Formal Methods for Trusted Space Autonomy: Boon or Bane?*

Steve A. Chien[1 [0000−0003−1023−9480]]

Jet Propulsion Laboratory,
California Institute of Technology,
Pasadena CA 91109-8099, USA

Abstract. Trusted Space Autonomy is challenging in that space systems are complex artifacts deployed in a high stakes environment with complicated operational settings. Thus far these challenges have been met using the full arsenal of tools: formal methods, informal methods, testing, runtime techniques, and operations processes. Using examples from previous deployments of autonomy (e.g. the Remote Agent Experiment on Deep Space One, Autonomous Sciencecraft on Earth Observing One, WATCH on MER, IPEX, AEGIS on MER, MSL, and M2020, and the M2020 Onboard planner), we discuss how each of these approaches have been used to enable successful deployment of autonomy. We next focus on relatively limited use of formal methods (both prior to deployment and runtime methods). From the needs perspective, formal methods may represent the best chance for reliable autonomy. Testing, informal methods, and operations accommodations do not scale well with increasing complexity of the autonomous system as the number of text cases explodes and human effort for informal methods becomes infeasible. However from the practice perspective, formal methods have been limited in their application due to: difficulty in eliciting formal specifications, challenges in representing complex constraints such as metric time and resources, and requiring significant expertise in formal methods to apply properly to complex, critical applications. We discuss some of these challenges as well as the opportunity to extend formal and informal methods into runtime validation systems.

Keywords: Verification and Validation · Flight Software · Space Autonomy · Artificial Intelligence.

1 Introduction

From the dawn of the space era, software has played a key role in the advancement of spaceflight. In the Apollo program, flight software in the Apollo Guidance Computer [16] enabled the astronauts to safely land on the Moon despite a radar configuration switch being set incorrectly.

* Contact author: steve.a.chien@jpl.nasa.gov. https://ai.jpl.nasa.gov
(c) Copyright 2022 California Institute of Technology.

Yet even with this success, the Apollo flight software development process encountered tremendous challenges [31], many of which would be quite familiar to flight software teams of today:

- inadequate memory available for software to meet stated requirements,
- evolving requirements,
- unit software being delivered to integration without any unit testing,
- late software deliveries jeopardizing project schedule (even the launch dates), and
- challenges in coordination between the teams distributed at NASA (Houston, TX and Huntsville, AL) and MIT (Cambridge, MA).

The Apollo program mitigated these challenges using methods that would be familiar to current flight software teams:

- revolutionary use of an interpreted "higher order language" rather than machine or assembly code
- requirements driven software development,
- reduction in scope of the software (reducing the fidelity of the Earth model used in lunar orbit, some attitude maneuver computations),
- development of significant infrastructure to support significant software testing (e.g. hardware and software simulations),
- institution of change control boards to restrict scope changes, and
- mitigating the distributed teams by having key personnel spend time co-location with other team elements.

In the end, the Apollo flight software delivered spectacularly, in no small part because of the tremendously talented team. The lessons learned from the Apollo flight software effort [31] would also come as no surprise to current flight software practitioners:

- documentation is crucial,
- verification must proceed through several levels,
- requirements must be clearly defined and carefully managed,
- good development plans should be created and executed, and
- more programmers do not mean faster development.

The Apollo flight software can be considered the "first" space autonomy flight software. The verification and validation process for this consisted primarily of extensive unit and system level testing. Although it is not described explicitly as such [31], informal methods must also have been heavily used in the form of code reviews and algorithm reviews.

But if we are to realize the incredible promise of autonomy in future space missions [10], which relies on reliable, trusted, autonomy flight software, what are the prospects for such software moving forward? We argue that all three major elements of validation and verification techniques will be critical as we move into an era of greater autonomy flight software: formal methods, informal methods,

and testing. More precisely defined, verification typically refers to ensuring that the software meets a specification and validation ensuring that the software meets the customer/user needs. For the purposes of our discussion, the focus is on verification but some elements of user studies, acceptance testing and informal design reviews would also address validation. Also for the purposes of this paper we use the following informal definitions.

Testing - exercising software artifacts - units, combinations of units, and system level on inputs both within and beyond the design specifications.

Formal Methods - analytical and search based methods intended to prove specific positive or negative properties of software or algorithms. Examples formal methods include model checking and static code analyzers.

Informal Methods - includes design reviews, code reviews, safety analysis, and coding guidelines. Informal methods tend to be people and knowledge intensive which is both a strength and a weakness. Some application of Formal Methods that requires expert translation or re-implementation of an algorithm into a different modelling language might best be considered hybrid formal/informal methods with the manual translation being an informal method.

In the remainder of this paper, we first describe major autonomy software that has been flown in space (including development of Mars 2020 Autonomy Flight Software scheduled for deployment in 2023) and discuss the use of informal methods, formal methods, and testing to Verify and Validate said software.

We then discuss the promise and the challenges in growing the role of formal methods in developing increasingly robust, verified and validated autonomy flight software.

2 Past Verification and Validation of Autonomy Flight Software

While only a small fraction of space missions include significant autonomy flight software, because of the large number of space missions there have been numerous flights of autonomy software. In this section we survey prior flights of autonomy/artificial intelligence software and describe the use of testing, informal methods, and formal methods in their deployment.

2.1 Remote Agent Experiment

The Remote Agent Experiment (RAX) [30] flew a planner-scheduler, task executive, and mode identification and recovery software onboard NASA's Deep Space One mission for two periods totaling approximately 48 hours in 1999. RAX represented the first spaceflight of significant AI software. RAX made extensive

use of multiple software and hardware testbeds of varying fidelity [4] to Verify and Validate the RAX software.

The verification and validation of the onboard planner used novel methods for testing including definitions of test coverage, use of a logical domain specification to check plans for correctness (derived from the planner model) and also checks automatically derived from flight rules [33] [13].

RAX was not only a significant advance in autonomy but also demonstrated significant use of formal methods for verification. Specifically, the executive was verified pre-flight using the SPIN model checker which identified several concurrency bugs [18]. Additionally, when an anomaly occurred during flight, an experiment was conducted to use formal methods to isolate the issue in a java surrogate for the flight code [17]. These successes are an excellent indicator of the utility of formal methods for AI/Autonomy software.

2.2 Autonomous Sciencecraft on Earth Observing One

The Autonomous Sciencecraft (ASE) flew onboard the Earth Observing One (EO-1) Mission and enables significant science-driven autonomy [9, 27]. ASE flew the CASPER onboard planning system, the Spacecraft Command Language (SCL) task executive, and also Onboard Data Analysis software (including Support Vector Machine Learning). ASE later flew the Livingston 2 (L2) Mode Identification and Recovery software as a further flight experiment but L2 was not used operationally [19, 20]. ASE enabled onboard analysis of acquired imagery and modification of the future mission plan to acquire more images based on image analysis. ASE originally was slated as a 6 month technology demonstration, but was so successful that it was approved for continued operational usage and was the primary missions operation software for EO-1 for the remainder of the mission 2004-2017 (over a dozen years). ASE represented flight of a considerable code base (over 100 K source lines of code (SLOC), primarily in C++ and C. Preparing this large code base for flight required overcoming significant software issues including memory allocation and code image size [34].

ASE was verified and validated using a combination of informal methods, formal methods, and testing [11]. Significant testing was performed on a range of software and hardware platforms of varying fidelity and included: requirements-based testing, unit testing, system-level testing, and scenario-based testing - including nominal, off nominal, and extrema scenarios.

ASE made heavy use of informal methods as well. A safety review was conducted studying over 80 potential ways in which incorrect operations could harm the spacecraft. ASE used a layered software and operations architecture with multiple redundant layers of: operations procedures, planner, executive, base flight software, and hardware. Therefore every layer could be used to redundantly enforce flight rules to protect the spacecraft. This layered architecture was very effective in enabling reliable operations.

ASE Verification and Validation had limited use of formal methods. Multiple static code checkers were used to check all ASE code. Automated code gener-

ation was used to generate of SCL checks from CASPER activity and resource specifications (this could be considered a form of runtime validation).

For a description of anomalies encountered during ASE operations and causes see [35]. It is worth noting that the majority of these anomalies could be considered systems engineering issues that were manifested in software, not core software errors (like pointer de-referencing or memory allocation issues).

2.3 WATCH/SPOTTER on Mars Exploration Rovers

WATCH/SPOTTER is image analysis software that was operationally qualified on the Mars Exploration Rovers (MER) mission [5] (WATCH is the MER software module name and SPOTTER is the name designated in publication(s)). WATCH was tested at the unit and subsystem level on testbeds ranging from workstation to the actual MER ground rover testbed. Informal methods were also used: coding guidelines, code walkthroughs, and software design document reviews. Standard code static analyzers were also applied as part of the project standard software process.

2.4 AEGIS on MER, MSL, and M2020

AEGIS is software used on the MER, MSL, and M2020 rover missions that allows the rover to acquire wide FOV imagery, find targets according to user specified science criteria, and target with narrow FOV sensors. AEGIS was originally developed for the MER mission Mini-TES and Pancam instruments [1] [12], updated for MSL with the Chemcam instrument [14], and is now in use on M2020 with the SuperCam instrument. AEGIS represents a significant code base at just under 30K lines of source code (SLOC).

Prior to deployment on all three rover missions, AEGIS was subjected to testing on testbeds ranging from workstations to actual ground rover testbeds. Informal methods were also applied such as code walkthroughs, software module reviews, and requirements analysis. Formal methods static code analyzers were also used as part of the normal software development process.

2.5 MSL FSW

While technically not all autonomy software, the Mars Science Laboratory (MSL) flight software development practices are worth considering as they represent the state of the practice for flight software development [21].

MSL heavily used a range of informal methods to ensure software quality including:

- risk-based coding rules (such as assertion density),
- design and code walkthroughs, and

[1] Unfortunately the Mini-TES instrument failed before AEGIS-MER operational qualification so AEGIS was never able to be used with Mini-TES on MER on Mars.

– documentation requirements and reviews.

Notably, the MSL project automated checking of the above software requirements.

MSL also conducted an extensive testing program on testbeds ranging from WSTS/linux workstation to flight testbeds.

Finally, MSL used formal methods in several ways. First, the SPIN model checker was used to search for concurrency issues in critical multithreaded code [21]. Second, significant amounts of code were automatically generated from higher-level specifications (such as controllers from statecharts). Third, MSL used the Coverity, Codesonar, Semmle, and Uno static code analyzers.

2.6 Intelligent Payload Experiment (IPEX)

IPEX [7] was a cubesat technology demonstration mission that demonstrated high throughput onboard processing for the HyspIRI Intelligent Payload Module (IPM) concept [8]. IPEX used the CASPER planner, a linux shell-based task executive, and numerous onboard instrument analysis software modules.

IPEX followed the same software processes as ASE. However, because IPEX was a much less complex spacecraft than EO-1 (specifically no active attitude control) the overall operations constraints were less complex. For IPEX the flight processor was running linux. This simplified the Verification and Validation process because there was very little difference between workstation and flight testbed environments - greatly facilitating testing. As with ASE, unit and system level testing, including nominal, off nominal, and extrema cases were performed. Informal methods included code, software module, and safety-based walkthroughs and reviews. Use of formal methods was limited to static code analyzers.

3 Current Validation of Autonomy Software: Onboard Planner for M2020

The Mars 2020 Mission is deploying an onboard scheduler to the Perseverance rover as this paper goes to press (Spring 2022) with a target operational date in 2023. This onboard scheduler would control most of the activities of the rover - including rover wake/sleep [32] [28]. This onboard scheduler must be fit within limited rover computing resources [15]. The onboard scheduler also utilizes flexible execution (which can be viewed as taking on the role of an executive) [1] and also supports a limited form of disjunction in plans [2]. The onboard planner represents a sizeable, complex code base at approximately 56K source lines of code (SLOC). The ground-based version of the automated scheduler [36] also has an explanation capability [3] to assist the ground operations team in understanding possible plans and outcomes.

The onboard planner is being verified using a combination of testing, informal methods, and formal methods. Testing includes unit test, systems test, and

scenario tests. Specifically scenario testing includes approximately 1 year of operations data of the Perseverance rover since landing. Informal methods includes code walkthroughs, coding guidelines and rules (see MSL above), as well as design reviews and software documentation. Finally, formal methods include the use of static code analyzers as part of the M2020 software development process.

4 Discussion of Competing Verification and Validation methods

In some sense, formal methods may be seen as more promising to achieve robust Verification and Validation to large scale, complex, autonomous systems. Consider the weaknesses of Testing and Informal Methods.

4.1 Limitations of Testing and Informal Methods

Testing can only reveal bugs, it cannot prove a software artifact bug-free. Residual defect rate refers to the defect rate in released software (e.g. post validation). Even the highly verified NASA space shuttle avionics software experienced 0.1 residual defects per KLOC [26] and leading-edge software companies experience a residual defect rate of 0.2 residual defects per KLOC [25]. A more broad reliability survey showed a residual defect rate of 1.4 per KLOC [29] and a Military system survey [6] showed a residual defect rate of 5-55 residual detects per KLOC. Additionally, testing can be extremely expensive both in terms of infrastructure (test drivers, simulators, oracles to evaluate tests) as well as time and computing power.

Informal methods can leverage significant human expert knowledge but are also incredibly time, labor, and expertise intensive and therefore add considerable expense to the software validation process.

4.2 Limitations of Formal Methods

Given the considerable weaknesses of testing and informal methods, one might consider why formal methods are not used. However consider the following challenges for application of formal methods to validation of autonomous space systems.

The Formal Specification Problem Typically in order to apply formal methods, one needs three formal specifications: the target artifact, the algorithm/semantics, and the conditions to check. For example, when analyzing a computer program for race conditions, the target artifact is the program itself, the algorithm/semantics are the semantics of the programming language, and the conditions would be a formal specification of the "race conditions" one wishes to identify. If one is validating that a space system planner will generate valid plans, the target artifact might be the planner model, the algorithm/semantics

might be the target planner algorithm for generating plans, and the conditions might be some specification of soundness or termination. The challenge of this approach is twofold. First, it is a tremendous amount of effort to derive the second and third specification, whose primary purpose is to enable the application of the formal methods analysis. Second, even if one is able to derive these specifications, they themselves are suspect and the process is only as good as these input specifications. E.g. recursively one might require a Verification and Validation process on these inputs as well.

The Representation Problem Formal methods are challenged by expressive representations. Specifically, space applications are demanding in their requirements for: complex spatial representations of location, free space, pointing and geometry; mixed discrete and continuous quantities and resources; and use of multiple, variable resolution time systems. Any one of these presents considerable challenges for formal methods, space applications often include most if not all of these representational challenges simultaneously. On the other hand, practical problems are typically propositional (or at least bounded instances) so that the truly general representations (such as first order predicate logic) are not strictly required. Still, in order for formal methods to make further headway in Verification and Validation of space autonomous systems, further advances in domain modelling capability are needed.

The Tractability Problem A formal methods proof that a property holds often is achieved by exhaustive search of some execution space (such as proving non concurrency of two elements may require searching the entire space of element orderings). For many space autonomy problems complete search of such a problem space is computationally intractable.

In some cases static source code analysis and logic model checking can been used to study the dual problem. Instead of exhaustively searching a problem space to prove a property, one searches in the problem space for violations of the property. In this way, even partial search can identify issues in the code [23]. This in some ways is more akin to testing but can achieve much greater coverage more rapidly (e.g. this approach can be considered a more efficient means of testing). Unfortunately, such approaches suffer similar drawbacks as testing - eg that they can only find issues and cannot (without complete search) indicate that no such issues exist.

Note also that increasing computing capabilities and swarm-based distributed methods of validation [24, 22] spread computational difficulties of these approaches may be mitigated. However, because many of these search problems scale exponentially om problem specification (e.g. code size) progress can be elusive.

The Expertise Problem Because of the above challenges, it often requires considerable expertise to apply formal methods to good effect. For example, the MSL concurrency analysis was performed by world class experts in formal

methods. Because of the challenges described, one must not only be able to develop formal specifications, but one must understand how to build specifications that model the correct aspects of the application and are amenable to efficient analysis (e.g. this deeper application of formal methods is far from out of the box static code analyzers). In many respects, this is analogous to the situation with autonomy for space applications, in which considerable expertise in software, space, and operations is needed to develop and deploy critical autonomy software.

5 Conclusions

This paper has discussed prospects for an increasing role for formal methods in the verification and validation of autonomy flight software. We first surveyed a number of prior and ongoing developments of autonomy flight software and described their use of testing, informal methods, and formal methods. In all of these cases, the bulk of the effort consisted of testing and informal methods. With only a few notable exceptions (such as MSL code generation and Model checking of critical code), usage of formal methods was restricted to use of static code analyzers. We then discussed several challenges in application of formal methods that restrict its usage: The Formal Specification Problem, The Representation Problem, The Tractability Problem, and The Expertise Problem. Yet because of the inherent limitations of testing and informal methods, we are still optimistic and believe that formal methods are an essential tool in the development of space autonomy software in the future.

6 Acknowledgments

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

References

1. Agrawal, J., Chi, W., Chien, S.A., Rabideau, G., Gaines, D., Kuhn, S.: Analyzing the effectiveness of rescheduling and flexible execution methods to address uncertainty in execution duration for a planetary rover. Robotics and Autonomous Systems **140 (2021) 103758** (2021), https://doi.org/10.1016/j.robot.2021.103758
2. Agrawal, J., Chi, W., Chien, S.A., Rabideau, G., Kuhn, S., Gaines, D., Vaquero, T., Bhaskaran, S.: Enabling limited resource-bounded disjunction in scheduling. Journal of Aerospace Information Systems **18:6**, 322–332 (2021), https://doi.org/10.2514/1.I010908
3. Agrawal, J., Yelamanchili, A., Chien, S.: Using explainable scheduling for the mars 2020 rover mission. In: Workshop on Explainable AI Planning (XAIP), International Conference on Automated Planning and Scheduling (ICAPS XAIP) (October 2020), https://arxiv.org/pdf/2011.08733.pdf

4. Bernard, D.E., Gamble, E., Rouquette, N.F., Smith, B., Tung, Y.W., Muscettola, N., Dorias, G.A., Kanefsky, B., Kurien, J., Millar, W., et al.: The remote agent experiment. In: Deep Space One Technology Validation Symposium. Pasadena, CA (February 1999), https://ntrs.nasa.gov/api/citations/20000116204/downloads/20000116204.pdf
5. Castano, A., Fukunaga, A., Biesiadecki, J., Neakrase, L., Whelley, P., Greeley, R., Lemmon, M., Castano, R., Chien, S.: Automatic detection of dust devils and clouds at mars. Machine Vision and Applications **19 (5-6)**, 467–482 (October 2008)
6. Cavano, J., LaMonica, F.: Quality assurance in future development environments. IEEE Software **Sep** (1987)
7. Chien, S., Doubleday, J., Thompson, D.R., Wagstaff, K., Bellardo, J., Francis, C., Baumgarten, E., Williams, A., Yee, E., Stanton, E., Piug-Suari, J.: Onboard autonomy on the intelligent payload experiment (ipex) cubesat mission. Journal of Aerospace Information Systems (JAIS) (April 2016)
8. Chien, S., Mclaren, D., Tran, D., Davies, A.G., Doubleday, J., Mandl, D.: Onboard product generation on earth observing one: A pathfinder for the proposed hyspiri mission intelligent payload module. IEEE JSTARS Special Issue on the Earth Observing One (EO-1) Satellite Mission: Over a decade in space (2013)
9. Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davies, A., Mandl, D., Frye, S., Trout, B., Shulman, S., Boyer, D.: Using autonomy flight software to improve science return on earth observing one. Journal of Aerospace Computing, Information, and Communication (JACIC) pp. 196–216 (April 2005)
10. Chien, S., Wagstaff, K.L.: Robotic space exploration agents. Science Robotics (June 2017), http://https://www.science.org/doi/10.1126/scirobotics.aan4831
11. Cichy, B., Chien, S., Schaffer, S., Tran, D., Rabideau, G., Sherwood, R.: Validating the autonomous eo-1 science agent. In: International Workshop on Planning and Scheduling for Space (IWPSS 2004). Darmstadt, Germany (June 2004)
12. Estlin, T., Bornstein, B., Gaines, D., Anderson, R.C., Thompson, D., Burl, M., Castaño, R., Judd, M.: Aegis automated targeting for the mer opportunity rover. ACM Transactions on Intelligent Systems and Technology **3(3)** (2012)
13. Feather, M.S., Smith, B.: Automatic generation of test oracles—from pilot studies to application. Automated Software Engineering **8**(1), 31–61 (2001)
14. Francis, R., Estlin, T., Doran, G., Johnstone, S., Gaines, D., Verma, V., Burl, M., Frydenvang, J., Montaño, S., Wiens, R.C., Schaffer, S., Gasnault, O., DeFlores, L., Blaney, D., Bornstein, B.: Aegis autonomous targeting for chemcam on mars science laboratory: Deployment and results of initial science team use. Science Robotics (June 2017)
15. Gaines, D., Rabideau, G., Wong, V., Kuhn, S., Fosse, E., Chien, S.: The mars 2020 on-board planner: Balancing performance and computational constraints. In: Flight Software Workshop (February 2022)
16. George, A.: Margaret hamilton led the nasa software team that landed astronauts on the moon (2019 (retrieved 25 March 2022)), https://www.smithsonianmag.com/smithsonian-institution/margaret-hamilton-led-nasa-software-team-landed-astronauts-moon-180971575/
17. Havelund, K., Lowry, M., Park, S., Pecheur, C., Penix, J., Visser, W., White, J.L.: Formal analysis of the remote agent before and after flight. In: Lfm2000: Fifth NASA Langley Formal Methods Workshop (2000)
18. Havelund, K., Lowry, M., Penix, J.: Formal analysis of a space-craft controller using spin. IEEE Transactions on Software Engineering **27**(8), 749–765 (2001)

19. Hayden, S.C., Sweet, A.J., Christa, S.E.: Livingstone model-based diagnosis of earth observing one. In: AIAA Intelligent Systems Technical Conference. AIAA (2004), https://doi.org/10.2514/6.2004-6225
20. Hayden, S.C., Sweet, A.J., Shulman, S.: Lessons learned in the livingstone 2 on earth observing one flight experiment. In: AIAA Infotech@Aerospace. AIAA (2005), https://doi.org/10.2514/6.2005-7000
21. Holzmann, G.J.: Mars code. Communications of the ACM **57**(2), 64–73 (2014)
22. Holzmann, G.J.: Cloud-based verification of concurrent software. In: International Conference on Verification, Model Checking, and Abstract Interpretation. pp. 311–327. Springer (2016)
23. Holzmann, G.J.: Test fatigue. IEEE Software **37**(4), 11–16 (2020)
24. Holzmann, G.J., Joshi, R., Groce, A.: Swarm verification techniques. IEEE Transactions on Software Engineering **37**(6), 845–857 (2010)
25. Jones, C.: Applied Software Measurement. McGraw-Hill (1991)
26. Joyce, E.: Is error free software possible? Datamation **Feb**(18), 749–765 (1989)
27. JPL-Artificial-Intelligence-Group: Autonomous sciencecraft web site (2017 (retrieved 25 March 2022)), https://ai.jpl.nasa.gov/public/projects/ase/
28. JPL-Artificial-Intelligence-Group: Mars 2020 onboard planner web site (2017 (retrieved 25 March 2022)), https://ai.jpl.nasa.gov/public/projects/m2020-scheduler/
29. Musa, J., et al: Software reliability: measurement, prediction, application. McGraw-Hill (1990)
30. Muscettola, N., Nayak, P.P., Pell, B., Williams, B.C.: Remote agent: To boldly go where no ai system has gone before. Artificial intelligence **103**(1-2), 5–47 (1998)
31. NASA: Chapter two: Computers on board the apollo spacecraft. In: Computers in Spaceflight: The NASA Experience. NASA (retrieved 27 March 2022), https://history.nasa.gov/computers/Ch2-6.html?mod=article_inline
32. Rabideau, G., Wong, V., Gaines, D., Agrawal, J., Chien, S., Kuhn, S., Fosse, E., Biehl, J.: Onboard automated scheduling for the mars 2020 rover. In: Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation for Space. i-SAIRAS'2020, European Space Agency, Noordwijk, NL (2020)
33. Smith, B.D., Feather, M.S., Muscettola, N.: Challenges and methods in testing the remote agent planner. In: AIPS. pp. 254–263 (2000)
34. Tran, D., Chien, S., Rabideau, G., Cichy, B.: Flight software issues in onboard automated planning: Lessons learned on eo-1. In: International Workshop on Planning and Scheduling for Space (IWPSS 2004). Darmstadt, Germany (June 2004), https://ai.jpl.nasa.gov/public/papers/tran_iwpss2004.pdf
35. Tran, D., Chien, S., Rabideau, G., Cichy, B.: Safe agents in space: Preventing and responding to anomalies in the autonomous sciencecraft experiment. In: Safety and Security in Multi Agent Systems Workshop (SASEMAS), Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2005). Utrecht, Netherlands (July 2005), https://ai.jpl.nasa.gov/public/papers/tran_sasemas2005_PreventingResponding.pdf
36. Yelamanchili, A., Agrawal, J., Chien, S., Biehl, J., Connell, A., Guduri, U., Hazelrig, J., Ip, I., Maxwell, K., Steadman, K., Towey, S.: Ground-based automated scheduling for operations of the mars 2020 rover mission. In: Proceedings Space Operations 2021 (May 2021), https://spaceops.iafastro.directory/a/proceedings/SpaceOps-2021/SpaceOps-2021/6/manuscripts/SpaceOps-2021,6,x1385.pdf