Flight Software Issues in Onboard Automated Planning: Lessons Learned on EO-1

Daniel Tran, Steve Chien, Gregg Rabideau, Benjamin Cichy
Jet Propulsion Laboratory, California Institute of Technology
Contact: {firstname.lastname}@jpl.nasa.gov

Abstract. Planning and scheduling systems for spacecraft operations have traditionally been an important step in the ground operations of mission planners. In the Autonomous Sciencecraft Experiment (ASE), this step is moved onboard the Earth Observing-1 spacecraft. The ASE features several advanced technologies: onboard image processing, a robust execution engine, and onboard planning and scheduling. This paper focuses on the onboard planner and scheduler CASPER, whose core planning engine is based on the ground system ASPEN. Given the challenges of developing flight software, we discuss several of the issues encountered in preparing the planner for flight, including reducing the code image size, determining what data to place within the engineering telemetry packet, and performing long term planning.

1 Introduction

The Autonomous Sciencecraft Experiment (ASE), currently flying onboard the Earth Observing-1 (EO-1) [4] spacecraft, demonstrates the future in space missions using several integrated autonomy software technologies to perform onboard image processing, robust execution, onboard planning and scheduling, and autonomous spacecraft re-tasking. The onboard planner generates a detailed operations plan from high-level requests sent from the ground. This plan is sent to the robust execution engine, which issues commands to the flight software. The onboard science algorithm analyzes acquired images and autonomously issue new requests to the planner to generate a new observation plan, and re-task the spacecraft.

The ASE onboard flight software consists of three main software components:

- Image processing algorithms that analyze onboard data to detect dynamic events such as volcano eruptions, flooding, lake freeze/thaw, sea-ice breakup, and autonomously request updates to the mission operations plan.
- Robust execution software using the *Spacecraft Command Language* (SCL) to enable spacecraft telemetry processing and event-driven commanding.
- Planning and scheduling software using the *Continuous Activity Scheduling Planning Execution and Re-planning* (CASPER) software to schedule science observations, ground contacts, and re-plan based on requests from onboard image processing.

The image processing algorithms analyzes onboard data to detect dynamic science events. Based on its output, it submits a request to update the mission operations plan. This request can be removing the image from the solid state recorder (SSR) to free up space for acquiring more images, or re-tasking the spacecraft to acquire more images of the target or an adjacent target on subsequent orbits.

The robust execution software (SCL) [5] accepts CASPER planned activities as input and issues individual spacecraft commands. SCL also monitors the state of the spacecraft by maintaining a database of the current spacecraft telemetry. This database is constantly monitored for any deviations during plan execution and SCL can quickly respond to any situations that may endanger the spacecraft.

The planning and scheduling software (CASPER) is a model-based planner that generates detailed mission operations plans from requests provided by the science team or onboard image processing. CASPER represents the operations constraints of the spacecraft in a general modeling language and reasons about these constraints to generate the detailed plan. CASPER also has the capability to continuously monitor the SCL database for any anomalous spacecraft situations and can modify the plan accordingly.

With all flight projects, developing software for space missions introduces many key challenges.

- *Limited communication* – EO-1 has approximately 8 ground contacts a day, each approximately 10-15 minute long with an uplink/downlink rate of 2Kbits/2Mbits. Also, with EO-1 in extended mission, the Mission Operations Control Center (MOCC) is only staffed 12 hours a day. Given these conditions, EO-1 is able to upload 300Kbytes of data a day.
- *Limited observability* – Because processing telemetry is expensive, onboard storage is limited, and downlink bandwidth is limited, engineering telemetry is limited. Thus the ground operations team must be able to operate the spacecraft with limited information. CASPER is allocated 248 bytes of data in its telemetry packet to store all real-time information about the state of the planner and averages an output of 1 packet every 60 seconds.

- *Limited CPU* – The ASE software operates on a MIPS R3000 Mongoose V running at 8MIPS, much slower than typical desktop workstations. Our CPU allocation on EO-1 is 4 MIPS, which is shared among SCL, CASPER, and image processing.
- *Limited memory* – The Mongoose V contains 256MB of RAM, with 32MB allocated to the original flight software. Of the remaining memory, 40MB is allocated for the CASPER code and heap space, with the rest set aside for SCL, image processing, and the filespace
- *Limited filespace* – Developers chose not to include a filesystem as part of the original flight software. ASE requires the use of a filesystem and included an 8MB ramdisk. CASPER was allocated 1MB of filespace for output log files, with the remaining set aside for project files and output files for SCL and onboard image processing.

In the remainder of this paper, we provide a general description the CASPER planner, discuss our methods to reduce the CASPER image size, our strategy for determining what data to store within the engineering telemetry packet, and our approach to performing long term planning.

2 A General Description of CASPER

The spacecraft planning and scheduling process is traditionally performed as one step of ground operations to schedule science observations and downlink opportunities. The output of this process is a detailed sequence of commands to be issued to the spacecraft for execution. In order for the ASE to autonomously satisfy new science requests from image processing algorithms, this step is done onboard by the CASPER [2] planning software. CASPER is able to represent the operations constraints in a generic modeling language and reasons about these constraints to generate a detailed mission operation plan while respecting mission constraints and resources.

CASPER uses a local search [3] approach to develop the detailed operation plan. The main algorithm for planning and scheduling is based on a technique called *iterative repair*. During iterative repair, the conflicts in the plan are detected and resolved one a time, until no conflicts exists. A conflict is considered to be any violation of the mission or spacecraft constraints and resolved through several predefined methods. These methods include: moving, adding, removing, detailing, or abstracting a scheduled operation. The repair algorithm may use any of these methods in an attempt to resolve a conflict.

Repair Algorithm

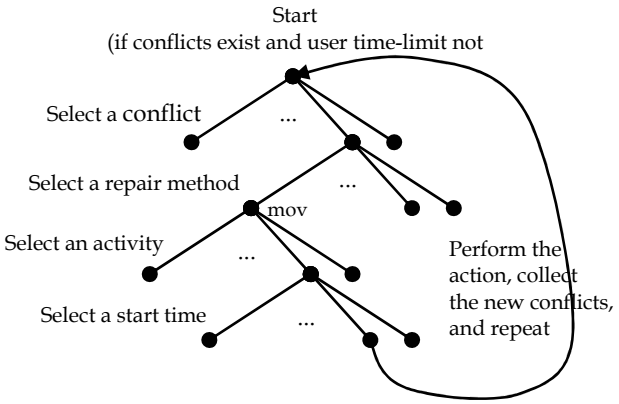

Figure 1 – The main CASPER planning and scheduling algorithm

Here is an example of the repair algorithm after introducing a set of new science observations into the plan:

1. Several conflicts are introduced and recognized by CASPER because the requests do not satisfy all operations constraints.
2. Given the set of the conflicts, CASPER will choose one conflict to resolve. In our example, its will be a resource conflict because the available memory on the solid state recorder (SSR) is oversubscribed.
3. Given the list of repair methods available for this conflict (add an activity replenish the memory, remove a user of the resource, move a user of the resource), CASPER selects move to resolve the conflict.
4. From the list of activities that subscribe to the SSR memory (all science observations), CASPER selects one activity to move.
5. An available start time to resolve this conflict is selected.
6. The science observation is moved to the new start time, resolving the conflict.
7. This algorithm is then repeated until all conflicts within the schedule are resolved.

3 Preparing CASPER for Flight

Given the many challenges to developing flight software, this section discusses several issues encountered in preparing the CASPER planner for flight. Specifically, we describe:

- *Reducing the CASPER image size* – With infrequent and short ground contacts and limited available memory, we needed to reduce the CASPER image size. We discuss our strategies to reduce the CASPER image size.

- *CASPER engineering data* – With limited communication and engineering data, we describe our criteria for selecting points of data to collect.
- *Approach to long term planning* – CASPER must be able to autonomously plan for a week's worth of EO-1 activities, which includes over 100 science observations. We discuss how this is achieved within the available memory and CPU.

3.1 Reducing the CASPER image size

CASPER's core planning engine is the Automated Scheduling and Planning Environment (ASPEN) [3] ground-based planner. ASPEN is a re-usable framework which is capable of supporting a wide variety of planning and scheduling applications. It provides a set of software components commonly found in most planning systems such as: an expressive modeling language, resource management, a temporal reasoning system, and support of a graphical user interface. ASPEN and CASPER is flexible enough to be used for multiple applications such as MAMM [12], MISUS [11], and CLEaR [13]. Though CASPER for ASE requires only a subset of the software components provided, all components are still included as part of the CASPER image.

CASPER developers took two approaches to reducing the image size: removing unneeded components and reducing code image size inefficiencies. A large portion of this work was done for the Three Corner Satellite mission (3CS) [see related work], which had a more constrained RAM allocation. 3CS contained 16MB of RAM, with 8MB allocated for the CASPER code and heap space. Prior to this work, the image size of CASPER was at 12MB.

The CASPER development team went through the core software and removed each software component deemed unnecessary for flight. Several modules removed from the CASPER code include:

- *Backtracking Search* – The ASPEN framework provides several search algorithms that perform backtracking search. On ASE, we have decided to use the repair search algorithm, so these other algorithms are not needed.
- *Optimization* – CASPER provides the capability to optimize the schedule based on several preferences [10] defined by mission planners. However, we have decided not to use this functionality for ASE.
- *GUI Sockets* – Because ASPEN is a ground-based planner, it provides a GUI for visualizing the schedule and interacting with it. Communication with this GUI is done through the ASPEN socket interface. In flight, support for a GUI is not necessary.
- *General Heuristics* – The ASPEN core contains multiple sets of generic heuristics that have been found to be useful across multiple projects. CASPER for ASE requires a subset of these heuristics; therefore, the unused sets can be removed.
- *Generalized Timelines* – Generalized timelines provides a general infrastructure to model complex state variables and resources. This infrastructure is not required for ASE.

Removing software components trimmed approximately 3MB from the CASPER image size, reducing it to 9MB.

CASPER also makes heavy use of the Standard Template Library (STL), specifically the containers provided. STL templates are widely known to increase code size in C++ because for each container defined in CASPER, the code may be duplicated several times. There exist various compiler techniques available that attempts to minimize the duplication. However, CASPER was compiled using the VxWorks 5.3 GNU compiler, the compiler used in the original flight software build, which does not provide any support to reduce duplication. To minimize the impact of code bloat, we re-implemented the STL container and functions used in the CASPER code. This re-implementation, dubbed "lite STL", was developed to minimize the code generation, trading space for execution time. For example, the STL map container uses a red-black tree as the underlying implementation to allow O(log N) lookups, while the "lite" implementation uses lists with a lookup time of O(N). The implementation of red-black tree is more sophisticated than lists, requiring larger amounts of code. This "lite STL" implementation also provided another added benefit; we did not need to modify any of the core software with ASPEN because the API did not change. For projects that operate on desktop workstations, the full STL implementation can be used, while other projects requiring a lightweight version of CASPER can use the "lite STL" implementation. We were able to remove approximately 3MB from the CASPER image using this strategy.

Along with simple compiler optimization, removing unneeded software components, and reducing the impact of code duplication, the final size of the CASPER image was reduced to 5MB.

To improve the time required to uplink the ASE software, given the uplink rate of 2Mbits/sec, we also utilized onboard decompression. We compressed the ASE software on the ground, and uploaded this compressed version to memory. The existing flight software was then patched to perform the decompression routine. This step provided a huge savings in uplink time. The full ASE flight software, originally at 9MB, compressed to a 1.5MB image size. Given our short and infrequent contact times, this reduced the uplink time from an estimated 30 days to approximately 5 days.

3.2 CASPER engineering data

On traditional ground planning systems, there are several ways of collecting data about the state and actions of the planner. Most systems provide a graphical user interface that allows ground personnel to immediately determine conflicts within the schedule, and the disk drive allows for large amounts of data to be stored for review. These are the methods we had used for developing and debugging issues with CASPER.

Collecting data for spacecraft operations is done much differently. ASE has two methods of collecting data: a telemetry packet and log files. Telemetry is output by each subsystem at various frequencies (a packet every 1 to 8 seconds) and provides information about the health and state of the spacecraft. The telemetry values are stored on the spacecraft local recorder and automatically downlinked during each ground contact. While in ground contact, the real-time telemetry data produced by each subsystem is immediately available to the ground operations team, but collected data has approximately a 24 hour turnaround time before it is available. Engineering data is the only method the EO-1 operations team used to collect data on the spacecraft. ASE introduced an 8MB ramdisk into the system as a means of collecting output log files. However, processing and viewing log files introduces extra work for the ground operator. In order to downlink log files from the spacecraft, operators need to specify the file to downlink and initiate the dump at the start of the ground contact. Also, if the file is too large, the operator may need to downlink the file across multiple ground stations and reconstruct it afterwards.

To fit within the framework of normal EO-1 operations and reduce the requirements to ground operators, we decided that engineering data would be the main method of extracting information about the health and status of CASPER. Output log files are still available for downlink, in case an anomalous situation occurs that cannot be explained through the telemetry packet.

There were several objectives in determining what points of data to include in the CASPER telemetry packet. We wanted to be able to identify an anomalous situation within the planner and replay that data to replicate what occurred in flight. This was achieved by partitioning the packet into three sections.

- *Health and Status* – contains a short summary of the planning software status.
- *Autonomous Decisions* – these decisions occur during modifications to the schedule in an iteration of repair.
- *Uncontrollable Inputs* – all un-controlled, un-planned inputs to the planner are logged. With CASPER, these inputs are updates to plan variables that differ from the modeled value.

A maximum of 248 bytes are available in a single telemetry packet. In the latest build of the ASE software, the CASPER telemetry packet is utilizing 224 out of the 248 bytes available. Below, we list out a few of the data points for each section, and provide a brief reason for each. To maximize the packet space, each telemetry point listed below is either 2 or 4 bytes in size.

Health & Status	
Data Point	**Description**
Heartbeat	Up-counter to indicate planner is still active
Errors	Number of errors
Warnings	Number of warnings
Current Stack Usage	Current amount in use from allocated stack space
Maximum Stack Usage	High amount used from allocated stack space
Current Heap Usage	Current amount allocated from heap manager

Table 1 – A sample of the health and status section of the CASPER telemetry packet

The health and status section (see Table 1) of the packet allows us to determine the status of CASPER. The error and warning counters indicates if an unexpected situation occurred within the planning software. The stack usage indicates how much margin exists before overflowing the allotted stack space. The heap usage will indicate if a memory leak is occurring.

Repair Iteration	
Telemetry Point	**Description**
Iteration counter	Number of repair iterations
Success	Indicates if the last plan modification was successful
Seconds Elapsed	Elapsed time for the last iteration
Pre Conflict Count	Number of conflicts prior to plan modification
Post Conflict Count	Number of conflicts after plan modification
Conflict Type	Type of last conflict
Conflict Start Time	Start time of conflict
Conflict End Time	End time of conflict
Resolution Method	Method used to modify the plan
Activity Instance ID	Instance of activity modified
Activity Schema ID	Schema of activity modified
Parameter Schema ID	Activity parameter being modified

Table 2 – A sample of the repair iteration section of the CASPER telemetry packet

The repair iteration section (see Table 2) of the packet contains data about how CASPER modified the plan. It does not contain information on what options were available at each choice point to assist in understanding why decisions were made. For example, it would be useful to collect the list of all conflicts considered in the repair iteration, prior to CASPER's selection. However, the number of conflicts is variable and unbounded. It would be impossible to store an unbounded set of data points within a finite telemetry packet, without creating an artificial upper bound. Instead, the conflict selected is stored to determine what schedule modifications were done. This strategy of selecting what was chosen is done for all choice points. The output log files contain the detailed list of possibilities for each choice point for a repair iteration.

With all autonomous plan modifications logged, we are able to reproduce in our ground testbed what occurred in flight. However, because we are not able to store the full state of CASPER, we assume that it is possible to reproduce the initial conditions of the system. When loading the initial set of goals into the planner, there needs to be the same number of conflicts on our ground testbed as there would be in flight in order to reproduce what occurred.

TimeLine Updates	
Telemetry Point	**Description**
Time Updated	Time plan was updated
Advanced Land Imager (ALI) Cover	The values of each telemetry point indicate how the schedule was updated
ALI Power State	
ALI Data Gate	
Hyperion Power State	
Hyperion Image Mode	
WARP mode	
WARP Free Blocks	
WARP Files In Use	
Downlink Rate	

Table 3 – A sample of the schedule update section of the CASPER telemetry packet

The timeline update section (see Table 3) of the telemetry packet contains all un-modeled updates to the CASPER planner during execution. As activities are inserted into the CASPER schedule, the future values of the spacecraft telemetry are modeled as timelines. Each timeline is constantly monitored to ensure what CASPER modeled is a true reflection of the state of the spacecraft. When the model differs from the spacecraft, updates to the timelines are inserted into the plan.

A common update to the CASPER planner occurs for the number of free memory blocks in the solid state recorder after executing a science observation. Due to scarce computing resources, it is not uncommon for the instruments to collect data several seconds longer than planned, thus consuming more memory blocks. Therefore the spacecraft telemetry value for the number of free memory blocks differs from the value CASPER modeled. CASPER then updates the timeline to the new value.

Several other points of data were considered but omitted from the CASPER telemetry packet due to its limited size and time constraints.

Activity information – activity state information, start time, and unique identifiers are several of the parameters that could be saved in the telemetry packet.

CASPER input commands – the last ground command issued to CASPER can be logged in the packet as verification of receipt. Currently, we are examining the consequences of the command to determine if it was successful received.

Heuristic information – at each decision point in the repair iteration, weighted heuristics are used to select the correct value. Data indicating which heuristics were used would help in determining why decisions were made during repair.

Code execution – within critical areas of the CASPER planning code, saving what section of the code is executing would help in debugging. For example, if CASPER were to enter a section a code and loop forever, we are not currently able to determine where the code is "stuck".

3.3 Approach to long term planning

One of the scenarios planned for ASE is autonomous control of EO-1 for a week. This requires CASPER to support generation of a valid schedule for a week's worth of EO-1 operations. During a nominal week, EO-1 averages over 100 science observations and 50 S-Band/X-Band ground contacts. The size of this problem presents a challenge to CASPER, given the limited memory and CPU constraints.

While most desktop workstations have several GB's of memory available, CASPER on EO-1 is constrained with a 32MB heap. As result, we need to ensure that generation of a week's plan does not exhaust all available heap space. A science observation is the most complex activity within the CASPER model, consisting of over 78 activities. Planning a week's worth of operation would require scheduling over 7800 activities (not including downlink and momentum management activities) and exhaust our heap space.

Also, as the number of goals in the schedule increase, the computation time to schedule a goal will also increase, due to the interaction between goals. On EO-1, this problem is exacerbated with a 8MIPS processor, of which 4MIPS are shared by SCL, CASPER, and science processing.

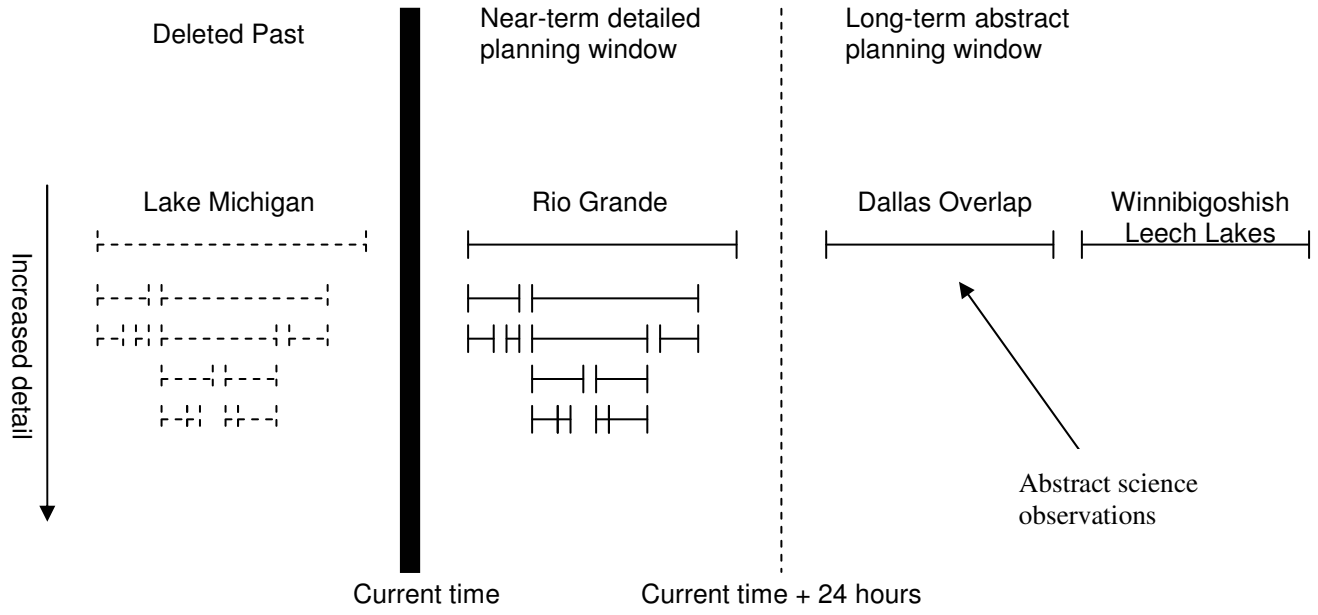

Figure 2 – Strategy for long term planning

To resolve the problems with CPU and memory consumption, CASPER will utilize a hierarchal planning approach with focused planning periods. CASPER will perform abstract planning and scheduling of observations for the entire week, such as ensuring a constraint of one science observation per orbit. It will also perform near-term planning for the next 24 hours by detailing the science observations to the low-level activities. This near-term planning window is continuously updated to include the next 24 hours of the schedule and as past observations exit the planning window, they are automatically removed from the plan. By reducing the number of science observations that need to be scheduled and detailed to a 24 hour period, we reduce memory and CPU consumption.

This strategy has not been implemented for the current version of CASPER operating onboard EO-1, but is being developed for our next ASE software release, where we expect to demonstrate long-term autonomous planning.

4 Flight Status

The ASE software has been steadily progressing to full operations with the major milestones listed in Table 4.

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.

An additional effort includes teaming with the NASA Ames Research Center to fly the Livingstone 2 Mode Identification and Diagnosis software to be added to ASE in the June 2004 timeframe. The Livingstone 2 experiment would demonstrate tracking of multiple fault hypotheses, a capability not demonstrated in the Remote Agent Experiment in 1999. This effort is in earlier stages but is making good progress.

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)

Table 4 – Flight tests of the ASE

5 Related Work

The Three Corner Sat (3CS) University Nanosat mission [1] will be using the CASPER onboard planning software

integrated with SCL and the flight software. Though significantly less complex that EO-1, this mission represented a significant step in preparing CASPER for flight. 3CS required CASPER to fit within a more constrained memory footprint, as only 8MB of RAM was allocated. CASPER was developed to operate under the VxWorks operating system, provide a simplified set of telemetry points, and interfaced with the SCL execution software. The 3CS mission was scheduled to begin in late 2003, but has since been rescheduled for launch on a Delta IV rocket in July 2004.

In 1999, the Remote Agent experiment (RAX) [6] executed for a few days onboard the NASA Deep Space One mission. RAX is an example of a classic three-tiered architecture [7], as is ASE. RAX demonstrated a batch onboard planning capability (as opposed to CASPER's continuous planning) and RAX did not demonstrate onboard science.

6 Summary

The ASE onboard EO-1 is demonstrating several advanced software components capable of autonomously re-tasking the spacecraft to respond to several types of science events. This represents the future of spacecraft missions. We've discussed several issues in developing the onboard planning system, CASPER, given the many challenges imposed in designing flight software.

References

[1] S. Chien, B. Engelhardt, R. Knight, G. Rabideau, R. Sherwood, E. Hansen, A. Ortiviz, C. Wilklow, S. Wichman , "Onboard Autonomy on the Three Corner Sat Mission," Proc i-SAIRAS 2001, Montreal, Canada, June 2001. (also see http://threecs.colorado.edu)

[2] 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. (also see http://casper.jpl.nasa.gov)

[3] 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.

[4] Goddard Space Flight Center, EO-1 Mission page: http://EO-1.gsfc.nasa.gov

[5] Interface and Control Systems, SCL Home Page, interfacecontrol.com

[6] NASA Ames, Remote Agent Experiment Home Page, http://ic.arc.nasa.gov/projects/remote-agent. See also "Remote Agent: To Boldly Go Where No AI System Has Gone Before", Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian Williams. *Artificial Intelligence* 103(1-2):5-48, August 1998

[7] S. Chien, B. Cichy, S. Schaffer, D. Tran, G. Rabideau, R. Bote, Dan Mandl, S. Frye, S. Shulman, J. Van Gaasbeck, D. Boyer, Validating the EO-1 Autonomous Science Agent, Working notes of the Workshop on Safe Agents, AAMAS-2003.

[8] E. Gat et al., Three-Layer Architectures. in D. Kortenkamp et al. eds. *AI and Mobile Robots*. AAAI Press, 1998.

[9] S. Chien, B. Cichy, S. Schaffer, D. Tran, G. Rabideau, R. Bote, D. Mandl, S. Frye, S. Shulman, J. Van Gaasbeck, D. Boyer, Validating the EO-1 Autonomous Science Agent, Working notes of the Workshop on Safe Agents, AAMAS-2003.

[10] G. Rabideau, B. Engelhardt, S. Chien, "Using Generic Preferences to Incrementally Improve Plan Quality," in Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000), Breckenridge, CO. April, 2000.

[11] T. Estlin, T. Mann, A. Gray, G. Rabideau, R. Castano, S. Chien and E. Mjolsness, "An Integrated System for Multi-Rover Scientific Exploration," Sixteenth National Conference of Artificial Intelligence (AAAI-99), Orlando, FL, July 1999.

[12] B. Smith, B. Engelhardt, D. Mutz. "Reducing Costs of the Modified Antarctic Mapping Mission through Automated Planning", Fourth International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations, 2001.

[13] F. Fisher, M. James, L. Paal, B. Engelhardt, "An Architecture for an Autonomous Ground Station Controller," Proceedings of the IEEE Aerospace Conference, Big Sky, MT, March 2001.

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.