Validating the Autonomous EO-1 Science Agent

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

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
Firstname.Lastname@jpl.nasa.gov

Abstract. This paper describes the validation process for the Autonomous Sciencecraft Experiment, a software agent currently flying onboard NASA's EO-1 spacecraft. The agent autonomously collects, analyzes, and reacts to onboard science data. The agent has been designed using a layered architectural approach with specific redundant safeguards to reduce the risk of agent malfunction to the EO-1 spacecraft. This "safe" design has been thoroughly validated by informal validation methods supplemented by sub-system and system-level 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 used to validate agent safety.

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 react autonomously to maximize science return. For example, our Autonomous Science Agent can monitor active volcano sites and schedule multiple observations when an eruption has been detected. Or monitor river basins, and increase imaging frequency during periods of flooding.

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 (such as EO-1) has 8 10-minute 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 and ground operations teams must be able to operate the spacecraft with even more 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. Our CPU allocation for ASE on EO-1 is 4 MIPS and 128MB RAM.
5. High stakes. A typical space mission costs hundreds of millions of dollars, any failure has significant economic impact. The total EO-1 Mission cost is over $100 million dollars. 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 it provides a framework for agent safety.
2. We describe our knowledge engineering and model review process including identification of safety risks and mitigations.

3. We describe our testing process designed to validate the safe design of our agent's architecture and model.

We describe these areas in the context of the Autonomous Sciencecraft Experiment (ASE), an autonomy software package adapted to NASA's New Millennium Earth Observer One (EO-1) spacecraft [4] from a design originally proposed for flight on the Air Force's Techsat-21 Mission [2].

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] plans activities to achieve long-term mission objectives. CASPER submits the planned sequences of activities to the Spacecraft Command Language (SCL) system [10] for execution. Using an internal model, SCL expands the activities into sequences of EO-1 commands, which are then executed through the EO-1 Flight Software (FSW).

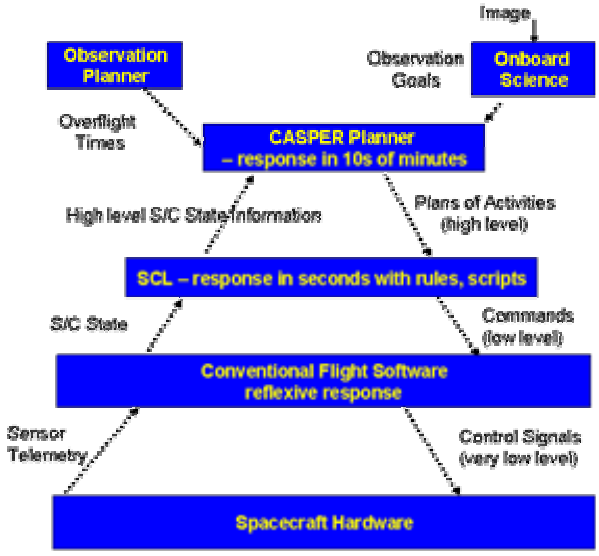

Figure 1. Autonomy Software Architecture

Operating on the tens-of-minutes timescale, CASPER responds to events that have multiple-orbit effects, including scheduling science observations and ground contacts. CASPER commands activities traditionally initiated through sequences uplinked by the EO-1 ground operations team. Consulting internal models of the spacecraft, CASPER searches for plans that combine these basic activities to satisfy high-level goals consistent with spacecraft operational and resource constraints.

Plans generated by CASPER are given to SCL at this basic-activity granularity. SCL expands the CASPER plan to detailed sequences of EO-1 spacecraft commands. 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 to expand activities and rules that monitor and enforce flight constraints.

SCL sends commands to the EO-1 FSW [9], the basic flight software that operates the EO-1 spacecraft. The interface from SCL to the EO-1 FSW is at the same level as ground generated command sequences – in other words the FSW does not know, or care, whether commands were issued by SCL or EO-1 ground operations.

SCL implements the commanding interface through a special component called the Autonomy Flight Software Bridge (FSB). The FSB takes autonomy software messages and issues corresponding FSW commands. The FSB also implements a new set of FSW commands to perform functions such as startup and shutdown of the autonomy software. This single interface point allows the EO-1 operations team to easily turn on and off the ASE commanding path, and thus ASE control, of EO-1.

The FSW 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 FSW as real-time commands. As part of its core, the FSW has a full fault and spacecraft protection functionality designed to:

1. Reject commands (from any source) that would endanger the spacecraft.
2. Execute pre-determined sequences to enter a "safe" mode upon detection of a hazardous state thereby stabilizing the spacecraft 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; 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.

Finally, the ASE software package includes a suite of science analysis algorithms. These algorithms process, interpret, and suggest reactions to onboard science observations. CASPER converts the science analysis suggestions to activities, and adds them to the onboard schedule for execution.

This layered architecture enables each lower layer to validate the output of the higher layers – SCL checks CASPER activities prior to sending the corresponding commands to the FSW, while the FSW fault protection checks the command sequences from SCL. This multiple-layer safety check emboldens confidence in the safety of our agent.

3 Model Building & Validation

Both CASPER and SCL rely on high-fidelity internal models of the EO-1 spacecraft. CASPER uses these models to delineate what goals can be achieved, and the scope of possible reactions. SCL uses models to generate command sequences and monitor activity execution. Any inaccuracies in these models could lead to ASE failing to achieve science objectives, or in the extreme, issuing unsafe sequences of commands. As such, these models were the product of a methodical development and review process designed to ensure they correctly encoded the relevant operational and safety constraints of EO-1.

The CASPER and SCL models share many of the same EO-1 constraints – including properties of physical subsystems, operation modes, valid command sequences, command prerequisites, and impacts of commands on spacecraft state. As a general rule however, CASPER models EO-1 at a higher level of abstraction than SCL. The activities commanded by CASPER are more abstract usually requiring tens or hundreds of spacecraft commands to achieve. Conversely, SCL activities sometimes expand to only a few spacecraft commands.

CASPER models the basic activities that must be assembled to complete the high-level mission goals including science observations and downlinks. The decomposition from goals to activities continues until a suitable level is reached for planning – a level that allows CASPER to model spacecraft state and its progression over time, discrete states such as instrument modes, and resources such as memory available for data storage. At this level of abstraction CASPER can commit activities in order to generate and repair schedules, track state, and monitor resources against predicted evolution.

SCL continues to model spacecraft activities and state at finer levels of detail. These activities are modeled as SCL scripts, which when chained together and executed, result in commands to the EO-1 FSW. SCL models spacecraft state through an internal database 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 transient changes. SCL uses its model to generate and execute sequences that are valid and safe in the current context. But while SCL has a detailed model of spacecraft state and resources, it does not generally model future evolution of state or resources.

The ASE team developed the CASPER and SCL models using an iterative multiple step process, that defined, modeled, reviewed, and validated EO-1 activities. Each of these steps focused on creating a high-fidelity model that was consistent with existing ground operations and constraints of the EO-1 spacecraft.

3.1 Model Development

The model development process began when a new high-level goal was tasked to ASE. At first ASE modeled simple goals, such as instrument calibrations. As we gained experience with the spacecraft, our modeling activities evolved to more complex multi-activity objectives including science observations, data downlinks, and spacecraft pointing.

With a new goal in hand, the ASE team would first identify the set of activities required to achieve the objective. Primarily this process was driven by a review of existing operations documents and engineering reports. For example, when ASE was tasked to begin collecting science data, prior EO-1 data collects were analyzed to see what sequences of commands, and thus activities, were required to image a science target. A science data collect requires activities to calibrate instruments, manage hardware operational modes, and command data recording from both the Hyperion and Advanced Land Imager (ALI) instruments.

With the activities defined, the ASE team reviewed formal EO-1 operations procedures to identify constraints on the selected 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 the ground station acquisition of signal (AOS) and loss of signal (LOS). Finally, fault protection documents listing fault monitors (TSMs) were consulted, reasoning that acceptable operations should not trigger any TSMs.

3.2 Model Reviews

Next, the ASE team conducted reviews where the latest iterations of the CASPER and SCL models were tabletop reviewed by a team composed of EO-1 spacecraft engineers and operators. Their working knowledge of the spacecraft, and experience over three years of operations, verified that no incorrect parameters or assumptions were 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, including the commands that the ASE software could execute, and derived a list of potential hazards ASE could pose to the spacecraft's health. For each of these hazards, a set of possible safeguards were proposed, and then implemented through operations procedures, and constraints embedded in the CASPER and SCL models. This analysis formed the basis for the testing of agent safety discussed in section 4. A sample 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.
FSW	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.

3.3 Code Generation

An interesting aspect of model development was the use of code generation techniques to derive SCL constraint checks from CASPER model constraints. In this approach, certain types of CASPER modeling constraints could be translated into SCL code to ensure consistency 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, a check could be auto-generated to verify a valid state 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 defined the constraint that "wfl" new files are created. We then auto-generated SCL code to verify that number of files can be created without exceeding the file limit before the WARP recording activity is allowed to execute.

```
// 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.

3.4 Sequence Generation

With the model defined, CASPER generated preliminary command sequences from past science requests representing a range of potential flight situations. These sequences were compared with the actual sequences generated and uplinked by the EO-1 ground team 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 potentially revealed an overlooked constraint on one or both of the commands. The EO-1 team also provided engineering telemetry from the onboard execution of these sequences. This telemetry allowed for execution comparisons to the telemetry generated by ASE. Additionally a novel "played back" capability was developed where the ASE software could be fed the results of commands using the actual effects observed onboard. 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. We converged on a consistent model after several iterations through this sequence generation process.

The sequence generation effort was in effect the crossover point between our model development process and the beginning of our system-level testing. While feeding directly into the iterative development process, it also allowed the first validation of the ASE model and software.

4 Testing Enforcement of Safety

Testing ASE against prior sequences would not be enough. We needed to show that onboard EO-1 the system would correctly plan, generate, and execute command sequences. Or, more importantly, that the generated command sequences would never endanger the safety of the spacecraft.

As demonstration software, the effort available for testing our agent was severely time and resource constrained. Therefore we decided early in the project that testing should focus primarily on ensuring that our agent executed safely. Missing a data collect would be an unfortunate although tolerable failure - endangering the safety of the EO-1 spacecraft would not.

Leveraging the completed safety analysis, we approached validation by breaking our testing strategy into three verification steps:

1. CASPER generates plans consistent both with its internal model of the spacecraft and SCL's model and constraints.
2. SCL does not issue any commands that violate the constraints of the spacecraft.
3. Both models accurately encode the spacecraft operational and safety constraints.

The first two steps build confidence that the ASE software executes within the constraints levied by the spacecraft model, while the third step verifies that the model encodes sufficient information to protect against potential safety violations.

We validated these requirements by extensive testing of the autonomy software on generated test-cases, using simulation and rule-based verification at each step. Note that the steps enumerated above, and the test cases described below, address only the top-two layers of the onboard autonomy software (CASPER and SCL). The existing EO-1 flight software testing and validation was addressed prior to ASE by a separate, more conventional, test plan. Additionally both CASPER and SCL are mature and tested software systems. The majority of the development effort for ASE was in the two internal models that adapt the systems to EO-1. Accordingly the testing strategy outlined below focuses the majority of the effort on exercising those models.

4.1 Test Case Parameters

Each EO-1 test case spans seven days of spacecraft operations covering multiple science observation and reaction opportunities. Each observation opportunity, referred to as a CASPER schedule window, represents an time period where ASE has been cleared to command EO-1. The test cases vary the state observed by ASE entering schedule windows (spacecraft state parameters), and vary the goals given to ASE through changes to mission and science objectives (mission scenario parameters). Additionally we employed simulators that changed the spacecraft state during test execution to simulate unknown environmental changes.

Mission scenario parameters represent the high-level planning goals passed to ASE. They are derived from a combination of the orbit and long-term science objectives. Mission scenario parameters specify when targets will be available for imaging, the parameters of science observations (i.e. number of targets to image and science analysis algorithms we wish to execute), and reactions to observed science events (i.e. follow-up observations).

Table 3. Mission-scenario parameters.

Parameter	Nominal	Off-nominal	Extreme
schedule 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
image start	anytime in orbit, 1 per orbit	1 per 3 orbits	any
image duration	8 s +/- 2	+/- 5	0,60
groundstation AOS	anytime in orbit, 1 per orbit	1 per 3 orbits	any
groundstation LOS	AOS + 10 min +/- 1	+/- 3	any
eclipse start	60 min after orbit start	+/- 5	any
eclipse duration	30 min	+/- 5	any
science algorithm	any	any	any
science goal start	fixed	not-specified	any
number of science goals	1 per orbit	1-2	>2
warp allocated	0	32K blocks	any

Spacecraft state parameters encode the relevant state of EO-1 at the start of a schedule window, and change as a result of commanded sequences. Changes to these parameters are simulated using a software simulator.

Table 2. Sample 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 number 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

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 2^{36} or over 68 billion test cases (each requiring on average a few hours to run). The challenge therefore becomes selecting a set of tests that most effectively cover the space of possible parameter variations within a timeframe that allows for reasonable software delivery.

4.2 Design of Test Cases

Traditional flight software is designed to be tested through exhaustive execution of a known set of command sequences. Command sequences usually must be run through a high-fidelity ground testbed before being cleared to run onboard.

Autonomy software however enables the spacecraft to execute in, and react to, a much wider range of possible scenarios. This flexibility enables new paradigms of operations and science, but comes at the price of complexity in testing and validation – tests that must attempt to intelligently cover the range of possible states and mission scenarios.

To trim the set of possible inputs, we took advantage of the scenarios identified by the model review process. For example, we never expect to take more than five science

data collects before a downlink (and usually exactly five as that is the limit of the WARP data storage). A downlink is almost always followed immediately by a format of the WARP. Science collections are always preceded by a slew and wheel bias and followed by a slew to nadir. Together these form a baseline mission scenario covering all the actions to be commanded by our agent.

Instead of testing every possible combination of spacecraft and mission parameters, we instead decided to vary parameters off of this baseline scenario, thus reducing the number of parameter variations for our test cases to consider. This is a similar approach to that used to validate the Remote Agent Planner for NASA's Deep Space 1 mission. [11].

We started the design process by using the nominal parameter values identified in the model review process. Using these assignments we generated test cases that varied each of the parameters across three distinct classes of values – nominal (single value), off-nominal (range of acceptable values), and extreme (failure conditions). For each parameter, 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.

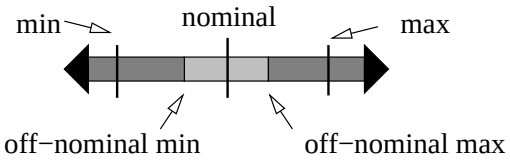

Figure 4. Parameter Decompositions

Using this decomposition we generated three sets of test cases:

1. Baseline scenario test cases that exercised just the baseline mission scenario.
2. Stochastic test cases, grounded in the baseline mission scenario, that varied parameters within nominal, off-nominal, and extreme ranges.
3. Environmental test cases that varied initial state, and inserted execution uncertainty.

4.2.1 Baseline-Scenario Test Set

The baseline mission scenario, identified in the model review process, was used for the first and most basic test set validating ASE.

This scenario provided exactly the expected sequences and parameter values to the ASE software. Any inconsistencies or anomalies in execution were easily traced back as the scenario was well understood and used previously to generate command sequences during the model review process.

4.2.2 Stochastic Test Set

Clearly the baseline test set did not fully exercise the autonomous planning and reaction capabilities of the system. In order to test more nominal scenarios, and also gain coverage in the off-nominal parameter ranges, we devised a procedure 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 simulated changes to each parameter of the current activity, and then varied 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 responded 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.

4.3 Testing Procedure

The test cases generated using the procedure outlined above were used in unit testing the individual agent layers and integrated system testing. Unit testing verified primarily the first two decompositions of our test plan – that CASPER commanded within its model, and that SCL did not violate any spacecraft constraints. Integrated testing verified that these constraints hold within the full system, and that the commanded sequences safely achieve the mission objectives.

The vast majority of tests were run on the Solaris and Linux platforms - as they were the fastest and most readily available. However, these test the software under a different operating system and processor, and therefore are primarily useful for testing assumptions in the CASPER and SCL models. The operating system and timing differences are significant enough that many code behaviors occur only in the target operating system, compiler, and processor configuration. Therefore every effort was made to extensively validate the agent 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 Mongoose M5, 12 MHz	2	High – runs Flight Software

On the Linux, Solaris, and GESPAC testbeds we used an automated test harness to setup, execute, and evaluate the results of each test run. Tests were run at accelerated speeds using the capabilities of our software simulator and the resources of the faster processors. The GESPAC and flight testbed configurations do not have similar acceleration capabilities, and therefore require tests to be run in real-time. The test harness ran over six years of autonomous operations during the first six months of our validation process.

To ensure stability, we implemented minimum requirements on the number of test cases that must execute without an identified failure before a build was cleared for flight. These requirements varied by platform as follows: 1 year of simulated operations on Linux/Solaris, 1 month on the GESPAC single board computers, and 1 week on the flight testbeds.

4.4 Success Criteria

To be considered successful a test run could not violate any spacecraft, operations, or safety constraints. On the Linux, Solaris, and GESPAC testbeds these constraints were checked by a software simulator that monitored activities committed by CASPER and executed by SCL. The simulator verified the timing, state, and resource constraints of the activities against those encoded in the CASPER model.

Recalling that our primary testing objective was to verify that our agent commanded EO-1 safely, we developed a separate "safety monitor" that watched only for violations of the safety and operations constraints. The safety monitor was developed with no knowledge of the CASPER or SCL models, and parsed the actual spacecraft commands issued by the autonomy software (isolated black-box testing). These commands were fed into state machines that monitored each of the safety and operations constraints – the same constraints that were derived from the safety and model review process. Any violations that were discovered were considered high-priority defects.

The flight testbeds used a higher-fidelity "Virtual Satellite (VSat)" simulator, developed independently from the autonomy software. The VSat simulator modeled the spacecraft at the subsystem level, including systems, states, and resources not modeled by CASPER or SCL.

5 Status & Deployment

The full ASE software has successfully commanded science observations onboard EO-1 since January 2004, As of April 2004, ASE has successfully collected target observations, analyzed science data onboard EO-1, and autonomously retargeted the spacecraft for subsequent observations.

Test Description	Test Date
First test of onboard cloud detection (science analysis)	March 2003
Verification of ASE-EO-1 FSW commanding path	May 2003
Onboard execution of CASPER ground-generated command sequences	July 2003
Full ASE software upload	August 2003
First ASE autonomously-commanded dark calibration image and downlink	October 2003
First ASE autonomous science observation	January 2004
First autonomous science analysis and subsequent reaction observation.	April 2004
Expanded EO-1 science operations automation.	May 2004-Present

6 Conclusions

This paper described the design and validation of a safe agent for autonomous space science operations. First, we described the 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. Finally, we described our test plan, with an emphasis on verifying agent safety.

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.

We would like to acknowledge the important contributions of Dan Mandl, Stuart Frye, and Stephen Ungar of NASA's Goddard Spaceflight Center, Jerry Hengemihle and Bruce Trout of Microtel LLC, Jeff D'Agostino of the Hammers Corp., Seth Shulman and Robert Bote of Honeywell Corp., and Jim Van Gaasbeck and Darrell Boyer of Interface and Control Systems.

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.