# Robust Periodic Planning and Execution for Autonomous Spacecraft *

**Barney Pell** † **Erann Gat** § **Ron Keesing** † **Nicola Muscettola** ‡ **Ben Smith** §

## Abstract

The New Millennium Remote Agent (NMRA) will be the first on-board AI system to control an actual spacecraft. The spacecraft domain raises a number of challenges for planning and execution, ranging from extended agency and long-term planning to dynamic recoveries and robust concurrent execution, all in the presence of tight real-time deadlines, changing goals, scarce resource constraints, and a wide variety of possible failures. NMRA is one of the first systems to integrate closed-loop planning and execution of concurrent temporal plans. It is also the first autonomous system that will be able to achieve a sustained, multi-stage, multi-year mission without communication or guidance from earth.

## 1 Introduction

We are developing the first on-board AI system to control an actual spacecraft. The mission, Deep Space One (DS-1), is the first in NASA's New Millennium Program (NMP), an aggressive series of technology demonstrations intended to push Space Exploration into the 21st century. DS-1 will launch in mid-1998 and will navigate and fly by asteroids and comets, taking pictures and sending back information to scientists on Earth. One key technology to be demonstrated is spacecraft autonomy, including on-board planning and plan execution. The spacecraft will spend long periods without the possibility of communication with ground operations staff and will, in fact, plan how and when it will communicate back to Earth. It must maintain its safety and achieve high-level goals, when possible, even in the presence of hardware faults and other unexpected events.

This paper describes our approach to planning and plan execution in the context of spacecraft autonomy. Our approach is being implemented as part of the New Millennium Remote Agent (NMRA) architecture [Pell *et al.*, 1997]. This architecture integrates traditional real-time monitoring and control with constraint-based planning and scheduling [Muscettola, 1994], robust multi-threaded execution [Gat, 1996], and model-based diagnosis and reconfiguration [Williams & Nayak, 1996].

The paper is organized as follows. Section 2 discusses the spacecraft domain and requirements which influence our design. Section 3 describes our approach to planning, execution, and robustness, and illustrates the top-level loop of our system. Section 4 addresses the issues involved in generating plans to support robust execution, and Section 5 shows how such plans are executed. We then consider related work and conclude.

## 2 Domain and Requirements

The autonomous spacecraft domain presents a number of challenges for planning and plan execution. Many devices and systems must be controlled, leading to multiple threads of complex activity. These concurrent processes must be coordinated to control for negative interactions, such as vibrations of the thruster system violating stability requirements of the camera. Also, activities may have precise real-time constraints, such as taking a picture of an asteroid during a narrow window of observability.

Virtually all resources on spacecraft are limited and carefully budgeted. The system must ensure that they are allocated effectively to goal-achievement. Some resources, like solar panel-generated power, are renewable but limited. Others, such as total propellant, are finite and must be budgeted across the entire mission. The planner reasons about resource usage in generating plans, but because of run-time uncertainty the resource constraints must also be enforced as part of execution.

The planner and plan execution system must reason

*This paper appears in the Proceedings of IJCAI-97.

‡Recom Technologies, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

†Caelum Research, NASA Ames Research Center, MS 269/2, Moffett Field, CA 94035.

§Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109.

about and interact with external agents and processes, such as the on-board navigation system and the attitude controller. These external agents can provide some information at plan time and can achieve tasks and provide more information at run-time but are never fully controllable or predictable. For example, the attitude controller can provide estimates of turn durations at plan time, but the completion of turns during execution is not controllable and can only be observed. Plans must express compatibilities among activities, and the plan execution system must synchronize these activities at run-time.

In addition, planning and the information necessary to generate plans are also limited resources. Because of the limited on-board processing capabilities of the spacecraft, the planner must share the CPU with other critical computation tasks such as the execution engine, the real-time control loops and the fault detection, isolation and recovery system. While the planner generates a plan, the spacecraft must continue to operate. Moreover, the plan often contains critical tasks whose execution cannot be interrupted in order to install newly generated plans. Thus inserting a planning activity means many other activities must be suspended or postponed. Since planning is expensive, plan failure is costly. Thus plan execution must be robust in the face of a wide variety of hardware faults and delays.

## 3 Approach

Our approach separates an extensive, deliberative planning phase from the reactive execution phase, executing infrequently generated plans over extended time periods. How frequently and how far in advance the system should plan is constrained by several factors, including uncertainty about the results of execution. For example, uncertainty about how much thrust has accumulated after a thrusting maneuver means the system can't reliably plan how many thrusts will be needed to reach the target, thus reducing how far in advance it is productive to plan such thrusting activities. While uncertainty motivates frequent planning with short *scheduling horizons*, the cost of planning motivates plans with long horizons.

In our approach the generation of the next plan is explicitly represented in the current plan as a task. By considering the costs and constraints of planning, the planner automatically optimizes future planning activities. When the executive reaches this task in the current scheduling horizon, it asks the planner to generate a plan for the next scheduling 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.

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 *et al.*, 1996], and by [Levinson, 1994]. 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. Third, it enables the planner to monitor execution in progress and project the likely course of actions, and then provide plan refinements which can be patched directly into the currently executing plan.

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

These simplifications have several consequences which impact our design. One important consequence is that the planner can no longer model or predict intermediate execution states. Since the executive is managing multiple concurrent activities at a level of detail below the planner's visibility, it is difficult to provide a well-defined initial state as input to the infrequent planning process. Also, a newly generated plan may be invalidated by execution activities that occur during or even after planning but prior to execution of the new plan, since the initial conditions of that plan may no longer be consistent with the state of the spacecraft.

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. For example, a plan failure causing us to enter standby mode during the comet encounter would cause loss of all the encounter science, as there is no time to re-plan before the comet is out of sight. Such concerns motivate a strong desire for plan robustness, in which the plans contain enough flexibility, and the executive has the capability, to continue executing the plan under a wide range of execution outcomes.

## 4 Planning

A principal goal of the NMRA is to enable a new generation of spacecraft that can carry out complete, nominal missions without any communication from ground. This is a great departure from previous and current missions (such as Voyager, Galileo or Cassini) which rely on frequent and extensive communications from ground. In traditional missions, ground operations routinely uplink detailed command sequences to be executed during subsequent mission phases. Such communications require costly resources such as the Deep Space Network, which makes them very expensive. Uplink independence is particularly important for missions that require fast reaction times (as it is the case for autonomous rovers, comet landers and other remote explorers); in this case detailed ground-based control is infeasible due to long communication lags. In case of loss of uplink capabilities, previous spacecraft could carry out a critical sequence of commands stored on board before launch. However, these sequences were greatly simplified when compared to the uplinked sequences and could only carry out a small fraction of all mission goals.

### Mission Manager

NMRA is launched with a pre-defined "mission profile" that contains a list of all nominal goals to be achieved during the mission. The detailed sequence of commands to achieve such goals, however, is not pre-stored but is generated on board by the planner. A special module of the planner, the Mission Manager, determines the goals that need to be achieved in the next scheduling horizon (typically 2 weeks long), extracts them from the mission profile and combines them with the initial spacecraft state as determined by the executive. The result is a specific planning problem that, once solved, yields detailed execution commands. This decomposition into long-range mission planning and shorter-term detailed planning enables NMRA to undertake an extended diverse mission with minimal human intervention.

### Requirements for Robust Execution

The NMRA must be able to respond to unexpected events during plan execution without having to plan the response. Although it is sometimes necessary to re-plan, this should not be the only option. Many situations require responses that cannot be made quickly enough if the NMRA has to plan them.

The executive must be able to react to events in such a way that the rest of the plan is still valid. To support this, the plan must be flexible enough to tolerate both unexpected events and the executive's responses without breaking. This flexibility is achieved 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.

The abstraction level of the activities in the plan must be chosen carefully. If the activities are at too fine a level of granularity, then the plan will impose too many constraints on the behavior of the executive, making plan execution more fragile. However, if the granularity is too coarse, then there may be interactions among the sub-actions of activities that the planner cannot reason about. In DS1, activities are abstracted to the level where there are no interactions among their subactivities. This level allows the planner to resolve all of the global interactions without getting into details that would over-constrain the executive.

The other mechanism by which the executive can respond to events without breaking the plan is having activities with flexible start and end times. Plans in DS1 consist of temporal sequences of activities. Each activity has an earliest start time, a latest start time, an earliest end time, and latest end time. The planner uses a least commitment approach, constricting the start and end times only when absolutely necessary. Any flexibility remaining at the end of planning is retained in the plan. This flexibility is used by the executive to adjust the start and end times of activities as needed. For example, if the engine does not start on the first try, the executive can try a few more times. To make time for these extra attempts, the end time is moved ahead, but not beyond the latest end time.

Changing the start or end time of an activity may also affect other activities in the plan. For example, if the spacecraft must take science data five minutes after shut-

ting down the engine, then changing the end time of the `engine firing` activity will change the start time of the `take science data` activity. To make the changes, the executive must know about the temporal constraint between the `fire engine` activity and the `take science data` activity. The plan therefore contains all of the temporal constraints among the activities.

Although the planner is typically *enabled* to leave flexibility in the activity start and end times because the times are under-constrained, it is sometimes *required* to provide such flexibility in order to operate the spacecraft successfully. For example, when the engine is commanded to turn on, it goes through a warm up procedure and turns itself on. The warm up procedure can take up to ten minutes, but the actual warm up time is not known at plan time. It is not known until the engine actually turns on. We currently handle such cases by providing enough time in activities to handle worst-case outcomes, although we are developing a method to plan explicitly about execution-time uncertainty.

## 5 Execution

From the point of view of the NMRA executive, a plan is a set of time-lines. Timelines consists of a linear sequence of tokens, each of which represents an activity which should be taking place during a defined temporal period. A token has a start and end window, a set of pre- and post-constraints. The start and end windows are intervals in absolute time during which the token must start and end. The pre- and post-constraints describe dependencies with respect to the starts and ends of tokens on other time-lines.

There are three different types of pre- and post-constraints: *before*, *after*, and *meets*. The semantics of these constraints is fairly straightforward. A before constraint specifies that the start of a token must come before the start of another token. An after constraint specifies that the end of a token must come after the end of another token. The amount of time that may elapse between these two related events is specified as an interval. A meets-constraint specifies that the start (end) of a token must coincide with the start (end) of another token.

### Issues

Plan execution would be relatively straightforward were it not for the fact that different token types have different execution semantics. In particular, there are different ways of determining whether or not a particular activity has ended. Some activities are brought to an end by the physics of the environment or the control system (e.g. turns) while others are brought to an end simply by meeting all its internal plan constraints (e.g. the periods of constant-attitude pointing between turns).

The situation is further complicated by the fact that a naive operationalization of these constraints leads to deadlock. Consider a `constant-pointing` token A followed by a `turn` token B. Token A (waiting for the turn) should end whenever token B (the turn) is eligible to start. However, B is constrained by the planner to follow A, and so B is not eligible to start until A ends. Thus, A can never end, and B can never start.

Another issue is that some tokens don't achieve their intended post-conditions until some time after they have started. For example, consider a time-line for a device containing a token A of type `device-off` followed by token B of type `device-on`. The intent here is that the executive should turn the device on at the junction between A and B, but this cannot be done instantaneously. Thus, a token on another time-line, constrained to start after B, may fail if it depends upon the device being on, since the device may not in fact be turned on until some time after B starts. One possible solution to this problem is to change the planner model so that it generates a plan that includes an intermediate token of type `device-turning-on`, but this can significantly increase the size of the planner's search space, and hence the time and resources required to generate a plan.

To solve these problems, we separate the execution of a token into three stages: *startup*, *steady-state*, and *ending*. The startup stage performs actions to achieve the conditions that the planner intends the token to represent. The steady-state stage monitors and maintains these conditions (or signals failure if the conditions cannot be maintained). The ending stage allows the token to perform cleanup actions before releasing control to the next token on the time-line. Tokens may have null actions in one or more stages. The algorithm for executing a token in this three-phase framework is as follows:

1. Wait for the beginning of the token's start window.
2. In parallel
   (a) wait for token's pre-constraints to be true, and
   (b) check that the end of the start window has not passed. If it has, signal a failure.
3. Signal that the token has started.
4. Execute the achieve-portion of the token.
5. Spawn the maintain-portion of the token as a parallel task.
6. Wait for the start of the token's end window.
7. Wait for the token's post-conditions to be true.
8. Wait for the pre-conditions of the next token to be true, except those that refer to the end of this token.
9. Stop the maintain thread spawned in step 5, and execute the cleanup-portion of the token

10. Check that the end of the end window has not passed. If it has, signal a failure. Otherwise, signal that this token has ended.

This algorithm allows all the token types to be executed within a uniform framework.

## 6 Related Work

NMRA is one of the first systems to integrate closed-loop planning and execution of concurrent temporal plans. It is also the first autonomous system that will be able to achieve a sustained, multi-stage, multi-year mission without communication or guidance from earth.

Bresina *et al.* (1996) describe a temporal planner and executive for the autonomous telescope domain. Their approach uses a single action representation whereas ours uses an abstract planning language, but their plan representation shares with ours flexibility and uncertainty about start and finish times of activities. However, their approach is currently restricted to single resource domains with no concurrency.

Drabble (1993) describes the EXCALIBUR system, which performs closed-loop planning and execution using qualitative domain models to monitor plan execution and to generate predicted initial states for planning after execution failures. The "kitchen" domain involved concurrent temporal plans, although it was simplified and did not require robust reactions during execution.

Currie & Tate (1991) describe the O-Plan planning system, which when combined with a temporal scheduler can produce rich concurrent temporal plans. Reece & Tate (1994) developed an execution agent for this planner, and the combined system has been applied to many real-world problems including the military logistics domain. The plan repair mechanism [Drabble, Tate, & Dalton, 1996] is more sophisticated then ours, although the execution agent is weaker and does not perform execution-time task decomposition or robust execution.

The Cypress system [Wilkins *et al.*, 1995] and the 3T system [Bonasso *et al.*, 1996] also address the closed-loop integration of planning and execution in the context of concurrency, although neither of these systems deals with temporal plans. It is interesting to compare how these systems differ from ours concerning the generation of execution context for the planner and the integration of new planning information back into execution. Cypress shares the same action formalism between planning and execution. This enables the planner to watch over execution and simulate the results forward, as discussed in section 3. The planner can detect problems in advance and send back a detailed plan refinement, and the executive can replace un-executed portions of its current plan with new portions and continue running uninterrupted.

In 3T, the planner maintains such tight control over execution that it does not even send the full plan it has developed. Instead, it sends directives to the executive one at a time, and the executive then responds to each directive in turn. This provides an interesting solution to the problem of keeping the planner informed about execution and also to the problem of integrating new planning information into the execution context. However, this approach is problematic in our domain as it places severe time constraints on the planner so that it can decide what to do before the executive runs out of activities, and it requires the computational and informational resources to be available for planning on a continuous basis. This is a luxury we could not afford on a spacecraft, as discussed in section 3.

Other systems integrating planning and execution in real-world control systems include Guardian [Hayes-Roth, 1995], SOAR [Tambe *et al.*, 1995], Atlantis [Gat, 1992] and TCA [Simmons, 1990]. These systems invoke planning as a means to answer specific questions during execution (like whether a particular treatment would take effect in time to heal the patient, which evasive maneuver will counter the opponents current attack plan, and which path to take to get to a particular room). This use of planning contrasts with our approach, in which the planner coordinates the global activity in the system. The local approach has the advantage of making use of special-purpose planners which can be built to answer narrow questions, but our global approach has the advantage of ensuring that the different activities undertaken at execution will not interact harmfully. It is not clear how the local approaches can be extended to provide similar guarantees.

## 7 Conclusion

A growing body of work is addressing issues of robust planning and execution in the face of failures and uncertainty. The Lockheed Underwater Vehicle [Ogasawara, 1991] uses decision-theoretic planning and execution to select courses of action which maximize utility. CIRCA [Musliner, Durfee, & Shin, 1993] considers a set of states, actions, and critical failures to be avoided. It then inserts a set of sense-act transitions into a real-time controller to ensure that the controller will never enter the critical failure states. Cassandra [Pryor & Collins, 1996], Buridan [Draper, Hanks, & Weld, 1994], O-Plan [Currie & Tate, 1991] and JIC [Drummond, Bresina, & Swanson, 1994] all consider actions with uncertain outcomes and produce plans that enable execution-time recovery without having to take time out for replanning.

We are currently working on extending our planning approach to support such capabilities in the context of concurrent temporal plans. Our present levels of robustness are achieved using the complementary approach of

flexible, abstract, and conservative plans which can be exploited by a smart executive.

A final distinction between NMRA and most other planning and execution systems is that our planner actually plans how and when it will plan for the next horizon. That is, it inserts a "plan next horizon" activity into the plan and plans other supporting activities around this goal. Such activities include information-gathering activities which will be necessary before another plan can be built. The executive then achieves these activities to enable this form of planning over multiple horizons. We believe this is a necessary capability of extended agency, and one which will become of growing concern as we design autonomous agents to achieve goals unassisted over years or decades of activity.

## References


[Bonasso *et al.*, 1996] Bonasso, R. P.; Kortenkamp, D.; Miller, D.; and Slack, M. 1996. Experiences with an architecture for intelligent, reactive agents. *JETAI*.

[Bresina *et al.*, 1996] Bresina, J.; Edgington, W.; Swanson, K.; and Drummond, M. 1996. Operational closed-loop obesrvation scheduling and execution. In Pryor [Pryor, 1996].

[Currie & Tate, 1991] Currie, K., and Tate, A. 1991. O-plan: the open planning architecture. *Artificial Intelligence* 52(1):49–86.

[Drabble, Tate, & Dalton, 1996] Drabble, B.; Tate, A.; and Dalton, J. 1996. O-plan project evaluation experiments and results. Oplan Technical Report ARPA-RL/O-Plan/TR/23 Version 1, AIAI.

[Drabble, 1993] Drabble, B. 1993. Excalibur: A program for planning and reasoning with processes. *Artificial Intelligence Journal* 62(1):1–40.

[Draper, Hanks, & Weld, 1994] Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of AIPS94*, 31–36. AAAI Press.

[Drummond, Bresina, & Swanson, 1994] Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Procs. of AAAI-94*, 1098–1104. Cambridge, Mass.: AAAI Press.

[Gat, 1992] Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Procs. of AAAI-92*. Cambridge, Mass.: AAAI Press.

[Gat, 1996] Gat, E. 1996. ESL: A language for supporting robust plan execution in embedded autonomous agents. In Pryor [Pryor, 1996].

[Hayes-Roth, 1995] Hayes-Roth, B. 1995. An architecture for adaptive intelligent systems. *Artificial Intelligence* 72.

[Levinson, 1994] Levinson, R. 1994. A general programming language for unified planning and control. *Artificial Intelligence* 76.

[Muscettola, 1994] Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Fox, M., and Zweben, M., eds., *Intelligent Scheduling*. Morgan Kaufmann.

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

[Ogasawara, 1991] Ogasawara, G. H. 1991. A distributed, decision-theoretic control system for a mobile robot. *ACM SIGART Bulletin* 2(4):140–145.

[Pell *et al.*, 1997] Pell, B.; Bernard, D. E.; Chien, S. A.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M. D.; and Williams, B. C. 1997. An autonomous spacecraft agent prototype. In Johnson, W. L., ed., *Proceedings of the First Int'l Conference on Autonomous Agents*. ACM Press.

[Pryor & Collins, 1996] Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *JAIR* 4:287–339.

[Pryor, 1996] Pryor, L., ed. 1996. *Proceedings of the AAAI Fall Symposium on Plan Execution*. AAAI Press.

[Reece & Tate, 1994] Reece, G., and Tate, A. 1994. Synthesizing protection monitors from causal structure. In *Procs. AIPS-94*. AAAI Press.

[Simmons, 1990] Simmons, R. 1990. An architecture for coordinating planning, sensing, and action. In *Procs. DARPA Workshop on Innovative Approaches to Planning, Scheduling and Control*, 292–297. San Mateo, CA: DARPA.

[Tambe *et al.*, 1995] Tambe, M.; Johnson, W. L.; Jones, R. M.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *AI Magazine* 16(1):15–39.

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

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