Linköping Electronic Articles in
Computer and Information Science
Vol. 5(2000): nr 17

Plan Execution, Monitoring, and Adaptation for Planetary Rovers

Richard Washington*
Keith Golden
John Bresina

NASA Ames Research Center
MS 269-2 Moffett Field, CA 94035
Telephone: (+1) 650-604-5000
Email: {richw | kgolden | bresina }@ptolemy.arc.nasa.gov

* RIACS contractor at NASA Ames

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/2000/017/
Revised version

Original version published on December 12, 2000
revised version on March 20, 2001 by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the authors using LaTeX
Formatted using étendu *style*

Recommended citation:
<Authors>. <Title>. Linköping Electronic Articles in
Computer and Information Science, Vol. 5(2000): nr 17.
http://www.ep.liu.se/ea/cis/2000/017/. *December 12, 2000.*

This URL will also contain a link to the authors' home pages.

Abstract

Planetary rovers must perform their missions in unknown environments with limited communication to ground controllers. To endow a rover with the capability for robust autonomous operation, we have designed an on-board executive architecture that incorporates robust flexible operation, monitoring of system and environmental state, and limited plan adaptation. The rover executive receives a plan with flexible time and resource constraints along with local and global contingency plans to handle deviations from the nominal plan. It internally monitors plan execution for communication and execution failures; through sensors and models of its operation, it determines its internal state, its resource usage, and its interaction with the environment. Based on the information it gathers from the sensors, it chooses the most appropriate course of action, potentially inserting contingency plans into its current plan, thus adapting its plan to fit the current situation.

Presented at the IJCAI'99 Workshop on Real-Time Monitoring.

1 Introduction

Unmanned exploration of distant planets will require rovers that are highly autonomous. The rovers will communicate infrequently with Earth, and the limited bandwidth and long distance will make teleoperation impractical if not impossible. In addition, the human effort required to control every move of a rover would be immense, and the command turnaround time would reduce the efficiency of the mission and thus the science return; for future Mars missions, the downtime for a rover with limited autonomy could reach 90% [Space Studies Board, 1999].

Traditionally, spacecraft have been controlled by time-stamped sequences of commands. The rigidity of this approach presents particular problems for rovers. Rovers interact with their environment in complex and unpredictable ways. Unlike a deep space probe, whose complexity largely lies within its own mechanisms, a rover's complexity arises from the interface with the external environment. Since the environment is unknown or poorly modeled, the rover's actions are highly uncertain. A difference in soil composition may make a movement take longer than expected, or it may make the wheels slip and leave the rover in an unexpected position. A dust storm might reduce the solar array output and affect the feasibility of the planned sequence. A rock's interesting chemical composition might become apparent only when the rover passes nearby.

The uncertainty of the rover's interaction with its environment makes rigid sequences impractical for efficient rover operation. To make best use of the rover's limited lifetime, the rover should be given instructions that allow it to handle the uncertainty by noticing and reacting to changes in the environment or its own state.

In this paper, we present a design for a rover executive as well as an instantiation of the architecture that was demonstrated in 1999 during a rover field test in the Mojave Desert. The architecture exhibits three advances over existing technology. First, the language accepted by the executive consists of flexible, conditional plans that can respond to the changing execution situation. Second, the executive monitors the system health and resource utilization. Third, the executive can adapt its plan based on monitored conditions, so the capabilities work together to optimize the performance of the rover at execution time.

Our on-board executive is designed to work in concert with ground tools for generating and verifying rover plans. Because of limited computational resources available on current and near-term planetary rovers, we have concentrated on safe, predictable, and efficient operation. Thus the techniques and algorithms we employ on board are limited with respect to optimal and complete techniques that may be more appropriately used by ground tools. In particular, as opposed to many architectures for autonomous systems, we have no on-board generative planner: the computational overhead is impractical for current rovers, and the highly dynamic environment poses problems for a combined planner-executive that exceed the state of the art. Instead, we have developed plan-modification algorithms that are more closely tailored to the problems and resources of a rover.

This work was inspired in part by work on the Remote Agent (RA), an integrated agent architecture developed for spacecraft control and deployed as an experiment on the Deep Space One mission [Bernard *et al.*, 1998, Muscettola *et al.*, 1998]. The rover executive architecture demonstrates advances in conditional execution and resource management compared to the RA executive. The language of the RA executive does not accept conditional

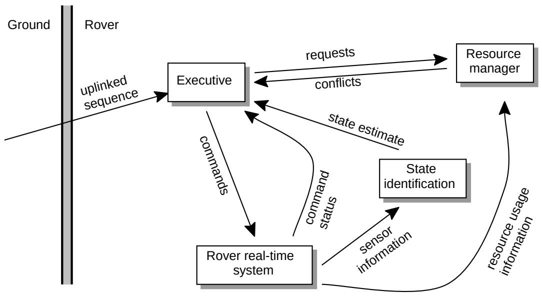

Figure 1: Rover autonomy architecture

sequences, which are critical to the success and effectiveness of a rover mission, given the highly variable interactions of the rover and the environment. Resource management in the RA executive is limited to discrete values of state variables that must be achieved and maintained for a task to be executed (for example, the camera power must be on to take a picture); in contrast, critical rover resources such as power and data storage are aggregate resources, rather than all-or-none. In the RA, the on-board planning and scheduling component reasons about aggregate resources, but not the plan execution component. The instantiation of the rover executive architecture presented in this paper does not, however, attempt to reproduce all of the capabilities of the RA; in particular, multiple concurrent activities, model-based state reconfiguration, and run-time resource selection for state variables are not currently included in our executive.

The remainder of the paper discusses the general rover executive architecture, followed by a more detailed description of the 1999 rover field test architecture instantiation and its capabilities for flexible plan execution, monitoring, and adaptation capabilities. Then this work is compared to related work in rover and robot autonomy. Finally, we present the current work on the rover architecture and how it is evolving.

2 Architecture Overview

The rover autonomy architecture is specifically designed to accommodate the particular constraints of planetary rovers while supporting autonomous operations. The complete rover autonomy system consists of the on-board executive architecture along with interactive ground tools for scientists and rover operators to construct a plan [Washington *et al.*, 1999, Bresina *et al.*, 1999]. Here we concentrate on the on-board component. The executive architecture consists of three major reasoning components (see Figure 1): a conditional executive (CX), a resource manager (RM), and a model-based state identification system (SI).

A contingent schedule is constructed on ground by scientists and rover operators working interactively with an automatic planning and scheduling system. The schedule is then uplinked to the on-board executive CX. These commands are sent to the real-time control system, with results coming back via state monitors into SI. SI infers the system state from the monitored information and updates the state estimate for CX. RM receives information about planned resource usage from CX and information about available resources from the real-time control system, and it signals potential or actual resource conflicts. CX notices when the execution deviates from the expected plan: in particular, when commands fail, schedule or resource constraints are violated, or science opportunities present themselves. CX adapts the rover's execution behavior using contingency plans.

Consider a plan for the rover to drive a predefined route while looking for rocks. This can be expressed as a number of short-duration driving and turning actions interleaved with actions to scan for rocks. On-board image analysis can indicate the potential presence of rocks, and if any are found, further actions can be performed, such as high-resolution images and spectrometry measurements. The potential presence of rocks is thus a contingency for opportunistic science. We may add constraints that certain waypoints in the motion should be reached by a given time, so that if time is too short, some on-board analyses will be omitted in favor of simply driving to the desired waypoint. Thus time constraints give rise to yet another type of contingency within the same plan.

2.1 Contingency Plans

Throughout a mission, detailed mission operations plans must be constructed, validated, and uplinked to a rover. In current practice, a mission operations plan takes the form of a rigid, time-stamped sequence of low-level commands. Unfortunately, there is uncertainty about many aspects of task execution: exactly how long operations will take, how much power will be consumed, and how much data storage will be needed. Furthermore, there is uncertainty about environmental factors that influence such things as rate of battery charging or which scientific tasks are possible. In order to allow for this uncertainty, current plans are based on worst-case estimates and contain fail-safe checks. If a task takes less time than expected, the spacecraft or rover just waits for the next time-stamped task. If a task takes longer than expected, it may be terminated before completion, potentially halting all non-essential operations until a new command plan is received. Either of these situations results in unnecessary delays and lost science opportunities.

To express plans that will handle these sources of execution uncertainty, we have designed the Contingent Rover Language (CRL), a flexible, conditional sequence language. CRL expresses time constraints and state constraints on the plan, but allows flexibility in the precise time that actions must be executed. Constraints include conditions that must hold before, during, and after actions are executed; failure of these conditions leads to an execution failure and potential plan adaptation.

The full command plan uplinked to the on-board executive contains a primary plan with contingent branches to handle potential problem points or opportunities in execution. In addition, the command plan may contain a library of *alternate plans* that specify a range of behaviors from general recovery plans to opportunistic science gathering.

The basic element of a plan is a *node*, which may be a single action, a choice point, or a composite action that is implemented by a CRL subplan. This hierarchical structure allows constraints to be placed on high-level actions as well as individual actions.

CRL describes the primary plan as a nominal sequence and a set of contingent branches (see Figure 2). The nominal sequence is the sequence that will be executed if there are no deviations from the *a priori* expectations of the environment and actions. The contingent branches specify alternative courses of action. Within any contingent branch there may be further contingent branches, so a plan is a tree of alternative courses of action. At this

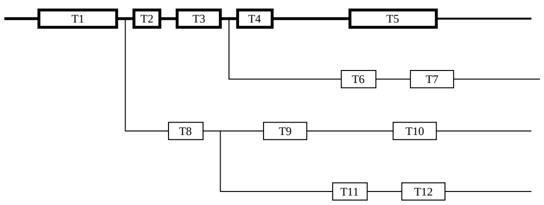

Figure 2: A plan with a nominal sequence (in bold) and a set of contingent branches.

time, CRL and the executive do not support concurrent activities, so there is a single path of execution through the plan structure. This is sufficient for current planetary rover operations, but as rovers grow in complexity, there may be an opportunity and need for concurrent action execution.

Alternate plans are applicable throughout execution rather than at specific points in the plan. When an alternate plan's eligibility conditions are satisfied, it is deemed enabled and will be considered as an alternative to the currently executing plan.

3 Architecture Instantiation — 1999 Rover Field Test

The general architecture described above was implemented for a rover field test in February 1999. In this section we describe in more detail the architecture components and capabilities as they were realized for the field test. We will refer to this specific version as the *1999 instantiation* in the remainder of the paper.

3.1 Flexible Plan Execution

The conditional executive (CX) is responsible for interpreting the command plan coming from ground control, checking run-time resource requirements and availability, monitoring plan execution, and potentially selecting alternative plan branches if the situation changes.

The input to CX consists of the primary plan and a set of alternate plans in CRL, as described above. CX executes each node while verifying that the preconditions, maintenance conditions, and end conditions are respected. CX can be instructed to wait for a subset of the preconditions (for example, the start time window) rather than failing the execution of the node. CX receives state information from the state identification module SI and resource information from the resource manager RM. It uses this information to check preconditions and maintenance conditions, as well as to check the eligibility conditions of the plans in the alternate plan library.

At each point in time, CX may have multiple options, corresponding to the eligible options of a branch point and the enabled alternate plans. CX chooses the option with the highest estimated expected utility, computed over the remainder of the plan. In the 1999 instantiation, the utility of successfully completing an atomic action is fixed and set by operators on the ground. From this atomic utility and a model of the probabilities of various events (such as a traverse taking longer than anticipated), the expected utility of an entire branching plan can be calculated.

When plan execution fails, CX reacts as specified in the node, either ignoring the action or aborting the execution and checking for applicable alternate plans. If execution is aborted and no alternate plans apply, CX aborts the entire plan set and puts the rover into a stable standby mode; all operation is suspended and the rover awaits further plans from ground control.

3.2 Monitoring

3.2.1 Command Status Monitoring

The most basic level of monitoring within the executive is command status monitoring. CX continually monitors messages from the real-time rover control system to confirm that a command has been received, that a command has succeeded, or that a command has failed. This information is used directly by CX to advance through its plan.

In the 1999 instantiation, communication between CX and the real-time rover system is based on a datagram model to avoid blocking the underlying real-time system, but that model does not ensure packet receipt. Thus, one internal execution fault that CX monitors for and reacts to is a communication failure between CX and the real-time rover system. CX monitors the command status messages, and if it appears that the command has not been received, it will resend the identical message. The rover control system ignores duplicate commands; unique IDs are used to avoid confusion between the cases of two instances of the same action (e.g., move 10 meters twice) and a resend of the same action.

3.2.2 Model-based State Identification

State identification involves interpreting the sensor information to determine the best estimate of the current state of the rover and the environment. We use a model-based approach that attempts to resolve the sensor information with an internal model of rover operation. Inconsistencies between the model and the sensor information lead to the consideration of anomalous (or fault) modes.

The approach that Pathfinder's Sojourner [Mishkin *et al.*, 1998] rover used to respond to faults is fairly typical of most fault-protection systems: monitors on specific sensors check whether given limits have been exceeded. Such monitors look at pitch and roll sensors, accelerometers, and the product of current and time in motors. This approach has a number of limitations.

- The appropriate thresholds for a given sensor depend greatly on context, not the least of which is the health condition of the sensor itself. Wheel motors will draw more current when the rover is going uphill, for instance.

- Rover engineers must trade the cost of false alarms against the risk of not responding to failures in time to prevent damage. On Sojourner, false alarms were quite common, resulting in lost science opportunities.

- Fault protection systems that rely only on monitoring individual sensors cannot cope well with multiple faults, and they are limited in the number of faults they can detect.

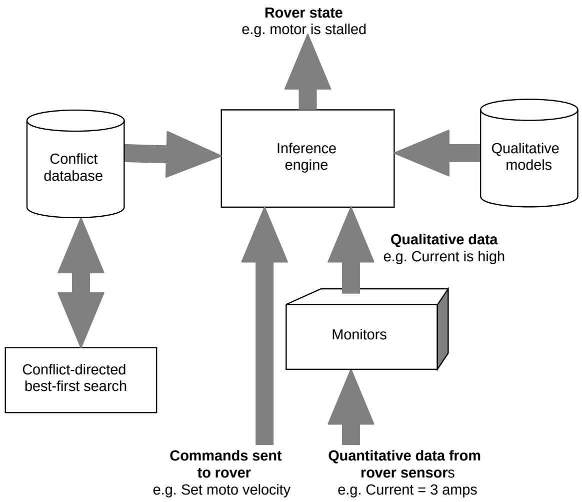

Figure 3: Information flow in SI.

- Simply halting and calling home for help is unnecessarily conservative for many common faults. Often, the rover could recover from the failure and keep going or else perform actions that do not depend on (or affect) the damaged component.

To address these problems, the rover autonomy architecture makes use of diagnosis, using all available sensor data to infer whether the rover is behaving correctly and, in some cases, to infer the specific fault. When it is possible to infer the fault, and that fault is recoverable, the rover can execute the appropriate recovery plan; otherwise, it can always stop and call home for help. Diagnosis, taking into account the complete state of the rover, allows faults to be detected earlier and also reduces the number of false alarms.

In the 1999 instantiation, we use Livingstone[Williams and Nayak, 1996] for the state identification (SI) component of the rover autonomy architecture. Livingstone was used as part of the Remote Agent Experiment on board the DS1 spacecraft [Bernard *et al.*, 1998].

SI eavesdrops on commands sent by CX to the rover. As each command is executed, SI receives observations from low-level monitors, which extract qualitative information from the rover sensors. For example, a current monitor may map the continuous-valued current into the set of qualitative values {low, nominal, high}. SI is informed whenever there is a change in the qualitative value returned by a monitor.

Based on monitor inputs, the commands executed on the rover, and a declarative model of the rover, SI infers the most likely current state (see Figure 3). SI also provides a layer of abstraction to the executive, allowing plans to be specified in terms of component modes, rather than in terms of low-level sensor values.

Livingstone uses algorithms adapted from model-based diagnosis [de Kleer and Williams, 1987, de Kleer and Williams, 1989] to provide the above functions. Livingstone extends the basic ideas of model-based diagnosis by modeling each component as a finite state machine (see Figure 4) and modeling the whole rover as a set of concurrent, synchronous state machines.

States in the state machines represent either "nominal" or "fault" modes, and transitions can be either default transitions or failure transitions. Fail-

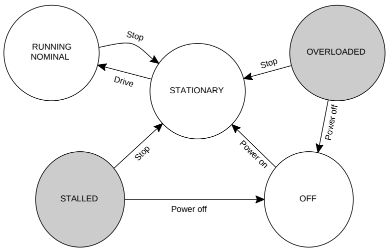

Figure 4: State diagram for motor control component. Shaded nodes represent fault modes. In addition to the explicit transitions, indicated by arrows, there are also implicit, random transitions to fault nodes, represented in the model probabilistically.

ure transitions are not explicitly represented in the user-defined models. Rather, it is assumed that any transition is possible, with a probability that depends only on the target state of the transition. During the course of rover operation, commands and sensors are monitored. The fact that those commands were executed is asserted, which causes the state machines that comprise the model to follow the default transitions corresponding to the commands that were issued. The observed sensor values are then asserted, and checked for consistency against the predicted sensor readings. If the model is consistent, then it is assumed that the actual state of the system is the predicted one.

How does Livingstone predict sensor values? Each component can have any number of properties, some of which are observable (i.e., determined by monitors), such as encoder values or ammeter readings, and some of which are not, such as velocities and torques. Each mode of a component specifies constraints on the properties of the component. These constraints can be stated in terms of properties of other components. For example, the output of an ammeter can be stated in terms of the current of a motor. When the ammeter is in the state WORKING, the two values will be the same. If the ammeter is WORKING and the motor is RUNNING-NOMINAL, then a monitor value of zero for the ammeter will result in an inconsistent model; the motor cannot have zero current when it is running, so either the ammeter must be broken or the motor must not be running.

When such an inconsistency is reached, Livingstone searches for the most probable set of state transitions consistent with the observations. The transitions of each state machine are assumed to be independent, so multiple low-probability failures will be assigned very low probability. For example, if a wheel encoder suddenly indicates that the corresponding wheel is not moving, the most likely explanation may be an encoder failure. If the ammeter also indicates that the wheel motor is not drawing current, then it is possible that both the encoder and the ammeter have failed, but more likely that the wheel motor has stopped running.

3.2.3 Resource Management

Resources on rovers are severely limited and, at the same time, critical for mission success. Solar energy is the primary source of power, but downlink events may be scheduled when there is little or no sunlight, so the power must be managed such that the on-board battery is sufficiently charged to communicate with ground. There are more opportunities to take pictures and instrument measurements than there is data storage on board or com-

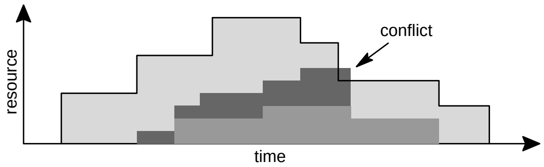

Figure 5: Resource management timelines. Resource availability is shown in the lighter shade, usage in darker shades. Conflicts occur where actual or predicted usage exceeds availability.

munication bandwidth, thus storage needs to be managed so that the most important data are stored and sent back.

Resources cannot be estimated precisely in advance; only during the execution of the command plan will it become clear how much of each resource is indeed available. If a traverse completes quickly because of unexpectedly easy terrain or good traction, more tasks may be possible with the surplus time and power. Conversely, if the rover traverses a ridge, slanted away from the sun, the accumulated energy may be insufficient to run all the planned experiments and also communicate with ground. The rover would then have to discard some of the experiments to reserve energy for communication.

In the 1999 instantiation, the resource manager is an on-board, run-time component that receives estimated resource profile information from tasks, monitors current and planned resource usage, and reacts to changes in resource availability. The resource manager is largely transparent to plan writers, while allowing them to take full advantage of the resources available.

The resource manager, RM, in the rover autonomy architecture communicates with the sequence execution component, CX. Each step in the nominal sequence has an expected resource profile associated with it. CX sends the expected profiles to RM, which records them and checks for conflicts. Any conflicts are signaled to CX.

As the plan is executing, the estimates of resource usage and availability are updated. Resource monitors gather the information about the real usage and availability and send that resource information to RM. Based on this new information, conflicts or opportunities may arise, which RM will in turn signal to CX.

RM stores resource profiles using a timeline-based representation. Each timeline is a set of disjoint time intervals over which a constant amount of the resource is used (see Figure 5). The predicted resource availability is produced from system resource information and models of resource availability over time (for example, solar flux models). Each resource request includes a profile of predicted resource usage. The sum of all the granted resource requests must be less than the resource available at all time points.

A conflict will arise whenever the requests exceed the resource available, either because the availability changes or because of a new request. RM sends conflict information to CX. CX in turn can respond in a variety of ways, depending on the severity and immediacy of the conflict. CX can fail the plan, select an alternate plan from its plan library, or ignore the conflict (for example, in the case that it is a conflict in the distant future). The reponse of CX depends on the conditions expressed in the CRL plan, as well as the availability of applicable alternate plans.

3.3 Plan Adaptation

The rover executive architecture supports a limited form of plan adaptation. In addition to the contingent branches within the plan, the plan may be augmented with *alternate plans* that may be substituted for the current plan or inserted into the current plan. Although this is much more limited than a full on-board replanning capability, it supports failure recovery and opportunistic science. The operational bounds of the plan adaptation remain under the control of ground operators, who specify the contents of the alternate plan library.

Although the alternate plan capability was implemented and available in the 1999 instantiation, issues of semantics and correct execution have limited its use (see Section 3.3.1). We are continuing to explore this capability (see Section 5).

Alternate plans are not attached to particular points in the primary plan; rather, they are applicable at any time their conditions are satisfied. Alternate plans can be viewed as global contingent branches, whereas the contingent branches within a plan are local to their position in the plan. Enabling events can include unexpected opportunities, plan failures, or conditions such as resource shortfalls and component degradation.

Currently, there are three subtypes of alternate plans, based on when they are considered as alternatives to the currently executing plan. Recall that a plan is built from nodes. As execution traverses the nodes, the alternate plans become eligible according to their type:

Node transition: Plans that are eligible between nodes. These behave exactly like branches in the primary plan, except that they are considered as an alternative at each node.

Node failure: Plans that are eligible when execution fails. These plans are meant to provide local failure recovery or overall backup plans. When an alternate plan is inserted into the current plan, it may remove the fault that caused node failure. If the anomalous condition remains, the node failure is propagated and handled within the primary plan, aborting parts or all of the primary plan.

When an alternate plan replaces the suffix of the current plan, the current plan is discarded. For node failures, this is intended for backup plans. For example, taking pictures to diagnose a fault condition, then downlinking them, would save the communication cycle that would be used to gather the diagnostic information.

Always: Plans that are eligible during an action's execution. These plans are appropriate for reacting to conditions that require interrupting the current action. For example, an interesting science opportunity discovered during a long traverse could be handled in this manner.

Some fault conditions can be handled by this method to avoid plan failure. For example, a degradation of solar power because of dust may trigger a dust-removal sequence before the power drops to a level where the plan will abort. The ability to monitor conditions continually to anticipate and avoid problems is important to achieve high productivity.

3.3.1 Semantic issues

The ability to insert plan fragments into the current plan and replace the current plan suffixes allows a wide range of plan modification, from local failure recovery to opportunistic science. This brings with it the danger of introducing undesired behaviors if the semantics of alternate plans are not carefully specified and implemented.

One potential problem arises from the fact that alternate plans could apply to failures within their execution. This is both powerful and dangerous, in that it could lead to uncontrolled and infinite recursion. We currently restrict alternate plans so that they will not apply within the context of their own execution.

One capability that we would like to support is to allow an action to continue execution after a local failure recovery removes the fault condition. Normally a node failure terminates the immediate action, at which point the error is caught and handled. Consider, however, the case of a long traverse with a momentary fault that is corrected. The ideal course of action would be to restart the traverse from the state where it broke. Allowing arbitrary action restarts could raise problems, however. Some actions may need to be restarted from the beginning (e.g., measurements that need to be contiguous in time), some may be continued from where the fault occurred (e.g., movement), and others may need some amount of resetting (e.g., data transfers). Because there is no single correct restart mechanism, we currently require that this be encoded into the failure plan.

3.4 Results

The rover autonomy architecture was demonstrated as part of a February 1999 field test [Bresina *et al.*, 1999] using the NASA Ames Marsokhod rover [Christian *et al.*, 1997]. The field test was designed to let scientists work in a mission-realistic setting; at the same time, it was designed to expose scientists to new technology that is intended for use in near-term missions and to give them a chance to interact with these technologies in a mission setting. The role of the rover autonomy architecture was to enable reliable operation of the rover during the field test and to illustrate the contribution of the rover autonomy elements to the science productivity of the rover.

Although the operational constraints of the field test limited the technology demonstrations that were possible, we successfully ran command plans with over a thousand commands, we demonstrated the usefulness of contingency plans, and we garnered interest in the possibility of on-board science processing enabled by the new technology. The Graduate Student on Mars (GSOM) project at NASA Ames [Gulick *et al.*, 1999], in particular, was interested in deploying their software for opportunistic use, and we began a continuing collaboration with the GSOM project during the field test effort.

Further outdoor and simulated tests have demonstrated the range of possible behaviors that the new technology enables. The rock-search plan described in Section 2 exhibits the potential utility of the GSOM opportunistic science tools as well as time-based contingencies. This scenario has been run on the actual rover in a test "sandbox," where both types of contingencies were exercised.

Another scenario, tested with a realistic simulation [Sweet *et al.*, 1999], demonstrates the ability of the contingencies to recover from failures during traverses. Lacking obstacle avoidance, the Marsokhod (and its simulation) will climb over rocks rather than driving around them. We constructed a plan to drive to a given point while monitoring the rover state for anomalous chassis positions that would indicate large and potentially dangerous obstacles. For rover safety we have fully exercised this only in simulation, where the rover can react to a dangerous obstacle by backing up, moving to the side, and continuing towards its goal (potentially repeating this procedure if it again encounters the same or a different obstacle). The same principle was used during the field test, but the rover operators avoided obviously dangerous situations, so the contingency was never exercised.

4 Related Work

4.1 Mission-ready rovers

Examples of mission-ready rover technology are scarce. The primary source is the Mars rover development program at JPL, which produced the 1997 Sojourner rover. Sojourner is the only deployed planetary rover to demonstrate a significant level of autonomous operations [Mishkin *et al.*, 1998]. Sojourner operated in a semi-autonomous mode, with periodic communication with Earth via the Pathfinder lander. Sojourner weighed about 23 pounds, operated for 83 sols (Martian days, which are 37 minutes longer than days on Earth), traveled 100 meters, performed 16 experiments on soil and rocks, and transmitted 550 images. Earlier, the Soviet space program produced two Lunokhod lunar rovers that were teleoperated from Earth in 1970 and 1973. The level of autonomy was basic, limited to stopping in response to excessive tilt, wheel blockage, or motor overheating.

NASA and the European Space Agency (ESA) have published mission designs calling for autonomous Mars rovers [Smith and Matijevic, 1989, Giralt and Boissier, 1992]. The space agencies recognize the necessity of integrated architectures that support autonomous operations for effective missions [Hayati and Arvidson, 1997]. More recent work on architectures for future rover missions [Volpe *et al.*, 2001] has included autonomous capabilities as an integral part of the overall architecture.

4.2 Research on outdoor robots

A number of rovers have been built for Earth-based missions or to test research ideas on realistic platforms. Much of the work on outdoor robotics has focused on advances in the robotic hardware necessary to operate in the unstructured, difficult environment that an outdoor setting presents. For example, the main contribution of the Ambler robot is a novel form of leg-based locomotion [Bares *et al.*, 1989]. That work is complementary to the work in this paper, which concentrates on the software architecture needed for robust autonomous operations.

Many outdoor robots are designed to be teleoperated [Christian *et al.*, 1997, Wettergreen *et al.*, 1993, Bapna *et al.*, 1998], concentrating on autonomous navigation to a waypoint while avoiding obstacles. Performance of these robots is impressive in terms of distance traveled [Wettergreen *et al.*, 1993] and environmental hazards overcome [Bapna *et al.*, 1998], but the robots did not have to exhibit robust behavior with respect to science goal achievement.

The ADS autonomous land vehicle [Linden and Glicksman, 1987] generated contingent plans over uncertain terrain. The work was restricted to path planning, and a prior map of the environment was assumed to exist. Recent work by the Carnegie Mellon University Robotics Institute has demonstrated automated path planning and navigation in unknown outdoor environments [Singh *et al.*, 2000].

4.3 Research on indoor robots

In contrast to deployed missions and even outdoor robots, results from research labs on indoor robots would suggest that mobile rovers can operate reliably and autonomously over a wide range of conditions, demonstrating impressive abilities for navigation, fault recovery, replanning, and science acquisition. However, the state of the art in outdoor rover technology is significantly more modest. The reasons lie in the considerations for mission-ready rovers stated above, along with the difficulty in assembling an integrated rover architecture from a number of disparate, individual research results.

Many indoor robots rely on accurate maps. Furthermore, the environment is often assumed to have straight walls, flat floors, and right angles at corners. These assumptions do not hold for outdoor environments, and the approaches are not easily extended to the general case. Map learning has been proposed as a method for overcoming the lack of an accurate map [Thrun *et al.*, 1998], but this is difficult in unstructured environments with significant motion error. Moreover, the usefulness of a map-learning phase is dubious in an environment where exploration and science tasks are foremost; the extra time necessary to build accurate navigation maps will have a severe impact on the overall science return. Many architectures have been proposed for autonomous robots, relying on artificial intelligence techniques for merging high-level and reactive control of the rover [Hayes-Roth *et al.*, 1995, Wilkins *et al.*, 1995, Simmons, 1990, Alami *et al.*, 1998, Bonasso *et al.*, 1995]. Very few architectures address the issues arising in remote, planetary rovers [Smith and Matijevic, 1989, Chatila *et al.*, 1995, Giralt and Boissier, 1992]. The more complex AI techniques that promise higher levels of autonomous operations often come with no guarantee on behavior, and their operation is not easily predictable. Bottom-up approaches that rely heavily on programming and interacting behaviors [Brooks, 1986, Gat *et al.*, 1994] become equally unpredictable when the system becomes large.

These architectures assume tightly coupled interaction between elements of the architecture (integrated planning and execution) or between a human and the rover system (teleoperation). Since it is not currently feasible to put a full planner on board a planetary rover, limited communication makes these approaches unrealistic at this time.

Research rovers, both indoors and out, are tested repeatedly under similar environments and situations to demonstrate the usefulness of the rover architectures under those conditions. Based on the test results, models are refined, parameters adjusted, and the rover control software modified. However, the only environment that truly matches that of a planetary mission is on the planet. Once there, the rover has a limited lifetime, which must be devoted to mission goals, leaving no time for further testing and refinement.

Rover research often addresses one problem, such as navigation, in isolation. A deployed rover must, however, balance considerations such as navigation with science, communication, resource consumption, and fault recovery. An integrated system presents challenges that single technologies do not address. Nonetheless, the research literature contains many results that are of potential value to missions. Some have influenced current rover designs; many more remain unused. Individual technologies, in addition to those mentioned above, include verifiable real-time control [Borrelly *et al.*, 1998, Schneider *et al.*, 1998, Musliner *et al.*, 1993] and human-computer interaction [MacKenzie and Arkin, 1998].

4.4 Research on autonomous planning and execution

The design philosophy behind the architecture presented in this paper is to concentrate on techniques that can assure safe, predictable operation of an autonomous rover with low computational overhead. This necessitates tradeoffs of optimality versus efficiency, and it sets this work apart from many other systems for planning and execution. The algorithms we have developed and fielded for fault diagnosis and resource management, in particular, are necessarily limited in comparison with complete deductive diagnosis engines and complete resource scheduling techniques, respectively.

The majority of autonomous planning and execution systems rely on on-board generative planning, loosely coupled with an execution system [Muscettola *et al.*, 1998, Wilkins *et al.*, 1995, Gat, 1992, Alami *et al.*, 1998, Munson, 1971]. The computational overhead of the on-board planning required for these systems is impractical for current rover technology; in addition, the semantics and practical implementation of such a system in a highly dynamic environment remains a problematic and largely unsolved problem. We have chosen to construct a system with a powerful, flexible execution language that can, nonetheless, be executed efficiently on board. We are investigating steps towards on-board plan modifications, but we remain sensitive to the real-time constraints and environmental uncertainty of the rover domain.

An alternative approach often presented in the literature consists of a powerful executive system with no associated planner [Firby, 1989, Firby, 1994, Brooks, 1986, Agre and Chapman, 1987]. For purposes of generation and validation of plans, we have adopted an execution language that is powerful and flexible, but intentionally limited to match the capabilities of planning and verification tools.

5 Current and Future Directions

The executive architecture for rovers presented in this paper is a step forward towards the goal of rovers that can autonomously accomplish mission goals. We have designed and implemented an on-board rover architecture that allows robust autonomous operation, where the possible range of actions is specified by the uplinked plan. This balances the need for autonomy with the need for verifiability and caution during missions. As advanced technologies gain acceptance and improve their reliability, more autonomy can migrate on board.

Between the 1999 rover field test, for which the system described in this paper was developed, and the paper publication date in early 2001, the rover architecture has continued to evolve. The implementation has changed from Lisp to C++. On the research side, the major directions that are currently underway are in the areas of state identification, resource management, and plan adaptation.

Our experience with the Livingstone state identification system has led us to investigate alternative techniques that are more specifically adapted to the rover domain. The Livingstone system, typical of qualitative diagnosis techniques, is particularly well-suited to domains where there are occasional transitions between stable states, as well as accurate sensors. In contrast, the dynamic nature of the rover environment and the high level of sensor noise can overwhelm the diagnosis engine and lead to both spurious faults and computational overload. Although some improvements can be achieved through better data filtering, we have instead begun to look at integrating filtering of continuous data and discrete, probabilistic state-estimation techniques [Washington, 2000].

We have also continued to investigate techniques for plan adaptation on board and at plan-execution time, with an emphasis on methods that provide gradual increases in autonomy while remaining computationally efficient. We have developed techniques for pre-compiling utility information at planning time (or periodically at execution time under certain conditions) based on possible execution behavior and using that to provide dynamic adaptation of conditional branching behavior [Bresina and Washington, 2000]. We have also continued work on alternate plans as a mechanism for plan adaptation; we are working to more carefully define their semantics and applicability [Bresina and Washington, 2001]. Finally, we are collaborating on a project for adaptive execution via pre-compiled policies, using a Markov Decision Process representation [Zilberstein and Mouaddib, 1999, Bernstein *et al.*, 2001].

The resource management focus has changed to match the dynamic, utility-based view of plan execution. In the 1999 instantiation, resource models were stored with actions and transmitted to a generic resource manager, which would reason about resource availability and inform the executive of existing and expected plan failures. We have moved to a framework where models for an individual resource exist in a specialization of a generic resource manager. Plan failure information feeds back into probability of action success and failure, which in turn affects action (and thus plan) utility. The advantages of this are: it allows a centralized model of resource usage and availability for each resource, which can be queried and updated by other components; it explicitly handles the interaction between resources and uncertain temporal information about execution; and it more cleanly defines the impact of future resource problems on current execution behavior.

Directions that remain for future research include concurrent actions, uncertainty in resource models, and more general treatment of different sources of uncertainty on execution behavior.

References

[Agre and Chapman, 1987] P. E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, pages 268–272. AAAI, 1987.

[Alami *et al.*, 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotic Research*, 17(4):315–337, April 1998.

[Bapna *et al.*, 1998] D. Bapna, E. Rollins, J. Murphy, E. Maimone, W. Whittaker, and D. Wettergreen. The atacama desert trek: Outcomes. In *IEEE International Conference on Robotics and Automation (ICRA-98)*, pages 597–604, May 1998.

[Bares *et al.*, 1989] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. Whittaker. Ambler: An autonomous rover for planetary exploration. *IEEE Computer*, June 1989.

[Bernard *et al.*, 1998] D. E. Bernard, G. A. Dorais, C. Fry, E. B. Gamble Jr., R. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, and B. C. Williams. Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the 1998 IEEE Aerospace Conference*, 1998.

[Bernstein *et al.*, 2001] D. Bernstein, S. Zilberstein, R. Washington, and J. Bresina. Planetary rover control as a markov decision process. In *Proceedings of the AAAI Spring Symposium: Game Theoretic and Decision Theoretic Agents*, 2001.

[Bonasso *et al.*, 1995] R. P. Bonasso, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In *Proceedings of IJCAI-95*, 1995.

[Borrelly *et al.*, 1998] J.-J. Borrelly, E. Coste-Manière, B. Espiau, K. Kapellos, R. Pissard-Gibollet, D. Simon, and N. Turro. The orccad architecture. *The International Journal of Robotics Research*, 17(4):338–359, April 1998.

[Bresina and Washington, 2000] J. L. Bresina and R. Washington. Expected utility distributions for flexible, contingent execution. In *Proceedings of the AAAI-2000 Workshop: Representation Issues for Real-World Planning Systems*, 2000.

[Bresina and Washington, 2001] J.L. Bresina and R. Washington. Robustness via run-time adaptation of contingent plans. In *Proceedings of the AAAI-2001 Spring Syposium: Robust Autonomy*, 2001.

[Bresina *et al.*, 1999] J. L. Bresina, K. Golden, D. E. Smith, and R. Washington. Increased flexibility and robustness of Mars rovers. In *Proceedings of i-SAIRAS '99, The 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 1999.

[Brooks, 1986] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[Chatila *et al.*, 1995] R. Chatila, S. Lacroix, T. Simeon, and M. Herrb. Planetary exploration by a mobile robot: mission teleprogramming and autonomous navigation. *Autonomous Robots*, 2(4):333–344, 1995.

[Christian *et al.*, 1997] D. Christian, D. Wettergreen, M. Bualat, K. Schwehr, D. Tucker, and E. Zbinden. Field experiments with the ames marsokhod rover. In *Proceedings of the 1997 Field and Service Robotics Conference*, December 1997.

[de Kleer and Williams, 1987] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:100–117, 1987.

[de Kleer and Williams, 1989] J. de Kleer and B. C. Williams. Diagnosis with behavioral modes. In *Proceedings of IJCAI-89*, 1989.

[Firby, 1989] J. R. Firby. *Adaptive execution in complex dynamic worlds.* PhD thesis, Department of Computer Science, Yale University, 1989. Available as Yale University Technical Report YALEU/CSD/RR #672.

[Firby, 1994] J. R. Firby. Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems*, June 1994.

[Gat *et al.*, 1994] E. Gat, R. Desai, R. Ivlev, J. Loch, and D. Miller. Behavior control for robotic exploration of planetary surfaces. *IEEE Transactions on Robotics and Automation*, 10(4), August 1994.

[Gat, 1992] E. Gat. Integrating planning and reasoning in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of AAAI-92*, pages 809–815. AAAI, 1992.

[Giralt and Boissier, 1992] G. Giralt and L. Boissier. The french planetary rover VAP: Concept and current developments. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS '92)*, pages 1391–1398, 1992.

[Gulick *et al.*, 1999] V. C. Gulick, R. L. Morris, M. Ruzon, and T. L. Roush. Autonomous science analysis of digital images for mars sample return and beyond. In *The 30th Lunar Planetary Science Conference*, 1999.

[Hayati and Arvidson, 1997] S. Hayati and R. Arvidson. Long range science rover (rocky 7) mojave desert field tests. In *Proceedings of i-SAIRAS*, 1997.

[Hayes-Roth *et al.*, 1995] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic. A domain-specific software architecture for adaptive intelligent systems. *IEEE Transactions on Software Engineering*, 21(4):288–301, April 1995.

[Linden and Glicksman, 1987] T. A. Linden and J. Glicksman. Contingency planning for an autonomous land vehicle. In *Proceedings of IJCAI-87*, pages 1047–1054. IJCAI, 1987.

[MacKenzie and Arkin, 1998] D. C. MacKenzie and R. C. Arkin. Evaluating the usability of robot programming toolsets. *The International Journal of Robotics Research*, 17(4):381–401, April 1998.

[Mishkin *et al.*, 1998] A. H. Mishkin, J. C. Morrison, T. T. Nguyen, , H. W. Stone, B. K. Cooper, and B. H. Wilcox. Experiences with operations and autonomy of the mars pathfinder microrover. In *Proceedings of the 1998 IEEE Aerospace Conference*, 1998.

[Munson, 1971] J. H. Munson. Robot planning, execution and monitoring in an uncertain environment. In *Proceedings of IJCAI-71*, pages 338–349. IJCAI, 1971.

[Muscettola *et al.*, 1998] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1/2), August 1998.

[Musliner *et al.*, 1993] D. Musliner, E. Durfee, and K. Shin. Circa: A cooperative, intelligent, real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), 1993.

[Schneider *et al.*, 1998] S. A. Schneider, V. W. Chen, G. Pardo-Castellote, and H. H. Wang. Controlshell: A software architecture for complex electromechanical systems. *The International Journal of Robotics Research*, 17(4):360–380, April 1998.

[Simmons, 1990] R. Simmons. An architecture for coordinating planning, sensing and action. In K. Sycara, editor, *Innovative Approaches to Planning, Scheduling, and Control*, pages 292–297. Morgan Kaufmann, 1990.

[Singh *et al.*, 2000] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr. Recent progress in local and global traversability for planetary rovers. In *Proceedings, IEEE Conference on Robotics and Automation*, April 2000.

[Smith and Matijevic, 1989] D. B. Smith and J. R. Matijevic. A system architecture for a planetary rover. In *Proceedings of the NASA Conference on Space Telerobotics*, 1989. JPL Publication 89-7.

[Space Studies Board, 1999] Space Studies Board. Scientific rationale for mobility in planetary environments. Technical report, National Research Council, 1999.

[Sweet *et al.*, 1999] A. Sweet, T. Blackmon, and V. Gupta. Simulation of a rover and display in a virtual environment. In *Proc. of the American Nuclear Society 8th International Topical Meeting on Robotics and Remote Systems*, 1999.

[Thrun *et al.*, 1998] S. Thrun, D. Fox, and W. Burgard. Probabilistic mapping of an environment by a mobile robot. In *IEEE International Conference on Robotics and Automation*, pages 1546–1551, May 1998.

[Volpe *et al.*, 2001] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The CLARAty architecture for robotic autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*, March 2001.

[Washington *et al.*, 1999] R. Washington, K. Golden, J. Bresina, D. E. Smith, C. Anderson, and T. Smith. Autonomous rovers for mars exploration. In *Proceedings of The 1999 IEEE Aerospace Conference*, 1999.

[Washington, 2000] R. Washington. On-board real-time state and fault identification for rovers. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, 2000.

[Wettergreen *et al.*, 1993] D. Wettergreen, C. Thorpe, and W. Whittaker. Exploring mount erebus by walking robot. *Robotics and Autonomous Systems*, 11(3-4):171–185, December 1993.

[Wilkins *et al.*, 1995] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental and Theoretical AI*, 7(1):197–227, 1995.

[Williams and Nayak, 1996] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of AAAI-96*, pages 971–978, 1996.

[Zilberstein and Mouaddib, 1999] S. Zilberstein and A-I. Mouaddib. Reactive control of dynamic progressive processing. In *Proceedings of IJCAI*, pages 1268–1273, 1999.