Safe Agents in Space: Lessons from the Autonomous Sciencecraft Experiment

Rob Sherwood, Steve Chien, Daniel Tran, Benjamin Cichy,
Rebecca Castano, Ashley Davies, and Gregg Rabideau

Jet Propulsion Laboratory, California Institute of Technology,
4800 Oak Grove Dr., Pasadena, CA 91109
{firstname.lastname@jpl.nasa.gov}

Abstract. An Autonomous Science Agent is currently flying onboard the Earth Observing One Spacecraft. This software enables the spacecraft to autonomously detect and respond to science events occurring on the Earth. The package includes software systems that perform science data analysis, deliberative planning, and run-time robust execution. Because of the deployment to a remote spacecraft, this Autonomous Science Agent has stringent constraints of autonomy and limited computing resources. We describe these constraints and how they are reflected in our agent architecture.

1 Introduction

The Autonomous Sciencecraft Experiment (ASE) has been flying autonomous agent software on the Earth Observing One (EO-1) spacecraft [5] since January 2004. This software demonstrates several integrated autonomy technologies to enable autonomous science. Several algorithms are used to analyze remote sensing imagery onboard in order to detect the occurrence of science events. These algorithms will be used to downlink science data only on change, and will detect features of scientific interest such as volcanic eruptions, flooding, ice breakup, and presence of cloud cover. The results of these onboard science algorithms are inputs to onboard planning software that then modify the spacecraft observation plan to capture high value science events. This new observation plan is then be executed by a robust goal and task oriented execution system, able to adjust the plan to succeed despite run-time anomalies and uncertainties. Together these technologies enable autonomous goal-directed exploration and data acquisition to maximize science return.

The ASE onboard flight software includes several autonomy software components:

- Onboard science algorithms that analyze the image data to detect trigger conditions such as science events, “interesting” features, changes relative to previous observations, and cloud detection for onboard image masking
- Robust execution management software using the Spacecraft Command Language (SCL) [6] package to enable event-driven processing and low-level autonomy

G.I. Webb and Xinghuo Yu (Eds.): AI 2004, LNAI 3339, pp. 51–63, 2004.

- The Continuous Activity Scheduling Planning Execution and Replanning (CASPER) [3] software that modifies the current spacecraft activity plan based on science observations in the previous orbit cycles

The onboard science algorithms analyze the images to extract static features and detect changes relative to previous observations. Several algorithms have already been demonstrated on EO-1 Hyperion data to automatically identify regions of interest including land, ice, snow, water, and thermally hot areas. Repeat imagery using these algorithms can detect regions of change (such as flooding and ice melt) as well as regions of activity (such as lava flows). We have been using these algorithms onboard to enable retargeting and search, e.g., retargeting the instrument on a subsequent orbit cycle to identify and capture the full extent of a flood. Although the ASE software is running on the Earth observing spacecraft EO-1, the long term goal is to use this software on future interplanetary space missions. For these missions, onboard science analysis will enable data be captured at the finest time-scales without overwhelming onboard memory or downlink capacities by varying the data collection rate on the fly.

The CASPER planning software generates a mission operations plan from goals provided by the onboard science analysis module. The model-based planning algorithms will enable rapid response to a wide range of operations scenarios based on a deep model of spacecraft constraints, including faster recovery from spacecraft anomalies. The onboard planner accepts as inputs the science and engineering goals and ensures high-level goal-oriented behavior.

The robust execution system (SCL) accepts the CASPER-derived plan as an input and expands the plan into low-level spacecraft commands. SCL monitors the execution of the plan and has the flexibility and knowledge to perform event-driven commanding to enable local improvements in execution as well as local responses to anomalies.

1.1 Typical ASE Scenario

A typical ASE scenario involves monitoring of active volcano regions such as Mt. Etna in Italy. (See Figure 1.) Scientists have used data from the Hyperion instrument onboard the EO-1 spacecraft for ground-based studies of this volcano. The ASE concept will be applied as follows:

1. Initially, ASE has a list of science targets to monitor that have been sent as high-level goals from the ground.
2. As part of normal operations, the CASPER planning software generates a plan to monitor the targets on this list by periodically imaging them with the Hyperion instrument. For volcanic studies, the infrared and near infrared bands are used.
3. During execution of this plan, the EO-1 spacecraft images Mt. Etna with the Hyperion instrument.
4. The onboard science algorithms analyze the image and detect a fresh lava flow. If new activity is detected, a science goal is generated to continue monitoring the volcanic site. If no activity is observed, the image is not downlinked.

5. Assuming a new goal is generated, CASPER plans to acquire a further image of the ongoing volcanic activity.
6. The SCL software executes the CASPER generated plan to re-image the site.
7. This cycle is then repeated on subsequent observations.

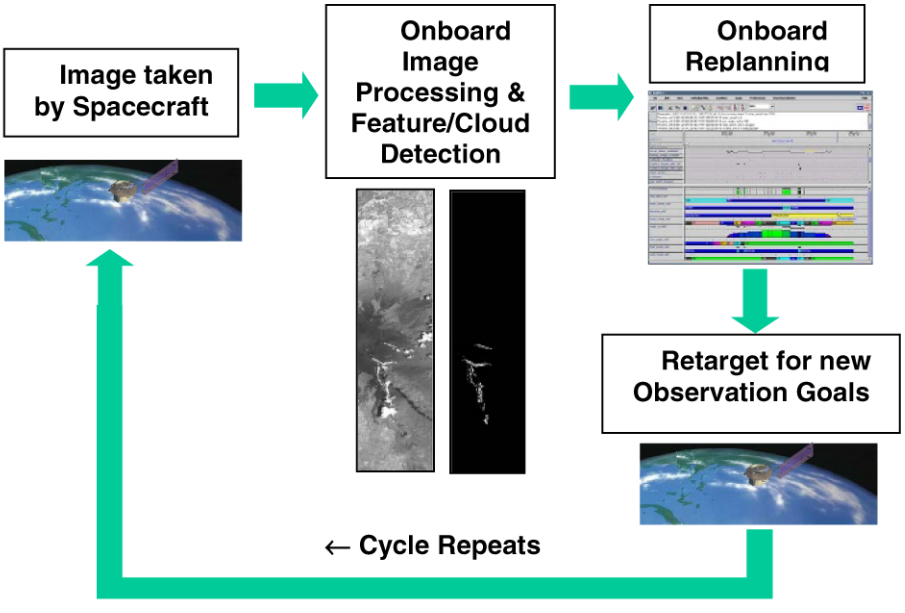

Fig. 1. Autonomous Science Scenario

1.2 Space-Based Autonomy Software Challenges

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. Some examples include:

1. Limited, intermittent communications to the agent. A typical spacecraft may communicate several times a day (Earth orbiter) to once or twice per week (planetary spacecraft).
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).
3. Limited observability. Because processing telemetry is expensive, onboard storage is limited, downlink bandwidth is limited, and engineering telemetry is limited. Thus onboard software must be able to make decisions based 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. On 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 spacecraft is 4 MIPS and 128MB RAM.

In the remainder of this paper we describe the ASE software architecture and components. We then discuss the issues of adapting the ASE software agent for space flight.

2 The EO-1 Mission

EO-1 was launched on November 21, 2000. EO-1 has 2 imaging instruments. Over 20-Gbits of data from the Advanced Land Imager (ALI) and Hyperion instruments are collected and stored for each image taken.

The EO-1 spacecraft has two Mongoose M5 processors. The first M5 CPU is used for the EO-1 spacecraft control functions. The secondary M5 CPU is a controller for the large mass storage device. Each M5 runs at 12 MHz (for ~8 MIPS) and has 256 MB RAM. Both M5's run the VxWorks operating system. The ASE software operates on the secondary CPU. This provides an added level of safety for the spacecraft since the ASE software does not run on the main spacecraft processor.

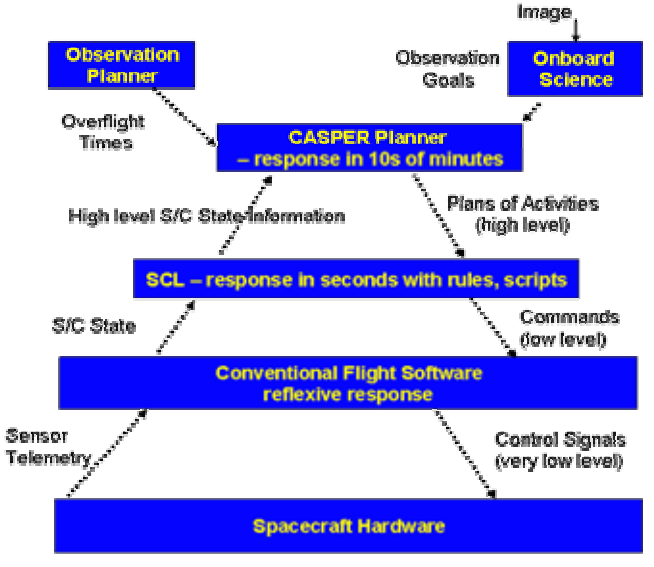

Fig. 2. Autonomy Software Architecture

3 Autonomy Software Architecture

The autonomy software on EO-1 is organized into a traditional three-layer architecture [4] (See Figure 2.). At the highest level of abstraction, the Continuous

Activity Scheduling Planning Execution and Replanning (CASPER) software is responsible for mission planning functions. CASPER schedules science activities while respecting spacecraft operations and resource constraints. The duration of the planning process is on the order of tens of minutes. CASPER scheduled activities are inputs to the Spacecraft Command Language (SCL) system, which generates the detailed sequence commands corresponding to CASPER scheduled activities. SCL operates on the several second timescale. Below SCL, the EO-1 flight software is responsible for lower level control of the spacecraft and also operates a full layer of independent fault protection. The interface from SCL to the EO-1 flight software is at the same level as ground generated command sequences. The science analysis software is scheduled by CASPER and executed by SCL in a batch mode. The results from the science analysis software result in new observation requests presented to the CASPER system for integration in the mission plan.

This layered architecture was chosen for two principal reasons:

1. The layered architecture enables separation of responses based on timescale and most appropriate representation. The flight software level must implement control loops and fault protection and respond very rapidly and is thus directly coded in C. SCL must respond quickly (in seconds) and perform many procedural actions. Hence SCL uses as its core representation scripts, rules, and database records. CASPER must reason about longer term operations, state, and resource constraints. Because of its time latency, it can afford to use a mostly declarative artificial intelligence planner/scheduler representation.
2. The layered architecture enables redundant implementation of critical functions – most notable spacecraft safety constraint checking. In the design of our spacecraft agent model, we implemented spacecraft safety constraints in all levels where feasible.

Each of the software modules operates at a separate VxWorks operating system priority. The tasks are shown below in Table 1 in decreasing priority. The ASE to flight software bridge is the task responsible for reading the real-time flight software telemetry stream, extracting pertinent data, and making it accessible to the remainder of the ASE software. The Band Stripping task reads the science data from the onboard solid state recorder and extracts a small portion of the science data (12 bands of Hyperion data) to RAM. The science analysis software then operates on the extracted data to detect science events.

Table 1. EO-1 Software Tasks in Decreasing Task Priority (e.g. upper tasks have highest priority for CPU)

Set of Tasks	Rationale for Priority
EO-1 Flight Software	Required for WARP hardware safety
ASE to FSW Bridge	Required to keep up with telemetry stream
Band Stripping	Utilizes WARP hardware while running
SCL	Lowest level autonomy, closes tightest loops
CASPER	Responds in tens of minutes timescale
Science Analysis	Batch process without hard deadlines

It is worth noting that our agent architecture is designed to scale to multiple agents. Agents communicate at either the planner level (via goals) or the execution level (to coordinate execution).

We now describe each of the architectural components of our architecture in further detail.

4 Onboard Mission Planning

In order for the spacecraft to respond autonomously to a science event, it must be able to independently perform the mission planning function. This requires software that can model all spacecraft and mission constraints. The CASPER [3] software performs this function for ASE. CASPER represents the operations constraints in a general modeling language and reasons about these constraints to generate new operations plans that respect spacecraft and mission constraints and resources. CASPER uses a local search approach [2] to develop operations plans.

Because onboard computing resources are scarce, CASPER must be very efficient in generating plans. While a typical desktop or laptop PC may have 2000-3000 MIPS performance, 5-20 MIPS is more typical onboard a spacecraft. In the case of EO-1, the Mongoose V CPU has approximately 8 MIPS. Of the three software packages, CASPER is by far the most computationally intensive. For that reason, our optimization efforts were focused on CASPER. Careful engineering and modeling were required to enable CASPER to build a plan in tens of minutes on the relatively slow CPU.

CASPER is responsible for long-term mission planning in response to both science goals derived onboard as well as anomalies. In this role, CASPER must plan and schedule activities to achieve science and engineering goals while respecting resource and other spacecraft operations constraints. For example, when acquiring an initial image, a volcanic event is detected. This event may warrant a high priority request for a subsequent image of the target to study the evolving phenomena. In this case, CASPER will modify the operations plan to include the necessary activities to re-image. This may include determining the next over flight opportunity, ensuring that the spacecraft is pointed appropriately, that sufficient power and data storage are available, that appropriate calibration images are acquired, and that the instrument is properly prepared for the data acquisition.

In the context of ASE, CASPER reasons about the majority of spacecraft operations constraints directly in its modeling language. However, there are a few notable exceptions. First, the over flight constraints are calculated using ground-based orbit analysis tools. The over flight opportunities and pointing required for all targets of interest are uploaded as a table and utilized by CASPER to plan. Second, the ground operations team will initially perform management of the momentum of the reaction wheels for the EO-1 spacecraft. This is because of the complexity of the momentum management process caused by the EO-1 configuration of three reaction wheels rather than four.

5 Onboard Robust Execution

ASE uses the Spacecraft Command Language (SCL) [6] to provide robust execution. SCL is a software package that integrates procedural programming with a real-time, forward-chaining, rule-based system. A publish/subscribe software bus allows the distribution of notification and request messages to integrate SCL with other onboard software. This design enables both loose or tight coupling between SCL and other flight software as appropriate.

The SCL "smart" executive supports the command and control function. Users can define scripts in an English-like manner. Compiled on the ground, those scripts can be dynamically loaded onboard and executed at an absolute or relative time. Ground-based absolute time script scheduling is equivalent to the traditional procedural approach to spacecraft operations based on time. SCL scripts are planned and scheduled by the CASPER onboard planner. The science analysis algorithms and SCL work in a cooperative manner to generate new goals for CASPER. These goals are sent as messages on the software bus.

Many aspects of autonomy are implemented in SCL. For example, SCL implements many constraint checks that are redundant with those in the EO-1 fault protection software. Before SCL sends each command to the EO-1 command processor, it undergoes a series of constraint checks to ensure that it is a valid command. Any pre-requisite states required by the command are checked (such as the communications system being in the correct mode to accept a command). SCL will also verify that there is sufficient power so that the command does not trigger a low bus voltage condition and that there is sufficient energy in the battery. Using SCL to check these constraints (while included in the CASPER model) provides an additional level of safety to the autonomy flight software.

6 Preparing the CASPER Planner for Space 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.
- *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.
- *Speed Improvements to Meet Autonomy Requirements* – Several model and code optimizations were performed to increase the running speed of ASE.

In addition, we have performed several optimizations on the data collected relating to the state and actions of the planner. These optimizations are not described in this paper but can be referenced here [11].

6.1 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. Because of limited onboard computing memory, we had to reduce the image size. CASPER developers took two approaches to reducing the image size: removing unneeded components and reducing code image size inefficiencies. Prior to this work, the image size of CASPER was 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 were 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 were removed.
- *Generalized Timelines* – Generalized timelines provides a general infrastructure to model complex state variables and resources. This infrastructure was not required for ASE and was removed.

Removing software components trimmed approximately 3MB from the CASPER image size.

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. 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. 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.

6.2 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 an 8 MIPS processor, of which 4 MIPS are shared by the SCL, CASPER, and science processing software.

To resolve the problems with CPU and memory consumption, CASPER utilizes a hierarchal planning approach with focused planning periods. CASPER performs abstract planning and scheduling of observations for the entire week, such as ensuring a constraint of one science observation per orbit. It also performs 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.

6.3 Speed Improvements to Meet Autonomy Requirements

The ASE experiment is constrained by the computing environment onboard EO-1. Because each of the EO-1 software builds is a single static image, all ASE components that dynamically allocate RAM require their own memory manager. SCL contains a memory manager previously used on the FUSE mission. CASPER uses a separate memory manager adapted from JPL's Deep Impact mission. However, performance from early flight tests indicated that the SCL memory manager was significantly hampering performance, so SCL was switched to use the same memory manager as CASPER (but with its own heap space). Note that these memory managers had to not only allocate and de-allocate memory quickly but also not suffer from longer term issues such as fragmentation.

The limited onboard computing power required changes to the SCL and CASPER models to meet operational timing constraints. For example, initially within SCL a much larger set of safety constraints was modeled and execution was designed to be much more closed loop. However, testbed runs and early flight tests indicated that telemetry delays and CPU bottlenecks meant that this design was delaying time-sensitive commands. Most importantly, instrument on-times were delayed (e.g. late) and too long (resulting in extra data acquired). The ASE team was forced to both streamline the code (including the memory manager modification) and streamline the model to speed execution.

The CASPER planner is a significant user of onboard CPU. When CASPER is planning future observations it utilizes all of the available CPU cycles and takes approximately 8 minutes to plan each observation. The CASPER model was designed to operate within a minimal CPU profile – and as a result observations are planned with less flexibility. By setting fixed values for temporal offsets between activities rather than retaining flexible offset times, search is reduced and response time improved at the cost of plan quality (in some cases). For example, an image take activity may require a camera heater warm up before the camera can operate. The heater may take 30-60 seconds to warm the camera up to its operational temperature. By setting the duration of the heater warm up activity to 60 seconds, the temporal offset between the heater warm up activity and the image data take activity is fixed at 60 seconds, rather than variable.

Other performance improvements for CASPER came from analysis of the running code. We found bottlenecks and made improvements in redundant calculations. In particular, this was critical for functions performed on every loop of CASPER (such as collecting conflicts). We made some simplifying assumptions to make some expensive calculations faster. For example, when initially scheduling activities, we ignore timeline constraints, assuming that temporal constraints are more critical than timelines (calculating valid start times for timelines can be expensive).

7 Flight Status

The ASE software has been flying onboard the EO-1 spacecraft since January 2004. We have steadily increased the level of autonomy during this period. In April 2004, we started the first closed-loop execution where ASE autonomously analyzes science data onboard and triggers subsequent observations. Since that time, we have run over 20 of these trigger experiments with over 100 autonomously planned image data takes. Our most recent focus has been to expand the duration of the tests until ASE is controlling the satellite for 7 days straight. This will involve over 100 image data takes and over 50 ground contacts.

8 Related Work and Summary

In 1999, the Remote Agent experiment (RAX) [8] executed for a few days onboard the NASA Deep Space One mission. RAX is an example of a classic three-tiered

architecture [5], as is ASE. RAX demonstrated a batch onboard planning capability (as opposed to CASPER's continuous planning) and RAX did not demonstrate onboard science. PROBA [9] is a European Space Agency (ESA) mission demonstrates onboard autonomy and launched in 2001. However, ASE has more of a focus on model-based autonomy than PROBA.

The Three Corner Sat (3CS) University Nanosat mission will be using the CASPER onboard planning software integrated with the SCL ground and flight execution software [1]. The 3CS mission is scheduled for launch on a Delta IV rocket July 3, 2004. The 3CS autonomy software includes onboard science data validation, replanning, robust execution, and multiple model-based anomaly detection. The 3CS mission is considerably less complex than EO-1 but still represents an important step in the integration and flight of onboard autonomy software.

More recent work from NASA Ames Research Center is focused on building the IDEA planning and execution architecture [7]. In IDEA, the planner and execution software are combined into a "reactive planner" and operate using the same domain model. A single planning and execution model can simplify validation, which is a difficult problem for autonomous systems. For EO-1, the CASPER planner and SCL executive use separate models. While this has the advantage of the flexibility of both procedural and declarative representations, a single model would be easier to validate. We have designed the CASPER modeling language to be used by domain experts, thus not requiring planning experts. Our use of SCL is similar to the "plan runner" in IDEA but SCL encodes more intelligence. For example, the plan runner in IDEA does not perform safety checks or have knowledge about how long to retry execution in the event of a failure. The EO-1 science analysis software is defined as one of the "controlling systems" in IDEA. In the IDEA architecture, a communications wrapper is used to send messages between the agents, similar to the software bus in EO-1. In the description of IDEA there is no information about the deployment of IDEA to any domains, so a comparison of the performance or capabilities is not possible at this time. In many ways IDEA represents a more AI-centric architecture with declarative modeling at its core and ASE represents more of an evolutionary engineered solution.

Using ASE on longer-range interplanetary missions would require more autonomy to deal with the longer gaps between communications periods. This would require higher fidelity models and more computing power. The current generation of radiation hardened spacecraft computers such as the PowerPC 750 provides more than adequate power to run the ASE software.

We have outlined several constraints on spacecraft autonomy software involving limited CPU and memory resources, strict timing for spacecraft control, and spacecraft safety. We have also described how we addressed those constraints through several optimizations we have performed on the ASE. These have included removing unnecessary code within the planner, changing memory managers, performing planner and executive model optimizations, and optimizing the running code. We have also devised a strategy for long term planning using very limited memory and CPU. In addition, we described our use of a three-layer autonomy architecture to increase the safety and performance of the ASE software. Specifically,

the three-layer architecture offers specific advantages for this application by allowing redundant safety checks at each layer, and allow the 3 layers to respond on appropriate time scales for spacecraft operations.

ASE on EO-1 demonstrates an integrated autonomous mission using onboard science analysis, planning, and robust execution. The ASE performs intelligent science data selection that will lead to a reduction in data downlink. In addition, the ASE will increase science return through autonomous retargeting. Demonstration of these capabilities onboard EO-1 will enable radically different missions with significant onboard decision-making leading to novel science opportunities. The paradigm shift toward highly autonomous spacecraft will enable future NASA missions to achieve significantly greater science returns with reduced risk and reduced operations cost.

Acknowledgement

Portions of this work were performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We would like to acknowledge the important contributions of Nghia Tang and Michael Burl of JPL, Dan Mandl, Stuart Frye, Seth Shulman, and Stephen Ungar of GSFC, Jerry Hengemihle and Bruce Trout of Microtel LLC, Jeff D'Agostino of the Hammers Corp., Robert Bote of Honeywell Corp., Jim Van Gaasbeck and Darrell Boyer of ICS, Michael Griffin and Hsiao-hua Burke of MIT Lincoln Labs, Ronald Greeley, Thomas Doggett, and Kevin Williams of ASU, and Victor Baker and James Dohm of University of Arizona.

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.
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 casper.jpl.nasa.gov)
3. 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)
4. E. Gat et al., Three-Layer Architectures. in D. Kortenkamp et al. eds. AI and Mobile Robots. AAAI Press, 1998.
5. Goddard Space Flight Center, EO-1 Mission page: http://EO-1.gsfc.nasa.gov
6. Interface and Control Systems, SCL Home Page, sclrules.com
7. N. Muscettola, G. Dorais, C. Fry, R. Levinson, and C. Plaunt, "IDEA: Planning at the Core of Autonomous Reactive Agents," Proceedings of the Workshops at the AIPS-2002 Conference, Tolouse, France, April 2002.

8. 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
9. The PROBA Onboard Autonomy Platform, http://www.estec.esa.nl/proba/
10. G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," Intl Symp Artificial Int Robotics & Automation in Space, Noordwijk, The Netherlands, June 1999.
11. D. Tran, S. Chien, G. Rabideau, B. Cichy, Flight Software Issues in Onboard Automated Planning: Lessons Learned on EO-1, Proceedings of the International Workshop on Planning and Scheduling for Space (IWPSS 2004). Darmstadt, Germany. June 2004