AIAA-99-4512

Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment

Douglas Bernard[*], Gregory Dorais[†], Edward Gamble[*], Bob Kanefsky[†], James Kurien[‡], Guy K. Man[*], William Millar[†], Nicola Muscettola[§], Pandurang Nayak[¶], Kanna Rajan[†], Nicolas Rouquette[*], Benjamin Smith[*], William Taylor[§], Yu-Wen Tung[*] [#]

In May 1999 state-of-the-art autonomy technology was allowed to assume command and control of the Deep Space One spacecraft during the Remote Agent Experiment. This experiment demonstrated numerous autonomy concepts ranging from high-level goal-oriented commanding to on-board planning to robust plan execution to model-based fault protection. Many lessons of value to future enhancements of spacecraft autonomy were learned in preparing for and executing this experiment. This paper describes those lessons and suggests directions of future work in this field.

INTRODUCTION

Robotic spacecraft are making it possible to explore other planets and understand the dynamics, composition, and history of the bodies that make up our solar system. These spacecraft enable us to extend our presence into space at a fraction of the cost and risk associated with human exploration. They also pave the way for 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 operations and control technology relies heavily on a relatively large and highly skilled mission operations team that generates detailed time-ordered sequences of commands or macros to step the spacecraft through each desired activity. Each sequence is carefully constructed on the ground in such a way as to ensure that all known operational constraints are satisfied. The autonomy of the spacecraft is limited.

An alternative approach to spacecraft commanding and control uses Remote Agent (RA) software technology and was demonstrated by the Remote Agent Experiment (RAX) on the Deep Space One (DS1) spacecraft. In the Remote Agent approach[1 2 3], Artificial Intelligence (AI) technology is used to encode the operational rules and constraints in the flight software. 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 to achieve in each period of time as well as how and when to communicate with the ground controllers.

The DS1 Remote Agent Experiment achieved multiple technology objectives. A primary objective of the experiment was to provide an on-board demonstration of spacecraft autonomy. This demonstration addressed nominal operations, including goal-oriented commanding, extended agency via back to back planning for sequential periods, and closed-loop goal-oriented plan execution responding to both time-driven and event-driven events. In addition, the experiment demonstrated extensive fault protection capabilities, including failure diagnosis, failure recovery using both repair and reconfiguration, on-board replanning following otherwise unrecoverable failures, and system-level fault protection.

An equally important, and complementary, objective of the experiment was to learn how to integrate a remote agent into the extensive flight and ground spacecraft control infrastructure. This objective was achieved by a three-pronged approach. First, a successful on-board demonstration required integration of the Remote Agent with the spacecraft flight software. This integration

[*] Jet Propulsion Laboratory, California Institute of Technology
[†] Caelum Research, NASA Ames Research Center
[‡] NASA Ames Research Center
[§] Recom Technologies, NASA Ames Research Center
[¶] RIACS, NASA Ames Research Center
[#] Copyright © 1999 by the American Institute of Aeronautics and Astronautics, Inc. The U.S. Government has a royalty-free license to exercise all rights under the copyright claimed herein for governmental purposes. All other rights are reserved by the copyright owner.

provided valuable information on required interfaces and performance. Second, the experiment forced the development of a testing methodology that was able to give the spacecraft engineers adequate confidence in the plans generated by the remote agent. Third, the experiment was operated with close cooperation between Remote Agent team members and DS1 operations engineers. Together, we learned about operating a spacecraft controlled by a remote agent.

In May 1999, the Remote Agent Experiment was executed successfully, demonstrating the applicability of Remote Agent technologies to spacecraft commanding and control. This experiment in spacecraft autonomy has allowed the remote agent team to learn many lessons about the technology needed for spacecraft autonomy, the process of technology infusion, and some of the obstacles remaining before spacecraft with a high level of autonomy will be flown. The intent of this paper is to share these lessons with the spacecraft and Artificial Intelligence communities so that we may work together to address these issues.

The Remote Agent was tremendously successful in showing what is possible. A major measure of its eventual success will be the extent to which our communities are able to apply what we've learned here to future challenges. We believe that highly-capable autonomous spacecraft are possible and their development can start soon; we also have quite a bit of work still to do to make that happen.

BENEFITS FOR MISSIONS

The real customers for the capabilities of the remote agent flight software are the ground operators. In this paper, we discuss a number of advantages that this technology offers from the customer's point of view.

High Level Commanding:

In today's environment, the operating model is that the behavior of the spacecraft in response to proposed commands is predicted and understood at a very detailed level. Typically each detail is reviewed and approved by cognizant representatives of each subsystem. We propose a paradigm shift: create on-board models of flight rules and constraints and command response behavior. Send high-level goals to the spacecraft and allow the spacecraft to create a plan that meets the goals while satisfying all constraints. The operators have the option to simulate the planning process ahead of time and predict the most likely spacecraft behavior as before. Eventually, it is assumed that it will be possible to reduce this simulation to spot checks when the circumstances are similar to those that the on-board planner has dealt with previously.

Robust Execution

Fixed time-based sequences are brittle: one unexpected problem is sufficient to cause the sequence to fail and then the spacecraft must take some contingency action, often switching to a "Safe" mode of some sort. In the remote agent approach, the "smart executive" is responsible for executing a plan rather than a sequence. The distinction between a plan and a sequence is that the plan contains not only the required tasks and the desired execution times, but also the event-relative constraints and information on how much flexibility each task has in its requested start and stop times. The remote agent executive chooses items for execution when all time- and event-relative preconditions have been met. Once the executive starts a task, it takes responsibility for its successful completion. If a problem occurs in execution, the executive has the ability to retry the same and different approaches to achieving a task.

Extensive On-Board Behavioral Models

Current spacecraft rely heavily on logical expressions to make on-board decisions. These may take many forms, such as: If Message A arrives, send message B. If M > N set Z to W, etc. These have the disadvantage of requiring new rules of each additional desired behavior. The remote agent relies on a combination of domain-specific behavioral models plus general-purpose reasoning engines. In the on-board planner, models include: state transitions required to set up a target state, resources required by tasks, flight rules and constraints, and detailed numerical planning experts that can respond to specific queries such as "How long does a turn from attitude X to attitude Y take"? For fault protection, the remote agent uses models of the behavior of the hardware and software in nominal and certain failure modes.

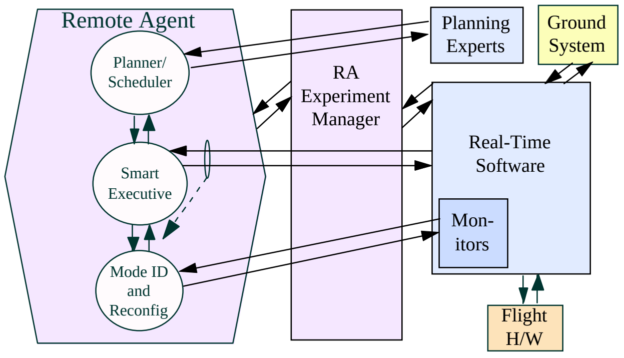
TECHNOLOGY

The Remote Agent is formed by the integration of three separate Artificial Intelligence technologies: an on-board planner-scheduler, a robust multi-threaded executive, and a model-based fault diagnosis and recovery system.

Remote Agent Architecture

The RA architecture and its relation to flight software are shown in Figure 1. Viewed as a black-box, RA issues commands to real-time execution flight software (FSW) to modify spacecraft state, and receives state information through a set of monitors that filter data streams into a set of abstract properties. The RA itself is comprised of three components: a Planner/Scheduler[4] (PS) that includes a Mission Manager, a Smart Executive[5] (EXEC), and a Mode Identification and Reconfiguration module[6] (MIR) also known as Livingstone.

Figure 1: RAX Architecture



PS formulates near-term planning problems based on a long-range mission profile representing the goals of the mission, extracts goals for the next scheduling horizon, combines them with a projected spacecraft state provided by EXEC, and formulates a planning problem. This decomposition into long-range mission planning and shorter-term detailed planning enables RA to undertake an extended mission with minimal human intervention.

PS then takes a plan request from MM and produces a flexible, concurrent temporal plan for execution by EXEC. PS constructs plans using domain constraints and heuristics in its knowledge base. Third-party software modules provide efficient computation for specialized problems.

EXEC executes a plan by decomposing the high-level plan activities into primitives, sending out commands, and monitoring progress based on direct feedback from the command recipient or on inferences drawn by MIR. If some task cannot be achieved, EXEC may attempt an alternate method or may request a simple recovery plan from Livingstone. If the EXEC is unable to execute or repair the current plan, it cleanly aborts the plan and attempts to bring the spacecraft into a safe state while requesting a new plan from MM.

Livingstone is responsible for mode identification (MI) and mode reconfiguration (MR). MI observes EXEC issuing commands, receives events from monitors, and uses model-based inference to deduce the state of the spacecraft and provide feedback to EXEC. MR serves as a recovery expert, taking as input a set of EXEC constraints to be established or maintained, and uses declarative models it shares with MI to recommend a single recovery action to EXEC.

The RAX manager, a software task belonging to the DS1 flight software, mediated all communication between the Remote Agent and the flight software on DS1.

FLIGHT QUALIFICATION

One of the most difficult challenges facing systems offering increased autonomy is that of demonstrating to the satisfaction of technologists and customer alike that the software will behave acceptably in a wide variety of situations for which it will not be pre-tested.

The remote agent has approached the qualification issue from a number of fronts:

Our primary testing approach was to exercise the RA on scenarios clustered around our expected experiment baseline. These were performed on platforms of varying fidelity. This proved effective for our experiment, but it is sensitive to changes in the baseline.

Formal methods can validate that the *design* (but not the implementation) of critical algorithms in the core engines meet certain requirements. We employed formal methods to validate some parts of the RA as feasibility studies, but it was not part of the testing process. Formal methods will need to be part of any full-scale testing effort of the Remote Agent.

Finally, we designed tools to automatically verify RAX output. One key tool was a “flight-rule checker” that converted the RAX execution traces into a form that could be checked by a DS1 tool for verifying that sequences obey the flight rules.

Testing Approach

Autonomous systems, such as the RA, need to respond robustly in a wide range of situations. Verifying that they respond correctly in all situations would require a huge number of test cases. To make matters worse, the tests should ideally be run on high-fidelity testbeds, which are heavily oversubscribed, difficult to configure correctly, and unable to run faster than real time, e.g., we could run only 10 tests in four weeks on one of DS1's high-fidelity testbeds. To address these problems, we employed a 'baseline testing' approach to reduce the number of tests, and exploited several lower-fidelity testbeds to increase the number of tests we could run[7].

We tested a number of nominal and off-nominal variations around two baselines. The variations comprised variations in spacecraft behavior that we might see during execution and changes to the baseline scenario that might be made prior to execution. These included variations to the goals in the mission profile, variations in when faults might occur, and variations in the FSW responses.

The tests were distributed among the low, medium, and high fidelity platforms. The two low-fidelity platforms were the 'babybed' and 'radbed'. The babybed had a PowerPC CPU, and the radbed, a flight CPU. Both ran with low-fidelity simulators written by the RAX team. The medium fidelity platform was the 'papabed', which had a flight CPU, bus, and memory and official DS1 simulators. The highest-fidelity platforms, the 'hotbench' and 'DS1 testbed', had flight computers and were connected to flight spare hardware or engineering models where feasible (see Table 1).

Table 1: DS1 Testbeds

Platform	Fidelity	CPU	Hardware	Availability	Speed
Spacecraft	Highest	Rad6000	Flight	1 for DS1	1:1
DS1 Testbed	High	Rad6000	Flight spares + DS1 sims	1 for DS1	1:1
Hotbench	High	Rad6000	Flight spares + DS1 sims	1 for DS1	1:1
Papabed	Med	Rad6000	DS1 simulators only	1 for DS1	1:1
Radbed	Low	Rad6000	RAX simulators only	1 for RAX	1:1
PowerPC	Lowest	PowerPC	RAX simulators only	2 for RAX	7:1

The architecture of RA allowed us to run certain tests on lower-fidelity testbeds and be confident that their results would hold on higher-fidelity testbeds. Specifically, the RA commands and monitors the spacecraft through well-defined interfaces with the FSW. Those interfaces were the same on all platforms, as were the range of possible responses. Only the fidelity of the responses improved with platform fidelity. This allowed us to exercise a wide range of nominal and off-nominal behaviors on the babybeds and radbed, test the most likely off-nominal scenarios on the papabed, and test only the nominal scenarios and certain performance and timing related tests on hotbench and testbed.

GROUND TOOLS

The primary set of ground tools developed to assist operators during the RAX experiment was very simple. Packetview, a tcl/tk based GUI was the basis of these tools and proved to be invaluable. It obtained the raw binary stream from the Deep Space Network (DSN), and displayed it as legible text. During the course of the flight experiment when the DS1 flight team needed exact times for specific events known to have happened, Packetview computed the actual spacecraft time when the event occurred and displayed the data along with relevant timing parameters.

In addition each of the RA modules had ground tools which were crucial in providing necessary detail during the course of the experiment to us. The tools are summarized below.

Ground Tool Modules

ExecView provided an insight into the actual execution of the plans that were generated onboard by showing the status of the spacecraft timelines that the Smart Executive was tracking.

Stanley gave an in-depth view of the Livingstone module by expanding the condensed data in these messages in concert with a ground-based twin

PSGraph is a simple tool to visualize the state space of the onboard planner’s search by displaying along an XY axis the set of nodes visited against the depth of the node in the planners depth first search tree.

Tlm2email was an outreach tool and successfully used in providing the general public with a view of what the spacecraft was doing during the experiment.

PredictedEvents would parse a downlinked plan generated on board and provide RAX and project team members with predictions on when specific events were to occur for purposes of tracking.

FLIGHT EXPERIENCE

This section describes the experience of executing the flight demonstration[8]. Some problems surfaced along the way, but it was still possible to achieve all technology validation objectives. Indeed, the unexpected problems allowed some additional level of validation of our capabilities.

RAX Flight Part 1

On Monday, May 17th, 1999, at 11:04 am PDT, we received a telemetry packet that confirmed that RAX scenario had started on DS1. Shortly thereafter, the first plan was generated correctly, but not before an unexpected circumstance created some apprehension in us.

Telemetry indicated that the planner was generating the plan following a different search trajectory than what we had observed in ground testing. Since the conditions on the spacecraft were practically identical to those on the ground testbeds, there was no apparent reason of for this discrepancy. It turned out that the spacecraft and papabed differed on the contents of the file containing asteroid goals and PS was actually solving a slightly different problem than it had solved on the ground! Thus, this unexpected circumstance allowed us to demonstrate that PS problem solving was robust to last minute changes in the planning goals, increasing the credibility of the autonomy demonstration.

The 2 day scenario continued smoothly and uneventfully with a simulated switch failure, the resulting replan, long turns to point the camera at target asteroids, optical navigation imaging during which no communication with DS1 was possible, and the start of IPS thrusting.

However, around 7:00 am on Tuesday, May 18, 1999, it became apparent that RAX had not commanded termination of IPS thrusting as expected. Although plan execution appeared to be blocked, telemetry indicated that RAX was otherwise healthy. The spacecraft too was healthy and in no apparent danger. The decision was made to use EXEC's ability to handle low-level commands to obtain more information regarding the problem. Once enough information had been gathered, the decision was made to stop the experiment. By this time an estimated 70% of the RAX validation objectives had already been achieved.

Troubleshooting and Recovery

By late Tuesday afternoon the cause of the problem was identified as a missing critical section in the plan execution code. This created a race condition between two EXEC threads. If the wrong thread won this race, a deadlock condition would occur in which each thread was waiting for an event from the other. This is exactly what happened in flight, though it had not occurred even once in thousands of previous races on the various ground platforms. The occurrence of this problem at the worst possible time provides strong impetus for research on formal verification of flight critical systems. Once the problem was identified, a patch was quickly generated for possible uplink.

Following the discovery of the problem, we generated a 6-hour RAX scenario to demonstrate the remaining 30% of the RAX validation objectives. This new scenario was designed, implemented, and tested, together with the patch, on papabed overnight within about 10 hours. This rapid turn around allowed us to propose a new experiment at the DS1 project meeting on Wednesday. The DS1 project decided to proceed with the new scenario. However, they decided not to uplink the patch, citing insufficient testing to build adequate confidence. In addition, based on the experience on various ground testbeds, the likelihood of the problem recurring during the 6 hour test was deemed to be very low. Nonetheless, we developed and tested a contingency procedure that would enable us to achieve most of our validation objectives even if the problem were to recur.

The DS1 project's decision not to uplink the patch is not surprising. What was remarkable was their ready acceptance of the new RAX scenario. This is yet more evidence that the DS1 project had developed a high level of confidence in RA and its ability to run new mission scenarios in response to changed circumstances. Hence, although caused by an unfortunate circumstance, this rapid mission redesign provided unexpected validation for RA.

RAX Flight Part 2

The 6-hour scenario was activated Friday morning. The scenario ran well until it was time to start up the IPS. Unfortunately, an unexpected problem in some supporting software failed to confirm an IPS state transition, thus causing RA to (correctly) stop commanding the IPS startup sequence. The underlying cause of this problem was still under investigation as of July 20, 1999. Since this situation was out of scope for RAX, the resulting RA state was inconsistent with spacecraft state. Fortunately, the discrepancy proved to be benign. Hence, RA was able to continue executing the rest of the scenario to achieve the rest of its validation objectives.

By executing the two flight scenarios, RAX achieved 100% of its validation objectives.

LESSONS LEARNED

The RA team learned valuable lessons in a number of areas including RA technology and processes, tools, and even autonomy benefits to missions. The following sections delve into each of these categories of lesson.

Remote Agent Technology

Team Structure for RA Model Development:

The RAX team was structured horizontally along engine boundaries. See Table 2. This meant that team members specialized in one of the PS, EXEC, and MIR engines, and each team was responsible for modeling all spacecraft subsystems for their engine. This horizontal organization was appropriate for RAX, since it was our first major experience in modeling spacecraft subsystems for flight. Hence, it made sense for engine experts to do all modeling for their engine.

Table 2: Horizontal vs. Vertical Team Structure

	Attitude Control	Power	Camera	Ion Engine	Etc.
PS	•	•	•	•	•
EXEC	•	•	•	•	•
MIR	•	•	•	•	•

However, this organization has several shortcomings. Perhaps the most significant shortcoming was that knowledge of any one spacecraft subsystem (e.g., attitude control, ion propulsion, MICAS camera) was distributed across the three teams; one needed discussions with three individuals to get a complete understanding of how a subsystem was commanded by RA.

Conclusions: These shortcomings suggest an alternate structuring for a future SW team. Instead of a horizontal structure, teams might be organized vertically along spacecraft subsystem or domain unit boundaries; e.g., a single team would be responsible for developing all models for the attitude control system. This would ensure internal coherence of the resulting model. Furthermore, since modelers would need to understand how to use all three engines, they can make well motivated decisions on how best to model a subsystem to exploit the strengths of each engine and avoid information duplication.

While a vertical team organization has its benefits, certain aspects of model development intrinsically involve managing and reasoning about global constraints, e.g., power allocation strategies, system-level fault protection. Hence, it is important to involve systems engineers to develop these global strategies.

Model Design, Development and Test:

One of the biggest challenges we faced was model validation. This was particularly true during validation testing, when even small changes in the models had to be carefully and laboriously analyzed and tested to ensure that there were no unexpected problems. In fact, in some cases we chose to forgo a model change, and instead decided to institute flight rules that would preclude the situation that required the model change from arising. A related issue was that methods do not yet exist to characterize RA's expected behavior in novel situations. This made it difficult to precisely specify the boundaries within which RAX was guaranteed to act correctly. While the declarative nature of RA models was certainly very helpful in ensuring the correctness of models and model changes, the difficulty stemmed from unexpected interactions between different parts of the model, e.g., different parts of the model may have been built under different, implicit, conflicting assumptions.

Conclusion: The central lesson we learned here was the need for better model validation tools. For example, the automated test running capability we developed proved to be enormously helpful, as it allowed us to quickly evaluate a large number of off-nominal scenarios. However, scenario generation and evaluation of test results were time consuming. In some cases, the laborious process we followed to validate model changes has provided us with concrete ideas for developing tools that would dramatically simplify certain aspects of model validation. Preliminary work in the area of formal methods for model validation is also very promising. Finally, we need to develop better methods for characterizing RA's behavior with a specific set of models, both as a way of validating those models and as a way of explaining the models to a flight team.

Robustness of the Basic System:

Model validation alone does not suffice; the rest of the system, including the underlying inference engines, the interfaces between the engines, and the ground tools, must all be robust. Given our resource constraints, we made the decision to focus our formal testing on model validation, with engine and interface testing happening as a side effect. This was a reasonable strategy: code that has been unchanged for years is likely to be very robust if it has been used with a variety of different models and scenarios. However, newer code does not come with the same quality assurance. Furthermore, as the deadlock bug in flight showed, subtle timing bugs can lay hidden for years before manifesting themselves.

Conclusion: The primary lesson is that the basic system must be thoroughly validated with a comprehensive test plan as well as formal methods, where appropriate, *prior* to model development and flight insertion. Interfaces between systems must be clean and well specified, with automatic code generation being used to generate actual interface code, telemetry, model interfaces, and test cases; code generation proved to be enormously helpful in those cases where we did use it.

On-Board Planning

Since the beginning of RA, on-board planning has been the autonomy technology that most challenges the comfort level of mission operators. Commanding a spacecraft with high-level goals and letting it autonomously take detailed actions is very far from the traditional commanding approach with fixed-time sequences of low-level commands. We believe that during RAX the flawless demonstration of on-board planning has provided a powerful existence proof of the feasibility of the approach. Our own discomfort with the discrepancy between tested behavior and in-flight behavior of PS during RAX was a surprising mirror of the objections of the critics of autonomy.

Conclusion: It is difficult to move past the mindset of expecting complete predictability from the behavior of an autonomous system. However, RAX has demonstrated that the paradigm shift is indeed possible. In the case of PS behavior during RAX, the point is not that the combination of pictures requested by NAV had never been experienced before, but that the problem-solving behavior that the planner used to achieve each individual picture goal had indeed been tested. Confidence in complex autonomous behaviors can be built up from confidence in each individual component behavior.

Design for Testability

System-level testing is an essential step in flight preparation. Designing the RA to simplify and streamline system-level testing and analysis can enable more extensive testing, thus improving robustness. In RAX, system-level testing proved to be cumbersome. The primary reason for this was the absence of efficient tools to generate new mission scenarios, so that all system tests had to be variations on the nominal scenarios. Hence, to test a particular variation, one was forced to run a nominal scenario up to the point of the variation, e.g., testing thruster failures during turns required at least 6 hours, since the first turn occurred about 6 hours into the scenario. (Check this example.)

Conclusions: The difficulty of generating new mission scenarios is easily addressed: a graphical tool allowing visual inspection and modification of mission profiles, as well as constraint checking to ensure consistency, can dramatically simplify the construction of new mission profiles. Such a tool is now being constructed. Nonetheless, overall RA validation is still necessary to ensure that RA will properly handle each new mission profile (see below).

Remote Agent Processes

A short development period, scarce resources, and ambiguity arising from the research issues inherent in the technology characterized the Remote Agent development. We learned several lessons in managing software development tasks of this nature that the software industry has been re-learning for years.

Priority of Objectives

Without a clear list of prioritized objectives, we had no way to regain schedule margin by dropping lower priority objectives, or to track our progress against those objectives.

Conclusion: establish prioritized objectives from the start and be clear about what you are willing to give up.

Early Use of Target Platform

A number of problems manifested each time we moved to a new target platform. Since we moved to these platforms very late, the problems were critical and we were limited to low-impact work-arounds. Many of these problems could have been detected by early end-to-end tests, even with a limited functionality system. Early knowledge of these issues may also have led to design changes that would have addressed the issues more robustly or elegantly than the eventual work-arounds.

Conclusion: Use as few platforms as possible, and run end-to-end tests on the highest-fidelity platforms as early as possible, even with a limited functionality product.

Remote Agent Tools

Systems Engineering Tools

Coding the domain models required substantial knowledge acquisition, which is a common bottleneck in Artificial Intelligence systems. It is better to have the domain expert code the models directly.

Conclusion: Develop tools and simplify the modeling languages to enable spacecraft experts to encode models themselves. Employ tools and languages already familiar to the experts. Organize the models around the domain (Attitude Control, Power, etc.) rather than around the RA technology (planner, exec, MIR).

Mission Profile Development

The RA is commanded by goals specified in a mission profile. For the experiment, constructing the profile was a "black art" that only one or two people on the RA team could perform. The mission planners and operations personnel must be able to specify goals themselves.

Conclusion: Simplify specification of goals. When possible, use approaches already familiar to mission planner, such as graphical timeline displays and time-ordered listings. Provide automated consistency checking.

Adaptability to Late Model Changes

The spacecraft requirements and operating procedures change throughout development, and even after launch. We were unable to encode late changes, due to the regression-testing overhead that each change required.

Conclusions: The validation cost of model-changes must be reduced. Some possibilities include tools to evaluate the consequences of model changes on testing. The models already support localized changes. Procedures are needed to uplink and install just those changes.

Ground Tools

Ground tools ought to be developed well in advance of the actual flight and be used as a primary means to test and understand how to operate complex systems. Given the late date of development of most of the ground tools, a good many of them felt not well integrated. As a result only the tools displaying or interpreting data in the most obvious way were of high value.

Telemetry

In addition to an on-board textual log file downlinked at the end of the experiment or on request, RAX sent a stream of binary telemetry packets, one for each significant event, that were displayed as color-coded text on the ground. Among other things, the telemetry allowed us to monitor all on-board communication among RAX modules and between RAX and the flight software. This proved valuable in allowing us to quickly diagnose the anomalies that occurred. We immediately knew that the reason RAX had failed to turn off the ion engine was that it had stopped executing the plan in some unanticipated manner; we knew RAX was still running and could also rule out a plan abort or a failure to send just one command. Similarly, we immediately narrowed down the second anomaly to a monitor message that was either not sent or not received.

Conclusion: Ensuring sufficient visibility on all platforms, including in flight, requires adequate information in telemetry. The best way to ensure this is to design the telemetry early, and to use it as the primary, if not the only, way of debugging and understanding the behavior of the system during integration, test, and operations.

Mission Benefits

Next we examine the future benefits of RA based on the lessons gained on DS1.

Customer Engagement

Until September 1998 the RAX design team was not able to engage the DS1 team in more than a limited manner. After that date, the engagement was focussed on issues related to RAX deployment and to demonstration of RAX safety with respect to spacecraft health. The DS1 team had only minimal exposure on the functionality of the RAX technology. As a consequence the DS1 flight team did not develop an understanding of how RAX would operate in various operational circumstances nor of what benefits it could provide for future missions. The RAX team consciously chose the strategy of "minimal interaction" from the beginning. Indeed, this was the only realistic option in a situation where the DS1 team was under extreme pressure to deliver baseline DS1 software and therefore could afford little distraction from their primary goal. Moreover, it was thought that the demonstration of AI technology taking control of the spacecraft itself would stimulate interest in learning more about the inner workings of the technology. This assumption proven to be wrong due to the heavy workload of the DS1 team.

RA technology therefore missed the opportunity to enlist the DS1 team as one of its knowledgeable supporters. After RAX, we found that the DS1 team still has a number of basic questions regarding RA, some no different now than they were at the beginning of the mission. In conclusion, although supportive of RAX, most members of the DS1 team have not become knowledgeable advocates for the RA technology.

Conclusions: future similar efforts should pay attention to educate the customer in the technology so that the customer can become a user and supporter of the technology—and give valuable feedback on ways to improve it. Understanding and support from the operational level (i.e., the people actually developing and deploying software) is important since it is at this level that the impact and usefulness of the technology is directly experienced. Support at higher levels of the mission management hierarchy is also needed especially early in the process, in order to clarify costs and trade-off of the technology with respect to the overall goal of the mission.

Remote Agent vs. Autonomy

There has always been a tight identification of the RA technology with “autonomy”. For example, one of the most highlighted aspects of RA is its ability to support “fail operational.” However, one could argue that the largest potential benefit of RA is in directly supporting system and mission engineering functions with formalized languages and models that can directly translate into operational, application code. This is a “process benefit” and is quite independent of whether or not the resulting flight software functionality is highly autonomous or more traditional. It will also be important to identify the value of using different RA components to solve localized problems faced by missions in an economically viable way. In fact, the decision of whether to adopt RA technology across the board, only for low-level control or only for the high-level observation management is a decision that depends on the specific cost/benefit tradeoff for the mission.

Conclusions: we need to more clearly articulate the value of this technology even if it is not used to achieve higher levels of autonomy. RA technologies can be used individually or together depending on the mission needs.

Robustness of Model Based Design

As mission development times becomes shorter and mission objectives become more ambitious, it is less and less likely that an accurate model of each spacecraft component will be available early in the flight and ground software development cycle. Dealing with this uncertainty is a major problem facing future missions. By emphasizing qualitative and high-level models of behavior RA can help solve this dilemma. Qualitative, high-level models can be captured early in the mission lifetime and should need only minor adjustments when the hardware is better understood. Our experience on RAX essentially confirms this hypothesis. Initial spacecraft models used by PS, EXEC and MIR were built early in the DS1 mission, before April 1997. During the following year and a half, EXEC and MIR models did not change and the PS model was only changed in order to support more efficient problem solving by the search engine, not in order to reflect new knowledge of the spacecraft behavior. In the last phase of the experiment preparation, when communications between the RAX team and the DS1 team resumed, adjustments were needed to finalize the interface between the low-level EXEC primitives and flight software.

Conclusion: Contrary to much concern, the type of qualitative, high-level models used by the RA requires little tuning throughout the project. The usefulness of the models for software development has been validated.

Uncertainty Handling

A precise characterization of how RA deals with uncertainty, however, requires additional work. At the start of RAX development we believed that RA was particularly suited to operate robustly under conditions that had not been explicitly tested. However, during development we discovered that, particularly in the case of PS, new scenarios sometimes highlighted flaws in the problem solving strategy. What usually happened is that, although in principle PS could solve the new problem (i.e., PS search was complete), in practice it could not always do so within the performance requirements of RAX, such as timing and memory, and therefore needed to be modified.

Based on our experience, we believe that robustness to uncertainty could be better characterized as the ability to operate according to specifications within the boundaries of validated capabilities. It is important that the boundaries of the validated capabilities be established in advance through formal validation or extensive testing. We believe that characterizing these boundaries without full path testing (which is impossible for typical applications of RA) is an important research and development goal for RA technology.

It is possible that sometimes RA will have to operate in conditions that have not been fully validated. In this case it will be important for the overall fault protection system to appropriately monitor the behavior of RA while guaranteeing the safety of the controlled system. When using RA in the context of a mission, system engineering, fault protection and mission engineering will have to identify which autonomous behavior should be guaranteed and which may not. This selection will have to be guided by the goals of a mission, the level of threat to system health posed by the environment, and the acceptable levels of system performance in different phases (e.g., launch and check out, cruise, encounter, landing).

Conclusion: Remote Agent's formal declarative models can facilitate the identification of classes of validated conditions under which RA performance is guaranteed. Outside the validated boundaries RA may sometimes fail to operate within performance limits and we should rely on good fault protection design to guarantee a safe continuation of the mission.

FUTURE WORK

Future work regarding Remote Agent can be divided into three categories: fundamental improvements in the capabilities of its components, improvements in usability or deployability, and upcoming demonstrations or applications.

A number of basic research areas are being pursued to improve future iterations of Remote Agent. Contingent planning and diagnostic ambiguity are just two examples. Contingent planning enables a planner to create a plan with branches that may be taken if any of a range of likely events occur, reducing the need to replan. Improved handling of uncertainty will allow Livingstone to better track multiple ambiguous trajectories the system may be following and recommend actions that are safe and goal-directed even though the system is in one of several possible states.

Applying Remote Agent to the Deep Space 1 spacecraft provided a wealth of practical lessons about what was needed to create a sustainable autonomy engineering process and make this technology usable for main-line mission development and operations. The Remote Agent team is now developing tools for graphically creating and debugging models, for automating much of the integration of Remote Agent with traditional flight software, and for allowing humans and autonomous software to interact more easily. The team is collaborating with software verification researchers at NASA Ames Research Center and Carnegie-Mellon University to allow certain Remote Agent models to be analyzed to prove they cannot recommend undesired behavior. The components are also in the process of being ported from Lisp to C++ to fit more seamlessly fit into flight environments, with the planner and Livingstone having achieved significant milestones in their ports. In short these research and development efforts are designed to make the Remote Agent and similar technologies more capable, easier to use, and easier to test and validate.

Remote Agent technology is successfully being transferred beyond the original team and several groups are currently building prototypes with Remote Agent in order to evaluate it. At NASA's Kennedy Space Center, Remote Agent applications are being developed to evaluate RA for missions involving in-situ propellant production on Mars on the 2003 lander or a future piloted mission. Applications for shuttle operations are being pursued as well. At the Jet Propulsion Laboratory, Remote Agent is being evaluated as the baseline autonomy architecture for the Origins Program Interferometry Instruments and is being used in the JPL interferometry testbed. The New Millennium Program's Deep Space Three, a space-based interferometry mission which includes two or three spacecraft cooperating to take science observations, may be one early customer of this development. At Johnson Space Center, components of Remote Agent are being integrated into an ecological life support testbed for human missions beyond Earth orbit. At Ames Research Center, Remote Agent technology is being incorporated into software for more robustly controlling planetary rovers. Working with Orbital Sciences Corporation, Ames is working to demonstrate Remote Agent as it applies to streamlining the checkout and operation of a reusable launch vehicle. This demonstration will fly on the X-34 vehicle.

SUMMARY

With the successful conclusion of the Remote Agent Experiment, we've taken a major step on the path to highly autonomous spacecraft. In addition to the end product, Remote Agent, the process of developing, deploying and demonstrating such a system on Deep Space One has also had an invaluable technological impact. For technology providers, it has provided valuable lessons about the critical issues for injecting autonomy technology into the NASA enterprises. For the NASA enterprises, it provided additional exposure to the concept of model-based and goal-directed operations for future NASA missions. The lessons learned will allow us to improve the technology, our processes, and perhaps most importantly, our approach to technology infusion. In short, the hope is that the experience and lessons learned during the Remote Agent Experiment will serve as the thin end of the wedge as far as the maturation and deployment of autonomy technology in aerospace applications is concerned.

ACKNOWLEDGEMENTS

This paper describes work performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration, and by the NASA Ames Research Center. This work would not have been possible without the efforts of the rest of the DS1 team.

REFERENCES

[1] Douglas E. Bernard et al., "Design of the remote agent experiment for spacecraft autonomy." In Proceedings of the IEEE Aerospace Conference, 1998.
[2] Nicola Muscettola, P. Pandurang Nayak, Brian C. Williams, and Barney Pell, "Remote Agent: To boldly go where no AI system has gone before." *Artificial Intelligence*, 103:5–47, 1998.
[3] 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*, 5(1), March 1998.
[4] Nicola Muscettola, "HSTS: Integrating planning and scheduling." In Mark Fox and Monte Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.
[5] Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith, "Robust periodic planning and execution for autonomous spacecraft." In Proceedings of IJCAI-97, 1997.
[6] Brian C. Williams and P. Pandurang Nayak, "A model-based approach to reactive self-configuring systems." In Proceedings of AAAI-96, pages 971–978, 1996.
[7] Benjamin Smith, William Millar, Julia Dunphy, Yu-Wen Tung, P. Pandurang Nayak, Edward B. Gamble Jr., and Micah Clark, "Validation and verification of the remote agent for spacecraft autonomy." In Proceedings of the 1999 IEEE Aerospace Conference, 1999.
[8] P. Pandurang Nayak, et al., "Validating the DS1 Remote Agent Experiment", Proceedings of the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS-99), 1999.