Plan Execution, Monitoring, and Adaptation for Planetary Rovers

Richard Washington* and **Keith Golden** and **John Bresina**
NASA Ames Research Center
MS 269-2 Moffett Field, CA 94035
(+1) 650-604-5000
{richw | kgolden | bresina}@ptolemy.arc.nasa.gov

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 respect to 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.

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% [Board, 1999].

Traditionally, spacecraft have been controlled through 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 can 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 that demonstrates three advances over existing technology. First, the language accepted by the executive consists of flexible, conditional plans that can adapt 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.

This work is inspired 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. On the other hand, the current rover executive architecture does not attempt to reproduce all of the capabilities of the RA; in particular, multiple concurrent threads, model-based recovery planning, and system reconfiguration are not currently included in our executive.

The remainder of the paper will discuss the elements of the rover executive architecture for flexible plan execution, monitoring, and adaptation.

2 Architecture Overview

The rover autonomy architecture is specifically designed to accommodate the particular constraints of plane-

*Caelum Research Corporation

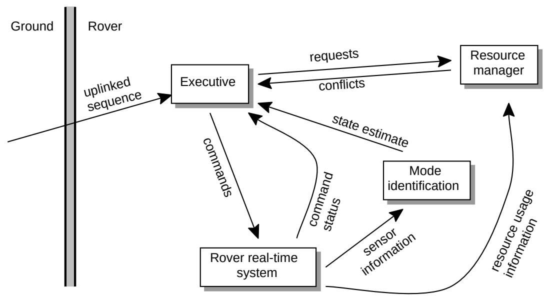

Figure 1: Rover autonomy architecture

tary 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]. 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 (MIR).

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 MIR. MIR's mode identification layer 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.

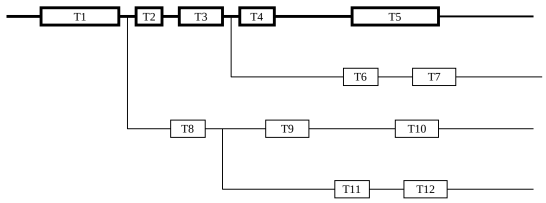

Figure 2: A plan with a nominal sequence (in bold) and a set of contingent branches.

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 resources are evaluated with respect to the nominal execution path. 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.

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 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). CX receives state information from the mode identification module MIR 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. The utility of successfully completing an atomic action is 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.

4 Monitoring

4.1 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.

One internal execution fault that CX monitors for and reacts to is a communication failure between CX and the real-time rover system. The communication is based on a datagram model to avoid blocking the underlying real-time system, but that model does not ensure packet receipt. 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.

4.2 Model-based State Estimation

State estimation 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 leads to the consideration of anomalous (or fault) modes.

The approach that Sojourner [Mishkin *et al.*, 1998] 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.
- 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.

The diagnosis component of the rover autonomy architecture is MIR, for Mode Identification and Reconfiguration. The rover executive architecture uses MI, the

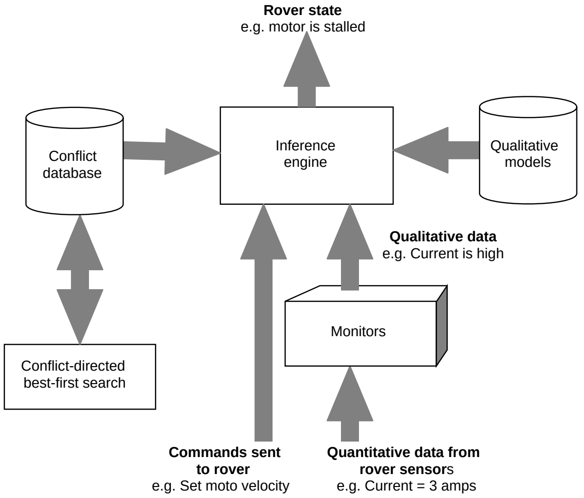

Figure 3: Information flow in MIR.

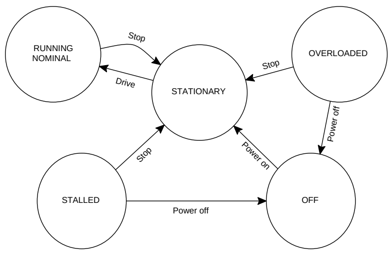

Figure 4: State diagram for motor control component. Shaded nodes represent fault modes. In addition to the explicit transitions, indicated by arrows, there are also implicit, random transitions to fault nodes, represented in the model probabilistically.

Mode Identification component of MIR; MR, the Mode Reconfiguration component, remains as future work (see Section 8). MIR was developed as part of the Remote Agent Experiment on board the DS1 spacecraft [Bernard *et al.*, 1998].

MI eavesdrops on commands sent by CX to the rover. As each command is executed, MI 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}. MI 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, MI infers the most likely current state (see Figure 3). MI 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.

MIR uses algorithms adapted from model-based diagnosis [de Kleer and Williams, 1987; 1989] to provide the above functions. MIR 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.

4.3 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 communication bandwidth, thus space 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.

We have designed an on-board, run-time resource manager 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,

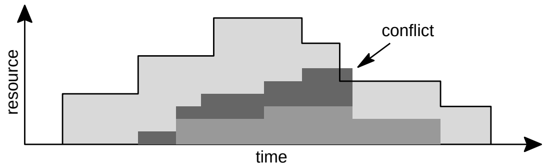

Figure 5: Resource management timelines. Resource availability is shown in the lighter shade, usage in darker shades. Conflicts occur where actual or predicted usage exceeds availability.

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).

5 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.

The 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.

5.1 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.

6 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 that collaboration during the field test.

Further outdoor and simulated tests demonstrate 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.

7 Related Work

7.1 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; Chatila *et al.*, 1995]. The space agencies recognize the necessity integrated architectures that support autonomous operations for effective missions [Hayati and Arvidson, 1997].

7.2 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]. This 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.

7.3 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].

8 Conclusions 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.

Near-term enhancements to this architecture involve moving some limited planning capabilities on board. The MIR system in the Remote Agent includes a recovery expert, MR, that takes as input a recovery request and returns a sequence of operations. This sequence, when executed starting in the current state, will move the executive into a state satisfying the properties required for successful execution of the failed activity.

Currently, when faced with multiple possible options, the executive follows the branch with the highest utility, where the utility is calculated on ground during construction of the plan. This utility can be situation-dependent, so an on-board dynamic utility revision component would improve the performance of the plan with respect to a changing situation.

The resource manager provides detailed information about which steps of a plan are predicted to cause conflicts and where additional resources remain. Whereas the current architecture can modify its behavior only at level of entire branches, the resource information could be used to revise a plan by adding and deleting individual steps. In effect this would be applying a limited form of rescheduling on board.

References

[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] Douglas E. Bernard, Gregory A. Dorais, Chuck Fry, Edward B. Gamble Jr., Bob Kanefsky, James Kurien, William Millar, Nicola Muscettola, P. Pandurang Nayak, Barney Pell, Kanna Rajan, Nicolas Rouquette, Benjamin Smith, and Brian C. Williams. Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the 1998 IEEE Aerospace Conference*, 1998.

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

[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 *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*, 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.

[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.

[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.

[Muscettola *et al.*, 1998] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian 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] David Musliner, Ed Durfee, and Kang 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.

[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.

[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.

[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.

[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.