Designed for Autonomy: Remote Agent for the New Millennium Program

Douglas E. Bernard* and Barney Pell**

*Avionic Systems Engineering, Jet Propulsion Laboratory, California Institute of Technology
MS 156-235, 4800 Oak Grove Drive, Pasadena, CA, USA 91109
e-mail: douglas.e.bernard@jpl.nasa.gov
Tel: (818) 354-2597 FAX: (818) 354-1425

Caelum Research, NASA Ames Research Center
MS 269/2, Moffett Field, CA USA 94035
e-mail: pell@ptolemy-ethernet@arc.nasa.gov
Tel: (415) 604-3361 FAX (415) 604-3594

ABSTRACT

This paper describes an new approach to spacecraft commanding and control, called the Remote Agent. In the Remote Agent approach, the operational rules and constraints are encoded in the flight software, and the software may be considered to be an autonomous "remote agent" of the spacecraft operators in the sense that the operators rely on the agent to achieve particular goals. The operators do not tell the agent exactly what to do at each instant of time. Instead, they tell the agent exactly which goals they want achieved in a period of time and how and when to report in.

Missions of NASA's New Millennium Program will mark the first use of remote agent technologies to autonomously manage spacecraft in flight. The Remote Agent technology is one of the highest priorities for validation and is targeted for flight on the first New Millennium Mission, Deep Space One, anticipated to be launched in July, 1998.

1. INTRODUCTION

As we attempt to better understand our planet, our solar system, and our universe, we rely increasingly on robotic spacecraft to be our eyes and ears. These spacecraft enable us to extend our presence into space at a fraction of the cost and risk associated with human exploration. Where human exploration is desired, robotic precursors can help identify and map candidate landing sites, find resources, and demonstrate experimental technologies.

Current spacecraft control technology relies heavily on a relatively large and highly skilled mission operations team that generates detailed time-ordered sequences of commands to step the spacecraft through each desired activity. Each sequence is carefully constructed in such a way as to ensure that all known operational constraints are satisfied. The autonomy of the spacecraft is limited.

This paper describes an alternative approach to spacecraft commanding and control, called the Remote Agent. In the Remote Agent approach, the operational rules and constraints are encoded in the flight software, and the software may be considered to be an autonomous "remote agent" of the spacecraft operators in the sense that the operators rely on the agent to achieve particular goals. The operators do not know the exact conditions on the spacecraft, so they do not tell the agent exactly what to do at each instant of time. They do, however, tell the agent exactly which goals they want achieved in a period of time and how and when to report in.

Missions of NASA's New Millennium Program (NMP) will mark the first use of remote agent technologies to autonomously manage spacecraft in flight. The New Millennium Program is designed to validate high-payoff, cutting-edge technologies to enable those technologies to become more broadly available for use on other NASA programs. The Remote Agent technology is one of the highest priorities for validation and is targeted for flight on the first New Millennium Mission, Deep Space One (DS1), anticipated to be launched in July, 1998.

On the New Millennium Program, we use the term "Remote Agent." or RA, to refer to that portion of the flight software responsible for interpreting high-level goals, planning and scheduling activities to meet those goals, and robustly executing that plan in a fault-tolerant manner. We make use of three separate AI technologies in building the RA: Planning and Scheduling; Smart

Executive, and Model-Based Mode Identification and Reconfiguration.

Section 2 discusses the reasons that the spacecraft community is interested in increasing spacecraft autonomy. Section 3 covers aspects of the spacecraft domain that bear on the ease of applicability of AI technologies. Section 4 describes the requirements placed on the remote agent for the DS1 Mission. Section 5 outlines the NMP RA design approach and architecture, and Section 6 describes plans for future remote agent development.

2. NEED FOR AUTONOMY ON SPACECRAFT

The desire to increase the level of spacecraft autonomy comes from at least three separate objectives of spacecraft customers: taking good advantage of science opportunities, reducing spacecraft operations costs, and ensuring robust operation in the presence of faults.

2.1. Taking Good Advantage of Science Opportunities

Our science customers would like the spacecraft to be able to modify its sequence of actions more quickly based on late-breaking information available on the spacecraft. Several possibilities have been proposed.

An ultraviolet spectrometer on a comet flyby mission might identify a region of particular interest for intense scrutiny. With current technology, scientists have to make do with whatever pre-planned sequence of observations has been stored on-board and cannot reprogram any of those to examine more closely the newly-identified region of interest. With the RA, plans may be revised based on this new information hours or minutes before flyby.

On a Mars polar orbital mission, each orbit flies over a different ground swath due to the rotation of Mars. If one particular site were to exhibit an interesting signature—sought after geological resources or a beacon from a planetary rover, it would be possible with on-board autonomy to modify the orbit so that imaging or listening to that site happens much sooner or more frequently than in the original plan.

2.2. Reducing Spacecraft Operations Costs

Our funding sources are insisting that means be found to reduce operations costs. A fixed amount of funding is available from NASA for solar system exploration spacecraft development and operations. When operations costs are reduced, more resources become available for developing a wider variety of interesting solar system exploration missions. Analyzing operations budgets shows that development of detailed spacecraft sequences accounts for the largest operations expenditure.

By commanding spacecraft at a higher level of abstraction, much of the sequence development task becomes the responsibility of the flight software, and ground operations costs are reduced. Some of the savings come from an change in how we think about operations planning. The old approach was that all spacecraft activities needed to be predicted and approved by ground controllers. The new thinking is that the ground controllers do not (always) need to know the low-level details of spacecraft activities but only the capabilities of the spacecraft and the high-level goals.

2.3. Ensuring Robust Operation in the Presence of Faults

Our customers still require high reliability and the ability to respond to problems in flight. For existing spacecraft, the Fault Protection system often represents the most autonomous system on-board.

For example, the Cassini fault protection design needs to be able to detect a fault in the prime main engine during Saturn orbit , shut down the burnBring the spacecraft to a safe state, configure the redundant engine for use, allow the system adequate cool down time, compute the additional burn duration required to get into orbit once the propulsion system is cool enough to use, and schedule the continuation of the burn to commence as soon as possible.

A point design has been developed to accomplish this [1], but the approach required great effort and does not generalize to other similar problems.

With the RA approach, model-based fault protection would diagnose the problem in a principled manner, shut down the burn and safe the spacecraft as before. But now there is a general purpose planner on board. The RA solves the problem by asking for a plan given the same goals but with three extra bits of information: 1. the old prime main engine is not usable; 2. the initial orbit has been modified by the earlier propulsion maneuver; and 3. the initial temperature of the propulsion system is high. The constraints are built into an on-board planner, so a simple request for a new plan will generate a plan that will work, and a detailed point design is never required. Point test cases are still important to demonstrate that the system has the desired behavior.

Another advantage of the Remote Agent derives from the nominal and failure modeling used by the fault diagnosis engine. For hard-coded fault protection designs, the domain knowledge is implicit rather than explicit. This means that we rely on the fault protection algorithm developers to understand the system, and abstract from that understanding a design for which symptoms to look for and what responses to take when they show up. In contrast, with model-based fault diagnosis, the fault protection software engineers explicitly model how the system behaves in nominal and failure cases. Fault diagnosis then becomes a search for likely diagnoses given observed symptoms. Since the spacecraft designers understand the details of the system behavior at design time, there is an advantage to having them encode their knowledge explicitly at design time.

3. SPACECRAFT DOMAIN

In designing a remote agent for a spacecraft, a number of issues must be kept in mind which differ from earth-based robotic applications.

3.1. Unattended Operation and Fault Tolerance

Planetary science missions may usually be described as long periods of low-activity cruise followed by variable duration planetary or small body encounters. Once all systems have been checked out, spacecraft operations during cruise are typically simple and routine. Ground controllers track spacecraft periodically rather than continuously when during low activity periods.

The need for unattended operation means that the spacecraft must be highly fault tolerant. This requires an ability to detect problems and either achieve a safe state and wait for further instructions or diagnose the cause of the problem and take corrective action, allowing the spacecraft to continue operations without operator involvement. The latter is preferred for autonomous spacecraft.

3.2. Simple World Model

In contrast with rover applications, which may confront unpredictable boulder fields, spacecraft have extremely simple, predictable world models.

The trajectory is typically well understood, though it may be necessary to model with accuracy solar radiation pressure, time and direction of any and all thruster firings, and atmospheric forces. The DS1 spacecraft caries a catalogue of star orientations and planet and asteroid ephemerides.

For the 3-axis controlled spacecraft, the spacecraft attitude responds only to control torques and a few environmental torques such as solar radiation torque, atmospheric torque, and gravity gradient torque. Propellant slosh and spacecraft flexibility can be significant modeling challenges. Spinning spacecraft bring additional complexity to the dynamics modeling.

The major uncertainty in the orbital and attitude dynamics occurs when the spacecraft passes close to a body with an atmosphere. Atmospheric density models are of low accuracy because some of the processes are not well understood and also because of large temporal and spatial variations from the nominal values. Spacecraft encounter atmospheres when flying by Earth or Venus for a gravity assist, when using the atmosphere of Mars or Venus to help slow the spacecraft so as to achieve planetary orbit or to adjust the orbit. Cassini will fly close to Titan to sample its atmosphere and perform high resolution radar mapping.

The most difficult models to create and the ones with the highest uncertainty are the models of the measurements to be taken by the science experiments. This is to be expected, and if the spacecraft do their jobs well and accurately report sufficient information back to scientists on Earth—then future missions will benefit from improved models of the planetary bodies they are studying.

3.3. Redundant Systems

Since planetary spacecraft operate for years without the possibility of hands-on maintenance, electronic component reliability models predict a credible possibility that some components will fail before the end of the mission. Depending on the length of the mission, some functional—or sometimes block—redundancy is likely to be part of the design. The 11 year Cassini mission relies on block redundancy (prime plus backup unit) for most systems to achieve sufficient mission reliability. The DS1 mission is only two years long and has little redundancy. Even on this mission, there are a few instance of redundancy. The power supplies for the transmitter are block redundant. The attitude control thrusters can are configured so that any one thruster failure can be accommodated if diagnosed (albeit with degraded performance). The suite of attitude control sensors include a sun sensor, star tracker, and gyro. A safe sun-pointed attitude can be achieved if any two of the three are working. This approach is known as functional redundancy.

3.4. Autonomous Subsystems

All autonomy does not need to reside at one level. In a number of instances, one or another component or subsystem has a degree of autonomy, allowing the system-level spacecraft software to interact with it at the goal level. For example, the DS1 mission is flying an autonomous star tracker that has its own embedded computer. When polled, the star tracker provides a complete attitude estimate—or an error message indicating that no reliable estimate is currently available and why. The complexity of attitude determination through star field pattern matching is completely masked from the system-level software.

3.5. Resource Management

Autonomous spacecraft will increasingly be called upon to manage a variety of resources. These include consumable resources such as propellant, shared resources such as power, renewable resources such as battery state of charge. One important is the attitude of the spacecraft. There are many competing demands on spacecraft attitude including power generation (solar panels to the sun), communications (antenna to the Earth), and others.

3.6. Success without Autonomy

Another significant aspect of the spacecraft domain is that current practitioners have shown spectacular success with limited autonomy approaches, so remote agent developers need to capture what is best about existing practices while demonstrating the improvements that increased autonomy enables.

4. REQUIREMENTS ON SPACECRAFT REMOTE AGENT

For DS1, we have placed the following requirements on the on-board RA:

- Achieve goal oriented commanding
- Generate plans based on goals and current spacecraft state expectations
- Demonstrate model-based failure detection, isolation, and recovery
- Avoid premature response to transient failure indications
- Coordinate hardware states and software modes; determine the health state of hardware modules
- Replan after failure given new context
- Allow low-level command access to hardware
- Ensure "call-home" behavior in severe situations

5. DESIGN APPROACH AND ARCHITECTURE

Early work identified the key contributing technologies: on-board planning and replanning, multi-threaded smart executive, and model-based failure diagnosis. Once these had been identified, a five-month prototyping effort was undertaken. The New Millennium Autonomy Architecture rapid Prototype (NewMaap) effort is described in Ref. [2]. In NewMaap, we learned how to take advantages of the strengths and weakness of these three technologies and merge into a powerful system. After successful completion of the prototype, the RA was selected as one of the NMP technologies for DS1

Fig. 1 shows communications architecture for the Remote Agent's interaction with the rest of the spacecraft flight software. Note that all interaction with the hardware is the responsibility of the real-time software. The RA is layered on top of that software, but also gathers information from all levels to support fault diagnosis.

Several spacecraft commanding styles are possible. Goal-oriented commanding is the intended operating mode for most of the mission; provision has been made for updating the goals in flight. In a typical planning cycle, the executive is executing a plan and gets to an activity that can be integrated as "time to plan the next segment." The executive calls the mission manager with the current and projected spacecraft state including the health of all devices. The mission manager sends the projected state and appropriate goals to the planner scheduler that generates a new plan using priorities, heuristics, and domain models. See Ref. [3]. The planner sends this plan to the executive that creates an agenda of plan items and executes the agenda. See Ref. [4]. Plan execution robustness is added by making use of the Model-based Mode Identification and Reconfiguration (MIR) system, see Ref. [5]. The MIR system includes monitors, mode identification for nominal and failure conditions, communication of state to the executive and proposals of reconfiguration actions to take in the event of failures.

The following subsections describe how the various parts of the RA work together to achieve the desired level of autonomy.

5.1. Period planning

Our approach separates an extensive, deliberative planning phase from the execution phase, executing infrequently generated plans over extended time periods. When the executive reaches the planning task in the current planning horizon, it asks the planner to generate a plan for the next planning horizon while it continues to

execute the activities remaining in the current plan. When the executive reaches the end of the current horizon, the plan for the next horizon will be ready and the executive will then install it and continue execution seamlessly.

5.2. Planning at an abstract level

Ideally, we would like to have the planner represent the spacecraft at the same level of detail as the executive. This approach is taken by Bresina [6] and by Levison [7]. The approach, when feasible, has a number of benefits. First, it enables the planner to simulate the detailed functioning of the executive under various conditions of uncertainty, and to produce a plan which has contingencies (branches) providing quick responses for important execution outcomes. Second, it enables the use of one language rather than two for expressing action knowledge, which simplifies knowledge engineering and helps maintain consistency of interfaces.

Unfortunately, in our domain this single representation approach is not practical because the complexity of interactions at the detailed level of execution would make planning combinatorially intractable. Thus, we have found it necessary to make the planner operate on a more abstract model of the domain. Examples of abstractions are:

- –hiding details of subsystem interactions controlled by the executive
- –merging a set of detailed component states into abstract states
- – not modeling certain subsystems
- –using conservative resource and timing estimates

5.3. Requesting and Executing back-to-back plans

We address the problem of generating initial states for the next planning round differently depending on the status of the currently-executing plan. Plans normally include an activity to plan for the next horizon. At this point, the executive sends to the planner the current plan in its entirety, with annotations for the decisions that were made so far in executing it. The current plan serves as its own prediction of the future at the level of abstraction required by the planner. Thus, all the planner has to do is extend the plan to address the goals of the next planning horizon and return the result to the executive. The executive must then merge the extended plan with its current representation of the existing plan. The net result is that, from the executive's perspective, executing multiple chained plans is virtually the same as executing one long plan. This has the useful consequence that it enables the executive to engage in activities which span multiple planning horizons (such as a 3-month long engine burn) without interrupting them.

In the event of plan failure, the executive knows how to enter a stable state (called a standby mode) prior to invoking the planner, from which it generates a description of the resulting state in the abstract language understood by the planner. Note that establishing standby modes following plan failure is a costly activity, as it causes us to interrupt the ongoing planned activities and lose important opportunities.

5.4. Achieving robust plan execution

Such concerns motivate a strong desire for having robust plans, on the one hand, and an ability to exploit those robust plans to continue executing in the presence of a wide variety of execution outcomes on the other hand.

Our approach achieves plan flexibility by (1) choosing an appropriate level of abstraction for the activities and (2) generating plans in which the activities have flexible start and end times. By planning at an abstract level, the planner doesn't worry about the details of particular activites. This leaves the executive free to achieve the activities in different ways depending on the current situation. By being flexible about the start and end times of activities, the plan will still succeed even if activities took longer than anticipated. This also leaves room for trying many different ways to complete a task, possibly in the presence of failures.

In the event of hardware failures during execution, the executive draws on the expertise of the model-based diagnosis and reconfiguration, to diagnose the problem and then suggest a recovery. The MIR engine, Livingstone, uses the same device models to do the diagnosis and to suggest recoveries which are consistent with both the inferred current state of the spacecraft and with the goals being maintained by the executive. Livingstone responds to each recovery request with a single action suggestion, and updates its diagnoses as each suggestion is executed. If the suggested recoveries put the system back on track with respect to the plan within the time and resources allotted, the executive continues executing the original plan. In the event of a timeout, or if the system runs out of possible recoveries, the executive abandons the current plan, enters a stable standby mode, and requests a plan for the new situation.

6. FUTURE WORK

A number of desirable remote agent features are planned for future remote agents that will not be part of the DS1 RA. These enhancements will will further increase mission robustness, refine diagnostic capabilities, and simplify the process of representing and integrating knowledge throughout the software.

In our discussion of mission robustness, we discussed flexible planning and recovery capabilities. These capabilities will not help in cases where some preventative or preparatory action needed to be taken in the past to enable recoveries in the current situation. For example, if the primary engine breaks, the system may only be able to switch to the backup engine if it has been warmed up. Future remote agents will have the capability to anticipate such possible failures, or even opportunities, and to then build plans which provide the necessary resources so the system is prepared for many possible futures. A related capability in this vein is for the executive to understand the priorities in the plan, so that it can abandon individual tasks or threads of activity without failing the entire plan. This will enable high-priority activities to be completed even if low-priority activities fail.

In our discussion of diagnosis, we pointed out that the diagnosis system makes new inferences every time an action is taken or a new observation is made. In the event of failures, it will generate recoveries which may improve the situation. However, sometimes these actions taken during normal execution or even recovery will not present the right information to isolate the fault to an optimal level of detail. Our future work will develop methods for active testing, in which the system will conduct tests whose sole purpose is to help it improve its understanding of the state of the spacecraft. Examples of this capability include turning the spacecraft to see if a gyro is measuring turn rates correctly, and turning selected devices on and off to detect shorts.

In terms of knowledge engineering, we discussed how the various reasoning engines in the RA use different representations of knowledge. In many ways this is a necessary and useful feature, as it allows the planner to reason at a more abstract level than the executive, and the diagnosis system to reason at a more detailed level. While heterogeneous representations have a number of benefits, they also raise some difficulties. Most significant of these are the possibility for models to diverge rather than converge, and the need to duplicate knowledge representation efforts. Ideally, we would like to head toward an increasingly unified representation of the spacecraft, but we intend to do so always generalizing from powerful models capable of handling the complexities of our real-world domain.

REFERENCES

[1] Brown, Bernard, Rasmussen Cassini

[2] B. Pell, D. E. Bernard, S. A. Chien, E. Gat, N. Muscettola, P. Nayak, M. D. Wagner, and B. C. Williams, " An Autonomous Spacecraft Agent Prototype," Proceedings of the First International Conference on Autonomous Agents, Marinal Del Rey, CA, 1997

[3] N. Muscettola, B. Smith, C. Fry, S. Chien, K. Rajan, G. Rabideau, and D. Yan, "On-Board Planning for New Millennium Deep Space One Autonomy," Proceedings of the IEEE Aerospace Conference, Aspen, CO, 1997.

[4] B. Pell, E. Gat, R. Keesing N. Muscettola, and B. Smith, "Plan Execution for Autonomous Spacecraft", Proceedings of the AAAI Fall Symposium on Plan Execution, AAAI Press, 1996.

[5] B. Williams, P. Nayak, "A Model-based Approach to Reactive Self-Configuring Systems," Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI Press, Portland, OR, 1996.

[6] J. Bresina, W. Edgington, K. Swanson, and M. Drummond, "Operational Closed-Loop Obesrvation Scheduling and Execution," Proceedings of the AAAI Fall Symposium on Plan Execution, AAAI Press, 1996.

[7] R. Levinson, "A general programming language for unified planning and control," *Artificial Intelligence*, vol. 76, Special Issue on Planning and Scheduling, 1994.

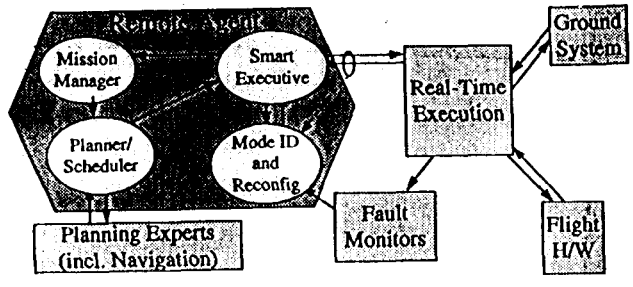

Figure 1: DS1 FSW Communications Architecture