

# 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

Firstname.Lastname@jpl.nasa.gov

Dan Mandl, Robert Bote, Stuart Frye, Bruce Trout, Seth Shulman, Jerry Hengemihle, Jeff D'Agostino

Goddard Space Flight Center

Firstname.Lastname@gssc.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 is currently flying onboard NASA's EO-1 spacecraft. The agent autonomously collects, analyzes, and reacts to onboard 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 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.

## 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 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 originally designed for flight on the Air Force's Techsat-21 Mission [2] in 2006 and now being flown on NASA's New Millennium Earth Observer One (EO-1) spacecraft [4].

## 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.

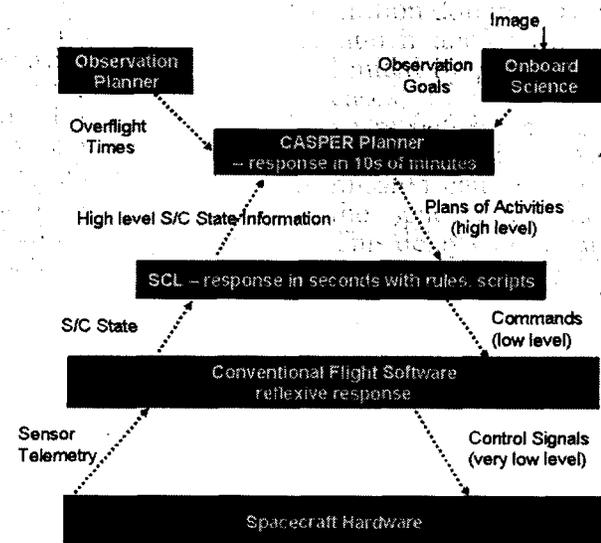


Figure 1. Autonomy Software Architecture

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 where the models were tabletop reviewed by a team of personnel with a range of operations and spacecraft background. This added confidence 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 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. 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
<b>Operations</b>	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.
<b>CASPER</b>	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)
<b>SCL</b>	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.
<b>FSS</b>	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

```

*not shared alone code*

**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

As demonstration software, the effort available for testing our agent has been 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 validate 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 is addressed 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. Therefore the testing strategy outlined below focuses the majority of the effort on exercising these models.

#### 4.1 Test Case Parameters

Each EO-1 test case covers seven days of spacecraft operations including multiple observation opportunities. Each observation opportunity, referred to as a CASPER schedulable window, represents an opportunity to schedule one or more science data collections or downlinks. The test cases 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.

Mission scenario parameters represent the high-level planning goals passed to CASPER. They are derived from a combination of the orbit and the science objectives uplinked from the ground. 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).

Spacecraft state parameters encode the state of EO-1 at the start of a schedulable window, and changes to the spacecraft as a result of our agent's actions. Changes to these parameters are simulated using a software simulator that models spacecraft state.

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 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
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

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 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 the possible scenarios would easily be intractable.

To trim the set of possible inputs, we can take 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 our test cases 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 using the nominal parameter values identified in the model review process. Using these assignments we generated test cases that vary 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.

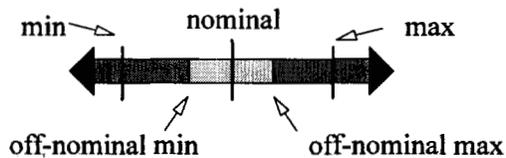


Figure 4. Parameter Decompositions

Next we generated three sets of test cases:

1. Coverage test cases that exercise the five boundary parameter-value assignments.
2. Stochastic test cases, grounded in the baseline mission scenario, that vary parameters within nominal, off-nominal, and extreme ranges.
3. Environmental test cases that vary initial state, and insert execution uncertainty.

#### 4.2.1 Parameter-Coverage Test Set

Using the parameter decomposition we designed test cases off the baseline scenario that exercised the five values for each parameter while holding all other parameters constant. Single-parameter variations allow for simple tests of off-nominal situations (variations that allow defects to be easily traced back to the source).

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 a baseline test set of approximately 150 test cases.

#### 4.2.2 Stochastic Test Set

Our Coverage test set exercises individual parameters, but does not exercise the interactions between multiple off-nominal parameters or even multiple parameters varied within their nominal ranges. 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 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 simulates changes to each parameter of the current activity, and then varies 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, as well as integrated system testing. Unit testing verifies primarily the first two decompositions of our test plan – that CASPER command within its model, and SCL did not violate any spacecraft constraints. Integrated testing verifies 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 are the fastest and most readily available. However, these test the software under a different operating system, 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 timing of interest. 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. The test harness ran over ten thousand simulations during our validation process.

The number of integrated test cases we ran before each upload was limited by available testing resources and the time remaining before each upload opportunity. 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: 3 years 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 must 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. This simulator checks the timing, state, and resource constraints of the activities against those that are encoded in the CASPER model. The flight testbeds used a higher-fidelity “Virtual Satellite (VSat)” simulator, developed independently from the autonomy software, for verification of command sequences. The VSat simulator modeled the spacecraft at the subsystem level, including systems, states, and resources not modeled by CASPER or SCL.

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. Any violations that were discovered were considered high-priority defects.

#### 5. STATUS & DEPLOYMENT

The ASE software has been steadily progressing to full operations with the major milestones listed below.

Test Description	Test Date
Onboard cloud detection	March 2003
Onboard commanding path	May 2003
CASPER ground generated commands executed onboard	July 2003
Software jumping and loading	August 2003
ASE autonomously acquires dark calibration image and performs downlink	October 2003
ASE autonomously acquires science images and performs downlinks	January 2004 - present
ASE autonomously analyzes science data onboard and triggers subsequent observations	April 2004 (expected)

The only step remaining for full operations is the flight of the integrated science with autonomous planning and execution. This software is currently in integration and test and is expected to be ready for flight in the April 2004 timeframe. When this software build is ready it will be flown until September 2004 and will be used to acquire

as many science-triggered scenes as resources allow.

## 6. CONCLUSIONS

This paper has 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.

## 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](http://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](http://ase.jpl.nasa.gov))
- [8] E. Gat, Three layer architectures, in *Mobile Robots and Artificial Intelligence*, (Kortenkamp, Bonasso, and Murphy eds.), Menlo Park, CA: AAI Press, pp. 195-210.
- [9] Goddard Space Flight Center, EO-1 Mission page: [eol.gsfc.nasa.gov](http://eol.gsfc.nasa.gov)
- [10] Interface and Control Systems, SCL Home Page, [sclrules.com](http://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.