Evolutionary Development of Mobile Robot Systems

Reid Simmons
School of Computer Science / Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
reids@cs.cmu.edu

Abstract

The development of any complex system is an iterative process. This is especially true for autonomous mobile robot systems that are designed to perform sophisticated tasks in unstructured and/or dynamic environments. The *structured-control* approach is presented, in which deliberative components that handle nominal situations are developed first, and then incrementally layered with reactive behaviors that handle exceptional situations. The Task Control Architecture facilitates the structured-control approach by providing control constructs for communication, planning, task execution, resource management, and error detection and recovery. One difficulty with the approach is that interactions between concurrent components can give rise to unpredictable behavior. We are beginning to address that by formalizing the control constructs in order to prove properties about the expected performance of robot systems.

Introduction

Three main factors influence the design of mobile robot systems: the tasks to be performed, the environment in which the system will operate, and the available hardware (computers, sensors, actuators). If the developer's knowledge regarding these factors is incomplete or uncertain, the designed robot system may not adequately perform its tasks when tested in the real world. This may be particularly apparent when mobile robot systems designed to handle a given set of tasks and environments are given new tasks to perform or are made to operate in different environments.

Incompleteness and uncertainty in design-time knowledge dictates an iterative development process: an initial system is designed according to the best available knowledge, the robot system is tested, failures are traced back to faulty design-time assumptions, and the design is modified to reflect the new, more accurate understanding of the specifications and requirements.

While this evolutionary process is inevitable, there are several ways to ease the process. First, the models of tasks, environments and hardware used to design the system can be made more explicit and more precise. This would tend to make the initial system more reliable, since the design knowledge would be more complete. For example, accurately modeling errors in the sensors and actuators and using algorithms that handle the inherent uncertainties can often lead to more reliable and competent systems (c.f. [Kosaka and Kak, 1992, Dean *et al.*, 1990]).

Another way to ease the incremental development process is to utilize techniques that help diagnose why a robot system failed to perform its tasks correctly. We anticipate that variants of model-based diagnostic techniques, developed in the domains of circuits and medicine [Davis and Hamscher, 1988], may be useful in pinpointing which changes need to be made.

A third method would be to provide tools that facilitate evolutionary design. The underlying system architecture would be structured so as to facilitate the upgrading of robot systems without requiring significant modifications to existing software. To support this incremental process, design verification techniques could be developed to help ensure that the complete system meet its specifications.

Structured Control

This paper focuses on the last of these methods: providing a system architecture and design verification tools to support evolutionary development. The approach advocated here, which we call *structured control*, involves first developing basic deliberative components that handle nominal situations, and then increasing reliability by incrementally layering on reactive behaviors to handle exceptions.

This layering of reactive behaviors on to a deliberative base provides an engineering basis for developing autonomous mobile robot systems. First, incomplete understanding of the tasks, environment or hardware is accommodated by separating the design into nominal, and presumably better understood, behaviors and the more numerous, but infrequently occurring, exceptional situations (which may become known and understood only after experience with the robot system). Second,

this separation of nominal and exceptional behaviors increases overall system understandability by isolating different concerns: the robot's behavior during normal operation is readily apparent, and strategies for handling exceptions can be developed separately and then added to the existing system with a minimum of effort. Finally, complex interactions are minimized by constraining the applicability of reactive behaviors to specific situations, so that only manageable, predictable subsets of the behaviors will be active at any one time (c.f. [Firby, 1992]).

The structured-control approach differs from the *behavior-based* approach, in which systems consist of collections of local behaviors that act according to direct sensing of the environment [Brooks, 1986, Connell, 1989, Gat, 1991]. The global behavior of such systems typically emerge from interactions between the local behaviors [Agre and Chapman, 1987, Brooks, 1989, Brooks, 1991]. A problem with the behavior-based approach is it assumes that robust primitive behaviors can be developed that act correctly in all, or most, situations. This can be very difficult in practice, given incomplete knowledge about the robot's hardware and environment. In contrast, the structured-control approach advocates developing complete components for limited environments, and incrementally updating the design to handle more challenging and diverse requirements.

The Task Control Architecture

To facilitate the structured-control approach, a developer should be able to add new reactive and/or deliberative components with little or no modification to existing software. The Task Control Architecture (TCA) has been developed to facilitate this process [Simmons, 1992a, Simmons, 1992b]. The term *task-level control* refers to the problem of coordinating perception, planning, and actuation to achieve a given set of goals. TCA provides a language for expressing task-level control decisions and software utilities for ensuring that those control choices are correctly realized by the robot.

In essence, TCA is a high-level robot operating system that provides an integrated set of commonly needed control constructs. The five major types of control constructs supported by TCA are:

- distributed communication
- task decomposition
- resource management
- execution monitoring
- exception handling

A system built using TCA consists of robot-specific modules that communicate by sending messages via a general-purpose *central control module*. The modules perform all task-dependent information processing, and the central control module is responsible for routing messages to modules that will handle them and for maintaining task control information. The modules specify control information by constraining the robot's behavior using the TCA control constructs. For example, a module can specify the order in which subtasks should be carried out or indicate when and how to monitor for exceptional conditions. TCA utilizes this control information, in turn, to schedule and coordinate the actions of other modules. TCA has been used to control several autonomous robot systems at CMU and elsewhere, in particular the Ambler six-legged planetary rover [Simmons and Krotkov, 1991] and an indoor mobile manipulator [Simmons *et al.*, 1990].

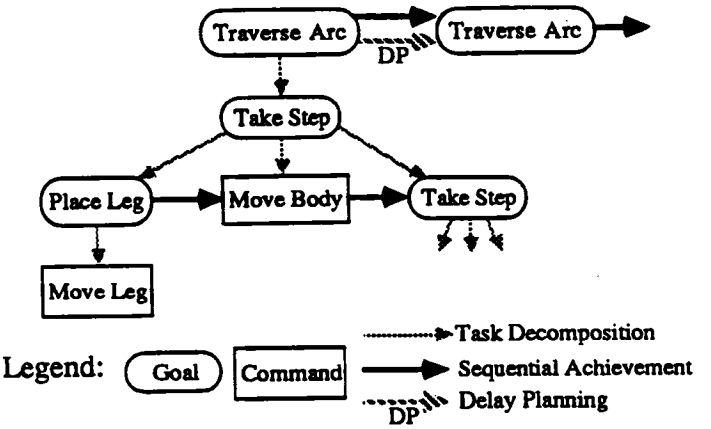

Figure 1: Task Tree for Ambler Walking

Central to the Task Control Architecture is the construction and maintenance of a hierarchical representation of subtasks called *task trees*. TCA constructs task trees dynamically: nodes in the task tree are associated with messages; when a message is issued by a module that is handling a message, a child is added under the node associated with the message being handled. In addition to defining this hierarchical goal/subgoal decomposition, modules can add *temporal constraints* between messages, indicating (partial) orderings on their execution.

Figure 1, for instance, illustrates a simplified version of the task tree for autonomously walking the Ambler rover. In the figure, narrow vertical arrows denote task decomposition and heavy horizontal arrows denote temporal constraints on task planning and execution. The task tree basically forms the walking "plan." It indicates that the Ambler sequentially traverses a series of arcs, where planning how to traverse one arc is delayed until the previous arc has been completely traversed. Traversing an arc consists of taking a sequence of steps, with each step consisting of a pair of leg and body moves. Unless the end of the arc has been reached, the planning module handling the "Take Step" message recursively issues another "Take Step" message. Note that the absence of a *delay planning* temporal constraint between the "Move Body" and subsequent "Take Step"

nodes indicates to TCA that planning one step can occur concurrently with the execution of the previous step. This use of concurrency enables the Ambler to achieve nearly continuous motion [Simmons, 1992a].

Incremental Development

TCA provides several mechanisms that directly support the structured-control approach. For one, exception handling strategies can be added incrementally without modifying existing components: a module can add control information to an existing task tree to indicate which procedure TCA should invoke in response to exceptions raised by other modules. The exception handling strategies typically deal with exceptions by modifying the currently executing plan.

For example, the Ambler real-time controller monitors force sensors in the feet and raises an exception when a threshold is exceeded (indicating unexpected terrain contact). A separate error recovery module handles this by modifying the current leg trajectory to surmount the obstacle, and then instructs TCA to reexecute the trajectory [Simmons, 1992b]. If one strategy fails to solve the problem, other strategies located higher in the task tree hierarchy are invoked until the exception has been dealt with successfully. For example, if modifying the leg trajectory fails to clear the obstacle, the complete move may be replanned, the Ambler's feet may be shuffled into a standard configuration, etc.

Similarly, execution monitors can be added incrementally using the TCA *wiretap* control construct. The wiretap mechanism enables a monitor to be associated with a class of messages, so that the monitor is automatically triggered whenever a message of that class is handled. For example, before every leg or body move the of Ambler, a stability monitor is invoked to verify that the move will not cause the robot to tip over; after every leg move a footfall monitor analyzes the force sensor data to detect possibly unstable footholds. These monitors were added after the basic walking component of the Ambler was designed and debugged, in order to enable the system to handle increasingly difficult terrain.

Concurrency in perception, planning and execution can also be added incrementally in TCA by adding and removing temporal constraints between task tree nodes and by managing access to robot resources (sensors, actuators and computers). Due to the difficulties in debugging concurrent systems, it is often desirable to start by designing and testing sequential systems, and then adding concurrency as performance requirements dictate. The Ambler system, for instance, was originally developed with a sequential sense-plan-act cycle: after sufficient testing, the temporal constraints of the walking plan (Figure 1) were changed to permit concurrent planning and execution [Simmons, 1992a]. Recently, TCA was used in making some of the perceptual processing concurrent as well [Hoffman and Krotkov, 1991].

Formal Models and Design Verification

A major difficulty with both the structured-control and behavior-based approaches is that as system complexity increases, interactions between the components increase as well, to the point where it becomes difficult to predict the system's overall behavior. This is where practice stops and theory must take over: analytical techniques are needed that enable developers to detect unwanted interactions at design time, rather than waiting until the problems manifest themselves during testing.

We are beginning to develop such tools for use with the Task Control Architecture. The first step is to formalize the TCA control constructs, so that their behavior and interactions can be analyzed. While the models will not capture robot-dependent interactions (such as a plan causing collisions between manipulator), there are many task-independent interactions that can be detected using such models. In particular, we anticipate they can be used to detect temporal inconsistencies in plan definitions, race conditions, deadlock, conditions under which exceptions may not be handled properly, non-termination of recursive or iterative plans, etc. Our goal is to embed the models into a design environment in which the various constraints on behavior can be displayed and manipulated graphically (c.f. [Erman *et al.*, 1988]), utilizing the analysis tools to verify the evolving system design.

Several formal models have been proposed that may be suitable for representing the Task Control Architecture. Petri Nets have been used to model (and coordinate) the execution of intelligent control systems, and to prove properties about the system, such as liveness and boundedness [Wang *et al.*, 1991, Cao and Sanderson, 1991, Kountouris and Stephanou, 1991]. Strengths of the Petri Net models are their intuitive, graphical correspondence to robot control tasks, their modular nature, and the well-developed theory that enables system-wide properties to be proven. The main disadvantage is that the language is restricted. In particular, it cannot easily handle dynamic topologies of components and is not well suited for sophisticated reasoning about time (although recently some temporal extensions to Petri Nets have been made [Freedman, 1991]).

Temporal logics are another class of formal models that have been applied to similar reasoning and verification tasks [Pratt, 1979, Clarke *et al.*, 1986, Lansky, 1987]. For our purposes, we desire a logic that can deal with both metric and relational (ordering) aspects of time. While modal temporal logics are concise languages for expressing temporal relationships, they have difficulties with encoding metric time information. On the other hand, languages such as the Duration Calculus [Chaochen *et al.*, 1991] deal well with metric information, but are somewhat awkward for representing purely relational aspects of time.

We are investigating a first-order temporal logic for representing and reasoning about TCA control constructs. The logic, based on the research of [Mc-

Dermott, 1982, Shoham, 1988, Dean and McDermott, 1987], represents time points explicitly and facilitates reasoning about both relational and metric time. The basic idea is that predicates such as *active(msg11)* are interpreted as terms denoting sets of time points (the times at which the associated predicate holds). Thus, *active(msg11)* represents the set of points (in this case a convex interval) during which message *msg11* is actively being handled. The predicate *Holds(p,t)* is syntactic sugar for $t \in p$, and is used to indicate that the "predicate" p holds at time point t. Thus, *Holds(active(msg11), t36)* indicates that *msg11* was being handled at time *t36*.[1]

In this logic, the standard logical connectives are interpreted using set-theoretic notations; for instance, $P \wedge Q$ holds exactly over the intersection of the time sets P and Q [McDermott, 1982]. The logic is sorted, with the sorts forming a type hierarchy. In particular, *message* is a type with different kinds of messages (*query, goal, command, monitor*) as subtypes. Under these are classes of messages, such as "Take Step" or "Move Leg."

Using this temporal logic, we have begun to formalize the notion of task trees and temporal constraints between task tree nodes. We are starting by formalizing task trees because they are so basic to TCA. We will then extend the model to include the other TCA control constructs (communication, resource management, monitoring, exception handling). We anticipate that modeling exception handling will be fairly difficult, since exception handlers may modify existing task trees. Thus, some form of non-monotonic reasoning may be needed [Shoham, 1988].

Several axioms are presented to give a flavor of this formalization. This is by no means a complete or final version, and is subject to major revision by the author at any moment.

1. After a message is sent by a module, it is added as a child of the task tree node associated with the message (p) that the module is handling.
   $\forall p, c : message; t, t1 : time$
   $Holds(send(p,c), t1) \wedge (t1 \leq t) \Rightarrow$
   $Holds(childOf(p,c), t)$
2. A message is *active* when it is being handled by a module. The time points *start(msg)* and *end(msg)* denote the ends of the interval during which the message is active (strictly speaking, the relational predicates, such as $\leq$, should be written using the *Holds* notation; for clarity, this is suppressed).
   $\forall m : message; t : time$
   $start(m) < end(m) \wedge$
   $start(m) \leq t \leq end(m) \Rightarrow Holds(active(m), t)$
3. A message is *completed* after it is active.
   $\forall m : message; t : time$
   $t > end(m) \Rightarrow Holds(completed(m), t)$
4. A message is *scheduable* when it is ready to be handled (it has been issued and any temporal constraints are met). Query messages are always scheduable. Goal, command and monitor messages are scheduable when the messages preceding them are completed.
   $\forall m : query; t, t1 : time$
   $Holds(send(m), t1) \wedge (t1 \leq t) \Rightarrow$
   $Holds(scheduable(m), t)$
   $\forall m, m1 : goal \cup command \cup monitor; t, t1 : time$
   $Holds(send(m), t1) \wedge (t1 \leq t) \wedge$
   $Holds(\sim precedes(m1, m) \vee completed(m1), t) \Rightarrow$
   $Holds(scheduable(m), t)$
5. One message *precedes* another when the first is constrained to be handled before the second can begin.
   $\forall m1, m2 : message; t : time$
   $Holds(precedes(m1, m2), t) \Leftrightarrow$
   $Holds(\leq (end(m1), start(m2)), t)$
6. We also define the concept of when a message is being achieved. The time points *ach.start(msg)* and *ach.end(msg)* denote the achievement interval. The idea of *achieving* is the time during which all the commands and monitors in the subtree rooted at the message are active (being executed). The exact semantics of *achieving* are detailed in the next several axioms.
   $\forall m : message; t : time$
   $ach.start(m) < ach.end(m) \wedge$
   $ach.start(m) \leq t \leq ach.end(m) \Rightarrow$
   $Holds(achieving(m), t)$
7. A command or monitor message is being achieved exactly when it is active. A goal messages can start being achieved after it is active, and continues being achieved at least until the message is completed.
   $\forall m : command \cup monitor$
   $start(m) = ach.start(m) \wedge end(m) = ach.end(m)$
   $\forall m : goal$
   $start(m) \leq ach.start(m) \wedge end(m) \leq ach.end(m)$
8. A message is being achieved at least while all of its children are being achieved.
   $\forall p, c : goal \cup command \cup monitor; t : time$
   $Holds(childOf(p,c), t) \Rightarrow$
   $Holds(\leq (ach.start(p), ach.start(c)), t) \wedge$
   $Holds(\leq (ach.end(c), ach.end(p)), t)$

The purpose of the formalization is to prove properties about the Task Control Architecture, in general, and about specific robot systems that use TCA. For example, the intent of the *sequential achievement* constraint is that all commands and monitors in the first subtree should precede any of those in the second (Figure 1). TCA implements sequential achievement using the constraint $ach.end(m1) \leq ach.start(m2)$. We outline a proof below showing that this does in fact implement the intended control construct.

More formally, we want to prove that
$\forall m1, m2 : goal, m3, m4 : command \cup monitor; t : time$
$Holds(SeqAch(m1, m2) \wedge$
$desc(m1, m3) \wedge desc(m2, m4), t) \Rightarrow$

[1] While it is probably cleaner to have *Holds* apply to intervals of time, rather than points, as [Shoham, 1988] does, we will use the simpler formulation here for illustrative purposes.

$Holds(precedes(m3, m4), t)$

where *desc* is the "descendant" relationship:

$Holds(desc(anc, msg), t) \Leftrightarrow$
$Holds(childOf(anc, msg), t) \vee$
$\exists m2[Holds(childOf(anc, m2) \wedge desc(m2, m), t)]$

From the definition of *precedes* (5), we need to show that $end(m3) \leq start(m4)$. From (7), it suffices to show that $ach.end(m3) \leq ach.start(m4)$, since both $m3$ and $m4$ are either command or monitor messages. To do this, we will show that $ach.end(m3) \leq ach.end(m1)$ and $ach.start(m2) \leq ach.start(m4)$. Then from the definition of the sequential achievement constraint and transitivity we get the desired proof.

To show $ach.end(m3) \leq ach.end(m1)$, we do a case analysis of the *desc* relation. If $m3$ is an immediate child of $m1$ (first disjunct), then the result follows immediately from (8). If $m3$ is a descendant of some immediate child (c) of $m1$ (second disjunct), then $ach.end(c) \leq ach.end(m1)$ from (8). By induction, $ach.end(m3) \leq ach.end(c)$, so by transitivity the desired "achieving" relationship holds between $m1$ and $m3$. A similar argument can be used to show $ach.start(m2) \leq ach.start(m4)$. Thus, the sequential achievement constraint in TCA achieves its desired purpose: ensuring that the command and monitor messages of one subtree completely precede those of another (note that this result does not hold for goal messages; this is intentional, in order to enable planning in advance of execution).

We can also prove properties about the performance of specific robot systems. For example, we could show that the temporal constraints described in [Simmons, 1992a] are sufficient to ensure that the Ambler walking system will plan at most one step in advance. To ensure that it will plan at least one step in advance, however, we need to utilize metric information about the relative durations of the various planning, sensing and actuation steps. The logic we are advocating is well suited for this purpose, and we intend to extend our methods to handle metric cases as well.

Conclusions

The structured-control approach provides a principled methodology for the evolutionary development of complex mobile robot systems. Due to uncertainties and incomplete knowledge of the robot's tasks, environment and hardware, an iterative approach is advocated in which components that handle nominal situations are developed and tested first, and then reactive behaviors that handle exceptional situations are incrementally added to increase safety, reliability and competence. The Task Control Architecture supports the structured-control approach by enabling reactive components to be added to robot systems with little or no modification to existing software.

While TCA and the structured-control approach have proven useful, in practice developing complex autonomous mobile robot systems is still a time-consuming and very heuristic process. We hope to go beyond the trial-and-error method of development by formalizing the TCA control constructs and by providing tools for analyzing designs formed from collections of interacting components. We have investigated several formalisms, including Petri Nets and temporal logics, and have settled (for now, at least) on a first-order temporal logic, due to its expressiveness in representing both relational and metric temporal information. While the formalization process is still preliminary, results to date are encouraging.

We are moving toward formalizing all the TCA control constructs. This will enable us to reason about such things as communication delays, resource contention and deadlock, expected system performance, the ability of the robot to detect and recover from errors, etc. While currently proofs of system properties are done by hand, we hope to automate some of the reasoning processes eventually. As mentioned earlier, formalizing the way task trees are modified during replanning is likely to be difficult, due to its non-monotonic nature. We need to find good ways of addressing this, however, since task tree modification is a critical aspect of the structured-control approach.

A crucial hypothesis in our approach is that robot systems can be constructed and analyzed as collections of interacting deliberative and reactive components. In addition, it is assumed that the TCA control constructs are sufficient for constraining the behavior of the systems to eliminate all undesirable interactions between components. We hope that experience developing additional TCA-based mobile robot systems and results from our formalization efforts will provide positive support of that hypothesis.

Acknowledgments

Many thanks to Chris Fedor who implemented much of TCA, and members of the Planetary Rover project who have developed the Ambler and its autonomous walking system. Thanks also to David Garlan for many useful ideas and discussions. This research is supported in part by NASA under Contract NAGW-1175 and by OSD under Contract F08635-92-C-0019.

References

[Agre and Chapman, 1987] Phil Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proc. National Conference on Artificial Intelligence*, pages 268–272, Seattle, WA, 1987.

[Brooks, 1986] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), 1986.

[Brooks, 1989] Rodney Brooks. A robot that walks: Emergent behavior from a carefully evolved network. *Neural Computation*, 1(2):253–262, Summer 1989.

[Brooks, 1991] Rodney Brooks. Intelligence without reason. In *Proc. International Joint Conference on AI*, Sydney, Australia, August 1991.

[Cao and Sanderson, 1991] Tiehua Cao and Arthur Sanderson. Automatic decomposition of assembly sequence plans. In *Proc. Third Annual Conference on Intelligent Robotic Systems for Space Exploration*, pages 19–28, RPI, Troy, NY, November 1991.

[Chaochen *et al.*, 1991] Zhou Chaochen, C.A.R. Hoare, and Anders Ravn. A calculus of durations. Technical report, ProCoS, February 1991.

[Clarke *et al.*, 1986] Edward Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[Connell, 1989] John Connell. A behavior-based arm controller. *IEEE Journal of Robotics and Automation*, 5(6):784–791, 1989.

[Davis and Hamscher, 1988] Randall Davis and Walter Hamscher. Model-based reasoning: Troubleshooting. In H. Shrobe, editor, *Exploring Artificial Intelligence*, chapter 8, pages 297–346. Morgan Kaufman, San Mateo, CA, 1988.

[Dean and McDermott, 1987] Tom Dean and Drew McDermott. Temporal data base management. *Artificial Intelligence*, 32(1):1–55, 1987.

[Dean *et al.*, 1990] Tom Dean, Kenneth Basye, Robert Chekaluk, Seungseok Hyun, Moises Lejter, and Margaret Randazza. Coping with uncertainty in a control system for navigation and exploration. In *Proc. National Conference on Artificial Intelligence*, pages 1010–1015, Boston, MA, 1990.

[Erman *et al.*, 1988] L. Erman, J. Lark, and F. Hayes-Roth. Abe: An environment for engineering intelligent systems. *IEEE Transactions on Software Engineering*, 14(12):1758–1770, December 1988.

[Firby, 1992] R. James Firby. Building symbolic primitives with continuous control routines. In *AI Planning Systems*, College Park, MD, June 1992.

[Freedman, 1991] Paul Freedman. Time, petri nets, and robotics. *IEEE Transactions on Robotics and Automation*, 7(4):417–433, August 1991.

[Gat, 1991] Erann Gat. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots*. PhD thesis, Jet Propulsion Laboratory, July 1991.

[Hoffman and Krotkov, 1991] Regis Hoffman and Eric Krotkov. Perception of rugged terrain for a walking robot: True confessions and new directions. In *Proc. of the International Workshop on Intelligent Robots and Systems*, pages 1505–1510, Osaka, Japan, November 1991.

[Kosaka and Kak, 1992] Akio Kosaka and Avi Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. In *Proc. IEEE International Workshop on Intelligent Robots and Systems*, pages 2177–2186, Raleigh, NC, July 1992.

[Kountouris and Stephanou, 1991] V. G. Kountouris and H. E. Stephanou. Dynamic modularization and synchronization for intelligent robot coordination: The concept of functional time-dependency. In *Proc. IEEE International Conference on Robotics and Automation*, pages 508–513, Sacramento, CA, April 1991.

[Lansky, 1987] Amy Lansky. A representation of parallel activity based on events, structure and causality. In M. Georgeoff and A. Lansky, editors, *Reasoning About Actions and Plans*, pages 123–160. Morgan Kaufmann, 1987.

[McDermott, 1982] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Pratt, 1979] V. R. Pratt. Process logic. In *Proc. 6th POPL*, pages 93–100, January 1979.

[Shoham, 1988] Yoav Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*. MIT Press, 1988.

[Simmons and Krotkov, 1991] Reid Simmons and Eric Krotkov. An integrated walking system for the Ambler planetary rover. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2086–2091, Sacramento, CA, April 1991.

[Simmons *et al.*, 1990] Reid Simmons, Long Ji Lin, and Chris Fedor. Autonomous task control for mobile robots. In *Proc. IEEE Symposium on Intelligent Control*, Philadelphia, PA, September 1990.

[Simmons, 1992a] Reid Simmons. Concurrent planning and execution for autonomous robots. *IEEE Control Systems*, 12(1):46–50, February 1992.

[Simmons, 1992b] Reid Simmons. Monitoring and error recovery for autonomous walking. In *Proc. IEEE International Workshop on Intelligent Robots and Systems*, pages 1407–1412, July 1992.

[Wang *et al.*, 1991] Fei Yue Wang, Konstantinos J. Kyriakopoulos, Anthanasios Tsolkas, and George N. Saridis. A petri-net coordination model for an intelligent mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):777–789, July/August 1991.