Validating the Autonomous Science Agent

Steve Chien, Benjamin Cichy, Steve Schaffer, Danny Tran, Gregg Rabideau, Rob Sherwood

Jet Propulsion Laboratory, California Institute of Technology

Firstname.Lastname@jpl.nasa.gov

Robert Bote, Dan Mandl, Stu Frye, Seth Shulman

Goddard Space Flight Center

Firstname.Lastname@gsfc.nasa.gov

Jim Van Gaasbeck, Darrell Boyer

Interface and Control Systems

{jimv, dboyer}@interfacecontrol.com

ABSTRACT

This paper describes the validation process for the Autonomous Science Agent, a software agent that will fly onboard the EO-1 spacecraft from 2003-2004. This agent will recognize science events, retarget the spacecraft to respond to the science events, and reduce data downlink to only the highest value science data. The autonomous science agent has been designed using a layered architectural approach with specific redundant safeguards to reduce the risk of an agent malfunction to the EO-1 spacecraft. This "safe" design is also in the process of being thoroughly validated by informal validation methods and extensive testing. This paper describes the analysis used to define agent safety, elements of the design that increase the safety of the agent, and the process being used to validate agent safety prior to the agent software controlling the spacecraft.

Keywords

Agent Safety, Autonomous Science, Automated Planning, Robust Execution, Agent Architectures

1. INTRODUCTION

Autonomy technologies have incredible potential to revolutionize space exploration. In the current mode of operations, space missions involve meticulous ground planning significantly in advance of actual operations. In this paradigm, rapid responses to dynamic science events can require substantial operations effort. Artificial Intelligence technologies enable onboard software to detect science events, replan upcoming mission operations, and enable successful execution of re-planned responses. Additionally, with onboard response, the spacecraft can acquire data, analyze it onboard to estimate its science value, and only downlink the highest priority data. For example, a spacecraft could monitor active volcano sites and only downlink images when the volcano is erupting. Or a spacecraft could monitor ice shelves and downlink images when calving activities are high. Or a spacecraft could monitor river lowlands, and downlink images when flooding occurs. This onboard data selection can vastly improve the science return of the mission by improving the efficiency of the limited downlink. Thus, there is significant motivation for onboard autonomy.

However, building autonomy software for space missions has a number of key challenges; many of these issues increase the importance of building a reliable, safe, agent.

1. Limited, intermittent communications to the agent. A typical spacecraft in low earth orbit typically has 8 communications opportunities per day. This means that the spacecraft must be able to operate for long periods of time without supervision. For deep space missions the spacecraft may be in communications far less frequently. Some deep space missions only contact the spacecraft once per week, or even once every several weeks.
2. Spacecraft are very complex. A typical spacecraft has thousands of components, each of which must be carefully engineered to survive rigors of space (extreme temperature, radiation, physical stresses). Add to this the fact that many components are one-of-a-kind and thus have behaviors that are hard to characterize.
3. Limited observability. Because processing telemetry is expensive, onboard storage is limited, and downlink bandwidth is limited, engineering telemetry is limited. Thus onboard software must be able to make decisions on limited information.
4. Limited computing power. Because of limited power onboard, spacecraft computing resources are usually very constrained. An average spacecraft CPUs offer 25 MIPS and 128 MB RAM – far less than a typical personal computer.

5. High stakes. A typical space mission costs hundreds of millions of dollars, any failure has significant economic impact. Over financial cost, many launch and/or mission opportunities are limited by planetary geometries. In these cases, if a space mission is lost it may be years before another similar mission can be launched. Additionally, a space mission can take years to plan, construct the spacecraft, and reach their targets. This delay can be catastrophic.

This paper discusses our efforts to build and validate a safe autonomous space science agent. The principal contributions of this paper are as follows:

1. We describe our layered agent architecture and how that enables additional agent safety.
2. We describe our knowledge engineering and model review process designed to enforce agent safety.
3. We describe our ground and on-orbit testing process designed to improve agent safety.
4. We describe our incremental flight validation process to enhance agent safety.

We describe these areas in the context of the Autonomous Sciencecraft Experiment (ASE), an autonomy software package originally designed for flight on the Air Force's Techsat-21 Mission [2] in 2006 and now being uploaded for flight on NASA's New Millennium Earth Observer One (EO-1) spacecraft in the summer of 2003 [4].

In this paper we address a number of issues from the workshop call.

Definition of agent safety and How to build safe agent – we define agent safety as ensuring the health and continued operation of the spacecraft. We design our agent to have redundant means to enforce all known spacecraft operations constraints. We also utilize declarative knowledge representations, whose models are extensively reviewed and tested. We use code generation technologies to automatically generate redundant checks to improve software reliability. Additionally, our experiment is also designed to fly in a series of increasing autonomous phases, to enable characterization of performance of the agent and to build confidence.

Robust to environment (unexpected) – our agent must be robust to unexpected environmental changes. Our agent uses a classic layered architecture approach to dealing with execution uncertainties.

How to constrain agents – because of strong agent safety concerns, our agent architecture is designed to enable redundancy, adjustable autonomy, and fail-safe disabling of agent capabilities. The layering of the agent enables lower levels of the agent to inhibit higher-level agent behavior. For example, the task executive systems (SCL) does not allow dangerous commands from the planner to be sent on to the flight software. The Flight Software bridge (FSW-bridge) can be instructed to disable any commands form the autonomy SW or to shutdown components of or the entire autonomy SW. The EO-1 Flight Software includes a fault protection function designed to inhibit potentially hazardous commands form any source (including the autonomy software, stored command loads from the ground, or real-time commands).

Validation of agent safety – we describe in some detail how we have or have not applied formal methods, informal methods, and testing to validate the performance of the ASE software.

Application area – we briefly describe some of the more salient features of the spacecraft control application area for autonomous agents.

The remainder of this paper is organized as follows. First we describe the ASE software architecture, with an emphasis on how it enhances safe agent construction. Next we discuss the modeling process and how we developed and validated models. Then we describe the testing process, including how we have designed test cases and the test infrastructure. Finally, we describe the incremental flight experiment setup and how that enables safer experiment operations.

2. AUTONOMY ARCHITECTURE

The autonomy software on EO-1 is organized as a traditional three-layer architecture [8] (See Figure 1.). At the top layer, the Continuous Activity Scheduling Planning Execution and Replanning (CASPER) system [3, 12] is responsible for mission planning functions. Operating on the tens-of-minutes timescale, CASPER responds to events that have widespread (across orbits) effects, scheduling science activities that respect spacecraft operations and resource constraints. Activities in a CASPER schedule become inputs to the Spacecraft Command Language (SCL) system [10].

CASPER models activities performed by the spacecraft and ground equipment and staff, and tracks activity effects on its model of spacecraft state and resources. CASPER then searches for plans that combine these basic activities to satisfy goals (such as downlinks and observation requests) while enforcing operations constraints.

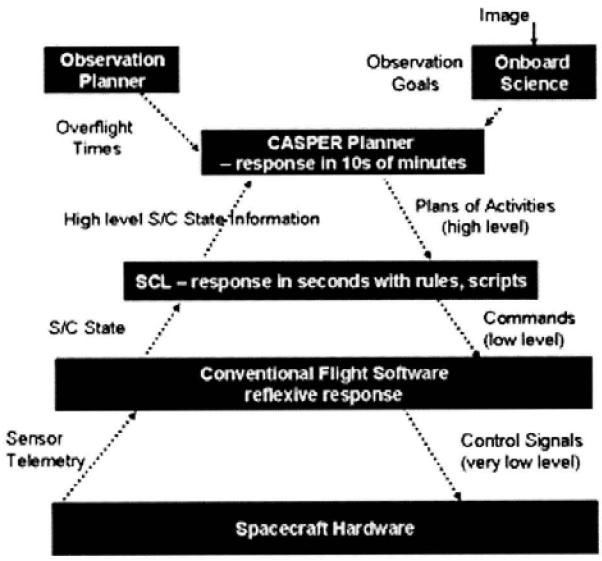

Figure 1. Autonomy Software Architecture

At the middle layer, SCL is responsible for generating and executing detailed sequence of commands that correspond to expansions of CASPER activities. SCL also implements spacecraft constraints and flight rules. Operating on the several-second timescale, SCL responds to events that have local effects, but require immediate attention and a quick resolution. SCL performs activities using scripts and rules. The scripts link together lower level commands and routines and the rules enforce additional flight constraints.

SCL sends commands to the EO-1 flight software system (FSS) [9], the basic flight software that operates the EO-1 spacecraft. The interface from SCL to the EO-1 FSS is at the same level as ground generated command sequences. This interface is implemented by the Autonomy Software Bridge (FSB), which takes certain autonomy software messages and issues the corresponding FSS commands. The FSB also implements a set of FSS commands that it responds to that perform functions such as startup of the autonomy SW, shutdown of the autonomy SW, switching from shadow to active mode, and other autonomy SW configuration actions.

The FSS accepts low level spacecraft commands. These commands can be either stored command loads uploaded from the ground (e.g. ground planned sequences) or real-time commands (such as commands from the ground during an uplink pass). The autonomy SW commands appear to the FSS as real-time commands. As part of its core, the FSS has a full fault and spacecraft protection functionality which is designed to:

1. Reject commands (from any source) that would endanger the spacecraft.
2. When in situations that threaten spacecraft health, execute pre-determined sequences to "safe" the spacecraft and stabilize it for ground assessment and reconfiguration.

For example, if a sequence issues commands that point the spacecraft imaging instruments at the sun, the fault protection software will abort the pointing activity. Similarly, if a sequence issues commands that would expend power to unsafe levels, the fault protection software will shut down non-essential subsystems (such as science instruments) and orient the spacecraft to maximize solar power generation. While the intention of the fault protection is to cover all potentially hazardous scenarios, it is understood that the fault protection software is not foolproof, Thus, there is a strong desire to not command the spacecraft into any hazardous situation even if it is believed that the fault protection will protect the spacecraft.

The science analysis software is scheduled by CASPER and executed by SCL. The results from the science analysis software generate new observation requests presented to the CASPER system for integration in the mission plan.

This layered architecture for the autonomy SW is designed such that each lower layer is validating the output of the higher layers. The planner activities are checked by SCL prior to being sent on to the FSS. The FSS fault protection is checking the SCL outputs as well.

3. MODEL BUILDING & VALIDATION

Because the control aspects of the Autonomy SW are embodied in the CASPER & SCL models, our methodology for developing and validating the CASPER and SCL models is critical to our safe agent construction process. These models include constraints of the physical subsystems including: their modes of operation, the commands used to control them, the requirements of each mode and command, and the effects of commands. At higher levels of abstraction, CASPER models spacecraft activities such as science data collects and downlinks, which may correspond to a large number of commands. These activities can be decomposed into more detailed activities until a suitable level is reached for planning. CASPER also models spacecraft state and its progression over time. This includes discrete states such as instrument modes as well as resources such as memory available for data storage. CASPER uses its model to generate and repair schedules, tracking both the current state & resources and the expected evolution of spacecraft state and resources based on planned activities.

SCL continues to model spacecraft activities at finer levels of detail. These activities are modeled as SCL scripts, which when executed, may execute additional scripts, ultimately resulting in commands to the EO-1 FSS. Spacecraft state is modeled as a database of records in SCL, where each record stores the current value of a sensor, resource, or sub-system mode. The SCL model also includes flight rules that monitor spacecraft state, and execute appropriate scripts in response to changes in state. SCL uses its model to generate and execute sequences that are valid and safe in the current context. While SCL has a detailed model of current spacecraft state and resources, it does not generally model future planned spacecraft state and resources.

Development and verification of the EO-1 CASPER and SCL models was a multiple step process.

1. First a target set of activities was identified. This was driven by a review of existing documents and reports. This allowed the modeler to get a high-level overview of the EO-1 spacecraft, including its physical components and mission objectives. Because EO-1 is currently in operation, mission reports were available from past science requests. These reports were helpful in identifying the activities performed when collecting and downlinking science data. For example, calibrations are performed before and after each image, and science requests typically include data collection from both the Hyperion (hyperspectral) and Advanced Land Imager (ALI) instruments.
2. Once the activities were defined, a formal EO-1 operations document was reviewed to identify the constraints on the activities. For example, due to thermal constraints, the Hyperion cannot be left on longer than 19 minutes, and the ALI no longer than 60 minutes. The EO-1 operations team also provided spreadsheets that specified timing constraints between activities. Downlink activities, for example, are often specified with start times relative to two events: acquisition of signal (AOS) and loss of signal (LOS). Fault protection documents listing fault monitors (TSMs) were also consulted, using the reasoning that acceptable operations should not trigger TSMs.

3. With the model defined, CASPER was able to generate preliminary command sequences from past science requests that were representative of flight requests. These sequences were compared with the actual sequences that were uplinked for the same request. Significant differences between the two sequences identified potential problems with the model. For example, if two commands were sequenced in a different order, this may reveal an overlooked constraint on one or both of the commands. We were also provided with the actual downlinked telemetry that resulted from the execution of the science observation request. This telemetry is not only visually compared to the telemetry generated by ASE, but it can also be "played back" to the ASE software to simulate the effects of executing sequences. The command sequences were aligned with the telemetry to identify the changes in spacecraft state and the exact timing of these changes. Again, any differences between the actual telemetry and the ASE telemetry revealed potential errors in the model. A consistent model was defined after several iterations of generating commands and telemetry, comparing with actual commands and telemetry, and fixing errors. These comparisons against ground generated sequences were reviewed by personnel from several different areas of the operations staff to ensure acceptability (e.g. overall operations, guidance, navigation and control, science operations, instrument operations).

4. Model reviews were conducted (or are planned) where the models are tabletop reviewed by a team of personnel with a range of operations and spacecraft background. This is to ensure that no incorrect parameters or assumptions are represented in the model.

Finally, a spacecraft safety review process was performed. In this process, experts from each of the spacecraft subsystem areas (e.g. guidance, navigation and control, solid state recorder, Hyperion instrument, power, ...) studied the description of the ASE software and commands that the ASE SW would execute and derived a list of potential hazards to spacecraft health. For each of these hazards, a set of possible safeguards was conjectured: implemented by operations procedure, implemented in CASPER, implemented in SCL, and implemented in the FSS. Every safeguard able to be implemented with reasonable effort was implemented and scheduled for testing. Such analysis for two risks is shown below.

Table 1. Sample safety analysis for two risks.

	Instruments overheat from being left on too long	**Instruments exposed to sun**
Operations	For each turn on command, look for the following turn off command. Verify that they are within the maximum separation.	Verify orientation of spacecraft during periods when instrument covers are open.
CASPER	High-level activity decomposes into turn on and turn off activities that are with the maximum separation.	Maneuvers must be planned at times when the covers are closed (otherwise, instruments are pointing at the earth)
SCL	Rules monitor the "on" time and issue a turn off command if left on too long.	Constraints prevent maneuver scripts from executing if covers are open.
FSS	Fault protection software will shut down the instrument if left on too long.	Fault protection will safe the spacecraft if covers are open and pointing near the sun.

An interesting aspect of model development is the use of code generation techniques to derive SCL constraint checks from CASPER model constraints. In this approach, certain types of CASPER modeling constraints can be translated into SCL code to ensure activity validity at execution time. If the CASPER model specifies that activities use resources, this can be translated into an SCL check for resource availability before the activity is executed. If the CASPER model specifies a state requirement for an activity, one can auto-generate a check to see if that state is satisfied before executing the activity. Additionally, if the CASPER model specifies sequential execution of a set of activities, code can be generated so that SCL enforces this sequential execution.

For example, in calibrating the Hyperion instrument, the solid state recorder (WARP) must be in record mode and the Hyperion instrument cover must be "open". Below we show the CASPER model and the generated SCL constraint checks.

```
// Hyperion calibration
activity hsi_img_cal
{
  durat caldur;
  // schedule only when the WARP is in record
  // mode, recording data, and
  // when the hyperion cover is open
  reservations =
    wrmwmode must_be "rec",
    ycovrstat must_be "closed";
  // start and stop the instrument
  decompositions =
    yscistart, yscistop
    where yscistop starts_after
          start of yscistart by caldur;
}

-- Hyperion calibration
script hsi_img_cal caldur
  -- verify that the WARP is in record
  -- mode, recording data, and
  -- that the hyperion cover is open
  verify wrmwmode = rec
     and ycovrstat = closed
         within 5 seconds
  -- start and stop the instrument
  execute yscistart
  wait caldur sec
  execute yscistop
end hsi_img_cal
```

Figure 2. Sample model and script for Hyperion calibration.

Note that this generated code also enforces the sequential execution of the "yscistart" and "yscistop" activities, separated by "caldur" seconds. This shows how code is automatically generated from a CASPER defined temporal constraint over two activities.

As another example, when initiating the WARP recording, there is a limit on the total number of files on the WARP recorder (63). In CASPER we define the constraint that "wfl" new files are created. In SCL, code is auto-generated to verify that that many files can be created without exceeding the file number limit before the WARP recording activity is allowed to be executed.

```
// Start the WARP recording
activity wrmsrec
{
  ...
    reservations =
      // reserve the required number of
      // files on the WARP
      wrmtotfl use wfl,
      // change the warp to record mode when
      // complete
      wrmwmode change_to "rec" at_end,
  ...
}

-- Start the WARP recording
script wrmsrec
  ...
       verify
        wrmfreebl wrmtotfl + wfl <= 63
        and wrmtotfl + wfl >= 1 and
  ...
end wrmsrec
```

Figure 3. Sample model and script for WARP recording.

4. TESTING ENFORCEMENT OF SAFETY

Gaining confidence in the safety of the EO-1 autonomy software requires extensive testing. We structured our testing methodology such that it would verify the protections provided by our layered architecture, and compliment the phases of the model development process. Specifically, the test plan is intended to validate the following system properties:

1. CASPER generates plans consistent both with its internal model of the spacecraft and SCL's model and constraints (as checked by SCL).
2. SCL does not issue any commands that violate the constraints of the spacecraft (as checked by our spacecraft simulator).
3. Our model satisfies spacecraft operational and safety constraints enumerated by the model safety-review process.

We will validate these three requirements by extensive testing of the autonomy software on generated test-cases, using checks at each layer to validate performance. The test cases described below address only the top-two levels of the onboard autonomy software (CASPER and SCL), with the flight software and spacecraft hardware replaced by a software simulator. Flight software testing and validation is addressed by a separate, more conventional, test plan.

4.1 Test Case Parameters

Each EO-1 test case covers seven days of operations containing multiple schedulable windows separated by a variable number of orbits. Each schedulable window represents an opportunity to schedule one or more science observations. The test cases must account for variations in the mission and science objectives (mission scenario parameters), initial state of the spacecraft (spacecraft state parameters), and changes to the spacecraft state during execution.

Since the autonomy software has no control over what happens outside of a schedulable window, we must be certain that our software performs reliably over a range of possible initial states. We cover these cases by using the simulator to vary the spacecraft state parameters as tracked within SCL and monitored by CASPER. The simulator also varies the spacecraft state parameters during execution to test the performance of our agent in the face of an uncertain environment.

Mission scenario parameters represent the high-level planning goals passed to CASPER. They are derived from a combination of the orbit of the spacecraft and the science objectives uplinked from the ground. They specify when targets will be available for imaging, as well as the parameters of a science observation (i.e. number of targets to image and science analysis algorithms we wish to execute).

The 22 spacecraft state parameters and 16 observation goal parameters used in the EO-1 test cases are shown in the tables below.

Table 2. Spacecraft state parameters.

Parameter	Expected Initial State
xband groundstation	unknown
xband controller	enabled
ACS mode	nadir
target selected	unknown
warp electronics mode	stndops
warp mode	standby
warp bytes allocated	0
warp num files	0
fault protection	enabled
eclipse state	full sun
target view	unknown
hyperion instrument power	on
hyperion imaging mode	idle
hyperion cover state	closed
ali instrument power	on
ali active mechanism	telapercvr
ali mechanism power	disabled
ali fpe power	disabled
ale fpe data gate	disabled
ali cover state	closed
groundstation view	unknown
mission lock	unlocked

Table 3. Mission-scenario parameters.

Parameter	Nominal	Off-nominal	Extreme
schedulable windows	0-3	3-5	5+
orbits between windows	2-7	1,8	0,8+
window start time	start of orbit	+/- 10 min	any
window duration	expected time of science analysis	+/- 10 min	any
target in-view start	anytime in orbit, 1 per orbit	1 per 3 orbits	any
target in-view duration	10 min +/- 1	+/- 3	any
groundstation in-view start	anytime in orbit, 1 per orbit	1 per 3 orbits	any
groundstation in-view duration	10 min +/- 1	+/- 3	any
eclipse start	60 min after orbit start	+/- 5	any
eclipse duration	30 min	+/- 5	any
imaging start	target view + 5 min	+/- 3	any
imaging duration	8 sec	+/- 4	any
science algorithm	any	any	any
science goal start	fixed	not-specified	any
number of science goals	1 per orbit	1-3	>3
warp allocated	0	32K blocks	any

To exhaustively test every possible combination of state and observation parameters, even just assuming a nominal and failure case for each parameter and ignoring execution variations, would

require a test set containing 2^{38} or 2.7×10^{11} test cases. Pruning the set of variations to just the sixteen observation parameters would still yield an impractically large set. The challenge thus becomes selecting a set of tests that most effectively cover the intractable space of possible parameter variations within a timeframe that allows for reasonable software delivery.

4.2 Design of Test Cases

Traditional flight software can be tested through exhaustive execution of a known set of sequences. Autonomy software however must be able to execute in, and react to, a much wider range of possible scenarios. As show above, testing all these possible scenarios would be intractable, however we can leverage the traditional nominal sequences and scenarios to baseline our tests – varying parameters off of a controlled scenario, and thus reducing the number of parameter variations our agent must consider. This is a similar approach to that used to validate the Remote Agent Planner for DS1. [11].

We started the design process by having spacecraft and operations personnel identify expected values for each parameter based upon the nominal mission scenario. Using these assignments we generated test cases by varying each of the parameters across three distinct classes of values – nominal (single value), off-nominal (range of acceptable values), and extreme (most likely failure conditions). For each parameter, based on this decomposition, we defined a set of five values at the boundaries of these classes – a minimum value, an "off-nominal-min" value at the boundary between the off-nominal and the extreme, a nominal value, an "off-nominal-max", and a maximum value.

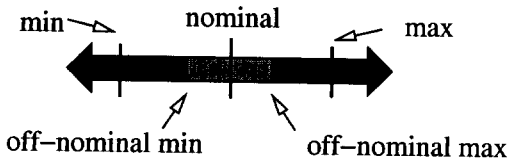

Figure 4. Parameter Decompositions

Using this decomposition of the test space, we generated three sets of test cases:

1. Coverage test cases that attempt to exercise a representative sample of all possible parameter-value assignments.
2. Stochastic test cases that verify nominal-operation scenarios.
3. Environmental test cases that evaluate how our agent performs in an uncertain environment.

4.2.1 Parameter-Coverage Test Set

Using the parameter decomposition we designed two sets of test cases, one that exercised the five values for each parameter while holding all other parameters within their nominal mission scenario, and another that exercised pair-wise combinations of parameter variations. Single-parameter variations allow for simple tests of off-nominal situations (variations that allow defects to be easily traced back to the source), while pair-wise combinations allow us to test the more complex interactions between parameters.

The single-parameter approach generates test sets that scale linearly with the number of parameters. Since we decomposed each of our parameters into five representative values, for N parameters, we have 5N test cases (or 4N+1 unique test cases as N of these will be the same nominal test set). For the EO-1 science agent this yields approximately 150 test cases. The pair-wise testing approach grows proportional to the number of pairs multiplied by the number of values for each pair or [k:2] * v * v for k parameters with v values. For EO-1, with 38 parameters each with 5 values, this gives us 17,575 test cases. Unfortunately even this number of test cases is impractical. Consequently we plan to use the method described in [6] and used by RAX [11], to reduce the number of pair-wise tests to a manageable level (under 100).

4.2.2 Stochastic Test Set

In all but one of the cases generated by our Coverage test set some parameter has an off-nominal or extreme value. While these tests give us confidence in the robustness of our system, they do not provide much evidence as to the correctness of execution in nominal scenarios. In order to test more nominal scenarios, and also gain coverage in the off-nominal scenarios outside of the five representative values, we devised a scheme for generating stochastic test sets based on parameter value distributions.

Parameters were given normal distributions around their nominal value, with standard deviations half the width of the off-nominal range (such that 95% of expected values will be either nominal or off-nominal). Nominal test sets were then generated assigning values to parameters based on the defined distributions. Furthermore, by modifying the construction of the parameter distribution, we were able to create off-nominal and extreme test sets that would stochastically favor some parameters to choose values outside of their nominal range.

4.2.3 Environmental Test Set

We further extended the stochastic test sets described above to include execution variations based on the parameter distributions. The spacecraft simulator was modified to allow as input variations to expected parameter values. During the execution of activities the simulator will simulate the change to each parameter of the current activity, and then vary the value returned based on the provided parameter distributions. Again nominal, off-nominal, and extreme test sets were generated that instructed the simulator to vary parameter values within the corresponding value class.

Finally we needed a way to test how the system responds to unexpected or exogenous events within the environment. These events could be fault conditions in the spacecraft or events outside of the CASPER model. Unlike the initial-state and execution-based testing described above, these events could happen at any time, and do not necessarily correspond to any commanded action or modeled spacecraft event. To accomplish this we added to our spacecraft simulator the ability to change the value of any parameter, at either an absolute time or time relative to the execution of an activity, to a fixed value or a value based on the distributions described above. We added small-variation events (within appropriate off-nominal and nominal classes) to our nominal and off-nominal stochastic test sets. Test cases are also currently in development that will use this capability to exercise the fault scenarios outlined in the Spacecraft Safety document. [5]

4.3 Testing Procedure

The number of test cases we plan to run is limited by available testing resources and the time remaining before mission inception. The EO-1 experiment has a compressed two month testing window with limited access to high-fidelity test beds. We forecast two-thirds of testing time will be spent evaluating output and running regression tests. The remaining one-third of our testing resources will be available for generated test cases. At two-hours per test run this gives us the capacity to run approximately 2400 tests.

At this time our automated test harness can detect "hard" test failures (i.e. crashes), and violations of system constraints (checked by the simulator). We also have in development and limited-deployment "goal-detection" software that evaluates whether CASPER successfully executed the goals specified in the mission scenario.

As an additional complication, for EO-1 we have a number of testbeds with varying degrees of fidelity to the actual flight environment. The vast majority of tests must be run on the Solaris and Linux testbeds, as they are the fastest and most readily available. However, these test the software under a different operating system, so are useful for testing of the model only. The operating system and timing differences are significant enough that many code behaviors occur only in the target operating system, compiler, and timing of interest. In order to validate aspects of the model dependent on precise timing we are forced to run tests on higher fidelity testbeds.

Table 4. Testbeds available to validate EO-1 agent.

Type	Number	Fidelity
Solaris Sparc Ultra	5	Low – can test model but not timing
Linux 2.5 GHz	7	"
GESPAC PowerPC 100-450 MHz	10	Moderate – runs flight OS
JPL Flight Testbed RAD 3000	1	Moderate
EO-1 Flight Testbed Mongoose M5, 12 MHz	1	High – runs Flight Software
EO-1 Autonomy Testbed (under construction, complete Sep 2003) Mongoose M5, 12 MHz	3	High – runs Flight Software

5. STATUS & DEPLOYMENT

Black box testing verifies that the system correctly follows those constraints encoded in the model and enforced by the layered architecture. However we also need a way to validate through testing that the model correctly encodes the requirements and constraints of the spacecraft. For EO-1 the first step in this process was the informal telemetry validation discussed previously in section 3. We have decided to augment this validation with a set of operational procedures that will help to build confidence in the safety of the EO-1 science agent before it is given full control to command the spacecraft. The autonomy software will be deployed as follows:

- Executive-only deployment (planned May 2003)

  In this phase SCL will be uploaded to the spacecraft and enabled to receive and process ground commands. SCL will monitor telemetry from the spacecraft but will be prevented from issuing any spacecraft commands across the software bridge. These commands will instead be downlinked as telemetry for review and safety inspection by the operations team.

- Ground generated flight test (planned May 2003)
  In this phase CASPER and SCL will generate a command sequence from ground-based testbeds for flight operations. This sequence will then be uplinked for execution onboard the spacecraft.

- Shadow flight operations (June 2003)

  Next we will upload CASPER and SCL to EO-1 and allow spacecraft telemetry and ground commands, but disable any spacecraft commands across the software bridge. This test will serve two purposes: test that the code executes as expected on the actual spacecraft, and validate that the generated command sequences conform to spacecraft safety and operations constraints.

- Full deployment (July 2003 – September 2004)

  After the sequences from shadow flight operations have been verified as correct, CASPER and SCL will be permitted to command the spacecraft. During this phase telemetry will be downlinked and checked on the ground to validate command sequences.

As this paper is being written, we are in integration of the full autonomy SW with the EO-1 FSW on the GSFC testbed. Additionally, the SCL-only software load is being uplinked to be tested May 22nd. While it is quite possible that there will be further delays from the schedule described above, the integration process is moving forward at a steady pace.

The majority of the testing effort has just begun. The ground-based testing has only begun in the last month, and additional analysis for test case generation is underway. Because of this, it is likely that the shadow mode of operations may need to be extended slightly, particularly if any of the flight tests cause anomalies.

6. CONCLUSIONS

This paper has described the design and validation of a safe agent for autonomous space science operations. First, we described the salient challenges in developing a robust, safe, spacecraft control agent. Second, we described how we used a layered architecture to enhance redundant checks for agent safety. Third, we described our model development, validation, and review. Fourth, we described our test plans, with an emphasis on reducing the number of test cases to a tractable set. Finally, we described our phased approach to flight, which provides additional safeguards.

7. ACKNOWLEDGEMENT

Portions of this work were performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

8. REFERENCES

[1] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, M. Slack, Experiences with an Architecture for Intelligent, Reactive Agents, Journal of Experimental and Theoretical Artificial Intelligence, 9:237-256, 1997.

[2] S. Chien, R. Sherwood, M. Burl, R. Knight, G. Rabideau, B. Engelhardt, A. Davies, P. Zetocha, R. Wainright, P. Klupar, P. Cappelaere, D. Surka, B. Williams, R. Greeley, V. Baker, J. Doan, "The TechSat 21 Autonomous Sciencecraft Constellation", Proc i-SAIRAS 2001, Montreal, Canada, June 2001.

[3] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO, April 2000. (see also casper.jpl.nasa.gov)

[4] S. Chien, R. Sherwood, D. Tran, R. Castano, B. Cichy, A. Davies, G. Rabideau, N. Tang, M. Burl, D. Mandl, S.Frye, J. Hengemihle, J. D'Agostino, R. Bote, B. Trout, S. Shulman, S. Ungar, J. Van Gaasbeck, D. Boyer, M. Griffin, R. Greeley, T. Doggett, K. Williams, V. Baker, J. Dohm, "Autonomous Science on the Earth Observer One Mission," ," *International Symposium on Artificial Intelligence Robotics and Automation in Space*, Nara, Japan, May 2003.

[5] S. Chien et al, EO 1 Autonomous Sciencecraft Experiment Safety Analysis Document, 2003.

[6] D. Cohen; Dalal, S.; Fredman, M.; and Patton, G.1997. The AETG system: An approach to testing based on combinatorial design. IEEE Transactions on Software Engineering 23(7):437-444.

[7] A.G. Davies, R. Greeley, K. Williams, V. Baker, J. Dohm, M. Burl, E. Mjolsness, R. Castano, T. Stough, J. Roden, S. Chien, R. Sherwood, "ASC Science Report," August 2001. (downloadable from ase.jpl.nasa.gov)

[8] E. Gat, Three layer architectures, in Mobile Robots and Artificial Intelligence, (Kortenkamp, Bonasso, and Murphy eds.), Menlo Park, CA: AAAI Press, pp. 195-210.

[9] Goddard Space Flight Center, EO-1 Mission page: eo1.gsfc.nasa.gov

[10] Interface and Control Systems, SCL Home Page, sclrules.com

[11] NASA Ames, http://ic.arc.nasa.gov/projects/remote-agent/, Remote Agent Experiment Home Page.

[12] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," *International Symposium on Artificial Intelligence Robotics and Automation in Space*, Noordwijk, The Netherlands, June 1999.