information about office assignments, general knowledge of how to locate people. There was to be no engineering of the environment to help the robot; it would have to deal with offices, corridors, and the humans that inhabited them, without any special beacons, reflective tags, markers, and so on. Any machine-human communication would use normal human-human modalities: speech and gestures.

The scenario was made more difficult by three factors: there were only 6 weeks to prepare; the supervisor would have no knowledge of robotics (it was Alan Alda, the program host); and the scenario was to be completed in one day, so the robot hardware and software had to be very robust. We converged on this scenario because it was the most open-ended one we could think of that would be doable with current equipment and algorithms, and because it would hint at what future mobile agents would be like.

We believe that any mobile agent able to perform well in an open-ended scenario such as this one must incorporate some basic features in its architecture. Abstractly, we have labeled these the

three C's: *coordination*, *coherence*, and *communication*.

Coordination. A mobile agent must coordinate its activity. At the lowest level there are effector commands for moving wheels, camera heads, and so on. At the highest level there are goals to achieve: getting to a destination, keeping track of location. There is a complex mapping between these two, which changes depending on the local environment. How is the mapping to be specified? We have found, as have others [5, 10, 8, 2], that a layered abstraction approach makes the complexity manageable.

Coherence. A mobile agent must have a conception of its environment that is appropriate for its tasks. Our experience has been that the more open-ended the environment and the more complex the tasks, the more the agent will have to understand and represent its surroundings. In contrast to the reactivists: "the environment is the model" [4], we have found that appropriate, strong internal representations make the coordination problem easier, and are indispensable for natural communication. Our internal model, the Local Perceptual Space (LPS), uses connected layers of interpretation to support reactivity and deliberation.

Communication. A mobile agent will be of greater use if it can interact effectively with other agents. This includes the ability to understand task commands, as well as integrate advice about the environment or its behavior. Communication at this level is possible only if the agent and its respondent internalize similar concepts, for example, about the spatial directions "left" and "right." We have taken only a small step here, by starting to integrate natural language input and perceptual information. This is one of the most interesting and difficult research areas.

In the rest of this paper we describe out approach to autonomous systems. Most of the discussion is centered on a system architecture, called Saphira, that incorporates the results of over a decade of research in autonomous mobile robots.

2 The Flakey Testbed

The Saphira architecture is implemented on Flakey, a custom research robot, and Pioneer, a small commercial robot from Real World Interface intended for research and educational uses.[1] Saphira uses a client-server method to isolate the specifics of the robot hardware from the agent architecture [14], and is accessible to any robot base that adheres to its protocol. Here we discuss the Flakey testbed, which was used in the Scientific American scenario.

Flakey is a moderately-sized mobile robot designed and built in the mid-1980's. Its shape is an octagonal cylinder, approximately .5 meters in diameter and one meter high. Its sensors include a ring of 12 sonar sensors on the bottom, and a stereo camera pair mounted on a pan/tilt head. Flakey also has a speaker-independent continuous speech recognition system called CORONA, developed at SRI, and a standard text-to-speech program for speech output.

All systems run on-board, using a 2-processor Sparcstation configuration.[2] One processor is dedicated to speech and robot control, the other runs the visual interpretation programs. These include stereo algorithms [33], which give full-frame dense stereo maps at a rate of about 2.5Hz. We use the results in two ways.

[1] More information on Flakey and Pioneer can be found at the web sites **http://www.ai.sri.com/people/{flakey,erratic}**.

[2] At the time of the demonstration, Flakey had a single onboard processor, which ran the vision algorithms and basic motor control. The rest of the work was done by an offboard Sparcstation connected through a radio Ethernet.

1. to identify surfaces that could be possible obstacles by matching their height against the ground plane, and

2. to find and track person-like objects.

A short description of the tracking algorithm is given in Section 5.3.

Flakey was used in early experiments with Stan Rosenschein's Situated Automata theory [24], which eschews representational models of the environment. In subsequent years we have develop a more classical environmental model we call the Local Perceptual Space (LPS). The LPS lies at the core of the Saphira architecture, which we discuss in the next few sections.

3 The Saphira Architecture

The Saphira architecture [29, 28, 7] is an integrated sensing and control system for robotics applications. At the center is the LPS (see Figure 1), a geometric representation of space around the robot. Because different tasks demand different representations, the LPS is designed to accommodate various levels of interpretation of sensor information, as well as *a priori* information from sources such as maps. Currently, the major representational technologies are:

- A grid-based representation similar to Moravec and Elfes' occupancy grids [19], built from the fusion of sensor readings.

- More analytic representations of surface features such as linear surfaces, which interpret sensor data relative to models of the environment.

- Semantic descriptions of the world, using structures such as corridors or doorways (*artifacts*). Artifacts are the product of bottom-up interpretation of sensor readings, or top-down refinement of map information.

The LPS gives the robot an awareness of its immediate environment, and is critical in the tasks of fusing sensor information, planning local movement, and integrating map information. The perceptual and control architecture make constant reference to the local perceptual space. The LPS gives the Saphira architecture its representational coherence. As we will show in Section 5, the interplay of sensor readings and interpretations that takes place here lets Saphira constantly coordinate its internal notions of the world with its sensory impressions. One can think of the internal artifacts as Saphira's *beliefs* about the world, and most actions are planned and executed with respect to these beliefs.

In Brooks' terms [3], the organization is partly vertical and partly horizontal. The vertical organization occurs in both perceptual (left side) and action (right side). Various perceptual routines are responsible for both adding sensor information to the LPS and processing it to produce surface information that can be used by object recognition and navigation routines. On the action side, the lowest level behaviors look mostly at occupancy information to do obstacle avoidance. The basic building blocks of behaviors are fuzzy rules, which give the robot the ability to react gracefully to the environment by grading the strength of the reaction (e.g., turn left) according to the strength of the stimulus (e.g., distance of an obstacle on the right).

More complex behaviors that perform goal-directed actions are used to guide the reactive behaviors, and utilize surface information and artifacts; they may also add artifacts to the LPS as control points for motion. At this level, fuzzy rules blend possibly conflicting aims into one smooth action sequence. Finally, at the task level complex behaviors are sequenced and their progress is monitored through events in the LPS. The horizontal organization comes about because behaviors can choose

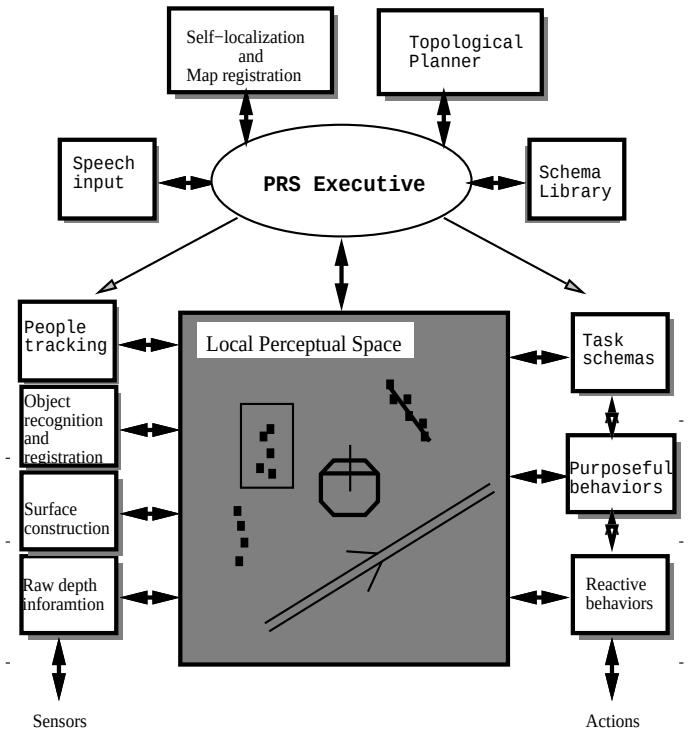


Figure 1: Saphira system architecture. Perceptual routines are on the left, action routines on the right. The vertical dimension gives an indication of the cognitive level of processing, with high-level behaviors and perceptual routines at the top. Control is coordinated by the Procedural Reasoning System, which instantiates routines for task sequencing and monitoring, and perceptual coordination.

appropriate information from the LPS. Behaviors that are time-critical, such as obstacle avoidance, rely more on very simple processing of the sensors because it is available quickly. However, these routines may also make use of other information when it is available, e.g., prior information from the map about expected obstacles.

4 Coordination

In the ensuing subsections, we describe in more detail how the behavior-based routines work, and how control is exercised by the PRS executive. This is the part of Saphira that deals with *coordination*: making the low-level motor commands responsive to the goals of the agent.

4.1 Behaviors

At the control lever, the Saphira architecture is behavior-based: the control problem is decomposed into small units of control called *basic behaviors*, like obstacle avoidance or corridor following. One of the distinctive features of Saphira is that behaviors are written and combined using techniques based on fuzzy logic (see [29] and [27] for a more detailed presentation). We use fuzzy control rules of the form

$$A \rightarrow C\,,$$

where A is a fuzzy formula composed by fuzzy predicates and the fuzzy connectives AND, OR and NOT, and C is a control action. Controls typically refer to forward velocity and angular orientation. A typical control rule might be:

> If the wall is too close on the left, then turn right moderately.

A fuzzy predicate can be partially true of a given state of the world (in our case, the content of the LPS). Truth values are represented by numbers in the interval $[0, 1]$. For example, the predicate "too-close" has truth value 0.8 if the distance is 700 mm. AND, OR and NOT are evaluated on the min, max, and complement to 1, respectively. The truth value of the antecedent A of a control rule determines the desirability of applying the control C in the current state. In general, C is a fuzzy set of controls: the C sets generated by the different rules in a behavior are weighted by the truth value of the rule antecedents, and then combined using fuzzy set operations. Correspondingly, each behavior actually outputs a full *desirability function*: a measure of how much each possible control c is desirable given the current input state s [25].

Given a desirability function $Des(s, c)$, we eventually need to select one control $\hat{c}$ to send to the effectors for actual execution. In our experiments, we have used a simple "defuzzification" function (weighted average):

$$\hat{c} = \frac{\int c\, Des(s, c)\, dc}{\int Des(s, c)\, dc}.$$

For averaging to make sense, the rules in a behavior should not suggest dramatically opposite actions in the same state. Our coding heuristic has been to make sure that rules with conflicting consequents have disjoint antecedents. Other authors have preferred to use more involved choice functions (e.g., [32]).

Basic behaviors take their input from the LPS. Simple reactive behaviors, like obstacle avoidance, have rules whose antecedent depends on low-level information, like occupancy information, which is quickly available. However, behaviors can also inspect more complex data structures in the LPS: this is the way *goal-seeking behaviors* are implemented. Goal-seeking behaviors take their input from

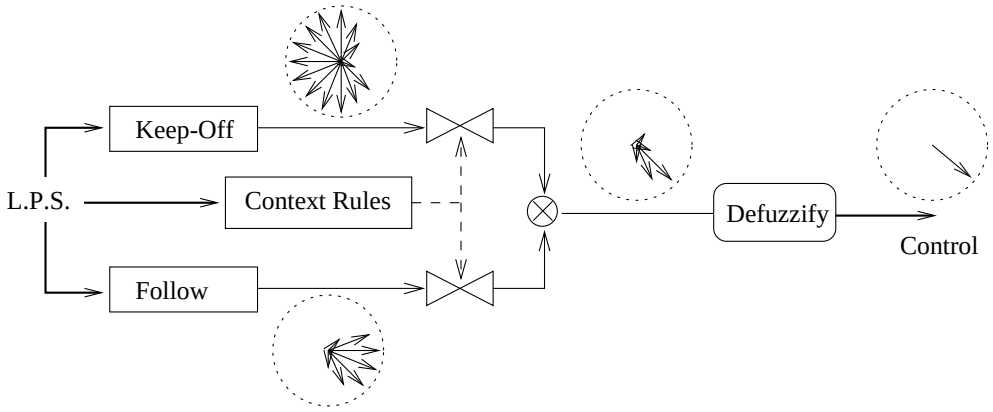


Figure 2: Context-dependent blending of follow-corridor and avoid behaviors.

artifacts in the LPS that represent the objects with respect to which the behavior must operate. For example, the behavior "Cross-Door" uses the coordinates of a door artifact in the LPS as input. Other than this, purposeful and reactive behaviors have the same form.

Using artifacts to control purposeful movement is a convenient way to bring strategic goals and prior knowledge into the controller. For example, the door artifact above is typically put in the LPS by the planning and execution levels in order to orient the control toward the crossing of a given door; the properties of the door artifact are based on information taken from a map of the environment. In order to preserve a closed loop response to the environment, however, artifacts need to be registered against the information coming from the sensors: we shall see how this is done by Saphira in Section 5.

Basic behaviors can be combined to form *complex behaviors*. To combine two behaviors, we simply take the desirability function generated by each one of them, and combine them through a minimum operation. The resulting desirability function gives preference to the controls that are desirable to both behaviors. Defuzzification is then applied to choose one tradeoff control value. For instance, a "follow-corridor" behavior and a "go-fast" behavior can be combined in this way to produce a behavior to go down an hallway quickly.

Care must be taken, however, of possible conflicts among behaviors aiming at different, incompatible goals. These conflicts would result in desirability functions that assign high values to opposite actions: simple min composition should not be applied in these cases. For example, suppose that we want to combine a corridor following behavior, named "Follow", with an obstacle avoidance one, named "Keep-Off". In the state where the robot is facing an obstacle, Follow would prefer controls that go forward, while Keep-Off would favor controls that turn the robot, say, right. The key observation here is that each behavior has its own *context of applicability*, and each desirability function should be considered only when appropriate. In the previous example, the Follow behavior can be sensibly applied only in situations where the space in front of the robot is free. When the obstacle is detected, this behavior is outside its area of competence, and we should (partially) disregard its preferences.

In general, we build composite behaviors by *context dependent blending* of simpler ones: the output of each behavior is weighted according to the truth value of its context, and all the outputs and then merged by fuzzy AND. More precisely, given a set $\{B_1, \ldots, B_k\}$ of behaviors, we define

Figure 3: Context-dependent blending in operation.

their *context-dependent blending* to be the composite behavior described by the following desirability function

$$Des(s,c) = (Cxt_1(s) \wedge Des_1(s,c)) \ \vee \cdots \vee \ (Cxt_k(s) \wedge Des_k(s,c)).$$

where $\wedge$ and $\vee$ denote min and max, Des_i is the desirability function of B_i, and Cxt_i is its context.

In practice, we use context meta-rules of the form

$$\text{IF } A' \text{ THEN activate } B$$

where A' is a fuzzy formula describing the context and depending on the content of the LPS, and B is a behavior. For example, the two meta-rules

IF obstacle-close THEN activate KEEP-OFF
IF NOT(obstacle-close) THEN activate FOLLOW

produce a behavior for following a corridor while avoiding obstacles on the way. Figure 2 illustrates context-dependent blending schematically. The groups of arrows represent the desirability functions for the turning angle produced at various stages; the length of each arrow is proportional to the degree of desirability of the corresponding turn. Note that defuzzification is applied *after* combination to choose one preferred tradeoff control from the overall desirability function

Figure 2 shows a run of the previous blending on Flakey. On the right, we plot the evolution over time of the truth value of the contexts, hence the level of activation of the corresponding behaviors in the blending. In (a), the obstacle has been detected, and the preferences of KEEP-OFF begin to dominate, thus causing Flakey to slow down and abandon the center of the hallway; later, when the path is clear (b), the goal-oriented preferences expressed by FOLLOW re-gain importance, and Flakey re-gains the midline at full speed.

Context-dependent blending proved to be an effective technique for coordinating reactive and goal-oriented behaviors. Behaviors are not just switched on and off: rather, their preferences are combined into a tradeoff desirability. An important consequence is that the preferences of the goal-seeking behaviors are still considered during reactive maneuvers, thus biasing the control choices toward the achievement of the goals. In the example above, suppose that the obstacle is right in front of the robot (e.g., the robot ended up facing the wall) and can thus be avoided by either turning right or left; then, the combined behavior prefers the side that better promotes corridor following.

It is interesting to compare context-dependent blending with the so-called "artificial potential field" technique, first introduced by Khatib [13] and now extensively used in the robotic domain [15], [2]. In the potential field approach, a goal is represented by a potential measuring the desirability of each state from that goal's viewpoint. For example, the goal of avoiding obstacles is represented by a potential field having maximum value around the obstacles; and the goal of reaching a given location is represented by a field having minimum value at that location. At each point, the robot responds to a pseudo-force proportional to the vector gradient of the field. Potential fields are combined by linear superposition: one takes a weighted vector sum of the associated pseudo-forces. Each force is a summary of the preferences that produced that force (e.g., which direction is best to avoid an obstacle), and the combined force is a combination of the summaries. In contrast, when combining two desirability functions, we *first* combine the component desirability functions, effectively forming

a full preference function, and *then* chose one preferred control from the combined function through defuzzification (cf. Fig. 2). This may give results different from the ones obtained by combining pseudo-forces. Intuitively, desirability functions carry more information than pseudo-forces, in that they also measures the desirability on non-optimal controls.

4.2 Controlling Executive: PRS-lite

Behaviors provide low-level situated control for the physical actions effected by the system. Above that level, there is a need to relate behaviors to specific goals and objectives that the robot should undertake. This management process involves determining when to activate/deactivate behaviors as part of the execution of a task, as well as coordinating them with other activities in the system. PRS-Lite [22], a reactive controller based loosely on the Procedural Reasoning System (PRS-CL) [11, 21] fills this role within Saphira.

PRS-Lite and PRS-CL are similar in spirit. Both manage the invocation and execution of procedural representations of the knowledge required to achieve individual tasks. Both provide the smooth integration of goal-driven and event-driven activity, while remaining responsive to unexpected changes in the world. However, the embodiment of the procedural knowledge philosophy in the two systems is markedly different. PRS-CL is a large, general-purpose, mature system that was designed for use in a broad range of control applications. It provides many sophisticated services, including a multiagent architecture, multitasking, metalevel reasoning capabilities, and rich interactive control via graphical interfaces. PRS-Lite is a minimalist redesign that omits certain of these features for reasons of compactness and efficiency. For example, while metalevel reasoning can be valuable in certain situations, its support incurs a heavy cost of deliberation. The key objective in designing PRS-Lite was to retain the mixture of goal-directed and reactive activity, but in a more streamlined setting.

PRS-Lite is not simply a subset of PRS-CL. Indeed, certain of the requirements for robot control are absent from PRS-CL. One problem is PRS-CL's assumption of atomicity for its primitive actions, making it unsuitable for the control of continuous processes. A related problem is its goal semantics: goals either succeed or fail, with their outcome affecting the overall flow of control in the system. As has been noted [9, 10], this semantics is inappropriate for managing continuous processes. PRS-Lite employs an alternative goal semantics that supports both atomic actions and continuous processes, as well as a control regime divorced from any notion of goal success or failure.

The representational basis of PRS-Lite is the *activity schema*, a parameterized finite-state machine whose arcs are labeled with goals to be achieved. Each schema embodies procedural knowledge of how to attain some objective via a sequence of subgoals, perceptual checks, primitive actions, and behaviors. Activity schemas are launched by instantiating their parameters and *intending* them into the system. Such instantiated schemas are referred to as *intentions*. Multiple intentions can be active at once, providing a multitasking capability. An *executor* repeatedly operates a short cycle in which it polls each active intention to determine whether any actions can/should be taken to achieve the current goals of the intention. Processing of an individual intention consists of at most a single step in each cycle, in order to ensure responsiveness to new events in the world.

Different modalities of goals are supported, corresponding to different classes of operations: *testing* conditions, *waiting* for certain conditions or events, *executing* specific code, *intending* instantiated schemas, and *deactivating* (or *unintending*) schemas. At the most primitive level, goals are grounded in either executable functions, tests on the world, or the activation/deactivation of behaviors. Intentions can be launched in either blocking or non-blocking mode. For blocking mode, further activity along that branch is suspended until the launched intention completes. In non-blocking mode, a separate thread of execution is spawned. There are also higher-order control primitives for expressing

conditional goals, parallel sets of goals, and iteration. Overall, the language can be used to construct a forest of hierarchically structured parallel activities.

Some schemas are quite complex; for example, the delivery schema discussed below. Others are fairly simple; for example, to detect closed doors, a monitoring schema is fired up every time Flakey attempts to go through a doorway, and if no progress is made after a fixed amount of time, or if the sensors detect that the doorway is closed, the schema halts the current door-crossing behavior, updates a global map with the new information (more on this below), and signals the executor that the intention has failed.

Goal-directed behavior is produced by intending schemas for satisfying individual tasks. Reactive, event-directed behavior is produced by launching intentions that employ *waiting* goals to suspend until some condition or event transpires. For many essential robot operations, it is common to create an intention that invokes a lower-level intention to perform the task, and one or more accompanying monitor intentions that detect changes in the world that could invalidate the actions being taken for the current task. For instance, there is a `Plan-and-Execute` schema that first invokes a topological path planner to generate an appropriate plan, then launches both an intention to execute the plan and a monitoring intention that oversees the execution to determine when problems have arisen. When non-recoverable problems are detected, the monitoring intention aborts both the plan execution and itself.

Figure 4.2 summarizes the intention structures from a run in which Flakey was executing a delivery task. It displays a snapshot of the hierarchical structure of active intentions and their associated behaviors at a particular point during execution.[3] Each line in the display consists of: an initial marker, indicating whether the intention is blocking ($*$) or nonblocking ($\circ$), the name of the Intention (eg, `Deliver-Object`), a unique identifier for the particular instantiation of the activity schema (eg, `I3674`), and either the next state of execution (for an intention) or `B` (for a behavior). At the instant captured by this display, PRS-Lite has two intentions active at the highest-level (corresponding to two distinct user-specified objectives): `Deliver-Object` and `Avoid`. The `Avoid` intention has only one active thread at this point, namely the behavior for avoiding collisions (`Avoid-Collision`). Note though that in the past or future, this intention may trigger many other activities. Of more interest is the state of execution for the `Deliver-Object` intention. At its topmost level, this parent intention has the single child intention `Plan-and-Execute`, which in turn is executing the `Follow-Path` schema while simultaneously monitoring for execution failures (via `Monitor-Planex`). As part of the path-following schema, the robot is currently moving from a corridor to a junction, which in turn has activated an intention to move toward a specific target. At the lowest level, three behaviors are activate simultaneously, namely `Follow`, `Orient`, and `Keep-Off-With-Target`.

PRS-Lite provides a powerful and natural framework in which to specify and manage the purposeful activities of a robot. The system itself is compact (<500 lines of LISP code, including executor, compiler, and display manager), especially in comparison to PRS-CL. It has proven to be sufficiently fast over a broad range of tasks, with the intention execution loop easily fitting into Saphira's overall cycle time of 100 ms. Precise figures for the cycle time of the PRS-Lite executor are not available, but the combination of behavior and task control lies somewhere in the range of 5 to 30 ms (on a Sparc-2 processor). This is very fast, considering that on average there are 10 to 15 intentions in operation, monitoring various conditions and coordinating behaviors.

In our experiences of writing over 50 activity schemas for a variety of different tasks, the expressiveness of PRS-Lite has proven to be mostly adequate. One omission from the language would be

[3] To assist in the understanding of the actions of the robot, PRS-Lite maintains an *intention display* that summarizes the intention structures for each cycle (as in Figure 4.2). The intention display provides a concise overview of the motivation for the actions being undertaken by the system at any point in time, thus conveying to an observer *why* the robot is behaving in a certain manner.

```
* Avoid (Avoid1) END
      * Avoid-Collision (Collide) B

* Deliver-Object (I3674) S171
      * Plan-And-Execute (I3675) S146
            o Monitor-Planex (I3676) CLEANUP
            * Follow-Path (Follow-Path) S138
                  * Corridor-To-Junction (I3677) END
                        * Follow-To-Target (I3678) END
                              o Follow (Follow-To-Target) B
                              o Orient (Orient-To-Target) B
                              o Keep-Off-With-Target (Keep-Off) B
```

Figure 4: Snapshot of the PRS-Lite Intention Structures during Execution of Delivery Task

the ability to suspend/continue execution of an intention. This would enable certain activities to be postponed in case more urgent matters arise. For example, consider the situation where some event occurs during the execution of a navigation plan that requires immediate attention for a brief period of time. It would be advantageous to suspend the path-following (and its associated behaviors) until the problem is addressed, then resume with the path from the current situation. In the current framework, the path-planning must be aborted and restarted once the previous problem is solved. (The only alternative is to account for all possible problems directly in the plan execution intention – an unlikely prospect.)

Several advantageous features found in PRS-CL were consciously omitted from PRS-Lite, due to concerns that their associated overhead might eclipse the limited processing time available for each perceive-act cycle of the system. Of these, two would be valuable extensions to PRS-Lite. The first would be a database to encode a declarative specification of the current world state. In the current PRS-Lite, all such state information is stored in Lisp variables. While this approach works, it is somewhat inelegant. A more significant change would be to add some limited deliberation to enable selection among multiple candidate activity schemas rather than direct dispatch of a designated schema. Given that there is unused cycle time, it would be interesting to modify the system to be more declarative in this manner.

In theory, a PRS-Lite style controller is dispensable: all behaviors could be in use at all times, with their levels of activation modified to reflect different goals. However, this approach is undesirable for two reasons. The first is understandability: PRS-Lite provides a modular, hierarchical framework in which both to specify and manage the organization of low-level behaviors for tasks, and to aid human operators in interpreting the actions of the system in terms of goal-oriented activity. The second reason is that without a runtime goal structure to provide a context for activity, behaviors may be activated for inappropriate reasons (as discussed further in [22]).

5 Coherence

Reactive behaviors such as obstacle avoidance often can take their input directly from sensor readings, perhaps with some transformation and filtering. More goal-directed behaviors can often benefit

from using *artifacts*, internal representations of objects or object configurations. This is especially true when sensors give only sporadic and uncertain information about the environment. For example, in following a corridor, a robot will not be able to sense the corridor with its side sonars when traversing open doorways or junctions. It would be foolish to suspend the behavior at this point, since over a small distance the robot's dead-reckoning is good enough to follow a "virtual corridor" until the opening is passed.

In other situations, an artifact may represent an artificial geometric entity that guides the behavior. Such situations occur frequently in human navigation, e.g., in crossing a street one tends to stay within a lane defined by the sidewalks on either side, even when there is no painted crosswalk. Similarly, in the follow-corridor behavior, the robot is guided by a lane artifact that is positioned a foot or so in from the corridor walls.

In accordance with these behavioral strategies, artifacts in Saphira come from three sources:

- From *a priori* information. Typically, the robot will start with a map of the corridors and offices in its environment.
- From perceptual features. When a perceptual process recognizes a new object, it may add that object to the list of artifacts.
- Indirectly, from other artifacts or goal information. For example, if the user gives the command, "Move 3 feet forward," a goal artifact is created at a position three feet in front of the robot.

Artifacts always have a class (WALL, CORRIDOR, POSITION, etc), geometric information about their position and extent, and a unique identity, so that no two artifacts in the same class are the same. They may also have information about geometrical dependencies with other artifacts.

Normally, artifacts in the LPS are updated based on the robot's dead-reckoning mechanism, which is reliable only over short distances. *Coherence* is the property of updating artifact positions in the LPS based on perception, so that the robot's model of the environment stays registered with the robot's position as it moves. To understand how this works, we refer to the following diagram:

$$\text{feature} \iff \text{object hypothesis} \iff \text{artifact}$$

Features are constructs based on sensor information, usually representing surface information. A typical feature would be a linear surface, represented by a straight segment in the LPS.

An *object hypothesis* is a set of features that could correspond to a real-world object. For example, two breaks in a linear surface could be a doorway, and these two features might be grouped together to form a doorway hypothesis. Object hypotheses do not have any particular identity, i.e., a doorway hypothesis does not have information that it is the doorway to a particular office.

All three types of representations — features, object hypotheses, and artifacts — coexist in the LPS. As the diagram implies, there is no strict relationship in how one is created or manipulated by another. Depending on the task, it is possible to go in several different directions. In map-building, for example, features are grouped into object hypotheses, which are then recognized as artifacts. Although this process is mostly bottom-up, there is also an important top-down component, in which previously-recognized artifacts are matched against current hypotheses, and only those hypotheses which refer to possible new objects are given the status of artifacts.

In practice, we have found that the introduction of artifacts greatly simplifies the design of behaviors, by allowing us to decouple the problem of control from the problems of interpreting noisy sensor data. Our behavior-writing methodology has been to first write small rulesets for elementary types of movements based on simple artifacts, like follow a line, or reach a location; and then focus

on the strategies to keep these artifacts anchored to the right features in the environment. The resulting behaviors often proved to be more robust than purely reactive controllers.

In this paper, we cannot describe all of the ways in which the feature to artifact connection is made. We will concentrate on two areas: feature extraction from perceptual information, and the process of *anchoring*, in which current artifacts are kept coherent with the environment by matching against features or object hypotheses.

5.1 Extracting Features

To navigate through extended regions, Saphira uses a global map that contains imprecise spatial knowledge of objects in the domain, especially walls, doorways, and junctions of corridors. Using a map depends on reliable extraction of object information from perceptual clues, and we (as well as others) have spent many frustrating years trying to produce object interpretations from highly uncertain sonar and stereo signatures (see, for example, [6, 16, 19]). The best method we have found is to use extended aperture sonars readings, perhaps augmented with depth information from the stereo system. As Flakey moves along, readings from the side sonars are accumulated as a series of points representing possible surfaces on the side of the robot. This gives some of the resolution of a sensor with a large aperture along the direction of motion. By running a robust linear feature algorithm over the data, we can find walls segments and doorways with some degree of confidence.

The utility of this technique is that it yields reliable linear features with almost no false positives. False positives are difficult to deal with, because using them for localization puts the robot at the wrong position in its internal map, and subsequent matches against features will fail. In general it's better to miss a few features and err on the conservative side in accepting features, since dead-reckoning works reliably for short periods.

Extracting wall and doorway features makes it easy to build a global map automatically, by having Flakey explore an area. The map is imprecise because there is error in the dead-reckoning system, and because the models for spatial objects are linear, e.g. corridors are represented as two parallel, straight lines. As features are constructed they can be combined into object hypotheses, matched against current artifacts, and promoted to new artifacts when they are not matched. In practice, we have been able to reliably construct a map of the corridors in the Artificial Intelligence Center, along with most of the doorways and junctions. Some hand editing of the map is necessary to add in doorways that were not found (because they were closed, or the robot was turning and missed them), and also to delete some doorway artifacts that were recognized because of odd combinations of obstacles.

5.2 Anchoring

Artifacts exist as internal representations of the environment. When the physical object that an artifact refers to is perceived by the sensors, we can use this information to update the position of the artifact with respect to the robot. This is necessary to guarantee that behavior using the artifact operate with respect to the actual object, rather than with respect to an *a priori* assumption. We call *anchoring* the process of (1) matching a feature or object hypothesis to an artifact, and (2) updating the artifact using this perceptual information (see [26] for more on anchoring).

In Saphira, the structure of decision-making for the anchoring problem takes the following form:

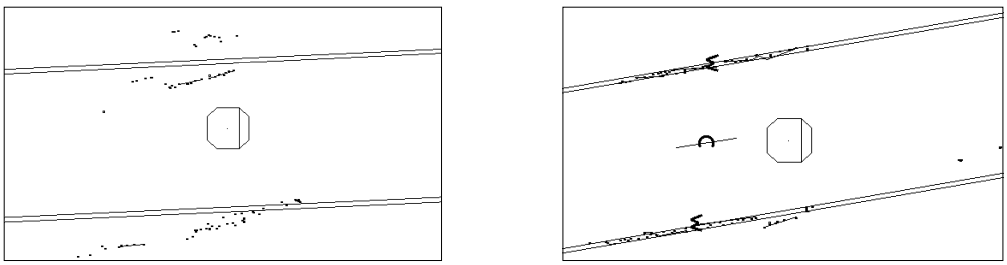
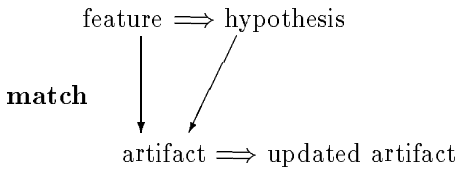


Figure 5: Anchoring a corridor artifact to sensor readings for corridor-following.



As features are perceived, Saphira attempts to convert them to object hypotheses, since these are more reliably matched than individual features. These hypotheses are matched against artifacts existing in the LPS. If they match against an artifact, the match produces information for updating (anchoring) the artifact's position. If not, they are candidates for inclusion as new artifacts in the map.

If an artifact that is in view of the perceptual apparatus cannot be matched against an object hypothesis, then Saphira tries to match it against individual perceptual features. This is useful, for example, when the robot is going down a hallway and trying to turn into a doorway. Only one end of the doorway is initially found because the other end is not in view of the side sonars. This information is enough to anchor the doorway artifact, and allow the robot to proceed with the door-traversing behavior.

Artifacts that have no perceptual support when they should (i.e., when they are in the range of the cameras or sonars), are candidates for reaping from the LPS. In general the reaping process must take into account the environment of the robot. For example, walls and corridors are fixed structures that are always present, while doors can be closed (and therefore not show up as features). The most obvious candidates for reaping are transient objects such as obstacles. At present we do not have any general theory of how to represent the dynamic aspect of objects and its relation to artifact recognition, but rely on special-purpose algorithms for each type of object.

Figure 5 shows an example of anchoring in Saphira. The picture shows the LPS in two consecutive moments during corridor following. The corridor-following behavior acts with respect to a corridor

artifact, represented by the two double lines.[4] This artifact is initially placed by the planning and executive levels based on map information (a). Note that the position of the artifact does not correspond to that of the actual wall, visible from the clusters of sonar readings (small dots): this may be because the map is incorrect or, more commonly, because of the inaccuracy of self-localization. As enough sonar readings are gathered, Saphira's perceptual routines infer the existence of two parallel walls (marked by "W") and hence form a corridor hypothesis ("C"). The artifact is then anchored to this hypothesis (b), and the movement now proceeds with respect to the actual corridor. If one wall is obscured by obstacles or a doorway, then the other will be used to anchor the corridor. Anchoring provides a closed-loop response whenever the relevant sensor data are available. When data are not available, e.g., if the walls become obscured, artifacts act as assumptions for movement.

Currently, anchoring is successfully used by individual behaviors or activity schemas that control behavior. For example, it is used by the doorway-traversing behavior to track the location of the doorway as it moves into or out of a room. It is also used to keep the robot globally registered with respect to a map. The registration process is sufficiently robust that the original map the robot makes does not have to be very precise (and it can't be, if dead-reckoning is poor). The anchoring process will keep the robot's position updated with respect to the relevant objects in the environment. This assumes, of course, that the robot can find and match objects of the right sort. For instance, going down a long corridor with no features, the robot will stay correctly registered in the center of the hallway, but the error in its longitudinal position will grow. So it's important to find doorways or breaks in the corridor at reasonable intervals. Using the anchoring strategy on corridors, doorways, and junctions, we have been able to keep the robot localized for arbitrarily long periods (i.e., until the batteries run low) as it navigates the corridors of the Artificial Intelligence Center.

Anchoring thus helps to correct uncertain prior information by using perception, and keep the robot localized on a map as it navigates. But there are still several problems we have not addressed with this scheme. One is that sonar percepts are, in general, not adequate to keep the robot localized in complex environments. For example, in crowded rooms, the robot quickly becomes confused because it cannot find a sufficient set of long linear segments to match. To solve this problem, we are considering correlation-based algorithms using the stereo vision system. By keeping track of a number of small surface patches that are sufficiently distinctive as features, we hope to be able to use the same registration algorithms to reliably localize the robot in complex indoor and outdoor environments.

A second problem is that of *place recognition*: localization when the robot has no knowledge of its current position. Our current ideas for solving this task are based on the use of fuzzy sets to represent locational uncertainty [30, 27].

5.3 Tracking people

Saphira incorporates a simple people-tracker based on stereo information. The tracking algorithm does not perform correlations on successive temporal frames to keep track of the object. Rather, it looks for person-like objects in each successive frame without reference to previous ones, formulates a hypothesis, and then passes it to the LPS registration routines. "Person-like objects" are detected by a matching process running on the stereo data. Three bands of range information at torso height are examined for a shape profile the width of an adult. If two of the three bands agree, then a person is detected at the center of the shape. Because we are only using a portion of the stereo information, we can run the person-tracker at a 10 Hz rate.

[4] It actually uses a lane artifact placed with respect to the corridor artifact, but we can ignore this complication for the purposes of the example.

The registration algorithm makes a decision about whether the hypothesized person is the same as one represented by an artifact, and either updates the artifact position or creates a new one. We keep a simple velocity model of person artifacts, which helps the registration process to make better matching decisions. It is also used by a *centering* process which can keep the camera center locked on the person being tracked. If a person artifact does not receive support for a period of time, it is removed from the LPS.

Interestingly, the registration process makes it possible to follow people around corners and through doorways, where they may be temporarily lost from sight. The artifact represents their most likely position, and the vision routines keep searching this area until the person is re-acquired, or cannot be found.

One limitation of the current system is its inability to distinguish different people. The range resolution of the system is coarse enough so that distinguishing people by shape is not possible, even without the complication of motion, arm position, and so on.

6 Communication

At this point, our ideas about communicating have been strongly influenced by the Natural Language Understanding community, especially the plan-based theory of speech acts [1]. Eventually, we would like to have a complete understanding system that would perform intention recognition on the speaker's utterance, and form appropriate plans. For the scenario, we concentrated on the area of referent identification, matching the speaker's terms to objects in the immediate environment of the robot. In this section we describe the speech input system, and the PRS schemas that carried out advice-taking and tasking commands.

6.1 Speech input

We were fortunate to have an excellent speech-recognition system, developed at SRI, called CORONA. CORONA has speaker-independent recognition models that need no training, although it will have better accuracy with individually-tuned models. CORONA accepts a BNF grammar of phrases, and produces strings of the words spoken. On a Sparc 10-51, it operates at about twice realtime on the simple grammar we used, i.e., a 5-second sentence would take about 10 seconds to recognize.

One of the hardest problems in voice communication is letting the supervisor know when Flakey has heard an utterance, and what the state of its understanding is. We employed several techniques:

Keying. There can be a lot of extraneous chatter during interactions. The grammar was created with a keyword initiation, so that only phrases beginning with the word "Flakey" were recognized. This worked extremely well; it was natural to address commands and statements to the robot in this fashion. A nice additional feature would be to use directional interpretation of sound sources for getting Flakey's attention; we currently do not have this capability.

Process status. CORONA employs an "endpointer," a period of low speech energy, to signal the end of a speaker phrase. From this point there is a delay until the speech is processed; the speaker can be confused about whether his input was received, and whether it was recognized. We implemented a simple end-of-speech flag: Flakey says "um" when the endpoint is reached. Thus the speaker can tell that his input was received and is being processed.

Results. It is important to give feedback to the speaker about the interpretation of his input. For example, if the speaker says "turn left," and it is interpreted as "go forward" (a not altogether unknown occurrence), the speaker will be mystified by the robot's behavior. So

Flakey would acknowledge each spoken command, either by paraphrasing it, or nodding its cameras in recognition. If the phrase was not recognized, Flakey said "duh?" or "what?" Crude, but effective.

6.2 Gestures

We were ambitious enough to want gesture recognition as part of the attention process. The idea was to use a mixture of speech and gesture for reference, e.g., "This office (*point to the right or left*) is Karen's." We did manage to extract enough information from the stereo system to recognize left or right-pointing arms, but did not have enough time to integrate it correctly with the speech input, even for simple reference. This would be a good project for future work, with a more elaborate natural-language understanding system.

We implemented a simple but surprisingly sufficient set of activity schemas to perform the scenario. They fall into several categories: direct motion commands ("turn around"), sensor-based movement ("follow me", "follow the corridor"), tasks ("get the file from Karen"), and information ("this is John's office").

6.3 Direct Motion

These are direct commands to move forward or turn, or to look (move the cameras) in certain directions, e.g., "look behind you." Direct motion commands are implemented as simple sequential behaviors with well-defined ending conditions and short timeouts. For example, if Flakey is told to move forward while facing a wall, it will turn away from the wall and move forward about 1 meter, since collision avoidance is active at all times. It would be smarter, of course, to recognize that it is not possible to move forward, and state this fact. But in our current implementation Flakey does not check for possibility of carrying out commands; it only tries to do the best it can.

6.4 Attending and Following

These commands involve coordination of sensing and acting, mediated by artifacts in the LPS. Even very simple activity schemas, when coordinated with sensing, can give the appearance of a well-trained robot with human-like capabilities.

The Attending schema finds the closest person-like object and brings Flakey to face the person at a distance of about 1 meter. The command "look at me" or "find me" triggers this schema. Flakey performs a scan from the current camera position to the extreme left and right position of the pan/tilt head, giving a full 360 degree field of view. The first person-like object detected by the stereo tracking algorithm (Section 5.3) is placed in the LPS, and an acitivity schema positions the robot to face the object and keep the cameras centered on it. Success of the Attending schema is indicated by saying "here I am;" failure, after a full scan, by "I can't find you!".

The Following schema uses the same kind of movement/sensor control to track a person. If there is no person currently in view, the Attending schema is invoked. Once found, the person is kept centered in the cameras, using the spatial information returned from the tracking algorithm. The activity schema keeps Flakey about 1 - 1.5 meters from the person as much as possible (top speed of 300mm/sec), while avoiding obstacles. If the person is lost, Flakey looks in the most likely area for awhile, but does not attempt to re-attend, since if there are other people around, it may acquire the wrong one. Flakey's vision system cannot distinguish individuals.

Both these behaviors worked well in practice. Mr. Alda was able to lead Flakey through several corridors and doorways at SRI, with only minor problems in re-acquisition.

6.5 Information

This is one of the most interesting areas for robot/human interaction. The hard problem is relating robot-centered representations of the world with human-centered ones. For example, when the supervisor says "..this office...", the robot must understand that there is an object in the current focus of dialogue that is being referred to. The point of contact, for the robot, is in the artifacts of the LPS. Flakey has enough background knowledge to infer that doors can lead to offices, even though it cannot recognize an office *per se*. So the phrase "this office" is linked to the nearest doorway in the robot's perceptual space.

In general the problem of reference resolution can be phrased in terms of abductive inference: find an object that, if assumed as the referent, would make the speaker's phrase carry reasonable information [12]. For the scenario demonstration, we did not use a general inference method to determine reference, since we decided beforehand that we would only refer to offices, people, and certain other objects. But in general the problem of reference for robots will require some of the same tools that have been used in computational linguistics. However, robots do enjoy one advantage: since they also perceive the world, they can draw on a repertoire of perceived objects as referents (in philosophical terms, they are *situated* in the word). Instead, the problem becomes one of matching the artifacts of the robot's representation with the linguistic expressions of the speaker. We have formulated a theoretical foundation for this interchange [20]; but it was not applied in the scenario. Other more complex theories of knowledge and action have been developed in a logical framework [18, 17]; we expect to see more application of these theories as robots become more sophisticated in their interactions.

Another type of information used by the robot is knowledge of how to locate people, since delivery tasks often involve finding someone. In general this is a planning problem, with reasoning about where someone is, or who might have information about where that person is. For the scenario, we implemented a simple routine that would first check the person's office (using speech output and waiting for a reply), and then start looking for people who might know where the person is. More sophisticated routines might check a personal calendar, or compile a list of likely places, or perform inference about knowledge prerequisites for action, and so on. Here again the problem of reference emerges. Phrases such as "Karen is around the corner" would be hard to interpret, so we settled on a set of place-names that could be used: offices, corridors, and open areas such as the library. These were all places that Flakey could learn from the supervisor, since there were artifacts (doors, corridors, junctions) that could be identified with the area.

One lesson we learned from attempting to give Flakey advice of this sort is that the closer a match between the robot's perceptual categories and the human's, the easier and more foolproof the exchange of information. For example, if Flakey had no concept of a corridor, it would be almost impossible to tell it the name of the corridor (e.g., "J-wing") and have it internalized in an effective way. Suppose Flakey were to store its current position in association with the corridor name. Then the command "Go to J-wing" would cause Flakey to go back to the same location, even if another location in the corridor were much closer. Commands such as "follow the corridor" wouldn't make any sense at all.

6.6 Tasking

A robot is supposed to perform useful tasks. For the scenario, Flakey was a delivery robot, whose main task was to deliver messages and manuscripts. While performing these tasks, Flakey maintained the heterostatic goals of obstacle avoidance and localization. These required no overt planning; PRS invoked the requisite behaviors automatically. In more complicated situations, for example where the robot must explore to find a new route, there might be explicit planning for localization.

Navigation plans were computed by graph search on the learned topological map, and then executed by PRS. More complicated procedures were constructed as conditional navigation plans: find person X, ask him/her where person Y is, and then go to that location. These plans were simple enough that we just created PRS schemata for them. A major failure in the plan (e.g., person X not found) caused PRS to instantiate an alternative schema. The most complicated plan Flakey executed was going to Karen's office and finding out she wasn't there; then going to John's office, asking him where Karen was, finding and getting a report from her, and returning to deliver it.

7 Discussion

There are two basic areas where our work on Saphira has differed in design and emphasis from others. The first is in the area of coherence: by mediating interacts between perception and action through the LPS, we have been able to abstract away some of the difficulties found in interpreting goal-directed behavior relative to the perceptual state of the robot. This is especially true of tasks that cannot rely on current perceptions to formulate correct action sequences. Typical here is the task of following a person around a corner. As the person turns the corner, the perceptual system can no longer track him: there is nothing in the perceptual space on which to base a "follow" behavior. By providing an artifact that represents the best estimate of the person's position, Saphira can still perform the follow behavior on the expectation of re-acquiring perceptual information after a short while.

The second distinctive aspect of our work in in the area of coordination. Context-dependent blending is a general methodology to form complex controllers by composing basic behaviors. Basic behaviors are simple functions on a local state written to satisfy a single goal over a small range of environments (the context). As such, they are relatively easy to write and to debug — they should also be easier to learn in the future. Complex behaviors to achieve multiple goals or operate over wider environmental conditions are composed out of simpler ones by using fuzzy context rules. This is in general an easy operation if the preconditions of the basic behaviors have been clearly stated. Interestingly, this compositional methodology can be formally analyzed using the tools of multivalued logics. In a related work [28], we have shown several properties that link behavior composition to goal decomposition: for example, we can prove, under certain assumption, that if two behaviors individually promote two goals, then their conjunctive blending promotes the conjoint goal in the conjoint context. Formal groundedness is an important feature of our compositional methodology to form complex behaviors.

Other forms of weighted combination of behaviors have been proposed in the literature, including the already discussed potential-fields methods, and Payton and Rosenblatt's architectures (see [23], and Rosenblatt's paper in this issue). Our approach, however, is peculiar in two respects. First, in context-depending blending we first combine behavior preferences, and then chose one summary control from the combination result. This contrasts with potential-field methods, which directly combine preference summaries, and resembles the combination schema used by Payton and Rosenblatt. However, Payton and Rosenblatt's schema is not grounded in any mathematical formalism, and thus has an ad-hoc flavor.

The second peculiarity of our approach is that we express behavior activation conditions through formulae in a (fuzzy) logical language. Complex conditions can often be described more easily in a logical form than in the analytical form of, say, a potential field function. Moreover, the ability to specify the context by a logical sentence makes integration of with symbolic planning easier: we have shown in [28] that our complex behaviors can be generated automatically by classical planning techniques.

Obviously, our approach also has its problems. Fuzzy behaviors implement a local "greedy" method, gradient descent, that is highly reactive and simple to compute. Like any local technique, however, fuzzy behaviors can be trapped in local minima. It is the responsibility of higher-level intentions to only instantiate behaviors for which gradient descent is appropriate based on some global analysis, or to monitor execution to detect local minima and failures. In practice, it is not always easy to find the right contextual condition for a behavior.

A related difficulty in our experience has been that tuning the parameters of the fuzzy rules may be difficult. Although the decomposition of complex behaviors into simpler ones makes them simpler to write, some behaviors required many days of experimental debugging. We are currently exploring the possibility to automatically synthesize basic behaviors from specifications; and the use of learning techniques to improve a behavior's performance.

Acknowledgments

Alessandro Saffiotti is supported by the BELON project, founded by the *Communauté Française de Belgique*. We thank the anonymous referees for many useful comments on a previous draft of this paper.

References

[1] J. F. Allen, Recognizing intentions from natural language utterances, in: *Computational Models of Discourse* (MIT Press, Cambridge, MA, 1983) 107–166.

[2] R. C. Arkin, Integrating behavioral, perceptual and world knowledge in reactive navigation, *Robotics and Autonomous Systems* **6** (1990) 105–122.

[3] R. A. Brooks, A layered intelligent control system for a mobile robot, in: *Proceedings of the IEEE Conference on Robotics and Automation* (1986) 14–23.

[4] J. Connell, *Minimalist Mobile Robotics: A Colony-style Architecture for an Artificial Creature* (Academic Press, 1990).

[5] J. Connell, SSS: A hybrid architecture applied to robot navigation, in: *Proceedings of the IEEE Conference on Robotics and Automation* (1992) 2719–2724.

[6] M. Drumheller, Mobile robot localization using sonar, A. I. Memo 826, Massachusetts Institute of Technology (1985).

[7] C. C. et. al., CARMEL versus FLAKEY: A comparision of two winners, *AI Magazine* **14** (1) (1993) 49–57.

[8] R. J. Firby, An investigation into reactive planning in complex domains, in: *Proceedings of the Conference of the American Association of Artificial Intelligence* (1987) 202–206.

[9] R. J. Firby, Task networks for controlling continuous processes, in: *Proceedings of the Second International Conference on AI Planning Systems* (AAAI Press, 1994).

[10] E. Gat, Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots, in: *Proceedings of the Conference of the American Association of Artificial Intelligence* (1992).

[11] M. P. Georgeff and F. F. Ingrand, Decision-making in an embedded reasoning system, in: *Proceedings of the Conference of the American Association of Artificial Intelligence*, Detroit, MI (1989) 972–978.

[12] J. R. Hobbs, M. Stickel, D. Appelt, and P. Martin, Interpretation as abduction", *Artificial Intelligence* **63** (1–2) (1993) 69–142.

[13] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, *The International Journal of Robotics Research* **5** (1) (1986) 90–98.

[14] K. Konolige, Erratic competes with the big boys, *AI Magazine* (Summer 1995).

[15] J. C. Latombe, *Robot Motion Planning* (Kluwer Academic Publishers, Boston, 1991).

[16] J. Leonard, H. Durrant-Whyte, and I. J. Cox, Dynamic map building for an autonomous mobile robot, in: *IROS* (1990) 89–95.

[17] Y. Lespérance and H. J. Levesque, Indexical knowledge and robot action — a logical account, *Artificial Intelligence* **73** (1–2) (February 1995).

[18] R. C. Moore, Reasoning about knowledge and action, PhD thesis, Massachusetts Institute of Technology, Cambridge, MA (1980).

[19] H. P. Moravec and A. E. Elfes, High resolution maps from wide angle sonar, in: *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, Washington, D. C. (1985) 116–121.

[20] K. Myers and K. Konolige, Reasoning with analogical representations, in: B. Nebel, C. Rich, and W. Swartout, eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*, San Mateo, CA (Morgan Kaufmann, 1992).

[21] K. L. Myers, User's guide for the procedural reasoning system, Technical note, SRI Artificial Intelligence Center, Menlo Park, California (1993).

[22] K. L. Myers, A procedural knowledge approach to task-level control, in: *Proceedings of the Third International Conference on AI Planning Systems* (AAAI Press, 1996).

[23] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey, Plan guided reaction, *IEEE Trans. on Systems, Man, and Cybernetics* **20** (6) (1990).

[24] S. J. Rosenschein, The synthesis of digital machines with provable epistemic properties, Technical Note 412, SRI Artificial Intelligence Center, Menlo Park, California (1987).

[25] E. H. Ruspini, Truth as utility: A conceptual synthesis, in: *Proceedings of the Conference on Uncertainty in AI*, Los Angeles, CA (1991).

[26] A. Saffiotti, Pick-up what?, in: C. Bäckström and E. Sandewall, eds., *Current Trends in AI Planning* (IOS Press, Amsterdam, Netherlands, 1994).

[27] A. Saffiotti, Using fuzzy logic for robot navigation, PhD thesis, Université Libre de Bruxelles, Brussels, Belgium (1996). In preparation.

[28] A. Saffiotti, K. Konolige, and E. H. Ruspini, A multivalued-logic approach to integrating planning and control, *Artificial Intelligence* **76** (1-2) (1995) 481–526.

[29] A. Saffiotti, E. H. Ruspini, and K. Konolige, Integrating reactivity and goal-directedness in a fuzzy controller, in: *Procs. of the 2nd Fuzzy-IEEE Conference*, San Francisco, CA (1993).

[30] A. Saffiotti and L. P. Wesley, Perception-based self-localization using fuzzy locations, in: *Procs. of the International Workshop on Reasoning with Uncertainty in Robotics*, University of Amsterdam (1995).

[31] R. Simmons, The 1994 aaai robot competition and exhibition, *AI Magazine* (Summer 1995).

[32] J. Yen and N. Pfluger, A fuzzy logic based robot navigation system, in: *Procs. of the AAAI Fall Symposium on Mobile Robot Navigation*, Boston, MA (1992) 195–199.

[33] R. Zabih and J. Woodfill, Non-parametric local transforms for computing visual correspondence, in: *3rd European Conf. Computer Vision*, Stockholm (1994).