https://ntrs.nasa.gov/search.jsp?R=19990052840 2020-06-15T21:26:55+00:00Z

Mission Operations with an Autonomous Agent

Barney Pell, Scott R. Sawyer, Nicola Muscettola, Benjamin Smith, Douglas E. Bernard

The Research Institute for Advanced Computer Science is operated by Universities Space Research Association, The American City Building, Suite 212, Columbia, MD 21044 (410) 730-2656

Work reported herein was supported by NASA via Cooperative Agreement NCC 2-1006 between NASA and the Universities Space Research Association (USRA). Work performed at the Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, CA 94035-1000

Mission Operations with an Autonomous Agent

Barney Pell[1], Scott R. Sawyer[2], Nicola Muscettola[3], Benjamin Smith[4], Douglas E. Bernard[4]

[5,1,3]NASA Ames Research Center,
Moffett Field, CA 94035
[1]Caelum Research @ Ames
[3]Recom Technologies @ Ames
650-604-4756
pell@ptolemy.arc.nasa.gov

[5]RIACS@Ames

[2]Lockheed Martin Missiles & Space
Advanced Technology Center
3251 Hanover St.
Palo Alto, CA 94304-1191
650-424-2291
scott.r.sawyer@lmco.com

[4]Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
818-354-2597
douglas.e.bernard@jpl.nasa.gov

Abstract—The Remote Agent (RA) is an Artificial Intelligence (AI) system which automates some of the tasks normally reserved for human mission operators and performs these tasks autonomously on-board the spacecraft. These tasks include activity generation, sequencing, spacecraft analysis, and failure recovery. The RA will be demonstrated as a flight experiment on Deep Space One (DS1), the first deep space mission of the NASA's New Millennium Program (NMP). As we moved from prototyping into actual flight code development and teamed with ground operators, we made several major extensions to the RA architecture to address the broader operational context in which RA would be used. These extensions support ground operators and the RA sharing a long-range mission profile with facilities for asynchronous ground updates; support ground operators monitoring and commanding the spacecraft at multiple levels of detail simultaneously; and enable ground operators to provide additional knowledge to the RA, such as parameter updates, model updates, and diagnostic information, without interfering with the activities of the RA or leaving the system in an inconsistent state.

The resulting architecture supports incremental autonomy, in which a basic agent can be delivered early and then used in an increasingly autonomous manner over the lifetime of the mission. It also supports variable autonomy, as it enables ground operators to benefit from autonomy when they want it, but does not inhibit them from obtaining a detailed understanding and exercising tighter control when necessary. These issues are critical to the successful development and operation of autonomous spacecraft.

TABLE OF CONTENTS

1. INTRODUCTION
2. MISSION OPERATIONS OVERVIEW
3. FULLY AUTONOMOUS AGENT PROTOTYPE
4. EXTENDED MISSION
5. MULTI-LEVEL COMMANDING
6. UPDATING INFORMATION
7. TRACKING STATUS
8. SUMMARY OF ARCHITECTURE REVISIONS
9. VARIABLE AUTONOMY
10. RELATED WORK
11. FUTURE WORK
12. CONCLUSION
13. ACKNOWLEDGMENTS

1. INTRODUCTION

NASA's New Millennium Program (NMP) is a series of technology validation missions designed to develop the technologies necessary for the next millennium of space exploration. Deep Space One (DS1) is the first NMP deep space mission. One technology to be demonstrated on the mission is the Remote Agent (RA), an Artificial Intelligence (AI) system which automates some of the tasks normally reserved for human mission operators and performs these tasks autonomously onboard the spacecraft. These tasks include activity generation, sequencing, spacecraft analysis, and failure recovery.

When we initially developed the RA prototype, we focused on automating as much as possible, so as to minimize the required effort of ground operators and enable the spacecraft to achieve mission-critical goals in situations in which human intervention was impossible.

However, as we moved from prototype into actual flight code and teamed with ground operators, we had to address the broader operational context in which RA would be used. This resulted in several major extensions to the RA architecture. First, we developed a method by which ground operators and the RA could share a long-range mission profile, which could be updated periodically from ground as new information becomes available. Second, we added support for commanding at multiple levels of detail, from goal-oriented commanding down to low-level time-tagged sequencing, or even commanding in both styles at the same time. Third, we added mechanisms for ground operators to provide additional knowledge to the RA, to augment or correct its interpretations of the current situation (for example, telling it that a sensor is faulty or that an actuator should not be used). This included addressing the problem of updating parameters without interfering with the activities of the RA or leaving the system in an inconsistent state. Fourth, we added support for monitoring spacecraft behavior at multiple levels of detail, from a high-level alert message when the spacecraft needs help down to low-level streams of debugging data, or even monitoring in both styles at the same time for different aspects of spacecraft behavior.

The resulting architecture enables ground operators to benefit from autonomy when they want it, but does not inhibit them from obtaining a detailed understanding and exercising tighter control when necessary. These issues are critical to the successful development and operation of autonomousspacecraft.

This paper describes our approach to mission operations with an autonomous agent. First, Section 2 provides an overview of traditional mission operations. Section 3 then describes the prototype agent we developed to address scenarios requiring completely autonomous operations. It also summarizes the new requirements we encountered as we began to address the use of this autonomous agent within the context of an entire mission. The next four sections describe how we addressed these requirements by architectural extensions. Section 4 shows how we extended the architecture to support long missions rather than an isolated scenario; Section 5 shows how we provided multi-level commanding rather than all-or-nothing autonomy; Section 6 describes the extensions to enable mission operators to provide the agent or the underlying flight software system with new knowledge and goals without having negative interactions; and Section 7 addresses our support for monitoring the spacecraft at multiple levels of detail. Section 8 summarizes the extensions, after which Section 9 discusses the more general themes in operating a mission with an autonomous agent at different levels of autonomy. Section 10 reviews related work and Section 11 discusses future work. Section 12 concludes the paper.

2. MISSION OPERATIONS OVERVIEW

In order to understand the advances offeredby the remote agent, first consider the traditional approach to mission operations for deep space missions. The following set of activities are typically part of successful mission operations:

Mission planning—Mission planning is the development of a high level mission plan. Since science missions are few and far between, the opportunity to collect data at various stages of the mission is a scarce resource and tradeoffsneed to be made among competing science opportunities. The mission plan collects the conclusions of the many tradeoffs and indicates which investigations are to be accomplished at what time.

Sequence development and spacecraft behavior prediction—Given an accepted mission plan, sequence development is the process of determining exactly how the (higher level) mission plan will be carried out. The product of sequence development is a time-ordered sequence of commands for the spacecraft. These commands may be at a low level, such as powering on a specified device, changing a parameter in the flight software, or opening a valve, or at a higher level such as instructing the spacecraft to turn its high gain antenna (HGA) to the Earth. As part of the sequence development task, the sequences are often tested on a simulation testbed to generate predictions of the spacecraft response to the sequence. If the predicted behavior is unacceptable or undesirable, the sequence may be revised.

Flight/groundcommunications—Sequences ready for use are delivered to the deep space network (DSN) for uplink to the spacecraft. The DSN also collects telemetry from the spacecraft for distribution to scientists and spacecraft engineers. Typically, communications passes are scheduled to occur at specific times. During these times, the spacecraft's high gain antenna and a DSN antenna are aimed at each other and configured for communication. Sequences are uplinked, while spacecraft and science data are downlinked. Outside of the scheduled passes, the spacecraft typically keeps its transponder powered and listens for low rate uplink from the ground with its omnidirectional antenna. The spacecraft may also broadcast low rate telemetry during these periods, but it has no assurance that anything has been received on the ground. Consequently, the spacecraft does not generally have the ability to request prompt help from the ground in the case of anomalies.

Spacecraft behavior monitoring and analysis—Engineering telemetry is monitored as it comes in for any alarm conditions indicating unexpected or serious conditions aboard the spacecraft. In addition, more extensive engineering telemetry is stored and analyzed by spacecraft engineering subsystem experts to monitor the health of their subsystems and spot any important trends in the data. The telemetry data is compared with the earlier predictions of sequence behavior and any significant differences are analyzed.

Anomaly investigation and recovery actions—In the event of an anomaly on the spacecraft, operations personnel investigate the situation, bringing in subsystem experts as needed. Action may be needed to assure that the spacecraft is kept safe until the anomaly can be fully understood and recovery actions developed and implemented.

Software upgrades—For long missions, software upgrades are a planned part of flight operations. Even for shorter missions, the capability to upgrade the flight software is an important insurance policy should correctable problems surface. Quantities that are expected to need adjustment periodically are managed through parameter updates, a much simpler process which traditionally involves writing new values directly into known memory addresses. Ground operators must also manage the storage of critical parameters and flight software in non-volatile memory (NVM), in order to recover from hardware/softwareresets. Therefore, there may be separate procedures for updating parameters and software in dynamic memory (DRAM) versus updating reset and recovery data in NVM.

Science data storage and retrieval—The purpose of the deep space missions is science. Part of the operations job is assuring that the scientists have timely and easy access to the returned data.

Sequence execution—Typically, a sequence containing dozens or hundreds of individual commands is uplinked to

the spacecraftas an entity and then an on-board process is responsible for metering out the commands each at its scheduled time. This is known as sequence execution. In addition, most spacecraft have the ability to accept real-time commands.

Safe Modes—Deep-space spacecraft design typically includes one or more safe modes. These are low capability operating modes requiring a minimum of hardware components and capable of maintaining spacecrafthealth for periods of days or weeks until ground operators can determine what actions are needed to return the spacecraft to full operation.

On-board fault protection—Given the long minimum response time for deep space missions, there are some possible component failures that require an on-board response in order to prevent loss of mission. Fault protection systems have been designed to handle these situations. In some situations, the on-board system will be able to reliably diagnose the situation, take reparative action, and return the spacecraft to operation. In other situations, the fault protection is designed to fail-safeand will put the spacecraft in one of its safe modes until operator intervention is possible. This requires first aborting any current sequence and results in loss of mission activities while the spacecraft is in this safe mode.

The mission operations approach described above makes use of many highly trained and capable individuals in order to pack as much science data gathering as possible into each unit of time and to establish a high level of certainty that the system is always working as intended.

3. REMOTE AGENT PROTOTYPE

In mid-1995, we developed a prototype of an autonomous agent for spacecraftcontrol, called the Remote Agent (RA). The RA prototyping effortfocused on automating as much of ground operations as possible, so as to minimize the required effortof ground operators and enable the spacecraft to achieve mission-critical goals in situations in which human intervention was impossible. If we could achieve full autonomy in a general and reusable manner, we could then proceed to extend this technology to support broader ranges of operational missions.

Cassini and SOI

Our development effort built on the work done for the Cassini mission, the most autonomous spacecraftdesign to date. The Cassini spacecraft launched in October, 1997, on a mission to study Saturn, Titan, and the rest of the Saturnian system. The spacecraft is designed to be in contact with the Earth for only 8 hours, once a week, except during high activity periods. Planning is done almost entirely on the ground leading to tightly packed commanded sequences of events that are uplinked to the spacecraft.

As a sequence is executed on-board, the commands are expanded into actions by the on-board executive. Activities of the objects comprising the flight software are coordinated by use of a central mode commander. If the resources required for a newly commanded mode are not available, the mode commander makes them available. An advanced configuration manager algorithm controls the power, health, and operating mode of all hardware assemblies. A geometric constraint monitor algorithm enforces sun-pointing and other constraints and readjusts pointing profiles that would violate those constraints [1].

Replanning is not generally available, but a primitive version of replanning has been created to handle plan-breaking failures during critical sequences. Consider any failure leading to main engine shutdown during Saturn orbit insertion (SOI). Here the prime engine is swapped to the backup, the (original) sequence is restarted, and the flight software uses stored information to know that this is a restarted sequence. The flight software also recomputes how much additional velocity change is required from the new burn given that some velocity change is already accumulated and that the new burn will occur in a differentlocation than that originally planned.

The Cassini design included just enough autonomy to solve the SOI problem, but the approach was a special, rather than general purpose one. The main limitation is that the Cassini softwarehad no capability for mission planning or for generating its own command sequence. Hence, the Cassini softwaredevelopers designed a special sequence for just the SOI scenario. They then designed the flight software (FSW) so that it could retry the same sequence several times if it failed. To do this, the FSW had to interpret commands differently depending on context. Hence, the flight software and the special SOI sequence were all designed carefullyand specifically for this one phase of full autonomy on this one spacecraft. In order to extend the approach to support full autonomy in different phases, and to make the approach applicable to other spacecraftwith less up-front design effort, it was necessary to endow the spacecraftwith an ability to generate sequences on-board. With this ability, the spacecraft could respond to failure of a sequence by generating a new sequence based on the current situation.

Remote Agent Prototype

The RA Prototype [2] extends the foundations of the Cassini autonomy approach by replacing some of the functions with technologies developed from Artificial Intelligence (AI). The RA architecture, shown in Figure 1, contains three components, with responsibilities as follows.

The Planner/Scheduler (PS) trades off goals, breaks down high-level goals into mid-level activities, orders activities to resolve interactions, and ensures the plan is consistent with resource constraints.

The Smart Executive (EXEC) invokes planning when necessary and then issues a sequence of commands to FSW to implement the high-level plan, making detailed sequencing choices in a context-sensitive fashion, and tracking resources and configuration as does Cassini's Executive.

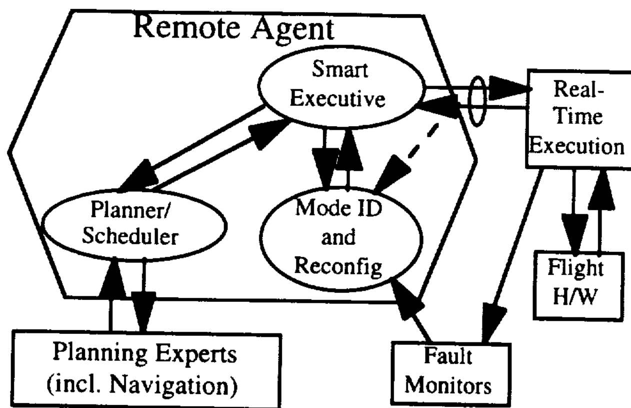


Figure 1. RA Prototype Architecture

The Mode Identification and Reconfiguration (MIR) component tracks execution activity. Based on the commands from EXEC and the outputs of the monitors, MIR infers the most likely current state of the hardware and passes its conclusions to EXEC. MIR also assists EXEC in repairing devices by suggesting actions based on the global context. All capabilities are performed using models of the spacecraft, rather than a set of hard-coded rules like those used in the Cassini approach.

The top-level operational cycle, including the planning loop, is described as follows. EXEC requests a plan, when necessary, by formulating a plan-request describing the current plan execution context, and then executes and monitors the generated plan. EXEC executes a plan by decomposing high-level activities in the plan into primitive activities, which it then executes by sending out commands, usually to the real-time flight-software control system (FSW). EXEC determines whether its commanded activities succeeded based either on direct feedback from the recipients of the commands or on inferences drawn by the Mode Identification (MI) component of MIR. When some method to achieve a task fails, EXEC attempts to accomplish the task using an alternate method in that task's definition or by invoking the Mode Reconfiguration (MR) component of MIR as a "recovery expert." If MR finds steps to repair the failing activity without interfering with other concurrent executing activities, EXEC performs those steps and then continues on with the original definition of the activity. If the EXEC is unable to execute or repair the current plan, it aborts the plan, cleans up all executing activities, and puts the controlled system into a stable safe state (called a "standby mode"). Since the prototype was designed for complete autonomy, EXEC then requests a new plan while maintaining this standby mode until the plan is received, and finally executes the new plan.

We now briefly describe the plan representation and plan execution capability in more detail. The planner represents spacecraft activities as a set of concurrent subsystems. Each independent component of a subsystem is conceptualized as a *state variable*, which can take on a series of different behaviors over time. A plan consists of one *timeline* for each state variable. Each timeline contains a sequence of constraints on the behavior of the state-variable. A *token* is a data structure which represents one part of a sequence on a timeline. A token has information about the desired behavior throughout the duration of the token, and also flexible constraints on when the token can start and finish. Each token has a unique name that unambiguously identifies it in the mission profile, and a type, an assertion that describes the activity and its parameters. Lastly, the plan contains constraints, called *compatibility* constraints, to coordinate behavior across tokens on different timelines. An example of a compatibility constraint is one which says that a "take-picture" token must be executed within the window during which the corresponding "keep-pointing-at-target" token is activated.

The EXEC is a multi-threaded process that is capable of asynchronously executing activities in parallel. EXEC has one thread for each timeline in a plan, and a procedure, called the *token definition*, for each type of token contained in the plan. A token definition procedure contains a precondition that must be met before the activity can start, a postcondition that must be met before the activity can finish, and a body which describes how the procedure is actually executed. To execute a plan, EXEC activates on the corresponding thread for each timeline the procedure corresponding to the first token on that timeline. When a new token is able to start (because the previous token has finished and all other constraints are satisfied), EXEC terminates the previous token procedure and transitions to the next one. For example, once the token for turning to a target has completed, the token for constantly pointing at the target can then be activated. This enables the "take-picture" token on the camera timeline to be activated. Only when the picture activity has finished will the EXEC terminate the "keep pointing at target" token and transition to the token for turning to the next target attitude.

Table 1 provides a summary showing how the RA supports some of the mission operations functions for a particular scenario of fully autonomous operation.

DS1 and Mission Operations

In October 1995, the Remote Agent prototype was successfully demonstrated on a simulated version of the SOI scenario. The agent autonomously generated and executed a plan based on the scenario goals and constraints and was able to replan in the face of a wide variety of failures (including unintended errors in the simulator) and still achieve the orbit insertion within the constrained time window. This success resulted in the inclusion of the RA as an autonomy experiment on the first NMP mission, Deep Space 1 (DS1), which is scheduled to launch in mid-1998. A successful demonstration on DS1 would make RA the first AI system to autonomously control an actual spacecraft.

Although the RA will only fly DS1 as a one-week experiment [3], we designed it to support DS1 as a full operational mission, including a usable flight/ground interface. Because the RA Prototype only dealt with full autonomy for one scenario, our prototype did not address issues to do with ground operation or the need to function beyond one short scenario.

Table 1. RA prototype's role in mission operations

Mission operations function	RA prototype approach in fully autonomous scenario
Mission planning	Humans supply goals for the scenario to EXEC
Sequence development	EXEC passes current state and any unachieved goals to the PS as part of a plan request. PS generates plan on-board; final level of sequence generation is performed by EXEC as part of executing the plan.
Spacecraft behavior prediction	PS generates high-level predictions as part of planning process; they are stored in plan. MIR generates detailed predictions on-board based on commands from EXEC and current state information.
Flight/ground communications	Content of uplink and downlink were not addressed in the full-autonomy scenario.
Spacecraft behavior	Monitored on-board by MIR and monitors; support for ground monitoring was not addressed.
Anomaly investigation and recovery actions	MIR assists EXEC in diagnosing and recovering from anomalies; ground is notified of anomaly and actions taken; RA will put spacecraft in standby mode if necessary, and replan when possible.
Software upgrades	Not addressed.
Science data storage and retrieval	Data was stored in traditional manner; retrieval was not addressed.
Sequence execution	Executive expands planner tokens into low level commands and manages execution.
Safe modes	RA standby modes are equivalent to safe modes.
On-board fault protection	Executive has reflex actions for critical failures; MIR assists in diagnosis and recovery if possible; if no recovery is possible within the current sequence, the sequence is aborted and EXEC asks PS for a replan.

To extend the RA prototype into an autonomous agent for mission operations, we had to address the following issues.

First, the agent would no longer function solely in an isolated scenario, and would need to be part of the overall mission design and operations philosophy. The agent would be required to execute a long-range mission plan. Issues to be addressed include: (a) When do we plan, and how far in advance, (b) Which goals do we give to the planner, and (c) How to make sure each short-term plan leaves sufficient resources for rest of mission?

Second, ground operators want to be able to interact with the spacecraft at various levels of detail, rather than accepting full-autonomy at all times. The following capabilities were determined to be important to mission operators concerning the use of an agent:

- Sequence the spacecraft as in traditional operations in some situations, use the on-board planner in other situations, or do both at the same time.
- Use the planning capability either from the ground or on-board the spacecraft, and change planning modes dynamically. It was also important to specify policies for when the agent should replan autonomously, and when it should wait for help from ground.
- Force the system into a stable, predictable state from which operators could perform analysis.
- Disable various fault protection responses in order to send low-level commands to repair the spacecraft.
- Provide additional knowledge to the agent, to augment or correct its interpretations of the current situation.
- Change any parameters in a consistent fashion: so that agent's plans and beliefs are consistent with the current state after parameter modification, and so that the parameter change is executed at a point where it does not interfere with the processes of the agent.
- Track spacecraft health and status.

As the spacecraft initiates activity at higher levels, it becomes harder for ground operators to predict which low-level state it will be in at any time. This makes it very difficult to coordinate the actions initiated by ground with those initiated by the spacecraft, to make sure they do not interfere with each other. There are four ways to address this synchronization problem:

- Force the spacecraft into a certain state before issuing a command.
- Set up an explicit mechanism to ensure that the command is executed in the appropriate situation.
- Provide information to the agent as advice, rather than commands, which the agent can address if and when it needs the information.
- Only change low-level details which the agent does not care about.

Each of these approaches has strengths and weaknesses. The forced-state approach causes the spacecraft to interrupt everything, which may cause loss of important mission activities. The synchronization mechanisms are potentially risky as well, as they rely on the correct situation coming about, but it is hard to guarantee situations in the presence of potential faults. The other two approaches are less problematic, but they also give operators less control.

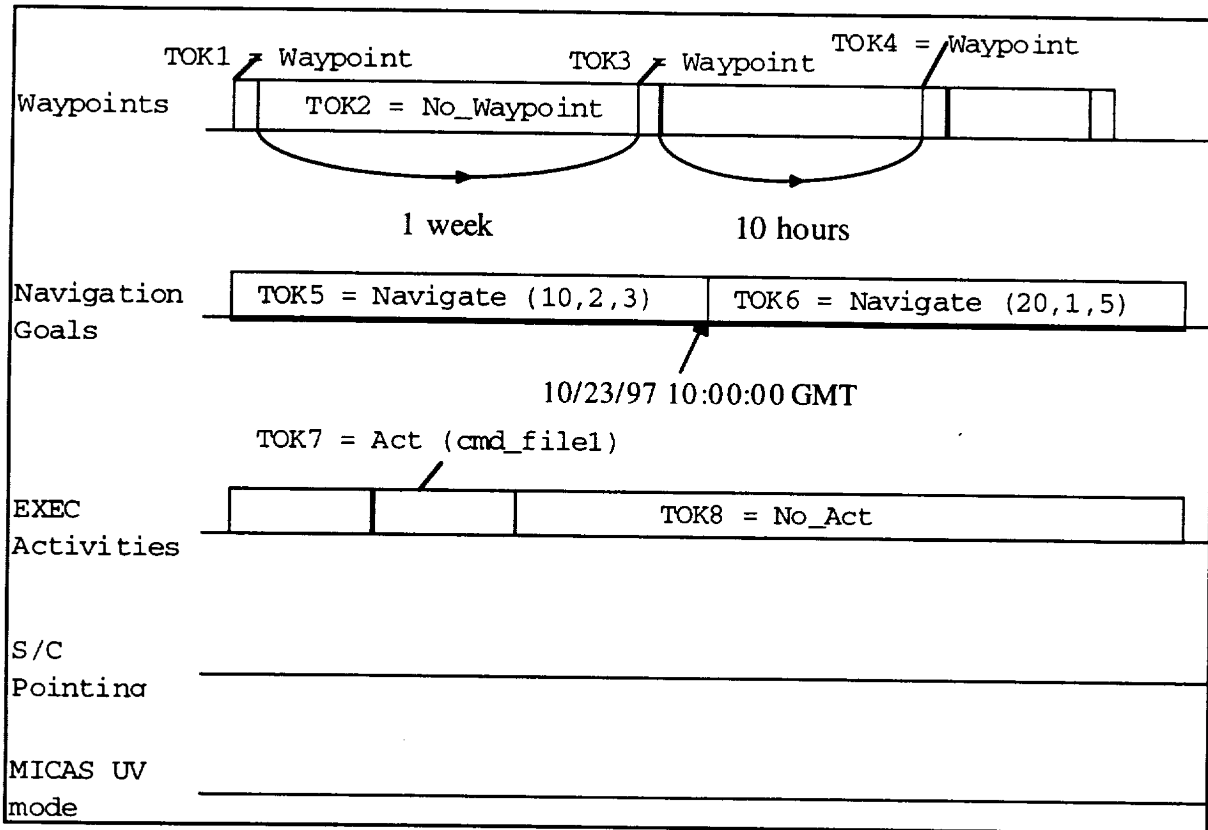


Figure 2. An example of a mission profile

The following sections describe how we addressed the top-level categories of extended missions, multi-level commanding, multi-level monitoring, and updating information. As we discuss each capability, we also point out which approach we took to the relevant synchronization problems.

Note that for the remainder of this paper, we will be describing the full RA architecture. The RAX architecture implemented for DS1 represents only a subset of the capabilities defined for the full RA architecture.

4. EXTENDED MISSION

To operate without any intervention over months at a time, an autonomous spacecraft needs a persistent memory of the goals to be achieved. For example, in a pure "fire and forget" mission, the goals would be loaded on the spacecraft at launch. The spacecraft would then achieve them at the appropriate times by generating on-board sequences of commands that take into account the current spacecraft operational conditions. In practice, goals will be modified or updated during the course of a mission, with the frequency of these updates decreasing as the level of autonomy of the spacecraft increases.

Mission Manager and Mission Profiles

In the RA, the software module responsible for storing and manipulating the mission goals is the Mission Manager (MM). Goals are stored in a temporal database called the *mission profile*. Figure 2 shows an example of a mission profile with the goals and types of timelines relevant to the RA experiment on DS1.

The mission profile is partitioned into a series of parallel timelines. Later in the paper we will discuss in detail the `Waypoints`, `S/C Pointing` and `MICAS UV mode` timelines. The `Navigation goals` goal timeline specifies the frequency, duration and placement slack of the "optical navigation windows", the times during which the spacecraft is requested to take a set of asteroid pictures to be used for orbit determination by the on-board Navigator. The `EXEC activities` timeline contains goals that specify a request for execution of a "mini-sequence", i.e., a set of lower-level commands that the executive will issue to the real-time software. The mini-sequence must be activated with certain synchronization constraints with respect to other planned activities.

The organization of the profile into parallel timelines is similar to that of a plan generated by the planner. Each mission profile timeline spans a long time horizon (potentially the entire mission) and can be (totally or partially) filled with tokens. Each token corresponds to a goal that needs to be achieved over an appropriate time interval.

Goals can be temporally constrained either relative to other tokens (e.g., in Figure 2 'TOK3 must start 1 week after the end of TOK1') or with respect to absolute time (e.g., again

in Figure 2 'TOK6 must start at 10:00:00 GMT on October 23, 1997'). Temporal constraints can include slack (e.g., 'TOK1 must start between 10 and 20 minutes after the end of TOK2'). This temporal flexibility in the mission profile is important. In fact the planner has more latitude in achieving a set of independently specified goals with flexible time than one with rigid achievement time. This increases the probability that the planner will be able to produce a consistent plan and therefore overall mission robustness.

The goal editing interface provides commands that ground operators can use to add, modify or delete goals from the mission profile. For example, in a mission in which the spacecraft communicates to Earth through the DSN, the final communication schedule allocated to the mission will become available only a few weeks in advance and it is possible that a schedule may change with a short notice (e.g., within a week). Ground will need to communicate both of these situations to the spacecraft by issuing appropriate edit commands to modify the mission profile.

By managing goals over very long time horizons, the mission manager achieves two functionalities that are important for autonomous spacecraft operations. First, a mission profile can contain goals that require synchronization that spans longer than a single plan scheduling horizon. For example, it is possible that a scientific experiment may require an instrument cool-down activity to start several weeks before the experiment is actually performed. By covering several of (and potentially all of) the scheduling horizons of a mission, MM will correctly dispatch the two goals to be performed at the appropriate time. Second, a long-term mission profile allows the minimization of up-link communication with Earth, potentially enabling "no uplink" missions if the profile covers the entire mission duration.

Waypoints in the mission profile

The mission profile is structurally homogeneous with a partially specified plan. For example, in Figure 2, the `S/C Pointing` and `MICAS UV mode` timelines correspond to low level activities that the planner will fill up. In fact, the mission profile must comply with the same domain model used by the PS. For example, the vocabulary used to define goal token type is restricted by the PS model, as are the timelines on which tokens of specific types can occur.

In principle, it would be possible for the PS to directly operate on the mission profile and generate a plan for the entire mission. In practice, this is undesirable. Generating a plan is computationally expensive and the cost typically grows exponentially with the number of goals to be achieved. Moreover, spacecraft subsystems are not always available or operating nominally (e.g., an on-board camera is temporarily unavailable due to a recoverable fault or the efficiency of on-board power generation degrades over time). An exact prediction of when system degradation and recovery will occur is clearly impossible.

For these two reasons, the mission profile includes a `Waypoints` timeline. This timeline is typically filled with tokens of two kinds; `Waypoint` and `No_Waypoint`. The first kind of token represents a synchronization event which marks the nominal start and end of each scheduling horizon. When asked to produce a new plan, the MM identifies the next applicable waypoint in the mission profile, selects all goals that can possibly fall between the requested horizon start and the waypoint, and then merges them with the initial state provided by the executive.

Waypoint tokens can appear in the mission profile with any frequency that is convenient for a particular mission phase. As mentioned before, the frequency is a function of the expected number of goals per profile and expected plan brittleness. For example, during an encounter phase several activities relating to scientific experiments need to be planned over a few hours. However, during cruise, only a few regular activities per week may be needed. Accordingly, waypoints should be separated by a few hours during encounter and by a few days during cruise.

Waypoints need not be fixed with respect to specific times of occurrence. This is because the technology used to implement the mission profile allows posting relative temporal constraints between waypoints (e.g., 'Waypoint token TOK3 starts one week after waypoint token TOK1'). This has important consequences on the robustness of the mission profile to variations of the temporal occurrence of significant mission events.

For example, it is not unusual for a mission to have its launch delayed. If waypoints in the mission profile were scheduled at fixed times, quite a bit of last minute editing of the profile could be needed. With relatively constrained waypoints, the actual time of occurrence of launch would be communicated to the MM immediately after launch. The MM would need only to post an absolute temporal constraint on the database that fixed the time of occurrence of a token representing the launch event. Temporal constraint propagation in the database will then infer for each waypoint the actual time of occurrence. This process occurs automatically, with no operator intervention. The elimination of manual intervention increases the precision of the timing of critical mission activities and removes a possible sources of human error, therefore increasing reliability. Besides marking the start and end of nominal scheduling horizons, waypoints also provide an important periodic checkpoint function.

For example, the consumption of non-renewable resources (such as fuel) needs to be appropriately distributed over the different mission phases. This avoids having excessive early use endanger important goals to be achieved later. The PS must take into account such budget constraints to determine which goals should be achieved during each scheduling horizon.

Budget constraints are translated into specific checkpoint conditions (e.g., the fuel level at the end of the scheduling horizon must be no more than 10 kg less than that at the

beginning) which are synchronized with the waypoints. Such synchronization can be achieved in two ways. For special conditions that occur only once in a mission, each checkpoint condition can be represented in the mission profile as an explicit token that is then relatively temporally constrained with respect to a waypoint. For routine checkpoint conditions, the domain model can include explicit compatibility constraints associated with the definition of a waypoint token type. When PS is called to generate a plan, it automatically generates tokens to satisfy the compatibility.

Uninterrupted operation across waypoints

It is important to note that waypoint tokens are completely transparent to the rest of the RA beyond MM and PS. In other words, execution of plans proceeds seamlessly without interruptions across waypoints. There is no need to put the spacecraft in a special state at a waypoint while the executive terminates the execution of the current plan and starts the next one.

Each timeline of the plan from the previous scheduling horizon ends with a token that straddles the waypoint token, i.e., a token whose start time is constrained to occur before the waypoint and whose end time is constrained to occur after the waypoint. When the executive requests a plan for the next scheduling horizon, it sends to the MM the last token of each timeline for the current plan and the constraints between each token and the end waypoint.

Since the end waypoint of the previous scheduling horizon is the start waypoint of the new scheduling horizon, the inclusion of the information passed by the executive in the plan request guarantees that the end of the old plan and the start of the new one always have tokens that completely agree (in name and type).

When installing the new plan, the executive possibly needs to update information on the duration of the ending tokens. Since the end of a straddling token is constrained to occur after the waypoint, these adjustments do not interfere with the execution of the current plan. Since the installation of a new plan occurs before the end waypoint of the current scheduling horizon, plan execution can be handed over from one horizon to the next without any interruption.

Types of mission profile timelines

As mentioned before, a mission profile has exactly the same timelines that a plan has. We can distinguish between four distinct kinds of timelines.

A *goal* timeline will contain the sequence of high-level goals that the spacecraftcan satisfy (e.g., the Navigate goal described before). Goal timelines can be filled either by ground operators or by on-board planning experts seen by the PS as goal generators.

For example, in order to generate the portion of the plan that commands the IPS engine, the PS interrogates NAV which returns two types of goals: the total accumulated thrust time for the scheduling horizon and the thrusting profile to be followed. These two types of information are laid down on separate goal timelines.

Expected device health information over time is tracked by *health* timelines. The expected profile is communicated by the EXEC to the MM in the initial spacecraftstate. EXEC can communicate that the health of a device has changed even if no fault has occurred.

For example, in a previous faulty situation, ground may have decided that the IPS engine is not trustworthy and therefore should not be considered operational until further tests have been run. PS will therefore generate plans that do not involve thrusting the engine. Ground may decide to run some tests by posting appropriate goals in the mission profile and therefore not break the nominal plan execution. After the tests, ground may decide that the IPS engine is trustworthy after all and send a message to EXEC that it is again OK to thrust. EXEC communicates this to the PS through the health timeline in the next scheduling horizon, without needing to interrupt regular plan execution and put the spacecraft in standby mode.

Another kind of state variable is an *internal* timeline. These are only used by the PS to internally organize goal dependencies and subgoaling.

Finally, an *executable* state variable corresponds to tasks that will be actually tracked and executed by EXEC.

The distinctions between types of state variables are important for software engineering and project organization purposes (e.g., the mission design team needs only to know about the structure of the goal timelines to command the spacecraft, but does not need to know the details of the executable timelines which are interesting to the EXEC team, and vice versa). However, notice that the mission profile treats all timelines homogeneously within the same data structures. The same editing and constraint propagation algorithms apply homogeneously to all timelines.

From the perspective of ground operations, this has a major advantage. If necessary (e.g., if once-a-mission special maneuvers are needed in special circumstances), it is quite possible to override the "organizational" boundaries. For example, the ground team could force certain behaviors out of the spacecraft by including in the mission profile explicit tokens on an executable timeline. Notice, however, that in this case. ground does not need to uplink a complete plan. The additional tokens will be treated by the PS as goals, will be checked against the internal PS model and missing supporting tasks will be automatically expanded to create an overall consistent plan. This will greatly facilitate the work of the ground team.

5. MULTI-LEVEL COMMANDING

A key requirement of the ground operators is the ability to control the level of autonomy granted to the spacecraft at any point in the mission. This allows the ground operators to

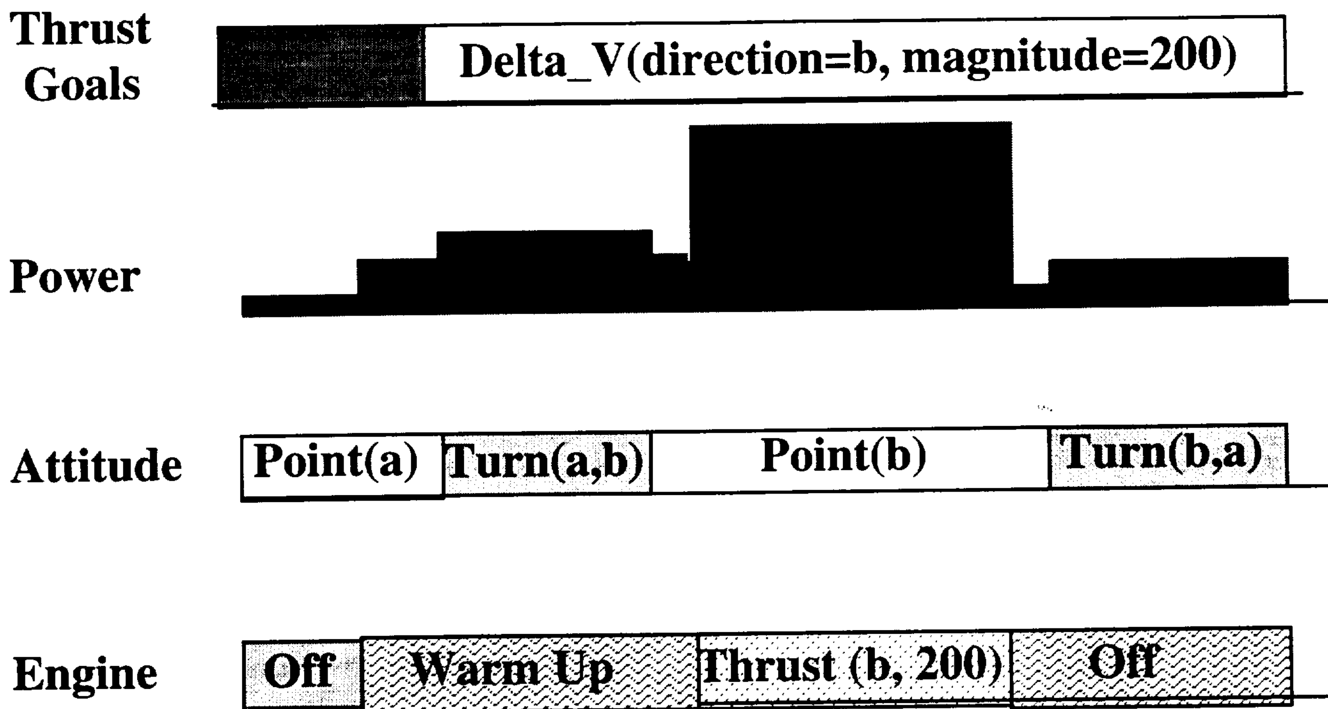


Figure 3. A plan fragment

build confidence in the RA by starting with very little autonomy and gradually increasing to higher levels. The ground team can vary the level of autonomy to match their risk/cost assessment. For a particularly critical and risky maneuver, they may want to pay the higher cost of total ground control. They may also want total control when diagnosing and repairing a particularly troublesome failure, such as a partially deployed solar panel or antenna. For other maneuvers, they may want to save money by granting the RA full autonomy. Of course, there will be some cases where autonomy is a necessity regardless of the cost/risk trade off. The Cassini SOI scenario is an excellent example of such a case.

The level of autonomy can be changed by the ground operators as needed throughout the mission, and even on an activity by activity basis. It is determined by how the spacecraft is commanded, rather than by changing the operating mode of the RA. If the ground provides only high level goals, the RA has maximal autonomy in determining how to achieve those goals. As the ground specifies *how* to achieve the goal in more detail, the RA has less autonomy. The various possible autonomy levels are described in more detail in the subsections below. The mechanisms used by ground operators to uplink the necessary information for commanding are discussed in the next section.

Full Autonomy

At one extreme, RA is granted maximal autonomy. The ground operators provide the RA with a set of high-level goals and the RA decides how to achieve them. The ground operators decide what the spacecraft should be doing, and the RA decides how to do it based on the current spacecraft state and the operation knowledge and policies encoded in the RA models. The spacecraftis most robust to faults at this level of autonomy. When a fault or contingency arises, the RA has the authority to choose alternate ways of achieving the goals that avoid or resolve the problem.

The RA is commanded at this level of autonomy as follows. Over the duration of the mission, the EXEC iteratively requests the PS to generate a plan for each scheduling horizon. The EXEC projects what the spacecraftstate will be at the start of the next horizon, based on the plan it is currently executing, and passes this to the PS as the initial state for the new plan. The MM extracts the goals for that planning horizon from the mission profile. The PS then generates a plan that achieves those goals from the initial spacecraft state provided by the executive. Ground still has full access to the mission profile editing capabilities so it can modify goals as desired.

Figure 3 shows a fragment of a plan generated by the PS. The `Delta_V` goal is satisfied by the `Thrust` token on the `Engine` timeline. The latter token requires the spacecraft `Attitude` timeline to hold the attitude `b` throughout the engine burn by executing token `Point(b)`. The other tokens on the `Attitude` and `Engine` timelines make sure that the spacecraft executes the appropriate auxiliary activities. The `Power` timeline guarantees that the power requested by the plan's activities does not exceed the total power available.

For most goals, the PS has flexibility in deciding how to achieve the goals. If the initial spacecraft state indicates a problem, the PS will generate a plan that achieves the goals in a way that avoids the problem. For example, if the initial state indicates that the camera is stuck on (it cannot be turned off), the PS will generate a plan that does not try to turn the camera off and makes accommodations for the additional power drain by rescheduling activities, turning off low-priority devices, etc.

The EXEC installs the generated plan at the end of its current plan. When the first plan finishes, it begins executing the second as if it were executing one continuous plan. The EXEC has autonomy in deciding how to execute the tokens in the plan. If a problem occurs, the EXEC has authority to try alternative command sequences to achieve the "main engine is on" state. For example, if a valve is stuck shut on the primary feed line, it can try to open the secondary feed line instead.

These recovery actions take time. The tokens in the plan have flexible start and end times. Recovery actions simply push back the end time of a token, and the EXEC adjusts the start times of downstream tokens accordingly. If the recovery actions take so much time that the EXEC cannot finish executing the token by its latest end time, then the token has failed. The rest of the plan is invalid at this point, so the EXEC aborts the plan and achieves standby state. This is a safe state that the spacecraft can remain in indefinitely. The EXEC then asks the PS for a new plan. If the token that failed could not be executed because of a failure diagnosed by MIR, then the appropriate timeline in the initial state passed to the PS is set to reflect the failure.

Partial Autonomy

At the next level of autonomy, the ground operators have more control over how the PS achieves its goals, but the EXEC still has full autonomy in deciding how to execute each token in the plan. This level of autonomy is achieved by specifying more details in the mission profile. As the partial plan in the mission profile becomes more detailed (closer to a complete plan), the PS has less autonomy in deciding how to expand the profile into a complete plan.

The profile can be detailed in some places, and less detailed in others. This allows the ground to determine just how much autonomy they want to grant the PS at any part of the mission, or even for particular activities. For example, the profile may specify the post-launch checkout and some critical science activities in great detail, but leave the cruise activities vaguely specified (e.g., cruise for three months stopping every third day for four hours to take optical navigation images).

At some parts of the mission, the ground operators may want to execute some activity that had not been considered before launch. Perhaps the solar panel is only partially deployed and they want to jar it loose with a high-speed turn that would normally be disallowed. Or perhaps they want to run a particular sequence of activities to diagnose a failure in more detail than the RA can provide. If the activities cannot be specified with a partial plan in the mission profile, then the ground can use a special token called `Exec-Activity`. This token executes a file of arbitrary commands in the executive sequencing language (ESL) [4]. This allows the ground to execute any sequence it wants. The token is added to the mission profile. It can be forced to execute at a specific time by specifying an absolute start time for the token. It can also be synchronized with other tokens in the mission profile by adding a temporal constraint (e.g., start 30 seconds after the next optical navigation window). Since the PS does not know what commands are in the file, the ground must ensure that the `Exec-Activity` token will not conflict with other activities in the plan. For example, a high-speed turn to jar loose ~~the solar array~~ should not be scheduled at the same time as an optical navigation image that requires the spacecraft to be in a fixed attitude.

Ground-based Planning

At the next level of autonomy, the plan is fully specified by the ground operators. The plan is generated on the ground, either by hand or with the help of a ground-based copy of the on-board planner. The plan is then uplinked to the spacecraft and the executive is placed in SCRIPTED planning mode. The transition to operations using a ground-based plan must be made from a standby mode in order to ensure that the ground generated plan has the correct initial state. The spacecraft may be explicitly placed into standby mode by commanding the executive to terminate the current plan and achieve standby mode. Otherwise, the spacecraft will enter standby mode automatically if a `Plan-Next-Horizon` token is encountered on the `Planner Processing` timeline when the spacecraft is in SCRIPTED planning mode.

A number of ground-generated plans can be chained together by including `Script-Next-Horizon` tokens on the `Planner Processing` timeline in the ground generated plans. This approach to chaining assumes that one script's final state (as predicted on the ground) can be used as the initial state for the next script. If there is any inconsistency, the chaining will fail and the RA will place the spacecraft in standby mode and seek assistance from ground. Typically, the last ground-generated plan in the chain resets AUTONOMOUS planning mode with an `Exec-Activity` token. This allows the on-board planner to resume plan generation for each new planning horizon.

The advantage of generating plans on the ground is that the ground operators can specify exactly how each goal is achieved, and they can verify the plan before it is executed. This is useful for special operations, such as resolving a particularly difficult failure (e.g., a stuck solar panel or partially deployed antenna). It is also useful for building confidence in the RA's plan execution capability before switching to autonomous operations.

Sequencer

Finally, the ground can send command files to be executed directly by the EXEC via the `Exec-Activity` token.

The file consists of time-tagged commands in ESL. In this mode of operation, the files are executed from standby state while no plan is running. This commanding mode is equivalent to the time-tagged sequencing used on traditional missions. The EXEC has no autonomy in deciding how to execute the commands, and the PS is not involved at all. The ground has complete control over the spacecraft. If the simplest ESL commands are used, the commands go directly to the spacecraft subsystems with almost no processing by the executive.

This mode will primarily be used during the early, confidence-building phase of the mission. Thereafter, it will remain as a fall-back mode. In case a problem develops with the higher functions of the RA, the spacecraft can be commanded directly at the "brain stem" level. It is also useful for testing. The various subsystems and hardware can be tested through low level command scripts without a fully functional RA.

Real-time commanding

Commanding in real-time is achieved using an implementation of the `Exec-Activity` functionality as an immediate function invocable by ground. The file containing the messages is uplinked to the spacecraft, just as it is for use with the `Exec-Activity` token. Then, the message EXEC_EVAL(filename) is sent to the EXEC. This causes immediate execution of the file contents without the overhead of a mission profile update or a plan generation. It has the disadvantage of running in parallel to all other spacecraft activities with no constraint checking. The ground operators must be fully responsible for any negative interactions resulting from the asynchronous commands.

Summary of operations mode transitions

Another way of looking at multi-level commanding is as follows. Both full autonomy and partial autonomy are levels in which the planning mode is AUTONOMOUS. The on-board PS is generating the plans. The difference between the two is entirely in the level of detail specified in the mission profile. Therefore, we can define an *operations mode* called 'autonomous operations.'

Next, consider the ground-based planning and sequencer levels of autonomy. In both, script files are uplinked and executed by the EXEC, entirely by-passing the PS. The planning mode is SCRIPTED. The differencebetween the two levels is only in whether any tokens other `Exec-Activity` and `Script-Next-Horizon` are present in the scripts. These levels can be collectively considered as an operations mode called 'scripted operations.'

When the spacecraft is in standby mode, that can be considered to be its operations mode. This mode can be reached not only when the planning mode is STANDBY, but also automatically in response to plan failures or anomalies. Note that the real-time commanding level of autonomy does not have its own operations mode, but can be invoked by ground operators in any operations mode.

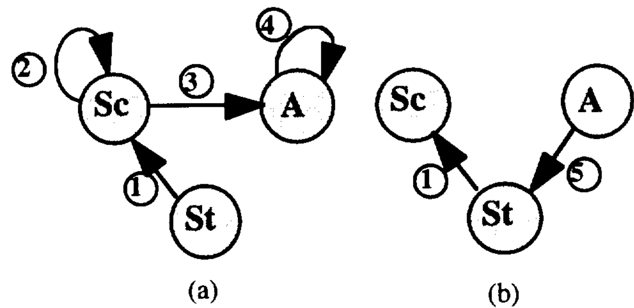


Figure 4. Operations mode transitions for normal operations
(A - autonomous, Sc - scripted, St - Standby)

Given the above defined operations modes, we can give a straightforward summary of spacecraft operations. Transitions between operations modes are typically controlled by changing the planning mode.

Figure 4a shows the expected sequence of operations modes during the early phases of a mission. Table 2 provides further details on the transitions. Initially, the spacecraft will be in standby operations. Transition (1) will take it to scripted operations mode at the sequencer level. The spacecraft may remain in scripted operations over several planning horizons (with `Script-Next-Horizon` tokens in each script), as shown by transition (2). During this period, timelines other than the `Exec Activity` timeline become populated, resulting in ground-based planning.

Eventually, the mission profile will be updated and transition (3) will move the spacecraft to autonomous operations mode at the partial autonomy level. Over successive planning horizons (transition (4)), the spacecraft eventually achieves full autonomy.

At times during the mission, it may be desirable to transition back to scripted operations mode. This can be achieved as shown in Figure 4b and Table 2.

If necessary, ground operators can transition the spacecraft into and out of standby mode when necessary, as shown in Figure 5a and Table 3.

Finally, anomalies will result in the transitions shown in Figure 5b and Table 3.

6. UPDATING INFORMATION

The means by which ground operators update information on-board the spacecraftare discussed in this section. This includes all of the uplink messages necessary to perform the various levels of commanding discussed in the previous section. These uplink messages are designed to have the same form as the internal messages used by the various flight software modules to communicate with each other. The internal messages have explicitly defined contents, a publisher and subscriber(s). The information update messages are essentially identical, but with the publisher being the ground segment.

Table 2. Transition descriptions for normal operations modes

Transition			Requirements for transition		
#	From operations mode	To operations mode	Planning mode condition	File/token requirements	Comments
1	Standby	Scripted	SCRIPTED	Script file available	Normal entry to scripted operations
2	Scripted	Scripted	SCRIPTED	`Script-Next-Horizon` token & script file available	Normal script chaining
3	Scripted	Autonomous	AUTONOMOUS	`Plan-Next-Horizon` token in current script	Normal transition to autonomous operations
4	Autonomous	Autonomous	AUTONOMOUS	`Plan-Next-Horizon` token in plan	Normal autonomous operations
5	Autonomous	Standby	SCRIPTED		Begin transition to scripted operations
			STANDBY		Force indefinite standby period

In the following subsections, we describe the various information updates in the RA architecture, the rationale for them, and the uplink messages we have defined to accomplish them. But before we begin, it is useful to consider two important, related distinctions between traditional and agent controlled missions.

Traditionally, command sequence uplinks occur regularly throughout missions. The exact time that a particular command or parameter update is uplinked and its exact position in the sequence of spacecraft activities is known by the ground operators. Its effect on the activities involved in the command sequence has been accounted for in the planning process. This is in contrast to the situation encountered in an agent controlled mission. Since spacecraft activities are planned and scheduled on-board and potentially a variety of activities may be carried out in parallel, ground operators do not know the exact spacecraft state at any given time. Even if operators arrange to have the current plan downlinked, the plan leaves a sufficient amount of flexibility to the EXEC for run-time resolution that the operators still cannot predict the precise state of the spacecraft as a function of time. This represents a fundamental cultural change in the way spacecraft will be operated using agents.

It also leads directly to the second distinction. In agent controlled missions, the spacecraft may be receiving two sources of input intended to influence its actions. The primary source is the on-board agent, which is generating and executing plans to accomplish mission objectives. The secondary source is command and parameter uplinks from the ground. Due to the lack of detailed state knowledge on the part of the ground operators, these uplinks are received asynchronously with respect to other activities aboard the spacecraft. For some types of updates, the new parameter value may adversely impact the ability of the agent to successfully complete previously generated plans. Consider

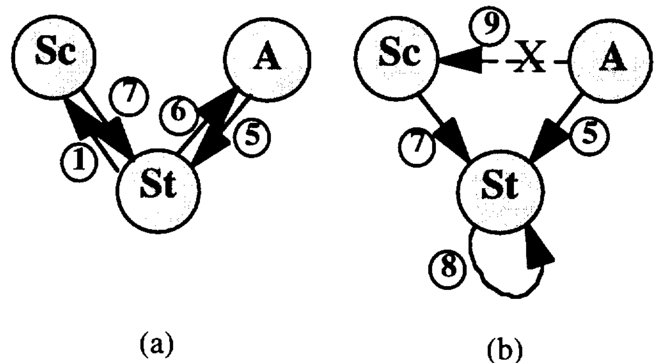


Figure 5. Operations mode transitions for (a) Standby and Abort operations, and (b) Anomalies (A - autonomous, Sc - scripted, St - Standby)

the example of a parameter update to change key attitude control system (ACS) parameters such that the time required to complete certain maneuvers would be increased. If an asynchronous update of these parameters occurred during execution of a plan including such maneuvers, the potential would exist for plan failure.

For each type of information update below, we will address how we dealt with these issues in the RA design.

Mission profile updates

During the fully autonomous phase of a mission, the principal method of interacting with the RA is expected to be at the goal level. The majority of routine activities, particularly during extended phases such as cruise, are expected to be well understood and modeled. Ground operators will specify mission activities at a high level of abstraction and rely on the RA to achieve them by whatever means it chooses. This allows them to focus on mission science objectives rather than low-level details of how to accomplish them.

Table 3. Transition descriptions for standby operations, aborts and anomalies

Transition			Requirements for transition		
#	From operations mode	To operations mode	Planning mode condition	File/token requirements	Comments
1	Standby	Scripted	SCRIPTED	Script file available	Return from standby or abort
					Recovery from failed plan
5	Autonomous	Standby	AUTONOMOUS	EXEC_ABORT_TO_MODE command	Abort current plan
					Failed current plan, or failure to generate a plan for next horizon
6	Standby	Autonomous	AUTONOMOUS		Return from standby or abort
					Recovery from failed plan
7	Scripted	Standby	AUTONOMOUS	No `Plan-Next-Horizon` token in current script	Go to Standby when script ends
			SCRIPTED	No `Script-Next-Horizon` token in current script	Go to Standby when script ends
				`Plan-Next-Horizon` token occurs in current script	Inconsistency between planning mode and token forces Standby
				No script file available via EXEC_SCRIPT_UPDATE	Wait for script in Standby mode
				EXEC_ABORT_TO_MODE command	Abort current script
					Failed current script
8	Standby	Standby	AUTONOMOUS		Failure to generate or install plan
			SCRIPTED		Failure to install script
9	Autonomous	Scripted			Prohibited transition

Although the mission profile has the capability of storing an entire mission's goal set prior to launch, there are two principal reasons why the ability to modify the mission profile is necessary. The first is that some activities cannot be scheduled in advance of launch and the goals for these activities must be uplinked during the mission. An example of this is the scheduling of communications passes using the spacecraft's high gain antenna. The spacecraft cannot control the schedules of DSN dishes; it must be informed of when to attempt communication with the ground. Since DSN schedules are not known far enough in advance to include the pass schedules for the entire mission in the initial mission profile, it is necessary to periodically uplink schedules for future passes. This is accomplished by creating an instance of a communications goal for each desired pass.

The second reason is that no matter how much effort is put into defining mission objectives prior to launch, somebody will always come up with a worthy modification during the mission. Typically, these modifications result from unexpected findings in downlinked mission data which result in the desire to re-prioritize data acquisition strategies. Requesting science observations of new targets, or of one target under different constraints or with different instrument settings, is accomplished by uplinking new goals to the spacecraft and/or deleting old goals. Science data acquisition goals are defined for each kind of instrument on-board.

A goal definition includes certain parameters which must be specified for each instance of the goal. For example, a `Communicate` goal has earliest and latest start time, earliest and latest end time, and a set of configuration parameters for the telecommunications equipment. A science observation goal might have earliest and latest start time, earliest and latest end time, target identification and/or coordinates, calibration settings such as filters and gains, and so on. Of course, the start and end time parameters for the communicate goal would be very tightly constrained to meet the DSN schedule, while the same parameters for the science goal might be much more flexible if there was no particular time constraint for acquiring the data.

Parameters which are associated with goals, which do not influence the state of the spacecraft outside of the goal activity, and which have potentially different values for each goal instance are included in the goal definition. When planning occurs, these parameters are sent to the appropriate planning experts, if necessary. The values are specified in the tokens generated by the planner when required.

Goal updates to the on-board MM are achieved as follows. A goal instance is created on the ground. Operators specify the values of the goal parameters or choose default values (when applicable). The resulting goal instance is placed in a file in the language of the PS. Ground operators verify the consistency of the new goal with the rest of the mission profile by submitting it to a ground version of the MM. Following the consistency check, the file is uplinked to the on-board file system during a scheduled pass. After the file uplink, the MM_GOAL_UPDATE(filename) message is sent to the on-board MM to notify it of the new update file to be processed.

Verifying the consistency of the new goal with the mission profile is not a complete guarantee that the PS will be able to successfully generate a plan which includes the new goal. The lack of complete spacecraftstate knowledge by ground personnel may occasionally result in an inability to successfully generate a plan. Therefore, the RA architecture has included the concept of goal prioritization. If the PS cannot achieve all the goals requested for a particular horizon, it will reject or defer lower priority goals until it can successfully generate a plan. It will then request guidance from ground on how to achieve the goals which were shed.

The mission profile update causes no synchronization problems with the current plan since the update was to the MM and the MM won't be invoked again until it is time to plan the next horizon. If ground wants to add an additional goal to the current plan horizon, it is necessary to force a replan for the remainder of the current horizon after the goal update. The strategy for accomplishing this is to send an EXEC_ABORT_TO_MODE(mode) message after the mission profile update messages. That forces the spacecraft to abandon the current plan, achieve the specified (or default) standby mode, and replan. There is always a small chance that the MM_GOAL_UPDATE message with a new goal for a particular planning horizon will arrive prior to the start of that horizon, but after the plan request for that horizon has gone from the mission manager to the planner. This does not result in any inconsistencies in the RA's behavior. In this case, the result is simply that any update information for the upcoming horizon will not be included in the new plan. We have adopted the policy that ground operators must take responsibility for forcing a replan when updating information in close temporal proximity to the waypoint where it becomes relevant.

The process is identical for mission profile updates in partially autonomous operations.

Ground-based plan uplinks

For ground-based planning, one or more fully specified plans must have been generated, validated and placed in files on the ground. One or more plan files must then be uplinked to the spacecraft, followed by an EXEC_SCRIPT_UPDATE() message. The argument of this message is a file list. If the EXEC is in SCRIPTED planning mode, it then searches through the files on this list whenever it needs a new plan to execute. It selects the first file within the list which has a start time consistent with the current time.

The EXEC_SET_PLANNING_MODE(mode) command is used to change planning modes. The possible modes are AUTONOMOUS, SCRIPTED and STANDBY. When in AUTONOMOUS planning mode, an on-board plan is requested whenever a `Plan-Next-Horizon` token is encountered on the `Planner Processing` timeline or when the spacecraft achieves a standby mode.

When in SCRIPTED mode, the EXEC selects a plan from the file list specified in the EXEC_SCRIPT_UPDATE() message whenever a `Script-Next-Horizon` token is encountered on the `Planner Processing` timeline or when the spacecraft achieves a standby mode. If an inconsistent combination of `Planner Processing` timeline token and planning mode selection occurs (e.g. a `Plan-Next-Horizon` token is encountered in SCRIPTED planning mode), the spacecraft will achieve standby mode and then proceed according to the current planning mode selection.

Typically, the last ground-generated plan in SCRIPTED planning mode chain resets AUTONOMOUS planning mode with an EXEC_SET_PLANNING_MODE command embedded in an `Exec-Activity` token. Then, if the final ground generated plan contains `Plan-Next-Horizon` token, the planner will generate a new plan which will merge seamlessly with the end of the ground generated plan. If the `Plan-Next-Horizon` token is missing from the ground generated plan, the spacecraftwill achieve a standby mode and then a new plan will be requested.

Synchronization of these messages with ongoing activities is managed because they have no effecton the system until it is time to acquire a plan for the next planning horizon. At that time, the executive decides how to proceed based on the current planning mode and `Planner Processing` timeline's token. They can only affectthe current planning horizon if an EXEC_ABORT_TO_MODE('standby mode') message is sent.

Sequence uplinks

This is just an instance of ground-based planning where the plans contain nothing but `Exec-Activity` tokens and, if extended operation in this mode is desired, `Script-Next-Horizon` tokens. The plan file(s) are uplinked, followed by the EXEC_SCRIPT_UPDATE() message with a file list argument and then the EXEC_SET_PLANNING_MODE('SCRIPTED') message. As in the previous subsection, the spacecraft must achieve standby mode before execution of the initial plan can begin.

Fault Diagnosis Overrides

Fault control is one of the primary mechanisms for ensuring the safety of the spacecraft. Although the RA has excellent fault detection and repair capabilities, the ground controllers will occasionally want to override the RA's diagnoses.

An override capability allows the ground controllers to change the RA's diagnosis if they disagree with it for some reason. It also allows the ground controllers to reserve failing devices before they fail completely. For example, the ground may detect signs that the data recorder is failing and decide not to use it for routine science but save it for a particular interesting opportunity that is coming up. This can be done by declaring the data recorder "unusable" for some period of time and then declaring it "usable" again during the important science window. This capability may also be used as a fault injection mechanism during testing.

The RA has two mechanisms for allowing the ground to override its fault diagnoses. The first is an MI_HEALTH_ UPDATE(device,status) message, which tells MIR to change the diagnosis of a device to a particular value (e.g., 'healthy' or 'failed'). This message allows ground to declare a device status to be any value which is defined for that device by the MIR. It can be sent at any time.

Once a device has been declared failed by the RA, the health update message is the only means of reviving it. Even if low-level commanding and troubleshooting by the ground team restores functionality to an ailing device, RA must still be notified that the device is available. However, declaring a device which has been previously classed as failed to be healthy does not immediately affect plan execution. Synchronization of this message with the ongoing spacecraft activities occurs when the spacecraftstate information is provided to the MM when it is time to plan the next horizon. If it is desired to take advantage of the restored capability immediately, a replan must be forced immediately following the health update.

If a device is declared healthy when it is actually failed, the RA will attempt to use the device as required by the current goals and end up repeating the fault diagnosis process that resulted in the failed declaration, eventually declaring the device failed again. Therefore, prematurely declaring a device healthy may waste time and resources while the diagnostic process is repeated, but it will not result in inconsistent behavior or dire consequences.

If a device is declared failed while it is in use, the MIR will treat the situation just as if it had detected the fault itself at that time. It will notify the EXEC of the failure and engage in the normal recovery activities for the failure.

The other mechanism for performing fault diagnosis overrides is the EXEC_POLICY_UPDATE(device, status) message. This is a message to the EXEC that declares a device usable or unusable. The EXEC passes the usable/unusable status of the device to the PS as part of the initial spacecraft state. The PS maintains health timelines that track the health of certain high-level devices: for example, the main engines, the camera, the attitude thrusters, etc. If a device is declared unusable, the PS generates plans that do not use the device, regardless of whether MIR thinks it is healthy or not. When the device is declared usable again, the PS can use the device. There are no synchronization issues because this message only affectsthe plan request from the EXEC to the MM. If the message is received after the EXEC has sent a plan request to the MM, it will not take effect until the next time a plan request is necessary. As always, the ground operators must force a replan in order to ensure that last minute changes are included.

Parameter updates

We will define a parameter as any piece of information that's used by the flight software, that is not changed by the lower level or more traditional segment of the flight software, but that the ground operators or RA may wish to change for some reason. Typically, parameter updates are intended for one specific software module and can include any kind of information. Ephemeris file updates for the navigation module or new calibration curves for instrument modules are simple examples of parameter updates.

During sequenced operations, messages required for parameter updates are built into the sequences by ground operators. Validation of sequences and synchronization of on-board activities is their responsibility and may be achieved by traditional means.

However, the discussion here will focus on parameter updates during autonomous operations. Table 4 shows that the way a parameter update is achieved in the context of autonomous operations depends on two properties of the parameter. One is whether the parameter is to be updated immediately or whether it is necessary to send the update at a particular time or with some other temporal constraint (e.g. after a specific event). The other property is whether or not the value of the parameter matters to the RA. This property assesses whether an asynchronous update could lead to plan failure. For example, if a particular parameter change caused the time required to complete a turn (attitude change) to become significantly longer, the token for the turn activity might time out, resulting in plan failure.

Let's take a closer look at the last example. The maximum turn rate of the spacecraft is controlled by a collection of ACS parameters which are interdependent and must be changed as sets or bundles. The process is not trivial and has an impact on other spacecraftactivities. It can only be accomplished by an ACS mode change or re-initialization.

Table 4. Mechanisms for achieving parameter updates based on parameter properties

	Affects RA	**Does not affect RA**
Immediate	Goal + replan	Parameter update messages within EXEC_EVAL message
Delayed	Goal	Parameter update messages within EXEC-ACTIVITY tokens

Parameters which are associated with reconfiguring a subsystem or module are often of importance to the PS and must be achieved at specific times. The proper method for updating such parameters is to define a goal which accomplishes the state change. The parameters (or sets of parameters) to be updated become goal parameters (i.e. part of the goal definition) and are specified for each instance of the goal. The goal is scheduled by the PS to occur at the required time under the necessary conditions, and the new parameter values are then maintained by the appropriate software module until the next update.

As discussed earlier, the new values may also need to be stored in NVM if the spacecraft is expected to recover to this state following a hardware reset. The options of updating parameter(s) in DRAM and/or NVM can be included in the goal definition, selectable by a goal parameter.

Using the goal mechanism to achieve these critical updates ensures robustness in the presence of faults. If a plan breaks during the time for which the update goal was scheduled, the PS will re-schedule the goal for the next horizon. However, the goal mechanism also results in the overhead of additional goal definitions and timelines for the PS. The asynchronous nature of the updates is handled by the MM, as described mission profile updates.

Parameter updates which are less critical may be achieved by simply sending messages to the target module. A parameter update message may be defined specifically to manipulate the value of a single parameter within a module. Alternatively, a file containing updated parameter(s) could be uplinked to the on-board file system, followed by a message to the appropriate module announcing the presence of the update file. This is useful for updating bundles of parameters; the entire parameter set may be written in the file. It is up to the target module to read the file, build a new parameter structure, and switch the structure pointer at the appropriate time. The asynchronous nature of these messages is addressed implicitly with the assumption that these messages do not affect plan execution and need only be synchronized internally to the target module. However, since any message may be sent using this mechanism, it is incumbent on the ground system to provide safeguards against sending messages which may disrupt on-board activities.

If the update is desired for a later time (not during a communications pass), the update is managed by the `Exec-Activity` token. The messages (as many as necessary) for the update are placed in a file and the file is uplinked. A new `Exec-Activity` token with the specified file as its argument is placed on the `Exec-Activity` timeline via a mission profile update. At execution time, the messages in the file specified by the `Exec-Activity` token are sent. The advantage of this approach is that only one PS timeline is required for an unlimited variety of updates, and multiple messages may be stored in each file. There are two disadvantages to this approach. First, there is no checking to insure that the update messages are consistent with other spacecraft activities. Second, if the plan breaks before the update occurs, there is no mechanism to insure that the update will occur after recovery. The `Exec-Activity` token for a particular update will not be rescheduled for a time past its latest start time, but will simply be dropped. During autonomous operations, this approach should be used sparingly for sending messages other than simple parameter updates. We believe that it is necessary to maintain a list of accepted "safe" update messages which may be uplinked at any time. All other messages must undergo a higher level of scrutiny and operator concurrence within the ground system prior to uplinking.

Parameter updates in real-time are achieved using the EXEC_EVAL message functionality. The file containing the messages is uplinked to the spacecraft. The message EXEC_EVAL(filename) is sent to the EXEC. This causes immediate execution of the file contents without the overhead of a mission profile update or a plan generation.

As with other updates, parameter updates to DRAM and to NVM are separate activities. The ground operator must specify what is desired.

Software upgrades

Although the RA architecture supports software upgrades, the process in the context of an agent-based mission is essentially identical to the process for a more traditional mission. Therefore, it will not be discussed further here.

However, a distinction can be made between the RA software and the information encoded in the RA models. The models used by the RA are likely to need occasional tuning and upgrading, particularly in the early phases of the mission. A mechanism to routinely update them without a full software upgrade would make such updates a more routine activity which can be performed without disrupting normal spacecraft activities. This would pave the way for the eventual capability to have the agent modify its own models as necessary. A capability currently exists to perform such upgrades for a partial set of the RA models. This is an area for future work.

7. TRACKING STATUS

We have described at some length how the RA works and how ground operators uplink new information in order to command the spacecraft at varying levels of autonomy. However, we have not described how the spacecraft communicates back with the ground and what kind of information is downlinked. Of course, there is generally a requirement to downlink the science data regardless of operations mode. Acquiring the science data and returning it to Earth is the primary purpose of the mission. However, the amount and type of spacecraft housekeeping, health and safety data that is downlinked must also be considered.

In partially autonomous modes or ground controlled operations modes, spacecraft status telemetry is necessary in order for ground operators to perform their duties. The amount and type of information required depends on the

operations mode. When operating in autonomous mode, the RA monitors the spacecraft state continuously throughout the mission. Unless an anomaly occurs which the RA cannot resolve on its own, there is no strict need to downlink any status telemetry since the ground operators are not directly controlling any aspect of the spacecraft operations. However, ground operators will still want to have some degree of visibility into the state of the system and the progress it is making toward achieving mission goals. The degree of visibility desired will be variable. Initially, operators may want to see relatively large amounts of information to reassure them that the spacecraft is functioning normally; after more experience is accumulated, they may become confident enough to require no more than a high level activity summary or eventually even just a report that all is well.

As with the multiple levels of spacecraft commanding described earlier, our mission operations approach includes multiple levels of spacecraft status monitoring which are selectable by ground operators. We will discuss the available levels of monitoring and how they relate to our operations concept below, but first we will give an overview of telemetry generation and management for missions involving autonomous agents, using the DS1 mission as a model.

Telemetry generation and management

Telemetry is generated on-board by each software module in streams. A single module may produce multiple streams. The contents and rates of each stream are determined by the module. Additionally, each module may have multiple modes with different telemetry streams generated for each. For example, a module may have a nominal mode and an anomaly mode which generates additional diagnostic data. Modes may be selectable by the module automatically or by command from the RA or ground. The telemetry manager module routes the streams to various telemetry pools. A given stream may be routed to a single pool or multiple pools.

Telemetry pools are managed by the telemetry manager. Each pool is a memory bufferof fixed size which contains related data. A pool may contain data from only telemetry stream or from multiple streams. For example, a Science Data pool may contain image files from an on-board camera and spectrograph files from an on-board spectrometer. There may be pools for Real-Time Engineering Data and Stored Engineering Data; both of these would contain streams from multiple modules. Each pool is sized such that it will not overflow if a single communications pass is missed. If more than one pass is missed, each pool has a set of deletion rules for managing overflow. Typical choices are to delete either the oldest or newest data when the bufferis full. Deletion rules may be modified by parameter update. For downlinking, a downlink priority table allocates each pool a priority and a percentage of the downlink bandwidth. Higher priority pools are emptied before lower priority pools are downlinked. Pools with identical priorities share the downlink according to their bandwidth percentages. Downlink priority tables are selectable by parameter update from the ground or the remote agent.

Variable levels of spacecraft status reporting

At the lowest level of autonomy, the ground operators are fully responsible for knowledge of the spacecraft status and for generating command sequences to accomplish mission objectives. They would therefore have a great need for visibility of data from all of the traditional spacecraft systems. Although not required to fly the spacecraft, additional data generated by the on-board autonomy software may offeroperators additional insight. For example, state identification and diagnostic analysis produced by the MIR and high level engineering data summarization produced by the beacon mode module (see below) may help to focus operator attention where it is most needed. For this mode, the downlink priority table may give raw telemetry higher priority while including summaries from the autonomy modules at relatively low priority.

For fully autonomous operation with no anomalies, the ground operators do not *need* any status information from the spacecraft, but they may require it. They will still have full access to the raw engineering data generated by each module. However, raw telemetry is difficult to interpret when ground operators do not have detailed knowledge of the spacecraft state or activities as a function of time. There are two approaches to addressing this problem, which may be adopted sequentially.

One approach is for the RA to provide additional products in the telemetry to give operators additional detailed visibility into spacecraftactivities. These products include plans, sequence traces from the EXEC, and logs of the MIR diagnostic analyses. In this approach, both the RA data pools and raw engineering data pools are downlinked with approximately equal priority. This process is bandwidth intensive, but it is feasible during the earlier phases of the mission, when the spacecraft is closer to Earth and data rates are relatively high. This is also the period when fully autonomous operations will be initiated and operators will want the greatest visibility possible to assess the RA's performance. Tuning of the RA models will be a priority activity during this period.

As operators gain confidence in the RA's performance, the second approach to providing visibility to operators may be adopted. In this approach, the downlink priority table is modified to give high priority to RA data pools and the data summarization data pool, providing the ground with activity and diagnostic logs, statistics, and trend information. Raw telemetry data may be downlinked at a very low priority as time permits.

At an even higher level of abstraction, when there are no anomalies, the downlink priority table may be modified to limit status data to an "executive summary" pool that verifies that all is well. It may also include a brief listing of science goals that were accomplished in the current reporting period and those which are scheduled for the next reporting period. This approach might be particularly useful in order

to limit downlink bandwidth during routine and trouble free phases of the mission. Late in the mission, when the spacecraftis distant from the Earth and data rates are low, this may be essential to maximize science data return.

During partially autonomous operations, the downlink priority table is set to give greater insight into some areas, while relying on summary data for others. For example, if the RA is managing the spacecraft housekeeping and ground is managing payload operations, science instrument data pools may be given priority equal to the RA and data summarization pools.

When anomalies occur, the RA switches to a downlink priority table optimized for anomaly investigation and resolution. The RA can switch telemetry modes in affected software modules (if they haven't detected the anomaly internally and switched themselves) to force generation of diagnostic data for routing to anomaly resolution telemetry pools. The RA can generate additional diagnosis and recovery action logs for routing to the anomaly resolution pools. The anomaly resolution and data summarization pools will be given highest priority and the science data pool will be given low priority until the anomaly is resolved or ground selects another downlink priority table manually.

Beacon mode operations

Having discussed the contents of the spacecraft telemetry, we will now describe the downlink strategy used for DS1. The RA manages periodic scheduled DSN communications passes, during which goals, parameters and other messages are uplinked, while spacecraft and science data are downlinked according to the current downlink priority table. However, while traditional missions do not generally have the ability to request prompt help from the ground in the case of anomalies, we do have a strategy for contacting the ground outside of normal scheduled passes when necessary.

The strategy is to continually power the communications system and low gain (omnidirectional) antenna whenever power constraints do not preclude it and to place the communications system in beacon mode [5]. The beacon mode module monitors spacecraft health update messages from the EXEC and MIR and compares these messages with an internal fault table in order to map the overall spacecraft health into one of four states. Each of these states is represented by a tone which indicates how urgent it is to track the spacecraft for telemetry. The selected tone is continuously broadcast via the low gain antenna. The definitions of tones correspond roughly to: (1) nominal, (2) something interesting to report (not urgent), (3) important to service spacecraft soon or state could deteriorate and/or critical data could be lost, and (4) spacecraft emergency, contact needed immediately. The beacon mode module also includes a data summarization component that computes running summaries of engineering data.

A low cost ground antenna is capable of monitoring these signals during a significant portion of each day. This gives ground operators daily visibility into the spacecraft's health. If a contact is requested that cannot wait until the next scheduled high gain pass (typically once per week), the ground operators can command an immediate high gain pass via the low gain omnidirectional antenna using an EXEC_ABORT_TO_MODE() message with the standby mode chosen to be an Earth communications mode such as 'HGA_TO_EARTH' (high gain antenna to Earth). If a high gain pass is not possible, operators can switch the spacecraft from beacon mode to low rate telemetry mode via the low gain antenna. Summary status data can then be downlinked at a lower rate. The summary data will be controlled by a special low-rate downlink priority table and will include the beacon mode data summarization pool and a remote agent data summarization pool. Note that the beacon mode technology can provide increased ground operator visibility into spacecraft health for both traditional and agent-based missions. It is a separate autonomy technology flown on DS1 which complements the RA extremely well.

Visualization tools

Because the ground operators must now have the ability to interpret a great deal of complex data from the spacecraft, such as plans, plan execution status and diagnostic summaries of the spacecraft state, it is clear that traditional numerical or data plotting displays are not adequate. Visualization tools have been developed for RA data. For viewing plans and plan execution status, there is a graphical display tool [6] that depicts the planner timelines, the tokens on them and the constraints between the tokens. Color coding of the tokens indicates which are complete, which are currently executing and which have failed, along with more detailed status information which can be summoned by clicking on tokens of interest. The MIR team has a graphical ground interface to the on-board MIR called Stanley which provides a schematic view of the spacecraft state information. These tools assist ground operators in determining if they need to take any action.

8. SUMMARY OF ARCHITECTURE REVISIONS

The current RA architecture diagram is shown in Figure 6. A number of revisions were found to be necessary as we transitioned from the SOI prototyping effort to the DS1 flight software development project. However, they fall into two basic categories as shown by the two new elements in the figure: the Mission Manager and the ground interface.

Mission manager

We added the MM because we needed to manage goals for an extended operational lifetime (2 years or more) rather than a 1-2 planning horizon scenario. The MM maintains the mission profile, a partially specified plan containing a database of goals for the entire mission (or a significant portion of it). The addition of the MM to the RA involved modifying the interface between the EXEC and the PS for requesting a new plan. The MM now takes initial state information from the EXEC's plan request, extracts the appropriate goals from its database, and formulates a

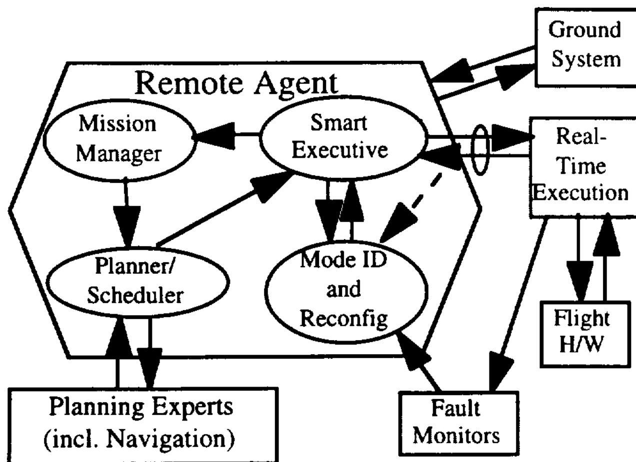


Figure 6. Current RA Architecture

complete new plan request for the PS. Additional internal modifications were required for the EXEC and PS to successfully implement the seamless chaining of plans over many planning horizons.

Ground interface

We also added a ground interface for the RA. This was not necessary for the SOI scenario, since that scenario was explicitly intended to demonstrate resolution of a failure under circumstances where ground intervention was not possible. However, for an extended mission, a great deal of ground interaction is anticipated and must be accommodated. The ground interface implementation can be further subdivided into several key capabilities.

Multiple levels of commanding

We implemented multiple levels of commanding in order to provide a means for a gradual hand-off of operations from ground operators to the on-board agent. At the most basic level, we provided a sequencing capability to allow the mission to be flown much as traditional missions are flown, with no on-board autonomy. A ground-based planning capability was implemented to allow exercising of the execution and monitoring functions of the RA on fully specified plans supplied by ground operators. A partially autonomous mode was implemented to allow the RA to gradually take over the planning and scheduling functions while the ground operators maintain greater control over key activities. A fully autonomous mode was then implemented. The lower levels of autonomy may also be used during other mission phases or activities as deemed appropriate by the ground operators.

Device health updates

We provided a means of performing device health updates which override the MIR diagnosis for the device. There is also a related means to declare a device unusable, regardless of its health status. This allows operators to perform troubleshooting if the MIR is making a diagnosis in error, or to preserve failing devices before they fail completely. It also provides a convenient fault injection mechanism for testing.

Information updates

Additionally, we provided a means of performing all of the information updates required to support the capabilities described above, as well as a general parameter updating capability. We explicitly addressed concerns about the impact of each type of information update on spacecraft activities.

Spacecraft monitoring

Finally, we provided support for monitoring spacecraft behavior at multiple levels of detail, from highly abstracted summary data to "traditional" raw telemetry, with many intermediate steps in between. We incorporated the features of beacon mode operations into the RA and provided the necessary visualization tools to interpret the downlinked health and status data.

Messages and tokens

The above ground interface was implemented with a small set of new messages, shown in Table 5, and three new tokens, shown in Table 6. With these few constructs, it is possible to function at any of the commanding levels and send any message (all defined internal flight software messages) to the spacecraft from the ground. Note that all of the other defined messages (including parameter updates) must be sent from the ground via either mission profile updates, `Exec-Activity` tokens, or EXEC-EVAL messages.

9. Variable Autonomy

Our operations-focused agent developments enable humans to interact with the agent at multiple levels. In this section, we elaborate on some different uses of this capability within a mission life cycle.

Incremental autonomy

One key issue concerns the timing and processes by which knowledge necessary to support autonomy gets encoded into the agent, on the one hand, and the processes by which the operators become familiar with the agent, start learning to use it, and gain confidence in it, on the other hand. Our prototype, based on complete autonomy, required operators to move from low-level management to high-level delegation in one big jump and all at once.

Some recent work in knowledge acquisition has focused on supporting increasing levels of assistance and competence from a knowledge-based system, rather than trying to build the ultimate capabilities into the system at the start. The "everything-from-the-start" approach has two problems. First, it takes a long time to add in all the knowledge that users would eventually like in the system, which can considerably slow down the deployment and adoption of a new system. Second, higher-level capabilities need to be developed based on feedback from users in the course of real-world use of the developing system.

Table 5. Messages defined to implement ground interface for RA

Message	From Ground to:	Argument(s)	Purpose
File uplink	File Manager	Filename	Uplink file to on-board file system
EXEC_ABORT_TO_MODE( )	Executive	HGA_TO_SUN HGA_TO_EARTH LGA_Z_TO_EARTH DEFAULT	Interrupt plan and place spacecraft in specified standby mode
EXEC_EVAL	Executive	Filename	Immediately send messages in file
EXEC_POLICY_UPDATE	Executive	Device, status	Directs the executive to declare a device status in the initial state it provides for planning; status may be 'usable' or 'unusable', regardless of device's health
EXEC_SET_PLANNING_MODE	Executive	AUTONOMOUS SCRIPTED STANDBY	Set planning mode
EXEC_SCRIPT_UPDATE	Executive	File list	Provide list of potential plans for SCRIPTED mode; EXEC chooses first plan file in list with start time consistent with current time
MI_HEALTH_UPDATE	MIR	Device, status	Sets device status to specified value
MM_GOAL_UPDATE	Mission Manager	Filename	Notify mission manager that a mission profile update file is available

Table 6. Tokens defined to implement ground interface for RA

Token	Timeline	Contents	Purpose
`Exec-Activity`	`Exec Activity`	Filename	Messages in specified file are sent when token is executed
`Plan-Next-Horizon`	`Planner Processing`	Nil	Set aside time in current plan to plan for next planning horizon
`Script-Next-Horizon`	`Planner Processing`	Nil	Allows chaining of multiple ground generated plans in SCRIPTED mode

In the "increasing levels of assistance" approach [7], users receive a system with a minimal set of capabilities, which they can start using immediately. Then, the system supports users in the incremental development of new knowledge on top of the older capabilities, in a smooth migration of responsibility for lower-level tasks from the user to the system.

In the context of autonomous systems, we are interested not only in increasing over time the levels of *assistance* the agent can provide to the operators, but also in increasing the level of *autonomy* the operators are willing to delegate to the agent. This involves developing not only capability, but trust as well. Hence, we describe the process *as incremental autonomy*:

- Deliver a system with minimal high-level knowledge, and relying on the operator to perform higher level capabilities.
- Encode increasingly higher-level knowledge into the system, which can then apply its own higher-level capabilities to this new knowledge to conduct increasingly autonomous operations.
- Continually modify the knowledge based on feedback from using the system.

Our new architecture lets ground operators use the agent in low-level mode, to interact with it in the traditional style. Then as ground operators gain confidence with the agent, they can define a few timelines and tokens for some automatable activity, but command the rest at a low level. They can even use the structure to generate detailed plans for ground-based verification, and upload the results to the spacecraft. Over time they can group the low level commands into `Exec-Activity` tokens, and then into full tokens with explicitly modeled constraints. Similarly, they can move capabilities from ground to on-board. Thus we support a delivery of minimal capability, with incremental increases in functionality all at the discretion of ground operators, on their schedule.

Variable autonomy

Incremental autonomy addresses increasing capability and increasing trust over time. If we focus only on this issue, it might seem as if the eventual goal is to produce an entirely autonomous system. Full autonomy is useful because it enables missions in which humans could not possibly assist

the agent due to limitations in communications and response speed, and also because it reduces operations costs which might prohibit human intervention even when it is physically possible. Put succinctly, the goal of full autonomy is to *minimize the necessity* for human interaction with the spacecraft.

However, we believe that full autonomy is not the complete goal. Scientists would like to be involved with the spacecraft, when possible, to suggest new targets based on new observations and to help get the most interesting data down first. Similarly, ground operators need to be involved with the spacecraft, when possible, to resolve problems using detailed engineering knowledge, to fix software problems or update the spacecraft with the latest algorithms. In addition, judicious reliance on human interaction, when possible, can have many benefits on overall system design. For example, we can rely on humans to solve the really hard cases, while letting the autonomous system address the more typical cases. This reduces the amount of knowledge that needs to be built into the onboard agent, thus reducing software design costs and accelerating the software development schedule. Since software development costs are becoming a major factor in the overall costs of running a mission, reductions in these costs actually make autonomy an affordable possibility. Hence, it is also a desirable goal to *maximize the capability* for human interaction with the spacecraft.

We unify these two goals in the concept of *variable autonomy*:

- Enable the operator to be in as much control as is possible and desired.
- Enable the operator to let the system do everything as autonomously as possible and desired.
- Enable the operator to choose for which aspects he wants which level of autonomy.
- Minimize the necessity, but maximize the capability, for human interaction with the spacecraft.

We provide two examples of the benefits of variable autonomy in spacecraft operations. The first example is in the area of fault protection. Of all the faults that could possibly happen, only a few will really happen in flight. Of those, only a few really need to be addressed on the spacecraft. If you don't support human involvement, the autonomous system needs to have knowledge to handle all possible cases (or risk losing the mission). If you support human involvement when possible, you can choose which faults to handle on-board beyond the minimum required to keep the spacecraft alive long enough to call for human help. These choices can be based on trading off the costs of up-front knowledge engineering necessary to recover from the fault on-board and continue with the mission against the benefits of increased mission returns since the spacecraft can continue execution in this situation without having to wait for human help.

The second example concerns relying on humans to handle the hard cases in sequence generation. Some mission phases, like cruising to a planet, occupy much of the lifetime of a mission, while other phases, like insertion into the planet's orbit, occur only once in the entire mission. In the case of a 10-year cruise to Pluto, it is almost certainly worth encoding the knowledge to enable the agent to function autonomously during as much of this period as possible, to save operations costs. But in developing a sequence for the critical Pluto flyby, where many considerations must be factored in that are unique for this one day period, it may well be cost-effective to rely on human operators to play a major role (and they'll want to). If we did not enable operators to vary their level of involvement across different phases, we would force them to choose between autonomous operation of the entire mission, including the potential high cost and risk to handle the orbit insertion, and traditional operation of the entire mission, including the high cost of staffing during a long and routine cruise period.

We can distinguish two types of variable autonomy. A simple form of variable autonomy is *phase-variable autonomy*, in which the operator can choose to use different levels of autonomy in different phases of a mission. This is the form illustrated in the sequencing example above, in which the operators may rely on the agent to plan its own activities for cruise phase, with relatively little attention or tracking from ground, but then they may design the sequence of activities for the orbit insertion phase entirely by hand and monitor the state before the sequence at a fine level of detail to make sure everything is set up just right.

The most general and flexible form is *activity-variable autonomy*, in which the operator can choose to use different levels of autonomy to control different activities at any given time. This is illustrated in the fault protection example above, in which the operator may desire that the agent handles some faults by detailed diagnosis followed by autonomous replanning, whereas the operator wants the agent to go directly to a standby mode and wait for human help in other situations. A similar example from sequencing would be to have the agent plan the navigation activities but rely on ground operators to manage detailed scheduling of the science instruments.

Our architecture supports both phase-variable and activity-variable autonomy. For example, operators can use the `Exec-Activity` tokens to install fine-grain activities within plans generated on-board by the agent, and can also use this to change parameters in the middle of autonomous plan execution. Similarly, the operators can choose the level of detailed telemetry they want to track, from high-level beacon tones down to debugging data, and they can choose the level of level of detail on a system-by-system basis and vary this over time.

Agent-variable autonomy

The preceding discussion has focused on enabling human operators to vary the autonomy granted to the agent. Our architecture also enables the agent to vary its own level of autonomy in some situations. For example, RA turns off its autonomous planning capability and waits for human

assistance if it is unable to find a plan, if it needs to plan from a situation for which its planning models are not defined or applicable, or if it detects repeated failures during plan execution. Such *agent-variable autonomy* can increase operator willingness to delegate autonomy to the agent because the agent will call back for help if the situation goes beyond its capabilities.

Simplicity vs. optimization

Another key theme concerns the tradeoffs between simplicity and optimization. In several cases in our design of the RA for operations, there were simpler ways to achieve some of the operational goals, but at the cost of interrupting the mission activities. For example, we could require that parameter updates can only be performed in standby mode. This solves the problem of inconsistency with the on-board planner with a simpler design than the one we described. However, operators might then be more reluctant to use autonomy in phases when they think they might want to update parameters, as they may be unwilling to sacrifice key science activities. While these considerations motivated us to develop a more complex and capable design, such simplifying alternatives should be kept in mind as useful de-scopes in the development of autonomous agents for mission operations.

10. RELATED WORK

This section relates our work to other work developing systems that are highly autonomous and which also support human interaction when necessary or desirable.

From the spacecraft perspective, Cassini [8] developed a highly autonomous system, but not a full agent in the sense that it was doing patchwork functions, supporting only what was necessary for that particular mission-critical scenario, rather than automating the capabilities (like planning) performed on ground.

Several other AI researchers have developed agents based around variable levels of autonomy. Mixed-initiative planning and scheduling has become a popular area of research. An emerging idea in this area is for operators to interact with the agent at the level of *constraints* on problem solutions. The agent is free to solve problems in any manner consistent with the operator-imposed constraints. If the solution is already fully-constrained, the agent is given minimal autonomy, whereas if the solution contains only constraints corresponding to high-level goal achievement, the agent has a high level autonomy. Our ground-to-planner interface adopts this constraint-based approach. Systems like Ozone [9] and O-Plan [10].take mixed-initiative constraint-based planning further by developing structured interfaces and dialogues between computer and humans and also by developing richer constraint representations including lists of open issues, rationales for decisions, and authority structures. However, these projects have focused on the planning capability, and do not go into issues of the entire execution architecture. Nor do they support the needs of full mission operations, including long missions, health updates, and parameter changes.

Lockheed's Tactical Planning and Execution System (TPES) [11] is a mixed-initiative system for planning and execution in the naval domain. Like RA, TPES supports high-level monitoring, in the form of alerts, current plans, and execution traces. It also supports mixed-initiative generation of plans, and the provision of new information during plan execution. Such new information includes command over-rides, policy changes, and situation updates. However, the mixed-initiative control in this model relies on the human operator being onboard the vehicle. Thus the operator can interact with the plan as it is being formed, gather immediate understanding of the current state of the system, and exercise immediate control when necessary. Thus, unlike the case for autonomous spacecraft, the approach did not need to address issues concerned with integrating changes within some future situation. However, the interaction model does account for the currently-executing plan leaving the controlled system with a set of commitments that may affect how the current plan can be modified, even if the current plan execution is aborted.

JSC's 3T architecture [12][13] provides support for activity-variable autonomy during operations. Each procedure implemented by the 3T executive may have alternate methods to execute depending on a user-specifiable *level-of-autonomy* parameter attached to the procedure. Defined autonomy levels include *autonomous*, *supervised*, *guided* and *tele-operated*. Autonomous methods execute without human assistance; guided methods follow user control but take limited autonomous actions such as obstacle avoidance; tele-operated methods monitor activity without taking actions; and supervised methods ask the user to decide which level of autonomy to grant as each method is executed. Like the Lockheed work, this approach seems most useful when the user is interacting with the agent in real-time (onboard an autonomous vehicle or co-located with a robot). For example, significant delays in communication to the agent reduce the value of asking the operator whether or not a particular activity should be performed autonomously. However, the approach is complementary to the constraint-based approach taken in RA and this type of capability could be incorporated in RA with minimal change to the rest of the architecture.

Many researchers have developed expert advisory systems, like the PI-in-a-Box [14], or the Pilot's Associate [15]. While such systems provide good support for human involvement, they do not provide a level of full autonomy. Because of this, they also do not have to address the issues of supporting different levels of autonomy at different phases, and providing for transitions across these commanding styles.

Baudin et al. [7] developed the concept of "increasing levels of assistance" in the development of knowledge-based systems. Their DEDAL document indexing and retrieval system is initially deployed with an index for each document and with a capability for users to add new indices. As the system gains experience with users searching for documents, it automatically creates new indices. Over time,

DEDAL can use the indexes and user feedback to support creation of new retrieval heuristics, vocabulary, and even domain models. Our work extends the notion of increasing assistance with the concept of incremental autonomy, in which the system gains not only capability but also trust over time, and with the concept of scaleable autonomy, in which we explicitly support multiple levels of interaction simultaneously throughout the evolution of the system.

Maes [16] addresses the need to increase trust as well as competence, and suggests that the best way is to advance both of these incrementally through experience with the user. We adopt this same philosophy. However, the learning interface agents developed by Maes perform only one style of reasoning, applied to one type of knowledge. As described above, our concept of variable autonomy explicitly supports multiple levels of interaction simultaneously. This requires a much more complex system, since user interaction at a low level may interfere with autonomous interaction at a higher level.

11. FUTURE WORK

The version of the RA to be flown on DS1 [3] demonstrates the basic requirements for operation with an autonomous agent. However, there are a number of useful features we would like to develop and demonstrate for future missions. In the current RA, the system is designed to abort the current plan execution if it loses capabilities needed for plan operation, and to then replan based on the degraded capabilities. This can be improved in two ways. First, we can provide a convenient mechanism for operators to specify dynamically that some goals are lower-priority and should be dropped if they cannot succeed, rather than leading to failure of the overall plan. Second, the RA will continue executing a degraded plan even if capabilities are subsequently restored that might permit a better plan, although these increased capabilities will be taken into account the next time the agent generates a plan. A mechanism for "opportunistic replanning" would enable the agent to distinguish contexts in which it is best to abort the plan and generate a new one to take advantage of the new opportunities. Another useful extension would be interactive support for robust planning, in which ground operators might inform the agent of particular faults they are concerned about (such as the potential failure of an antenna motor which has been behaving erratically), and the agent could then develop plans which would succeed even if that kind of fault comes up during execution.

The MM component of the RA maintains a mission profile and decides which mission goals are applicable to the current planning horizon. While the MM supports user editing of the mission profile, it currently does not do any actual mission planning, which must therefore be performed entirely on ground. Limited capabilities for on-board mission planning would enable the agent to dynamically prioritize activities based on higher-level mission goals and potentially free the operators from worrying about short-term way-points in the mission profile.

A final capability to mention concerns reconfiguring telemetry for anomaly resolution. We pointed out that all flight software modules support detailed debugging modes, and that operators can command the software to generate more debugging information to aid trouble-shooting. We suggested that the agent might make use of these facilities as well. This could enable the agent to improve its own ability to diagnose problems and could also help the ground to troubleshoot in the critical situations in which they are able to receive down-link but not uplink to the spacecraft. This feature is not implemented at present, but seems worthy of further exploration.

12. CONCLUSION

This paper has described the Remote Agent (RA), with particular emphasis on how it supports the wide range of capabilities required by ground operations. These include the need to support ground operators and the RA sharing a long-range mission profile with facilities for asynchronous ground updates; support ground operators monitoring and commanding the spacecraft at multiple levels of detail simultaneously; and enable ground operators to provide additional knowledge to the RA, such as parameter updates, model updates, and diagnostic information, without interfering with the activities of the RA or leaving the system in an inconsistent state.

The resulting architecture supports incremental autonomy, in which a basic agent can be delivered early and then used in an increasingly autonomous manner over the lifetime of the mission. It also supports variable autonomy, as it enables ground operators to benefit from autonomy when they want it, but does not inhibit them from obtaining a detailed understanding and exercising tighter control when necessary. While these concepts are similar to those addressed in the context of user interface agents, mixed-initiative planning systems, or expert advisory systems, the context of autonomous operations leads to some fundamentally different issues because the operator is only able to interact with the agent asynchronously and infrequently. The substantial delays in communication mean that the operation of the agent is harder to monitor at a detailed level, so the operator cannot always rely on taking immediate control when desired. Moreover, the strong desire to have the agent do something useful when we are not commanding it, to maximize its productivity, makes it all the more challenging and important to find effective ways to support ground interaction without interfering too much with the agent's ongoing activities.

These issues are critical to the successful development and operation of any autonomous agent to control spacecraft or other systems that must function autonomously for long periods of time, such as unmanned submarines, airplanes, rovers, and mobile robots operating under extreme conditions. As the agents we build become increasingly capable, we need to learn how to let them go out and do work for us without requiring constant management, but we also want to keep an open line of communication so that we are not shut out from the action. We want agents that

minimize the necessity for our attention, but yet still maximize the capability for us to interact with ,them to get what we want. Since the RA will be the first on-board agent to control a real spacecraft, the issues we had to address, and the methods we developed, should be of interest to researchers deploying a wide range of agent architectures for this domain.

13. ACKNOWLEDGMENTS

Thanks to Sandy Krasner for helping to develop many of the basic approaches we adopted. Thanks also to the NMP DS1 Flight/Ground Interface Working Group, particularly Tom Starbird, for many valuable ideas and discussions. Ron Keesing, Chris Plaunt, and Kanna Rajan are members of the remote agent team who were instrumental in implementing the ground interface architecture. Additional thanks are due to Bob Kanefsky and Pandu Nayak for defining the device health updates message. Finally, we would like to express our appreciation to Gregory Dorais for reviewing the manuscript.

REFERENCES

[1] J. Hackney, D.E. Bernard, and R.D. Rasmussen, "The Cassini Spacecraft: Object Oriented Flight Control Software," in *1993 Guidance and Control Conference*, Keystone, CO, 1993.

[2] Barney Pell, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams, "An Autonomous Spacecraft Agent Prototype," *Autonomous Robotics Journal*, 1998, (To Appear).

[3] Douglas E. Bernard, Gregory A. Dorais, Chuck Fry, Edward B. Gamble Jr., Robert Kanefsky, James Kurien, William Millar, Nicola Muscettola, P. Pandurang Nayak, Barney Pell, Kanna Rajan, Nicolas Rouquette, Benjamin Smith, Brian C. Williams, "Design of the Remote Agent Experiment for Spacecraft Autonomy", in *Proc. of IEEE Aeronautics Conference (Aero-98)*, Aspen, CO, IEEE Press, 1998.

[4] Erann Gat, "ESL: A Language for Supporting Robust Plan Execution in Embedded Autonomous Agents," in *Proceedings of the AAAI Fall Symposium on Plan Execution*, AAAI Press, 1996.

[5] E. J. Wyatt, R. L. Sherwood and M. K. Sue, "An Overview of the Beacon Monitor Operations Technology" in *Proc. of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, Tokyo, Japan, 1997.

[6] Reid Simmons and Greg Whelan, "Visualization Tools for Validating Software of Autonomous Spacecraft," in *Proc. of the Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, Tokyo, Japan, 1997.

[7] Catherine Baudin, Smadar Kedar and Barney Pell, "Increasing Levels of Assistance in Refinement of Knowledge-Based Retrieval Systems," *Knowledge Acquisition* **6(2)**, 179-196, 1994. Also appears as RIACS Technical Report 94.03.

[8] G.M. Brown, D.E. Bernard, and R.D. Rasmussen, "Attitude and Articulation Control for the Cassini Spacecraft: A Fault Tolerance Overview," in *14th AIAA/IEEE Digital Avionics Systems Conference*, Cambridge, MA, November 1995.

[9] Stephen F. Smith, Ora Lassila and Marcel Becker, "Configurable, Mixed-Initiative Systems for Planning and Scheduling," in *Advanced Planning Technology*, (ed. A. Tate), Menlo Park, CA, AAAI Press, 1996.

[10] Austin Tate, "Mixed Initiative Interaction in O-Plan". in *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, Stanford, California, USA, March 1997.

[11] Steven W. Mitchell, "A Hybrid Architecture for Real-Time Mixed-Initiative Planning and Control," in *Procs. of AAAI-97*, Cambridge, MA: AAAI Press, 1997.

[12] R. P. Bonasso, D. Kortenkamp, and T. Whitney, "Using a Robot Control Architecture to Automate Space Shuttle Operations," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI/IAAI 97)*, Menlo Park: AAAI Press, 1997.

[13] R. P. Bonasso, D. Kortenkamp, D. Miller, and M. Slack, "Experiences with an Architecture for Intelligent, Reactive Agents," *JETAI*, **9(1)**, 1997.

[14] R. J. Frainier, N. Groleau, L. R. Hazelton, S.P. Colombano, M. Compton, I. Statler, P. Szolovits, and L. R. Young, "PI-in-a-Box: a knowledge-based system for space science experimentation", *AI Magazine* **15(1)**, 39-51, 1994.

[15] Dan Ballard and Lisa Rippy, "A Knowledge-Based Decision Aid for Enhanced Situational Awareness," *in Procs. 13th Digital Avionics Systems Conference*, Phoenix, AZ: IEEE, 1994.

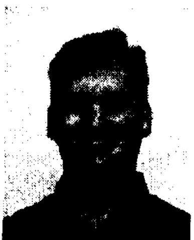
[16] Patti Maes, "Agents that Reduce Work and Information Overload", *Communications of the ACM*, Volume 37, Number 7, July 1994.

Dr. Douglas E. Bernard received his B.S. in Mechanical Engineering and Mathematics from the University of Vermont, his M.S. in Mechanical Engineering from MIT and his Ph.D. in Aeronautics and Astronautics from Stanford University. He has participated in dynamics analysis and attitude control system design for several spacecraft at JPL and Hughes Aircraft, including Attitude and Articulation Control Subsystem systems engineering lead for the Cassini mission to Saturn. Currently, Dr. Bernard is group supervisor for the flight system engineering group at JPL and team lead for Remote Agent autonomy technology development for the New Millennium Program.

Dr. Nicola Muscettola is a Senior Computer Scientist at the Computational Sciences Division of the NASA Ames Research Center. He received his Diploma di Laurea in Electrical and Control Engineering and his Ph.D. in Computer Science from the Politectnico di Milano, Italy. He is the principal designer of the HSTS planning framework and is the lead of the on-board planner team for the Deep Space 1 Remote Agent Experiment. His research interests include planning, scheduling, temporal reasoning, constraint propagation, action representations and knowledge compilation.

Dr. Barney Pell is a Senior Computer Scientist in the Computational Sciences Division at NASA Ames Research Center. He is one of the architects of the Remote Agent for New Millennium's Deep Space One (DS-1) mission, and leads a team developing the Smart Executive component of the DS-1 Remote Agent. Dr. Pell received a B.S. degree with distinction in Symbolic Systems at Stanford University. He received a Ph.D. in computer science at Cambridge University, England, where he studied as a Marshall Scholar. His current research interests include spacecraft autonomy, integrated agent architecture, reactive execution systems, collaborative software development, and strategic reasoning. Pell was guest editor for Computational Intelligence Journal in 1996 and has given tutorials on autonomous agents, space robotics, and game-playing.

Dr. Scott R. Sawyer is lead of the Mission Critical Systems group within the Intelligent Systems Center at the Lockheed Martin Advanced Technology Center, with interests in autonomous ground and flight systems for space applications. He was formerly the lead systems engineer for the DS1 flight/ground interface team. He has a B.Sc. in Physics from MIT, an M.S. in Engineering from Stanford University, and an M.A. and Ph.D. in Astronomy from the University of Texas at Austin.



Dr. Benjamin Smith is a member of the Artificial Intelligence group at JPL, and Deputy Lead of the JPL element of the DS1 planning team. He holds a Ph.D. in computer science from the University of Southern California. His research interests include intelligent agents, machine learning, and planning.