N90-22313

SYSTEM CONTROL OF AN AUTONOMOUS PLANETARY MOBILE SPACECRAFT

William C. Dias
Barbara A. Zimmerman
Sequence Automation Research Group
Mission Profile and Sequencing Section
Jet Propulsion Laboratory
California Institute of Technology

PHONE: Dias (818) 354-0153   Zimmerman (818) 354-6700
MAIL: Jet Propulsion Laboratory, MS 301-250D
4800 Oak Grove Drive, Pasadena, CA 91109

ABSTRACT

Our goal is to suggest the scheduling and control functions necessary for accomplishing mission objectives of a fairly autonomous interplanetary mobile spacecraft, while maximizing reliability. Goals are (a) to provide an extensible, reliable system conservative in its use of on-board resources, while (b) getting full value from subsystem autonomy, and (c) avoiding the lure of ground micromanagement. We propose a functional layout consisting of four basic elements: GROUND and SYSTEM EXECUTIVE system functions and RESOURCE CONTROL and ACTIVITY MANAGER subsystem functions. The system executive includes six subfunctions: SYSTEM MANAGER, SYSTEM FAULT PROTECTION, PLANNER, SCHEDULE ADAPTER, EVENT MONITOR and RESOURCE MONITOR. The full configuration is needed for autonomous operation on Moon or Mars, whereas a reduced version without the planning, schedule adaption and event monitoring functions could be appropriate for lower-autonomy use on the Moon. An implementation concept is suggested which is conservative in use of system resources and consists of modules combined with a network communications fabric. The paper introduces a language concept we have termed a "scheduling calculus" for rapidly performing essential on-board schedule adaption functions.

INTRODUCTION

Interplanetary mobile spacecraft (rovers) require more autonomy than spacecraft in planetary flybys or orbiters, if they are to be acceptably productive. This is essentially because knowledge of the environment changes over a much shorter time scale than the speed at which data could be received and analyzed and commands generated and sent from Earth, given the light-time delays (Wilcox, et al, 1987; Dias, et al, 1987). Even a low autonomy rover on the relatively nearby moon needs more autonomy in the control area than other spacecraft if it is required to move continuously (Pivirotto, et al, 1989).

This paper proposes a FUNCTIONAL SYSTEM CONTROL ARCHITECTURE in which a design or requirements for a design could be phrased. We address fairly autonomous rovers first of all. Second, adaption of the control architecture to a low-autonomy Lunar rover is discussed. Next, the paper has a section on the practicalities of implementing the control architecture. Last, we discuss ongoing research in the JPL Sequence Automation Research Group on the development of a language in which the rule base of vital parts of the control system could be phrased.

The design process, as well as the design itself, should be responsive to the needs of operations managers to ascertain reliability and functionality. This is because the control system partly substitutes functionally for ground operations. Fairly autonomous rovers would need to be able to reliably perform many of the spacecraft command and control functions now performed only on Earth (Linick, 1985). It will be seen the functional layout preserves some of the current division of responsibilities among traditional ground system and subsystem elements. Various parts of the COMMAND GENERATION process, including REQUEST GENERATION, REQUEST INTEGRATION, SCHEDULE GENERATION and COMMAND TRANSMISSION, are proposed for on-board implementation.

Our proposed architecture assumes a spacecraft with a complex and varied set of goals and activities only partly predictable during design. Activity schedules will require some parallelism and optimization, as now provided for on Voyager and Galileo. There will inevitably be a desire for a great degree of ground control, to maximize mission return. In fact the command and control system design must walk a tightrope among the three paradoxically competing concepts of system autonomy, subsystem autonomy, and maximal ground control, each proffered in the name of maximizing return.

The control system needs to be as VERSATILE as possible, because the exact desired operational modes and combinations of activities for a spacecraft are not always fully knowable in advance, and this will be especially so for a planetary rover. The less known in advance about the particulars of the environment, the more varied that environment, the more varied and general the set of tasks, and the larger the suite of approved instruments, the less predictable the final operational range will be.

To promote rover functional EXTENSIBILITY, the control system design should incorporate features able to enhance the software development environment. Quick changes may be needed in the software implementing traverse and sample acquisition functions after landing, whether due to unforseeable hardware failures or unexpected conditions. Depending of course on the mission design, sample return mission surface stay times could be as short as a few months, adding greatly to pressures for operational responsiveness (Bourke, et al, 1989). Rapid software turnaround presents a danger of its own in an operational environment. By being versatile and robust enough to comprehensively trap and correct fault conditions, a good control system design should make fast software development turnaround possible in the operational phase. We can define the architecture to maximize probability of success. We have done this by incorporating software verification in the fault protection scheme.

Our architecture differs considerably from other proposals, though it takes a layered approach often favored by other designers (IKI, 1988; Simmons, et al, 1989). Resulting as it does from the considerations in the above paragraphs, our proposal is oriented towards providing "general purpose" spacecraft functionality by representing what currently exists as ground operations functions on board. This approach differs from Subsumption Architecture (Brooks, et al, 1989) and from the Task Control Architecture (Simmons, et al, 1989). These appear to be oriented towards a predefined (but presumably robust) set of "behaviors", and towards missions which could be accomplished with rovers operating in a more narrowly defined functional envelope than the kind of mission we foresee. We feel early interplanetary rover missions will need to use mobile spacecraft which are as general in capability as is reasonable, for all the reasons in the above paragraphs. It seems likely there would be a place for both the "behavioral" and "general purpose" architectural philosophies in a well funded, longer term solar system planetary exploration program, however.

The functions required for the rover as a whole are taken generally from Smith and Matijevic (Smith, et al, 1989). We have added Global Navigation, Data Handling, and Pointing and Articulation to their System Executive, Telecommunications, Power, Thermal, Science, Mobility and Sampling. Thus, our idea on how these subsystem functions should be defined is slightly different, but exact subsystem delineations are not our purpose. Instead, our hope is to clarify the system / subsystem interface in general. We find no essential conflict between the Smith and Matijevic formulation and ours. The emphasis is different. Their method appears to provide a convenient means for designating the control, command, and data paths to be included in a roving spacecraft from high to low levels, before design begins. Where they provide a general purpose tool and framework, we try to provide and justify the functional relationships which need to be used to fill in the details in the Smith and Matijevic architecture matrix, with emphasis on the System Executive over the subsystems. We wanted to show the most meaningful functional interfaces and formulate them so that people can begin to think of allocation of functions to modules.

The control system needs to incorporate concepts from spacecraft FAULT PROTECTION (Riethle, 1983) in order to improve RELIABILITY over research and ground-based robots and robotic vehicles. In a "classic" form of fault protection, signals from one or a few subsystems are used to determine a fault and then pre-canned, simple and highly structured routines take control of the subsystem or spacecraft and throw it into a predetermined, safe, but usually non-productive state. This "classic" form of fault protection needs to continue to exist on planetary rovers, but its scope needs to be limited in such a way that more intelligent autonomy is not subverted by its sheer simple-mindedness. More refined forms of fault protection which take into account the higher level of intelligence on the spacecraft must be included, or the advantages of intelligent autonomy would be lost.

Finally, it may seem that this architecture is too complex, that it could never execute in a timely fashion. We do not take this potentially serious problem lightly. The functional description appears complex partly because we did not want it to appear incomplete through overgeneralization. We wanted to try to bring out a description of the functions and interrelationships that might be required, perhaps more complete than available in the past. The design and implementation could turn out considerably simpler than the functional description might cause one to expect, but all functions should be addressed in the design process. We also feel we have allayed some of these complexity concerns in the implementation section.

OVERVIEW

This section contains an abbreviated description of the overall control flow of the architecture. In the subsequent detailed section on selected elements, we give a fuller description of the behavior of the major elements.

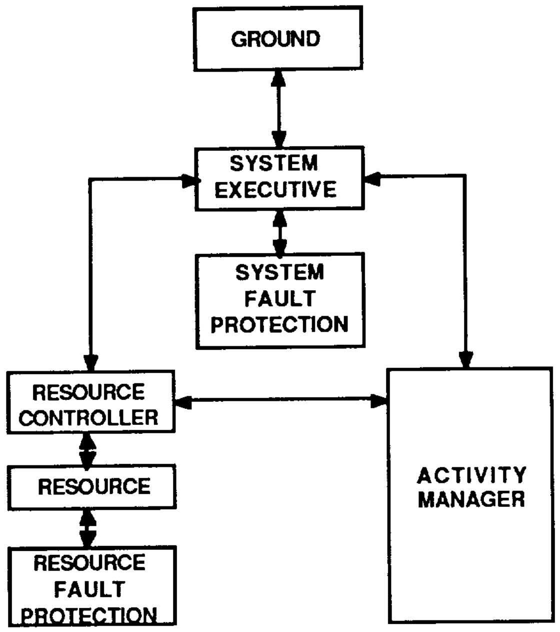

Figure 1. Basic Functional Control Architecture Layout.

The functional control architecture has four basic elements: GROUND, SYSTEM EXECUTIVE (SE), ACTIVITY MANAGERS (AMs), and RESOURCE CONTROLLERS (RCs). These share control as shown in Figure 1. Ground and SE in combination provide the system-level command and control functions, while the subsystem functions are performed by the AMs and RCs in various combinations for different operational modes. The SE and RCs each have separate fault protection functions, but the AMs are oriented more strictly towards command generation, command, and control, and have no fault protection role.

In the operational scenario for autonomous modes, which are the ones addressed in this paper, GROUND provides goals and schedule contraints -- general and specific -- to the rover SE. GROUND uses some sort of simulation based on whatever information it has from orbit, previous traverses, statistical likelihoods, and the rover's sensors to try to foresee likelihood of success. There is no guarantee that goals are achievable within time constraints, only some probability. Goals should try to encompass at least a few hours of activity, preferably a day or so, with fallback contingencies in case operations take longer than expected, and fill-in items in case activities take less time than foreseen.

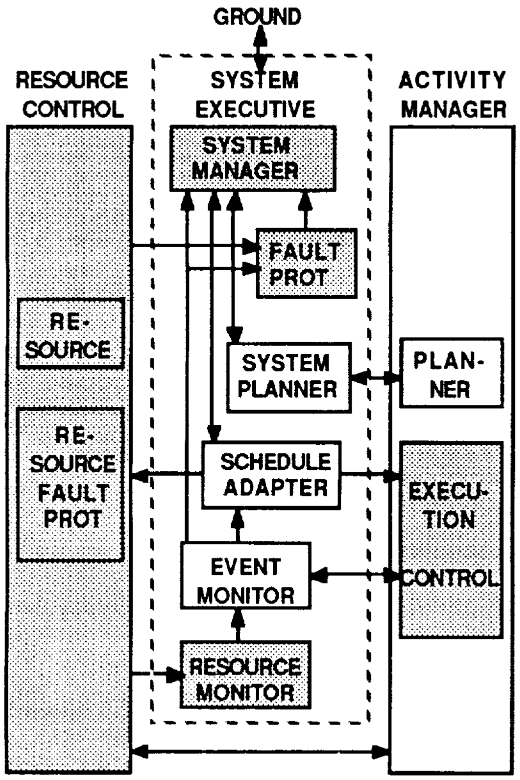

Figure 2. System Executive command/control architecture. All functions required on more autonomous rovers. Only shaded areas needed for command/control of low-autonomy rovers.

Figure 2 depicts the internal functional interfaces of the System Executive in more detail. The on-board SE co-ordinates rover operations through the SYSTEM MANAGER. First the SYSTEM PLANNER and the requisite subsystem AM PLANNERS agree on a plan to accomplish the goals. They do this by first having the SYSTEM PLANNER arrive at sub-goal and activity sequences and resource and time envelopes within which the AMs must plan, then having the AM PLANNERs arrive at sequences implementing the goals to the degree possible. A series of "events" would be part of any schedule agreed upon. These events would be more or less at the same level as those used by a Voyager or Galileo ground sequence team in its higher level scheduling activities today. The time relationships among these would be partly relative, that is, the time at which an event would occur would be phrased as relative to other events, not as an absolute time (though the system would always have a nominal absolute time for future events.) Sequences are phrased relative to events, with the exact time of commanding to be determined later, in execution.

In the execution phase of the plan, the AM EXECUTION CONTROLLER actually sends commands to (and reads data from) the RESOURCE CONTROLLERS, which are the only entities able to address the physical resources directly. The RESOURCE MONITOR (RM) keeps track of resource status (including progress on events) at the system level, by reports and / or polling techniques. The EVENT MONITOR keeps tabs on the progress of the schedule with respect to the events. It does this by receiving both timed and event-tagged progress reports from the AMs, and (redundantly) by RC reports relayed from the RM. Discrepancies in these reports result in fault protection, replanning, or other exception processing. AMs may optionally read event reports from the EVENT MONITOR. The SCHEDULE ADAPTER constantly modifies the timing of the system-level sequence, within agreed parameters, based on event-progress reports from the EVENT MONITOR. The AMs modify their schedules for data and for commanding the RCs based on near-realtime scheduling refinements from the SCHEDULE ADAPTER. The latter is also able to dynamically reconfigure the RCs based on schedule changes, and the AMs are then limited to commanding within those parameters.

Replanning can occur during execution, either because specific events in a schedule are designated as points for plan refinement, when the next increment of information is available for planning, or because unforeseen conditions caused an existing schedule to be invalidated. Sometimes, subsystem replanning will be necessary without system replanning being needed.

RESOURCE FAULT PROTECTION responses, including reflexes, may be invoked to safe a resource based on resource-internal information alone. Signals are then sent system-wide. SYSTEM FAULT PROTECTION may be invoked in response to conditions signaled by combinations of subsystems, the schedule becoming dangerously unworkable, disagreeing progress reports from the AMs and RCs, or signaling of computed status derivatives and trends indicative of faults from the RM. One would try to autonomously replan out of at least some fault protection response modes.

DETAILED FUNCTIONS OF SELECTED SYSTEM ELEMENTS

This section discusses selected elements of the architecture in more detail than in the overview.

SYSTEM PLANNER

First, the System Planner processes ground-supplied GOALS into a serial and parallel collection of MAJOR ACTIVITIES which will bring about the desired goals. Time factors are only applied in a gross way, which is enough to eliminate many schedule possibilities.

Second, the System Planner derives schedule constraints and resource utilization envelopes within which each major activity must be planned.

Next, the System Planner waits while subsystem AM Planner functions derive a SCHEDULE from the required activities (See Activity Managers, below). The System Planner then integrates the resultant subsystem-derived schedules, which may include changing the previously imposed resource and schedule constraints and asking for additional subsystem planning.

The SCHEDULE derived by the planning functions must have a form consonant with the need for on-board schedule maintenance in realtime. This is more than is required of a contemporary spacecraft schedule. In general a schedule might be defined as the timed series of events and states of all resources and subsystems which is determined to bring the desired activities to completion. In order to allow for later realtime adaption, the new type of schedule must include temporal relations among events, of a sufficiently economical nature to allow operations in realtime. In other words, an EVENT-DRIVEN SCHEDULE is needed. Each event or state change needs to have its time requirements described relative to when other events or state changes take place.

Stated more abstractly, the schedule consists of a set of functional relationships among the three elements of states, events and times, such that, where t is an instant in time and F1 and F2 are functions:

1. System State$_t$ = F1(Subsystem States$_t$)
2. Acceptable System State$_t$ = F2(System State$_{t-1}$)

and, where F3 is a function, Event$_a$ is next on the schedule and Events $b1$ through bx have already occurred:

3. Acceptable Time Interval(Event$_a$) = F3(Times (Event$_{b1}$,...,Event$_{bx}$))

Applied recursively, what this expresses is that acceptable activities or states of both the system and of each subsystem, for any instant in time, depend on the states of all those entities dynamically as matters progress. This is similar to a system which runs according to "control laws". In this way, schedules are derived without assigning all final times, but with needed time linkage among events and states. In the last section in the paper, we discuss a language and some of the rules of a "scheduling calculus" which could be used to express and enforce the functions in a real system.

The following are possible examples of "events" at the level of interest of the System Planner. These correspond roughly to a fairly high level of planning, specifically the initial sequence integration which occurs right after request generation for a spacecraft. They stop short of device management which in this conception is at the level of the subsystem AM Planner and RC.

start / finish a camera platform slewing operation
start / finish a single maneuver in the course of a longer traverse
start / finish a sample arm movement (or, maybe a joint movement)
start / finish warmup of a sample processing oven

SCHEDULE ADAPTER

The Schedule Adapter converts the schedule, phrased in expressions to be operated on by the scheduling calculus, into a high-level SEQUENCE with final times, exact states, etc., assigned. Whereas "schedules" include functional relationships among times, events and states, "sequences" include only the times, events and states which result from applying the functions based on knowledge at a given instant. Thus what is known about the spacecraft high level sequence might include only a few seconds or minutes of activity.

So the Schedule Adapter routinely uses the rules of the scheduling calculus to derive acceptable ranges for the next set of all subsystem states from the current state. In a closely related function, it continually changes the system's idea of when events will occur. These adaptions thus do not necessarily constitute a need for schedule changes (i.e., replanning) in our terminology.

The Schedule Adapter is very dependent on input from the Event Monitor to keep it informed of status of all relevant events, or uncertainty in that status. The Schedule Adapter must have a way to respond to unclear state or event status knowledge, informing System Fault Protection which may be the one to decide what to do.

The Schedule Adapter sends sequence revisions to the relevant AM execution control functions, which adapt in turn.

The Schedule Adapter provides configuration commands to the RCs. The AMs, which actually run the rover through an activity such as a rolling maneuver, are limited to commanding the RCs within the envelopes configured by the Schedule Adapter for each moment in time. This mechanism provides system level resource control while allowing the responsible AMs reasonable command authority over those resources needed.

There may have been specific points in the schedule designated for system or subsystem replanning or plan refinement. If so, these are honored in an event-driven fashion the same as other dependent events.

The Schedule Adapter may be called upon by System Fault Protection to invoke a special schedule implementing a fault protection schedule, and to abort a current schedule in an organized way.

Another part of the Schedule Adapter's function is to realize when incompatible combinations of conditions occur or are about to occur. For instance, if the rover is running behind schedule in getting to a site of scientific interest, low priority activities may need to be either rushed through, with controlled loss of data quality, or abandoned entirely. It is up to the Schedule Adapter to do this and signal the AMs and RCs accordingly. If conditions deteriorate further, the System Planner and AM Planners must be reinvoked to completely rework the schedule and salvage what is possible of the original goals. As a last resort, the rover should inform Earth of the problem at the next opportunity. Ground can then respond by recommanding with adjusted goals. This can be an expensive solution in terms of wasted rover time, a fact that needs to be worked into the original decision on what to do if a schedule fails under various conditions.

SYSTEM EVENT MONITOR

This function holds the current system knowledge of all events previously completed/aborted in the schedule or in progress. It accomplishes this by seperately monitoring both resource and activity status. This redundant approach provides cross-checking considered highly desirable in spacecraft fault monitoring (Riethle, 1983; Reiners, 1985).

First, AM Execution Controllers provide activity status to the Event Monitor, both at predetermined intervals and at status change points agreed to in the schedule. This is a high level check that activities are on schedule and status is acceptable. Activity status report frequency will undoubtedly vary by operational mode. Events to be reported can be unplanned. For instance when the AM Execution Controller becomes aware independently that its plan is no longer workable, that needs to be reported.

Second, event-related resource status reports are provided from the System Resource Monitor, which has collected these from the RCs. Unplanned events, such as the invocation of a reflex action by the RFPCs, also need to be reported.

The Event Monitor integrates these various sources of event status and reports to the Schedule Adapter to help it make decisions. System Fault Protection is informed in case event patterns reported can be used to infer fault conditions. The software validation function is served because some (perhaps most) software problems will show up as error reports or as inconsistencies in event reports among the various sources.

SYSTEM FAULT PROTECTION (SFP)

This function includes the separate areas of fault detection, fault analysis, and fault reponse. It detects system-level fault conditions from combined messages from the System Event Monitor, System Resource Monitor, Resource Fault Protection Controller, and Resource Controllers. Messages can include both status and data determined in the design process. It may execute hard-coded (or at least high-speed), high-reliability responses to faults detected, forcing the spacecraft into very well defined states from which recovery will be as easy as possible. It may conceivably also initiate slower fault responses requiring normal schedule planning channels. It seems likely System Fault Protection would include reliable, predesigned, canned schedules in a form able to be adapted to specific current conditions by the on-board Planners and/or Schedule Adapter.

The System Fault Protection functions need to respond directly from primary inputs -- otherwise they would be dependent on other functions and therefore less foolproof. At least some responses must be designed to operate without permission from the System Manager. The System Manager must in turn be informed as soon as fault protection is invoked. This requirement poses a problem faced by all systems with distributed authority and / or redundant data -- the possibility that different parts of the system will be working to cross purposes for some interval of time, with resultant system state ambiguities. The problem cannot be fully addressed until the design phase. Hopefully, the pre-canned fault protection schedules can be designed so as to be adaptable by the Schedule Adapter to the particulars of the current state.

SYSTEM RESOURCE MONITOR

The System Resource Monitor has three separate ways of monitoring resource status.

First, the SE receives a "heartbeat" -- or elementary status message -- from each RC on a regular, timed basis. These messages are independent of any specific task the resource has been commanded to perform. The total absence of a message, a message conveying an error status, or a message containing data from which an error condition is deduced, can be grounds to invoke system-level fault response. Heartbeat occurence and frequency may vary from subsystem to subsystem, but as a point of reference, Galileo heartbeats are at approximately 0.7-second intervals.

Second, the RCs report to the Resource Monitor in connection with the specific tasks they have been commanded to support. These messages are tagged to the portion of the command sequence that brought them about. A "resource event" in this sense includes any RC status change, or any change in the tasks being supported even though there may be no other resource status change. Reports are also required at regular intervals (much like the heartbeats) as a cross-check that status is as expected and things are on schedule.

Third, signals from the RFPC are received in the event resource-level fault protection is invoked.

Both planned and unplanned resource status changes are events and are duly reported to the System Event Monitor.

ACTIVITY MANAGERS (AMs)

Identified subsystem functions requiring AMs are: Science Payload, Sample Acquisition, Traverse, Global Navigation, Telecom, Data Handling, and Imaging. We discuss the AMs only to the degree necessary to put them in context with the rest of the rover control system. Their design is specialized, different for each subsystem function, and not the subject of this paper.

AMs provide a level of intelligence higher than the RCs for important spacecraft subfunctions (e.g., autonomous power subsystem, Fesq, et al, 1989; autonomous navigation subsystem, Gat, et al, 1989). They trade or share control depending on operational mode. On less autonomous spacecraft, subsystem planning and monitoring would be performed by ground subsystem engineers or scientists in coordination with system level engineers. On a more autonomous spacecraft such as a fairly autonomous rover, the AMs to some extent represent on-board the functions the subsystem engineers serve on the ground. AMs have NO FAULT PROTECTION RESPONSIBILITIES, because it would be redundant, and because they represent areas such as navigation where fast software development turnaround is desirable. AMs requiring data from other AMs for planning or execution functions must obtain and update information in common data base areas to maintain controls.

The fully implemented AM is presumed to have a PLANNER and an EXECUTION CONTROLLER.

The AM PLANNER utilizes specialized subsystem knowledge unavailable to the SE planner to derive (1) a sequence of time-driven and event-driven commands to be given to the RCs, and (2) a corresponding set of expectations (Gat, et al, 1989). The AM Planner co-ordinates provisional plans with the SE Planner. It should be noted that, as in the case of the SE planner, planning may be incremental. That is, a high level activity such as the acquisition of a sample will probably be worked out as a series of steps with estimated times, with final planning applied only as preceding steps complete.

The AM EXECUTION CONTROLLER co-ordinates execution of plans agreed to between the AM Planner and the SE Planner. It implements control by commanding the RCs and reading data and status from them. It may read event status, if desired, from the SE Event Monitor. It responds to schedule envelope changes provided by the Schedule Adapter in near realtime. It reports its version of events and status back to the Event Monitor on both a time- and event-driven basis. It may be interrupted by the SE if the latter decides things are not on track and invokes fault protection or replanning.

RESOURCE CONTROLLERS (RCs)

Every resource is governed by a Resource Controller (RC) which has about the same level of intelligence as a typical contemporary disk controller. All contact between a resource and other elements (except resource-level fault protection) is through its RC. RCs for different purposes could have different amounts of memory and CPU power, but they would all have the same qualitative functions: receiving commands, sending status, sending and receiving data, and keeping track of a time-linked stack of commands for one resource. The RC accepts reconfiguration commands from the Schedule Adapter and commands from the AM Execution Controllers, provided these are within the configuration envelope provided by the Schedule Adapter. It provides both time- and event-tagged status to the System Resource Monitor and System Fault Protection functions. Status messages returned by the RC include identifiers so that other functions may know which event(s) from the sequence the resource is currently working on, and the RC's progress on the sequence.

RESOURCE FAULT PROTECTION CONTROLLERS (RFPCs)

Any precanned, fixed routines which are designed to automatically, unconditionally and unilaterally change individual resource states based on sensor fault readings are handled by the RFPCs. This is a basic, conventional spacecraft fault protection strategy which will continue to be needed for some faults. Examples include fuse protection, automatic shutdown of electric heaters exceeding temperature specs, or trend analysis for individual resources. Other system elements such as the RCs, AMs and SE Event Monitor are informed of the fault status through normal channels after the fact.

In some cases, a system-wide fault response is needed, which must be processed by System Fault Protection (SFP). In those predefined cases, the duty of the RFPC is to simply send the status to the RC and the important fault data to the SFP for action.

In our view, RFPCs would also implement any required resource-level reflexes. These include any action or behavior, at the level of the individual resource, required on an unexpected basis and on too short notice for organized involvement by the planning functions. We believe reflexes should be considered fault reponses for spacecraft design purposes. Of course, not all reflexes result from true emergencies and will therefore sometimes result in situations easily handled by on-board software.

The invocation of any reflexes or resource-level fault protection presumably necessitates replanning after any further immediate spacecraft safing is complete.

RESOURCES

Resources are commandable elements providing services, conditions, or commodities to the requesting elements, through their RCs. All communication to a resource is through the RC in normal modes. The following commandable resources have been identified for a planetary rover: Power, Thermal State, Data Handling, Science Payload, Science Imaging, Sampling Mechanisms, Pointing and Articulation, Mobility / Vehicle State, Mass Storage, and Telecom Data.

At this stage of the design, there is always the possibility that some high-speed control requirement will later be found requiring direct communication with the resource. That discovery will have to await the testbed development stage.

LOW-AUTONOMY INTERPLANETARY ROVERS

True teleoperation, in which both command and low-level control is in the hands of an operator, is thought to be an unreasonable means of controlling rovers even on the Moon. The light time delay of around three seconds is too great (Pivirotto, et al, 1989). 300 milliseconds may be the maximum for teleoperation.

Our view is that all the functional elements discussed in this paper would be needed in some degree for fairly autonomous interplanetary rovers, regardless of the distance from Earth. However, considerable economy is possible for a Lunar rover with lower autonomy and interactive commanding from Earth, because the round-trip communication delay (light-time plus electronic delays) is likely to be only a few seconds. All planning (system and subsystem), schedule maintenance and event monitoring functions can be done on Earth. These would be tightly integrated in operator command terminals. Activity execution monitoring, mode switching, resource handling and fault protection would be on the spacecraft. See shaded areas on Figure 1. This is more autonomy and on-board control than would be provided by teleoperation.

It is likely that Lunar rover programs will naturally precede Mars rovers (*Report of the 90-Day Study*, 1989). This provides an opportunity to perfect the designs and techniques for autonomous schedule maintenance on the ground in an earlier program. Those functions would later be moved on-board to achieve the greater autonomy necessary for productivity at the up to 40-minute round trip light time delays presented by Mars, or by more autonomous Lunar rovers.

IMPLEMENTATION OF THE SYSTEM EXECUTIVE ARCHITECTURE

To the SE the planetary-vehicle domain appears to be made up of a two-level hierarchy of elements. The top level of the hierarchy represents the vehicle functions or activities. The lower represents the hardware that participates in carrying out the activities. This view of the rover domain suggests an architecture that supports multiple interacting tasks which can access a pool of resources. A suitable architecture can be modeled, superficially, by a modern computer operating system (Rashid, 1986). However, the computer operating system model is not complete or sufficient because the control procedures commonly invoked fall short of those required for a capable rover's operation. To control a fairly autonomous planetary vehicle the system design must specify a comprehensive self-analysis tool with which to track the state of the vehicle. In this section we describe an architecture for the SE's control functions, and a procedure for planning and diagnosis of the state of the rover as it carries out a task. The procedure, which we call a scheduling calculus, combines qualitative relationships with arithmetic expressions to render judgements about the rover's state, and the validity of a schedule or sequence given that state.

OVERVIEW

Preliminary system designs (Pivirotto, et al, 1989; Lambert, 1989) have typical planetary vehicle functions such as sample acquisition, navigation, data handling, science, imaging, and telecommunications controlled by several independent processing elements (See Figure 3).

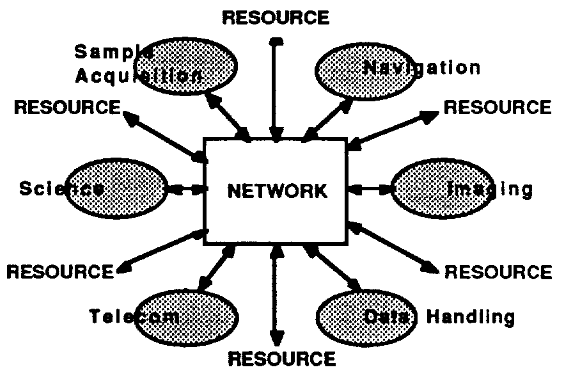

Figure 3. A Distributed Computing Network. Processing Elements (ovals) and Resources controlled by independent activity modules through a common system-level network.

In addition, the processing elements share some form of mass storage and system resources. The processors and the resources are joined by a communication fabric which connects each of the processing elements to one another and to the resources. We will refer to this communication fabric as a network. We envision a layered architecture whereby the activities direct their requests for resource usage through the SE (Figure 4). Access to the SE, the activities, and the resources is by message passing which is supported by SE service routines.

An important concept in the proposed design is that the SE's subfunction modules are modelled as a collection of one or more independent processes that communicate through message passing. The modular nature permits the SE to be dynamically configurable to accomodate the requirements of a mission. Further, we envision a system in which one or more of the SE's subfunction processes reside in each processing element. The distributed capability, which is independent of the number of processors, can be used to partition the demand on the rover's computer resources.

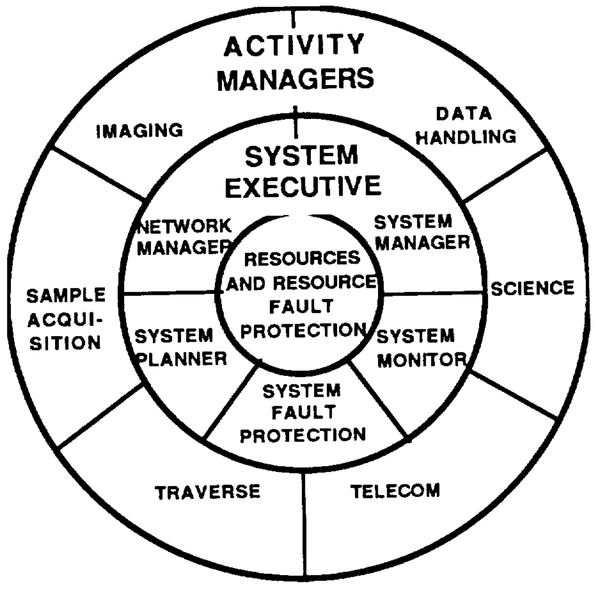

Figure 4. **Rover Command / Control System Layered Architectural Layout.**

SYSTEM EXECUTIVE MODULES

The following sections describe the five classes of modules that make up the SE. These are the SYSTEM MANAGER, SYSTEM FAULT PROTECTION, SYSTEM PLANNER, SYSTEM MONITOR, and SYSTEM NETWORK MANAGER. It is assumed that the system executive processes reside on top of some form of computer operating system environment which provides the low-level functionality with which the hardware is addressed and the data managed. In a distributed system there has to be some means by which data of a global nature are provided to the processes. While the SE, the RCs and the AMs depend on these services, they are not part of the SE functions and are not discussed in what follows.

System Manager

This element of the SE provides the interface between the ground and vehicle, and among some of the major on-board components. The system manager configures the control system to support the mission requirements. For a less autonomous rover that is to be commanded from the ground, the system manager directs the command sequence to the relevant AM Execution Controllers and Resource Controllers. For a more autonomous rover the System Manager activates the planner processes, and the event monitor.

System Fault Protection

Both lower- and higher-autonomy rovers need on-board fault protection. It is likely the great body of fault detection, analysis and response code could be thought of as stored in this separate class of modules. However, it is necessary for efficiency to distribute some of the fault detection responsibilities to the System Monitor and Network Manager modules, outlined below. In addition some elementary fault responses, somewhat redundant, will probably be required as part of the code running in each node in a multiprocessor network. As stated elsewhere, other fault protection is implemented at the subsystem level, entirely outside the SE. However, it is the responsibility of the system fault protection class of modules, in the event of a system fault, to activate fault protection, retrieve the necessary data to configure the resources and return the rover to a known state, and send the necessary commands to orchestrate controlled aborts of on-going processes. In the higher-autonomy rover, stored, configurable schedules can be provided on-board to the System Planner and Schedule Adapter for these purposes.

System Planner

The two planner processes, a Planner and a Schedule Adapter, provide a layered implementation of the scheduling processes. The system manager directs relatively high level goal and constraint data to the system planner. The goals are expanded by the planner into a time-ordered set of tasks for the Activity Managers to process. The Activity Managers produce the detailed sequence of events that accomplish the tasks. Before execution, this sequence is returned to the planner for verification and Event Token extraction (see below). The schedule adapter monitors the plan's execution, adjusting the sequence as needed.

There are several features that reflect the degree of sophistication represented by the SE planner and schedule adapter processes. The planner must automatically generate and maintain a schedule. This is a significant task which ground-based planners currently do not fully support. We are investigating the Remote Mission Specialist (RMS) system developed by the Sequence Automation Research Group at JPL (Rokey, et al, 1990), and the research on automatic planning by the same group (Eggemeyer, et al, 1990) for solutions to the planner requirements. In addition, our own work on the scheduling calculus can be used for the schedule maintenance task.

System Monitor

The system monitor provides for the required event and resource monitoring functions. This includes maintaining current resource status for system purposes, and serving as a central repository of event status around which other modules can synchronize. It is expected that some of the fault detection -- or preprocessing for that purpose, such as trend analysis -- would be offloaded from the system fault protection modules to the monitor modules, for efficiency.

Network Manager

All of the rover elements, including the SE, communicate via message passing. The network manager provides the system services to support this activity. The system services provide a message format that characterizes the nature of the communication. The message format includes fields for the message type, the sender, the event tag, and the command or data, and possibly the destination. Based upon message type, the messages are expeditiously forwarded to relevant processes. The sender of a message need not know how a request for data will be filled, or how a message will be routed. For example if the message type is that of a "heartbeat" message from a Resource Controller it is directed by the network manager to the Resource Monitor process of the SE.

The network manager provides for both added security and ease of application programming. The subsystem implementers access all of the rover functions through a uniform interface -- a message protocol. In addition, the network manager implicitly provides the capability to monitor message traffic over the network. Statistics gathered by this process can be used to measure the health of the subsystems on the net. However, care must be taken in the design of this type of support to insure that the service does not overwhelm the computer resources.

SCHEDULING CALCULUS

The overall function of the schedule adapter and monitor processes is to integrate the responses from multiple task activities to determine whether the activities are leading to the given objectives or to an undesirable state or possible fault condition. We are required to assign a qualitative value for the state of the vehicle, based upon quantitative information supplied by the resources. In addition, we are required to judge whether the events reported during the execution of a sequence match the goal set for the task, or whether conditions of the vehicle or in the environment require the sequence to be changed. We chose to investigate the methods of qualitative processing (Bobrow, 1985) as an approach to solving the above problem. In particular, we are in the process of developing a language that implements an arithmetic reasoning tool. The work follows closely that described in the paper *Commonsense Arithmetic Reasoning* (Simmons, 1986). We are using the UNIX[1] yacc procedure (Johnson, 1983) to build the scheduling calculus language. The language combines methods of graph search, interval arithmetic, relational arithmetic, constraint propagation, and function evaluation to arrive at decisions about the validity of a schedule step, and conclusions about the rover state. A capability of the

[1]UNIX is a trademark of AT&T.

language can be illustrated by one of Simmons' examples. Given the ranges of three quantities A, B and C we want to determine the acceptable range for a fourth quantity, D:

A is constrained to the interval [3,4]
B is constrained to the interval [1,4]
C always has the value of 2
D is related to A,B,C by (B*C)/(A+B)
We assert the relation B >= C
B is now constrained to interval [2,4]
Using interval arithmetic we determine D is constrained to [.5,1.6]

REQUIRED KNOWLEDGE BASE

The tool will require a knowledge base that provides functions, parameters, values, and constraints that describe an acceptable operating environment. We call this a model database. The high-level goals provided by the uplink process will provide further functions and constraints with which to reason about the system. The event tokens derived by the planning process and resource sensor data also contribute to the knowledge base. We call this the derived knowledge base. Using the model and derived bases, directed graphs can be built by the language as reasoning tools. Values for the functions are the nodes of the graphs and relations among the functions are the arcs.

REASONING ABOUT RESOURCE STATES

The procedure to reason about the resources' states is simple in principle. Resource Controller output data and state messages are examined to see if they fall within acceptable time and value intervals. The change or lack of change with respect to time is noted. The changes contribute to a table of derivatives that can be used to identify trends in the behavior of a resource or a set of related resources. This trend analysis serves in part as a fault detection device, allowing prediction of future conditions and intervention before some faults occur. Data for the trend analysis is collected and preprocessed locally by the monitor processes, to reduce network loading. Further fault analysis and response is performed by the System Fault Protection module.

EVENT TOKENS

The interactions of the system planner and Activity Managers produce a time ordered set of event tokens which represent steps in the schedule that indicate levels of progress. The event token format is a tag or label, a time interval and expectation values for rover state parameters. The event tokens are assembled into tree structures which are used by the schedule adapter to measure the progress of the sequence.

REASONING ABOUT THE SCHEDULE

Conditions derived from information provided by elements of the event token trees are used by the schedule adapter to trigger the scheduling calculus processes. Below are two simple examples. The first illustrates what may be a common condition in rover operation -- things taking longer than predicted.

With the current node of the event token tree, we have reached the end of one of a series of traverse segments. The duration of a traverse segment was predicted to be 1 hour. The actual time was 1.5 hours, exceeding the specified interval. The sequence calls for an obligatory downlink in an hour. The scheduling calculus procedures are invoked to determine how the sequence and constraint envelopes for the planning of the next traverse segment are to be changed for acceptable productivity while ensuring the vehicle stops for the downlink telemetry at the proper time.

The second example is one that has the vehicle supporting a science experiment. Again, this example is an illustration of the non-deterministic nature of planetary vehicle operations. This example illustrates the requirement for the scheduling calculus to reason about the rover environment.

Objective:	Take a multispectral image of a designated feature along a traverse.

Parameters: Exposure time vs. amount of ambient light, location coordinates, frequencies, objective, and sun angle.

When the rover arrives at the feature, based upon the sun angle and the position of the scan platform, the scheduling calculus procedure will reason whether there will be sufficient light during the time interval for the observation. If there is not, the method can calculate a new exposure time. However the method must also reason whether this exposure time can be used and still meet the overall objectives of the schedule.

STATUS

The scheduling calculus language currently supports the relational, interval, and functional arithmetic, as well as built-in typed functions and the ability to create typed symbols. It can perform operations illustrated by the Simmons example and will soon operate on preconstructed graphs. We are in the process of designing the database specification and interface, and the procedures for automatically constructing the graphs.

CONCLUSIONS

This rover system control architecture proposal is complex because we have tried to offer something which could eventually grow into a comprehensive solution to the problem of maximizing mission return of a rover very remote from Earth, by moving difficult, complex, time-consuming ground processes on-board. Whatever the ultimate solution to the apparent paradox among system, subsystem, and ground control, rover mission complexity will be reflected in the command and control system. It still remains to be proved (1) whether these functions can be implemented, and (2) whether the resultant implementation can be tested well enough so mission managers will allow the system to be used. We are optimistic on both counts.

ACKNOWLEDGEMENTS

The work described in this paper was carried out at the Jet Propulsion Laboratory / California Institute of Technology under a contract with the National Aeronautics and Space Administration.

While the authors have sole responsibility for the contents of this paper, we wish to extend special thanks to the following JPL personnel for ideas and review of the report: Jim Burke, Carol Collins, Sven Grenander, Ken Lambert, Jacob Matijevic, William McLaughlin, Donna Pivirotto, Dave Smith, and P. Richard Turner.

A complete list of credits would be impossible, however the authors wish to thank the following people for ideas: Charles Budney, Curt Eggemeyer, Matt Golombek, Mike Hollander, and Rob Manning.

REFERENCES

Bobrow, D., Editor, (1985). *Qualitative Reasoning About Physical Systems.*

Bourke, R., Kwok, J., and Friedlander, A. (Nov 1989). Design of a Mars Rover and Sample Return Mission, *Proc. of CNES International Symposium on Space Dynamics*, Toulouse, France.

Brooks, R., and Flynn, A., (Oct 1989). Rover on a Chip, *Aerospace America.*

Dias, W., and Gershman, R., (Oct 1987). *A Day in the Life of a Mars Rover.* Jet Propulsion Laboratory Internal Report, D-5075.

Eggemeyer, W., and Cruz, J., (to appear May 1990). PLAN-IT-2: The Next Generation Planning and Scheduling Tool, *Proc. of Goddard Conference on Space Applications of AI, 1990.*

Fesq, L., and Stephan, A., (1989). On-Board Fault Management Using Modeling Techniques, *Proc. Inter-Society Energy Conversion Engineering Conference (IECEC).*

Gat, E., Firby, R., and Miller, D., (July 1989). Planning for Execution Monitoring on a Planetary Rover, *Proc. of NASA Conference on Spacecraft Operations, Autonomy, and Robotics.* Houston, TX.

IKI - Space Research Institute (USSR) (1988). *Martian Rover Motion Control Principles Preliminary Concepts*, Pr-1422.

Johnson, S., (1983). YACC: Yet Another Compiler Compiler. *UNIX Programmer's Manual, Vol. 2.* Bell Telephone Laboratories, Murray Hill, N.J.

Lambert, K., (Oct 3, 1989). A Computational System for a Mars Rover, AIAA 89-3026. *AIAA Computers in Aerospace VII.*

Linick, T., (1985). Spacecraft Commanding for Unmanned Planetary Missions: The Uplink Process, *Journal of the British Interplanetary Society*, Vol. 28 No. 10.

Pivirotto, D. and Dias, W., (1989). *United States Planetary Rover Status-1989*, Jet Propulsion Laboratory Internal Report D-6693.

Rashid, R., (1986). Threads of a new System. *UNIX Review.* Vol 4, No. 8.

Reiners, T., (Aug 1985). *Autonomous Redundancy and Maintenance Management Subsystem (ARMMS) Executive Summary.* JPL internal report D-2414.

Report of the 90-Day Study on Human Exploration of the Moon and Mars, (Nov 1989). National Aeronautics and Space Administration.

Riethle, G., (Aug 1983). *A Summary Overview of Technology Applied to the ARMMS Demonstration Project. Fault-Tolerance Techniques.* JPL internal report D-948.

Riethle, G., (Oct 1983). *A Summary Overview of Technology Applied to the ARMMS Demonstration Project. ARMMS Executive Software.* JPL internal report D-1122.

Rokey, M., and Grenander, S., (to appear Jun 1990). Planning for Space Telerobotics: The Remote Mission Specialist, *IEEE Expert.*

Simmons, R., (Aug 1986). 'Commonsense' Arithmetic Reasoning, *Proceedings of AAAI-86*, Philadelphia, PA.

Simmons, R., and Mitchell, T., (Jul 25, 1989). A Task Control Architecture for Autonomous Robots. *Proceedings of SOAR '89 Conference.*

Smith, D., and Matijevic, J., (Oct 1989). A System Architecture for a Planetary Rover. *Proc. of the NASA Conference on Space Telerobotics, January 1989.* JPL Publication 89-7.

Wilcox, B., and Gennery, D., (1987). A Mars Rover for the 1990's, *Journal of the British Interplanetary Society (JBIS)*, Vol 40, No. 10.