On Board Planning for Autonomous Spacecraft

Nicola Muscettola
Chuck Fry
Kanna Rajan

Computational Sciences Division
NASA Ames Research Center
Moffet Field, CA 94035
{mus, chucko,kanna}@ptolemy.arc.nasa.gov

Ben Smith
Steve Chien
Gregg Rabideau
David Yan

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 525-3660
Pasadena, CA 91109-8099
{smith, chien, rabideau, yan}@aig.jpl.nasa.gov

Abstract

The Deep Space One (DS1) mission, scheduled to fly in 1998, will be the first spacecraft to feature an on-board planner. The planner is part of an artificial intelligence based control architecture that comprises a planner/scheduler, a plan execution engine, and a model-based fault diagnosis and reconfiguration engine. This autonomy architecture reduces mission costs and increases mission quality by enabling high-level commanding, robust fault responses, and opportunistic responses to serendipitous events. This paper describes the on-board planning and scheduling component of the DS1 autonomy architecture.

Introduction

The first mission of the New Millennium program (NMP)—Deep Space One (DS1), to launch in 1998—will feature an experimental on-board autonomy software system, the Remote Agent (RA). RA is an artificial intelligence based control system derived from the NewMaap technology demonstration [1]. RA has three components: the Executive (EXEC) [2], the Planner/Scheduler (PS), and the Mode Identification and Recovery engine (MIR) [3]. In this paper we describe PS.
RA is crucial for the future of space exploration because it enables reduction of mission costs and increases mission quality in several ways; some of these improvements are specific to RA's use of an on-board PS. First, because of its hierarchical architecture and its ability to generate plans on-board, RA enables commanding the spacecraft with fewer, high-level commands. Second, RA autonomously responds and recovers from failures that would normally require costly and time consuming ground intervention. PS supports failure response by looking in the future and deliberating about mission goal trade-off and global interactions: the rest of RA handles localized real-time failures that require quick reactions. Finally, on-board planning enables taking advantage of fortuitous events, such as better than expected resource consumption or serendipitous science discoveries (e.g., volcanoes on Io). This opens up fundamentally new and exciting unmanned space exploration missions where round-trip light time does not permit joy-sticking a spacecraft from Earth.
In this paper first we quickly describe the DS1 mission . Then we introduce the concept of high level commanding enabled by RA, contrast it to traditional sequencing, and highlight the role played by PS. Finally we describe more in detail the features and capabilities of the on-board PS.

The DS1 Mission

Spacecraft used by NMP missions are relatively inexpensive (e.g. the DS1 Mission is capped at $138.5 million) and must serve the primary objective of validating new technologies in flight; doing planetary science is an additional objective but one that also has the effect of stress-testing the technologies. This ordering of priorities allows the development and validation of technologies that would not pass the stringent reliability requirements imposed by typical planetary science missions. If the new technologies prove worthy on an NMP flight, then they will be used on future science missions.
The nominal DS1 mission is to launch in July 1998, fly by Asteroid 3352 McAuliffe in January 1999 taking a series of images, and then fly by Comet West-Kohoutek-Ikemura in June 2000. There will be comparatively infrequent ground communication coverage throughout the mission, only one Deep Space Network pass every two weeks. DS1 will carry aboard several new

technologies. During cruise preceding the first encounter each new technology will go through validation experiments. The RA is one of these technologies. Others include the on-board optical navigator (NAV), the ion-propulsion engine (IPS), and the Miniature Integrated Camera Spectrometer (MICAS).

RA is an experimental spacecraft control system. It consists of three components: the Planner/Scheduler (PS), the Executive (EXEC), and the Mode Identification and Recovery system (MIR). PS receives a set of high-level mission goals from an on-board *mission profile* and generates a *plan*—a set of synchronized procedures. Once executed, these commands will achieve the mission goals without violating resource, temporal, or safety constraints. The EXEC takes each plan procedure, decomposes it into low-level real-time commands and ensures the correct dispatching of these commands. MIR monitors device responses to commands, identifies faulty components, and suggests recovery actions to EXEC.

High Level Commanding

The RA architecture enables a new approach to spacecraft commanding, *high-level commanding*. PS is the primary module through which high level commanding happens. In this approach, ground interacts with a spacecraft through abstract directives or *goals* instead of detailed streams of instructions. The responsibilities of PS are: (1) to select among the proposed goals those to be achieved at any point in time; (2) to compromise between the level of achievement of the selected goals, and (3) to expand the *procedures* needed to achieve the goals. PS ensures the satisfaction of various synchronization constraints among procedures and resolves resource conflicts. The set of expanded procedures and constraints among them constitutes a *plan*.

In contrast, in the traditional approach a spacecraft is commanded with a sequence of time-tagged commands to the real-time device drivers. Such a sequence could be highly optimized to "squeeze" as much performance as possible out of the spacecraft. However, temporal and resource constraints and fault protection goals also have to be ensured at an extremely detailed level. The consequence is that developing a sequence is a very exacting and time consuming process, often requiring months of manual labor. Once generated, a sequence is very difficult to modify.

Three are the primary benefits of high-level commanding when compared to traditional sequencing: *modularity, execution flexibility* and *robustness*.

Modularity: a PS plan makes very explicit the hierarchical decomposition of responsibilities between the different flight software components. Each plan procedure is expanded by EXEC into fairly complex sequences of real-time commands on the basis of actual execution conditions. Since the plan already resolves synchronization and resource allocation constraints among procedures, this expansion is highly localized and, therefore, greatly simplified. Extensive validation of these small sequences is much simpler than the validation of those generated in the traditional approach.

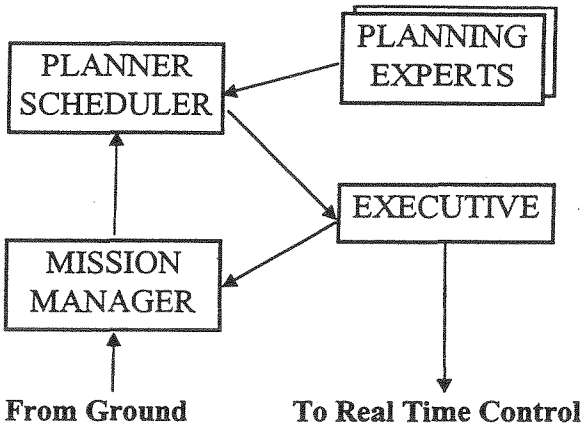

Figure 1: High Level Commanding

Execution flexibility: Procedures in a plan can be potentially executed in parallel. The plan explicitly represents and maintains temporal constraints between concurrent procedures. For example, a temporal constraint can express that procedure A must start from 30 to 60 minutes after procedure B, or that procedure B must execute while procedure C is executing, or that procedure A ends exactly when procedure C starts. PS ensures the consistency of the network of temporal constraints in the plan and infers *time ranges* during which a procedure can start and end. Unlike simple time tags, time ranges give EXEC the flexibility to compensate for execution delays caused by locally recoverable failures.

Robustness: an on-board planner can also make fault protection simpler and more robust than traditional sequencing. In the traditional approach, a sequence is infrequently uplinked to a spacecraft and therefore needs to include contingencies to handle a wide variety of failure conditions. In a "fail operational" scenario (e.g., a scenario in which the spacecraft autonomously recovers from a fault) the on-board sequence must be restarted, it must command the assessment of the new execution conditions, and react conditionally on the basis of this assessment. Because of the large number of possible failure conditions and the low level of the instructions in a sequence, the size of a robust sequence can be very large and the effort needed to build it very high. This is why "fail operational" scenarios are avoided as much as possible in traditional mission and are usually confined to critical mission phases (e.g., Cassini Saturn Orbit Insertion, encounter). In the RA approach, plans are valid only for the execution conditions known at the time PS was invoked and therefore the sequence of real-time commands issued is simpler and smaller. When execution conditions differ so much from the initial assumptions

that local failure recovery is insufficient, execution of the plan stops and PS is asked for a new plan that takes into account the new situation. Dealing with fault conditions on an as-needed basis simplifies the solution of the fault protection problem.

Generating Plans from Goals

Figure 1 describes how the DS1 PS implements high-level commanding within the RA architecture. A long-term plan containing goals for the entire mission, the *mission profile,* is stored and maintained on-board by the *Mission*

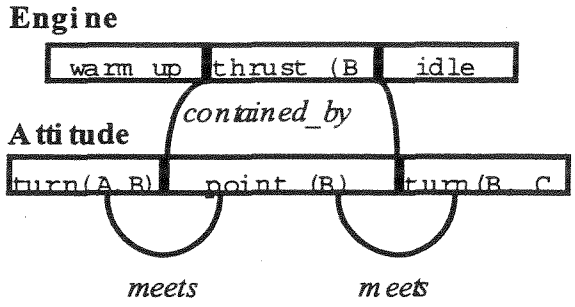

Figure 2: Example Timelines

Manager (MM). Ground operations interacts with MM to add, modify and delete goals in the mission. MM also responds to EXEC's requests for new plans by selecting a new set of goals from the mission profile, combining it with initial spacecraft state information provided by EXEC and sending it to PS. The time horizon covered by PS is typically two weeks during cruise and a few days during encounter. When a plan is ready, PS sends it to EXEC. When EXEC has almost completed execution of its current plan, it sends a new request to MM; this also happens when the EXEC is maintaining the spacecraft in standby mode after the occurrence of a major failure. In this case the initial spacecraft state will clearly identify which capabilities (if any) have been degraded by the fault. PS takes this information into consideration when generating the new plan.

Besides the mission profile, goals also come from other on-boards systems, the *planning experts*. For example, NAV communicates to PS of which beacon asteroids it needs pictures in order to estimate the current spacecraft position. The use of goals generated on board means that the spacecraft can modify its behavior, based on new information not known to ground. This capability is particularly important if the spacecraft has only infrequent contact with the ground or if ground reaction is too slow to take advantage of serendipitous opportunities. Goals generated on-board are incorporated in the plan during plan generation.

Plan representation

Both PS and MM use the same underlying system to represent, the *plan database.* A plan database is organized in several parallel *timelines*, each comprised of a sequence of *tokens.* A timeline describes the future evolution of a single component of the spacecraft's state vector. The set of tokens active at a given time represent the state vector value at that time. Goals and procedures are both represented as tokens. Each token consists of a state variable descriptor (specifying to which timeline the token belongs), a type (a symbolic representation of the goal or procedure and its parameters), a start-time, an end-time and a duration. Timelines can also represent renewable resources such as battery state of charge, non-renewable resources such as fuel, and aggregate resources (i.e., resources that can be allocated in parallel to several consumers) such as electric power. Temporal constraints synchronize resource allocation tokens with the corresponding consumer tokens. The type of the resource tokens indicate the amount requested and the modality of consumption (e.g., constant, linear depletion).

For example (Figure 2), a plan timeline may describe the state of the engine (warming up, thrusting, or idle) and another for the spacecraft attitude (e.g., pointing to a target, turning from target A to target B). The plan database also explicitly represents temporal constraints between tokens. These include constraints synchronizing tokens on separate timelines (e.g., "the spacecraft attitude must be pointing to target B while the engine is thrusting") and ordering tokens on the same or different time lines (e.g., "before the spacecraft can point to attitude A it must turn from its previous attitude B to A").

One important feature of the plan database is that decision variables (e.g., start or end time) and constraints among them are explicitly represented. The database then uses constraint propagation to infer valid ranges of values for variables and to detect inconsistencies (e.g., contradictory temporal constraints between tokens). This allows PS to concentrate on establishing constraints instead of selecting exact values for decision variables, an approach that often avoids over-commitment errors and therefore minimizes backtracking on earlier commitments.

The plan database can represent a plan at any stage of partial completion. Unlike complete plans, incomplete plans can have gaps between tokens on a timeline. Also, an incomplete plan may include an as-yet-unimplemented request for a constraint between tokens (see the section "The Domain Model"). The presence of these gaps and unfulfilled requests prompts PS to add tokens and constraints until the plan is complete. More representational details on the plan database can be found in [4].

The Domain Model

In a valid plan tokens must satisfy many constraints, including ordering (e.g., the catalyst-bed heaters must warm up for ninety minutes before using the reaction control thrusters), synchronization (e.g. the antenna must be pointed at the Earth during uplink,) safety (e.g. do not point the radiators within twenty degrees of the sun), and resource constraints (e.g. the MICAS camera requires fifteen watts of power). These are all expressed as temporal and parameter type constraint templates, or *compatibilities*, among token prototypes. The planning model is a set of compatibilities that must be satisfied in every complete plan. More formally, a.compatibility is a temporal relation that must hold between a *master token* and a *target token* whenever the master token appears in the plan. If the master token does not occur in the plan, the relation does not need to be satisfied. Compatibilities also specify relations between arguments in the master token type and value equivalence between arguments in the master and in the target token types. All compatibilities associated to a token are organized into a Boolean *compatibility tree.*

A simple compatibility tree is shown in Figure 3. It says that the state in which the MICAS camera is on must be preceded by a state in which it is turning on, and followed by one in which it is turning off. While the camera is on, it consumes fifteen watts of power.

```
(MICAS_On)
  :compatibilities
  (AND
    (met_by (MICAS_Turning_On))
    (meets  (MICAS_Turning_Off))
    (equal  (REQUEST (Power 15)))
```

Figure 3: A Compatibility Tree

Planning Algorithm

The planner searches in the space of incomplete or partial plans [5] with additional temporal reasoning mechanisms [6 and 4]. As with most causal planners, PS begins with an incomplete plan (given to it by MM) and attempts to expand it into a complete plan. The plan is complete when it satisfies all of the compatibilities in the plan model and there are no gaps on any timeline. The set of "defects" that need to be fixed in an incomplete plan in order for it to become complete is called the plan *conflicts.* Figure 4 summarizes the basic "conflict fixing" loop by PS. Each decision is typically made using heuristics and, when heuristic information is not particularly strong, using a uniform randomized rule (deterministic random number generator). If the wrong decision is made, PS will eventually reach a dead end, backtrack, and try a different path.

For example, consider one of the possible conflict types, *open compatibility.* An open compatibility is a temporal and parametric constraint that must exist between a *master* token already in the plan and a *target* token that may or may not be in the plan. For example, the compatibility `A meets B` is open if A is in the plan but B is not, or if both A and B are in the plan but the relation `A meets B` is not explicitly enforced. PS can satisfy an open compatibility with one of three resolution strategies. It can add the target token to the plan in such a way that it satisfies the temporal relation; it can adjust the start or end time of either the target or master token in order to satisfy the relation; or, it can decide that the relation will be satisfied by a token in the next planning horizon, and can therefore be ignored. These options are called *adding*, *connecting*, and *deferring*, respectively. Deferred compatibilities are maintained in the plan and carried forward to the next planning horizon as part of the initial state. PS will chose one of these options when it addresses an instance of an open compatibility conflict.

While plan has open compatibilities:
1.pick an open conflict;
2.select and apply a resolution strategy;
3.if no resolution is possible, backtrack

Figure 4: Planning Loop

Goal Prioritization

The overall mission goals depend on achieving a careful balance between potentially conflicting goals generated by independent sources (e.g., the science team, the navigation team). Conflicts typically arise because of over-subscription of limited resources (e.g., power, time). When a compromise is possible, PS appropriately distributes the use of available resources. When a compromise is not possible, then PS selects some of the lowest priority goals for postponement or outright rejection.

In PS tokens that have not yet been inserted onto a timeline, or *free token,* constitute a conflict category for which one of the possible resolution strategies is to reject the token. PS decides if the free goal token will be inserted. In DS1 PS does not explore all possible permutations of free token achievements but follows a statically assigned prioritization scheme (e.g., science goals have highest priority, followed by navigation goals and then by telemetry goals). This scheme, sufficient for DS1, avoids exponential backtracking but can yield sub-optimal solutions. Enhanced goal prioritization will be included in future PS versions.

Failure Response

RA provides a two level failure response — an immediate *reactive* response, and a longer term *deliberative*

response. This is typical of many autonomy architectures (e.g., Soar [7], Guardian [8]). The fast, real-time reactive behavior is implemented by EXEC and MIR. If this fails to solve the problem within the time and resource constraints of the current plan, then the failure can endanger future goals in the plan. In this case EXEC puts the spacecraft in standby, PS is called to assess the failure's impact on the remaining goals to decide how to best proceed. The deliberative response also addresses "advantageous failures" (e.g., serendipitous discoveries) and is the basis for enabling fundamentally new types of science missions.

This two level response results in simpler and more robust plans facilitating spacecraft commanding. The plans are simpler since they can address only the nominal case and trust that failures will be handled properly as they arise. Failures are either resolved by the reactive layer and allow the plan to continue, or cannot be resolved, in which case the plan breaks and the PS generates another nominal plan based on the new spacecraft state.

The plans are also more robust. This is partly due to the failure response mechanism, partly due to the hierarchical nature of the RA, and partly due to the plan representation. The hierarchy allows the tokens in the plan to describe fairly abstract procedures. The plan representation allows flexible start and end token times. Therefore EXEC has wide latitude in executing tokens, being allowed to respond to failures by retrying commands or trying alternate approaches. The extra failure response time needed is absorbed by the flexibility in the token's start and end times.

Conclusions

On-board planning is crucial for spacecraft autonomy. It can reduce mission costs and improve mission quality by allowing high-level commanding, enabling achievement of mission goals in the presence of failures without ground intervention, and taking advantage of fortuitous events. The DS1 mission marks the first on-board planner to fly on a spacecraft. The validation of this technology will open the way for future autonomous missions.

Acknowledgements

This work was performed in part at the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

References

[1] Pell, B., Bernard, D., Chien, S., Gat, E., Muscettola, N., Nayak, P., Wagner, M. and Williams, B. 1996. A remote agent prototype for spacecraft autonomy. In Proceedings of the SPIE Conference on Optical Science, Engineering and Instrumentation.

[2] Pell, B., Gat. E., Kesing, R., Muscettola, N., and Smith, B., 1997., Plan Execution for Autonomous Spacecraft in *IJCAI* 97 (forthcoming).

[3] Williams, B.C., and Nayak, P.P., 1996. A model-based approach to reactive self-configuration systems. In *Proceedings of AAAI-96*, pp 971-978.

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

[5] Weld, D.S., 1994. An Introduction to Least Commitment Planning, *AI Magazine* Winter 1994.

[6] Allen, J.F. and Koomen, J.A. 1983. Planning using a temporal world model. *IJCAI 83*. pp. 741-747.

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

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