JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 6, March 2009

Goal-Based Operations: An Overview

Daniel D. Dvorak* , Michel D. Ingham† , and J. Richard Morris‡
NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 91109

and

John Gersh§
Applied Physics Laboratory, Johns Hopkins University, Laurel, Maryland, 20723

DOI: 10.2514/1.36314

Operating robotic space missions via time-based command sequences has become a limiting factor in the exploration, defense, and commercial sectors. Command sequencing was originally designed for comparatively simple and predictable missions, with safe-mode responses for most faults. This approach has been increasingly strained to accommodate today's more complex missions, which require advanced capabilities such as autonomous fault diagnosis and response, vehicle mobility with hazard avoidance, opportunistic science observations, etc. Goal-based operation changes the fundamental basis of operations from imperative command sequences to declarative specifications of operational intent and termed goals. Execution based on explicit intent simplifies operator workload by focusing on what to do rather than how to do it. The move toward goal-based operations, which has already begun in some space missions, involves changes and opportunities in several places: operational processes and tools, human interface design, planning and scheduling, control architecture, fault protection, and verification and validation. Further, the need for future interoperation among multiple goal-based systems suggests that attention be given to areas for standardization. This overview paper defines the concept of goal-based operations, reviews a history of steps in this direction, and discusses the areas of change and opportunity through comparison with the prevalent operational paradigm of command sequencing.

I. Introduction

SPACE missions have traditionally been operated using time-based command sequencing, where commands are planned to be executed at prescribed instants in time, and where telemetry is returned for operators to determine if the planned activities were accomplished. This approach has worked well in missions that have a manageable number of spacecraft states and transitions for operators to reason about, that interact with relatively predictable environments, and that can resort to safe-mode to await operator intervention when unexpected behavior occurs. These characteristics have been true of many missions from the earliest days of robotic space exploration, including orbiters, planetary fly-bys, and stationary landers.

Received 21 December 2007; revision received 11 June 2008; accepted for publication 26 November 2008.

* Principal Engineer, Planning & Execution Systems Section, M/S 301-270, Daniel.L.Dvorak@jpl.nasa.gov, AIAA Member.
† Senior Software Systems Engineer, Flight Software Systems Engineering and Architectures Group, M/S 301-225, Michel.D. Ingham@jpl.nasa.gov, AIAA Senior Member.
‡ Software and Systems Engineer, Planning and Sequencing Systems Group, M/S 301-250D, John.R.Morris@jpl.nasa.gov
§ Principal Engineer, System and Information Sciences Group M/S 2-236, John.Gersh@jhuapl.edu, AIAA Member.

As missions have become ever more ambitious, the limitations of traditional mission operations have become more apparent. One limiting factor is complexity and workload. Preparation of command sequences and interpretation of telemetry are time-consuming processes, especially for the most hazardous phases of a mission. Human operators cannot keep a sufficiently detailed mental model of all the states and transitions safely and repeatedly to command a vehicle, so operations must move to higher levels of abstraction, leaving more details to automation. A second limiting factor lies in the assumption of predictability inherent in command sequencing. Some missions are already operating in less predictable environments, such as Mars surface explorations, and some missions cannot return up-to-the-moment telemetry to operators, either because of long light-time communication delays or infrequent communication schedules. A third limiting factor is the slow reaction time to interesting events or changes. Missions that perform scientific monitoring and military reconnaissance must capture and react to short-lived events—opportunities that would otherwise be lost while waiting for human-in-the-loop analysis and control. A fourth limiting factor exists in flight control architectures for sequence execution. "Sequencers," as they are called, do not inherently support the kinds of temporal flexibility and autonomous decision-making that are needed.

The answer to these challenges clearly suggests the need for more automation in ground systems and more autonomy in flight systems, but is that all that is needed? Is command sequencing still an appropriate operational paradigm for these more ambitious missions, or is there a need for—perhaps even a movement toward—a new operational paradigm? In this paper we claim that there *is* a more suitable paradigm, termed "goal-based operations", that there is already movement in that direction and that it differs in a fundamental way from command sequencing.

The key principle behind this evolution is the specification of operator intent. *Goal-based operation* addresses the above-mentioned limitations by *making operator intent explicit* and carrying it into uplink products. Providing a system with a specification of intent endows it with the capacity to check for successful accomplishment of objectives, and to use alternative methods to achieve objectives if necessary (see Fig. 1). Goal-based operation simplifies operator workload by allowing them to focus on *what* objectives the spacecraft should be achieving, rather than *how* it should be achieving them (command sequences). Goals also allow flexibility in the ordering and timing of activities, enabling event-driven execution: an operating paradigm of increasing use in modern embedded systems, particularly those that must operate robustly in unpredictable environments and in the absence of real-time communication with human operators. Goals facilitate adjustable levels of autonomy and a spectrum of fault responses short of simply entering "safe mode".

Importantly, goal-based operation facilitates in situ decisions needed for mission scenarios such as reconnaissance and exploration, making it easier to autonomously re-task assets in response to local observations. The Autonomous Sciencecraft Experiment (ASE) [1–3] on the Earth Observing 1 (EO-1) spacecraft provides an example of such capability. As a result of onboard image processing, ASE re-targets EO-1 on subsequent orbits based upon changes such as flooding, ice melt, and lava flows. EO-1 has been operating autonomously since November of 2004. Goal-driven

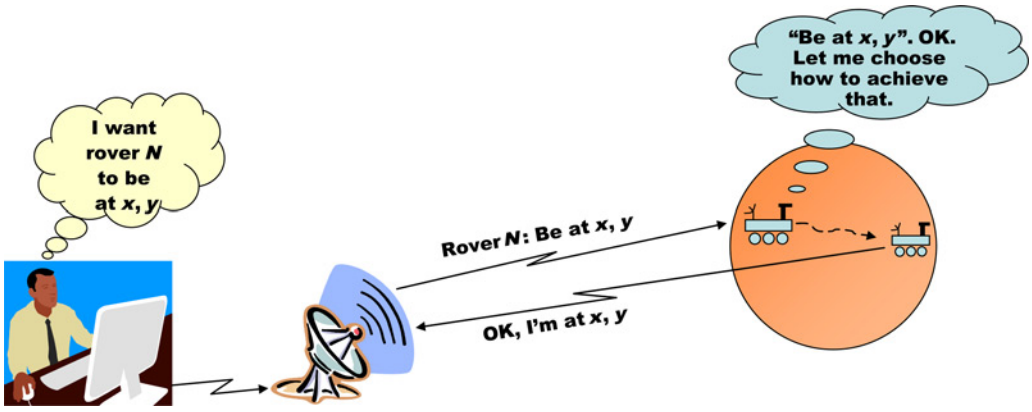

Fig. 1 Goal-based operations elevate control to the level of operator intent. A goal specifies a state to be achieved in the system under control; it specifies what to do, not how to do it, thus reducing operator workload and leaving options open for the control system.

operation supports the trend of increasing functionality in flight systems, with more components designed to achieve higher-level objectives and respond appropriately to off-nominal or opportunistic events, rather than simply execute timed commands.

Aspects of goal-based operations are already in use in a variety of systems, sometimes under different names such as "task-based commanding" or "activity-based operation", but mostly implemented in ad hoc ways. An important next step in this evolution of operations will be a general control architecture that (a) makes operational intent explicit in the form of *goals*; (b) manages interactions among multiple activities; and (c) establishes a uniform operations approach across the breadth of capabilities, including navigation, attitude control, observing, resource management, fault protection, and coordination of multiple assets.

Although goal-based operation is presented in this paper from the perspective of robotic space missions, the concepts apply equally to Earth-bound robots such as unmanned aerial vehicles (UAVs), autonomous ground vehicles (as in the DARPA Grand Challenge), unmanned underwater vehicles (UUVs), and control of industrial processes.

This paper presents a brief history of goal-based operations in unmanned space missions, describes how the concept affects different phases of the engineering lifecycle, and examines a representative software architecture for a control system designed for goal-based operation. The paper also explores how a goal-based approach affects operations process, tools and workflow, and then describes how operators can interact with a goal-driven system in terms of different views. Finally, the paper addresses concerns about determinism, reliability, V&V, and impact on established systems.

II. A Brief History of Goal-based Operations

The idea of operating systems at the level of explicit intent is not a completely novel concept—for example, thermostats have been used to control the temperature of building interiors for over a hundred years. The thermostat's set point is a simple form of goal—it specifies the desired temperature that the building's heating, ventilating, and air conditioning control system must achieve and maintain. In the context of space exploration, goals have actually been used for decades in limited fashion, particularly in the context of spacecraft attitude and articulation control systems, for the purposes of pointing science instruments, communication antennae, solar panels, etc. Vehicle reorientation and gimbal angles are "commanded" by specifying trajectories of desired angles and rotation rates; these state trajectories are explicit representations of intent. Until recently, however, such representations have not been used consistently across all spacecraft subsystems, and have not been integrated into coherent system-level control architectures and operations processes. This section provides a brief history of goal-based operations (GBO), highlighting a number of significant achievements from the space exploration domain.

One of the first full-scale (system-level) applications of goal-based spacecraft operations was the Remote Agent (RA) Experiment [4], which was flight-validated in 1999 on the Deep Space One (DS-1) spacecraft, the first deep space mission in NASA's New Millennium Program. RA is a model-based, reusable, artificial intelligence (AI) software system that enables goal-based spacecraft commanding and robust fault recovery. A simplified view of the RA software architecture is shown in Fig. 2. RA consists of general-purpose reasoning engines (both deductive and procedural) and mission-specific domain models. One of its key characteristics—and a main difference with traditional spacecraft commanding—is that ground operators can communicate with RA using goals (e.g., "During the next week take pictures of the following asteroids and thrust 90% of the time") rather than with detailed sequences of timed commands. RA determines a plan of action that achieves those goals; actions are represented as tasks that are decomposed on-the-fly into more detailed tasks and, eventually, into commands to the underlying flight software. The RA Experiment provided an invaluable proof-of-concept and lessons learned in a number of areas, including benefits and challenges associated with autonomous goal-based operations. These lessons have been documented in the Remote Agent Experiment DS-1 Technology Validation Report [5].

NASA's Mars Exploration Rovers [6] (MER), Spirit and Opportunity, also employ a certain degree of GBO capability, in both the ground system and onboard the rovers. In the ground system, operators use the Mixed-initiative Activity Plan GENerator (MAPGEN [7,8]) tool to plan each rover's science and engineering activities on a sol-by-sol basis. Given a set of user observation goals and their priorities, this tool enables operators to construct a plan that satisfies these goals and schedule the activities in the plan such that conflicts between incompatible activities and oversubscription of limited resources are avoided. MAPGEN leverages the automated planning and scheduling engine that was flight-validated as part of RA on DS-1, integrating it into a GUI environment that enables operators to

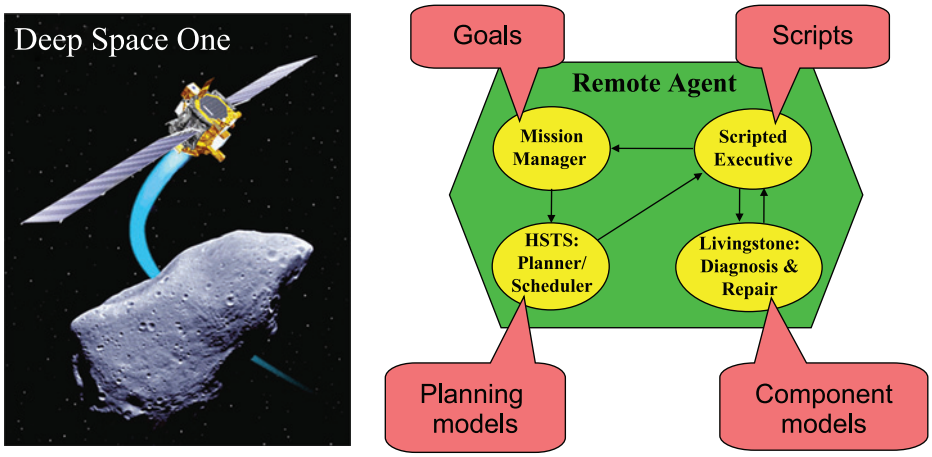

Fig. 2 Remote Agent Experiment on DS-1. The Remote Agent experiment demonstrated goal-based operations onboard the DS-1 spacecraft in 1999. In this architecture, the Mission Manager sends high-level goals to the Planner/Scheduler which then reasons through declarative planning models to generate detailed task plans consisting of lower-level goals. The Executive executes the task plans by invoking scripts associated with the lower-level goals, issuing commands as needed. The Diagnosis & Repair engine reasons through component models of the system to provide the Executive with estimates of the spacecraft state, and guidance on how to achieve the lower-level goals.

incrementally build and edit their plans. With this tool, a plan is refined through iterations of automated computation and judicious hand editing based on domain expertise, eventually converging to a final plan that the operator finds appropriate. Onboard each rover, the flight software is programmed to accept a combination of abstract goal-like directives, such as "drive to waypoint," and lower-level commands. In a remote and unpredictable environment such as the Martian surface, the rovers robustly achieve their ambitious science objectives by taking advantage of their ability to make certain decisions in-situ, and execute flexibly specified plans in an event-driven fashion. These are fundamental characteristics of goal-based systems.

The most recent and comprehensive space-based application of goal-based operations is the ASE [1–3]. ASE is an autonomous software agent currently flying onboard the EO-1 spacecraft, which has demonstrated several integrated autonomy technologies that together enable science-directed autonomous operations. The ASE software includes onboard continuous planning, robust task and goal-based execution, and onboard machine learning and pattern recognition. It has also more recently been augmented to demonstrate model-based diagnosis capabilities with RA heritage. Like RA, ASE began as a technology experiment within NASA's New Millennium Program, as part of the Space Technology 6 project. Early tests had the goal-directed planning and execution capabilities deployed as part of a ground-based sequencing system; the success of these tests built up confidence in the technology in preparation for ultimate deployment of the capabilities onboard the spacecraft. The technology was declared fully validated in May 2004. The ASE software now runs full-time onboard the EO-1 satellite, and has become its primary mission planning and execution system. Through automation of the operations process, ASE has contributed operational savings of approximately $1M per year, compared with EO-1's nominal operations cost before ASE deployment [3]. It has resulted in dramatic increases in science return, thanks to its intelligent downlink selection and autonomous retargeting capabilities, and increased flexibility in operations, thanks to the resulting streamlining of human-operator-in-the-loop activities. Another long-term benefit of the ASE project is documentation of the lessons learned [2], which will certainly be invaluable to future applications of onboard autonomy and GBO.

Not surprisingly, NASA is not alone in its desire to exploit the benefits of spacecraft autonomy and goal-based operations. In 2001, the European Space Agency (ESA) launched its first Project for Onboard Autonomy [9] (PROBA-1) spacecraft. The PROBA-1 technology validation mission successfully demonstrated both onboard and ground-based automation, including the ability to convey high-level goals (user requests) to the spacecraft via the internet. ESA is also investigating the use of GBO and onboard planning and scheduling for ExoMars [10], a Mars rover anticipated to be the first flagship mission in ESA's Aurora Exploration Programme.

Although this paper's focus is on the use of GBO of spacecraft, this approach has broad applicability to other domains, such as industrial robot control and autonomous unmanned air/underwater/ground vehicles. For example, the Defense Advanced Research Projects Agency (DARPA) has sponsored Grand Challenges, which have stimulated the development of various goal-directed planning and execution techniques and technologies. More broadly, GBO is the focus of much research and development in academic, governmental and industrial organizations.

III. Goal-Based Operations across the Project Lifecycle

Although the term "operations" refers specifically to the final phase of a space mission project, the concept of goal-based operations shapes engineering activities across the entire engineering lifecycle of a project (see Fig. 3), as described below.

Pre-Phase A (Advanced Studies) and Phase A (Mission and System Definition):

- Establish high-level mission objectives, which may be expressed as goals.
- Build up (goal-based) scenarios for accomplishing the high-level objectives.
- Develop initial Concept of Operations.

Phase B (Preliminary Design):

- Identify key goal types (parametric specifications) and key goal expansions or elaborations.
- Flesh out Concept of Operations and goal-based operations process.
- Identify goal-based operations tools that must be adapted for the mission.

Phase C (Design & Build):

- Define full set of goal types.
- Define full set of goal elaborations or expansions.
- Define logic needed for scheduling.
- Design & implement achievement software (flight & ground).
- Design & implement goal-based operations tools.

Phase D (Assembly Test and Launch Operations) and Phase E (Operations):

- Instantiate, elaborate, schedule and execute goals needed to achieve mission objectives.
- May define additional goals for system check-out beyond the set of operational goals.
- Update goal-based operations tools and achievement software as necessary.

IV. Software Architecture

Goal-based operations applies to a large class of systems that can collectively be referred to as robots, both mobile robots (spacecraft, surface rovers, humanoid robots, etc) and immobile robots (building environmental control systems, chemical processing control systems, power generation systems, etc). In all cases the purpose is control of a physical system to achieve specified objectives, and the complexity of those systems warrants careful attention to software architecture for monitoring and control. Indeed, there has been a lot of relevant architectural work in the overlapping fields of space systems, robotics, autonomous systems, and industrial process control. A general

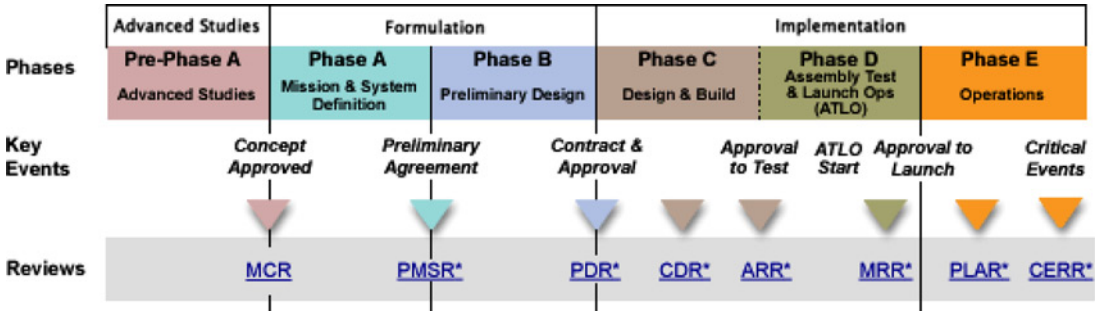

Fig. 3 Project lifecycle phases. The concept of a "goal" shapes engineering activities beginning in Pre-Phase A, where high-level objectives may be stated as goals and mission scenarios may be specified as sequences of goals, to Phase E, where goals are instantiated, elaborated, scheduled, and executed.

consensus has emerged around a three-layer architecture [11] consisting of reactive control at the lowest layer, plan execution in the middle, and deliberative planning in the top layer. This section overviews one such architecture specifically designed for goal-based operations, namely, the Mission Data System (MDS). This section describes MDS in terms of architectural concepts, with some attention to goal representation and processing mechanisms, but leaves many details to other papers [12–14].

A. Goal as a Specification of Intent

As stated earlier, a goal is a specification of operational intent, meaning that it specifies *what* we want a system to do. Although a command sequence tells a system what to do in terms of time-based commands, the sequence doesn't knowingly achieve operational intent. For example, a command sequence may close a switch at 1:00 pm and open it at 2:00 pm, but if the switch spontaneously opens at 1:30 pm, the operational intent may not be achieved, and operators won't be aware of that unless they check telemetry. Because we want to use goals in the real world where things don't always go as desired, a goal must have a success criterion that can be checked during execution. A goal-driven control system has a contract to achieve (or maintain) some condition, or report that it has failed to do so. This is the architectural concept of *cognizant failure* [15,16].

All of this raises the question of how to represent intent. Because we are going to use goals to operate a system, goals must have clear semantics that can be processed by a computer. By itself, this requirement would allow a kind of goal such as "execute procedure X with parameters a, b, and c, and return success or failure", but such goals are specific to the design of a control system, with no broader semantics. With goal-based operation we want to also use goals for interoperation and coordination of a system of systems, with elements potentially built by different teams, so goal semantics should be independent of control system design. Specifications of intent should come naturally from the problem domain and should have obvious meaning to systems engineers and operators. In this vein, "intent" must be about the *system under control (including its environment)*, not the control system. For example, a goal may specify a desired spacecraft attitude because that is a constraint on a state of the system under control, but a goal must not specify a mode of an attitude controller because that is an implementation-specific state of a controller in a control system.

The MDS control architecture defines a goal as a constraint on the value history of a state variable during a time interval, as depicted in Fig. 4. The term "state variable" refers to an element of the control system that corresponds to a physical state of the system under control. For example, a power switch in the system under control will physically be in an opened state, closed state or tripped state, and the control system will use evidence such as sensor measurements and issued commands to estimate the state of the switch as opened, closed, or tripped. A goal on the state of the switch is specified as a constraint on the switch's estimated state. For example, a goal could specify that the value of the state variable must be 'closed' continuously from 2:00 pm to 3:00 pm. A different goal could require that the switch be closed 90% of the time between 2:00 pm and 3:00 pm, thus tolerating temporary switch state transitions. In all cases, a goal specifies a requirement on a state history that must be satisfied. If the goal's constraint is violated, the control system detects the violation and handles it in ways to be described later.

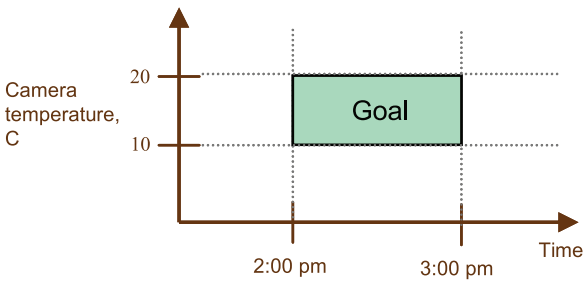

Fig. 4 Goal as a constraint on state and time. A goal represents operational intent as a constraint on the values of a state variable during a time interval, such as "Camera temperature is between 10°C and 20°C from 2:00 PM to 3:00 PM". As such, a goal can be visualized as a region of acceptable behavior in value and time, as depicted by the box.

Goals can be specified on uncontrollable states. For example, the light level from the Sun is not controllable, but activities that depend on some minimum light level—such as picture-taking—can include a goal on light level. Such a goal will not be ready to start until the estimated light level satisfies the goal's constraint. When such a goal is coupled appropriately with dependent goals, then the starting or ending time of whole activities can be based on such conditions. In this sense goals represent requirements, and forward progress in execution can be conditioned on events.

Goals can also encode flight rules that may be imposed throughout one or more phases of a mission. For example, an operational constraint like "The temperature of the science camera shall never exceed 30°C" can be included in the planning process to ensure that the mission's planned activities are projected to satisfy the flight rule. Furthermore, this goal can be loaded onboard the spacecraft, actively to monitor for its violation during execution and trigger an appropriate fault response.

B. State Variables and Their Timelines

The preceding definition of "goal" raises the question "Can all types of operational intent really be specified as constraints on the values of state variables?" We believe the answer is "yes" given that the whole purpose of operations is to change the state of a system under control in specified ways. Examples of physical states include device status (configuration, temperature, operating modes), dynamic states (vehicle position and attitude, gimbal angles, wheel rotations), resources (power and energy, propellant, data storage, bandwidth), and data (science observations, engineering measurements). Given the wide variety of kinds of state variables, goals can represent "low-level" objectives, as seen with the power switch goal, and they can also represent "high-level" objectives that drive an entire phase of a mission, such as "spacecraft has landed on Mars within ellipse x by time t." This high-level goal will get elaborated into a myriad of supporting goals, in a manner to be described later. Note that this "high-level" goal still fundamentally expresses a constraint on state—in this case, the position of the spacecraft with respect to some specified Mars surface-fixed reference frame.

As mentioned earlier, every state variable in the control system corresponds to a physical state in the system under control. Each state variable contains two timelines: an *intent timeline* that holds the goals, ordered according to time, and a *knowledge timeline* that holds estimated state up to the present time (see Fig. 5). During execution a goal that extends from time t_1 to time t_2 succeeds if the estimated state history between t_1 and t_2 satisfies the goal's constraint. Suitable displays of these timelines enable an operator to examine the past, present, and future, and see how the system under control behaved, insofar as it can be estimated from available evidence.

An important aspect of the state knowledge timeline is that estimates contain not only the 'best estimated value' of the physical state but also some representation of the amount of uncertainty. This inclusion of uncertainty is important for robust control because it acknowledges that sensors and actuators are imperfect, as are our models of devices and the environment. If the evidence available to a state estimator is noisy or conflicting or unavailable, then the uncertainty of its estimate must be correspondingly higher. This aspect of estimate uncertainty is important in two ways: it allows an estimator to be honest about the evidence and not pretend that its estimates are facts, and it

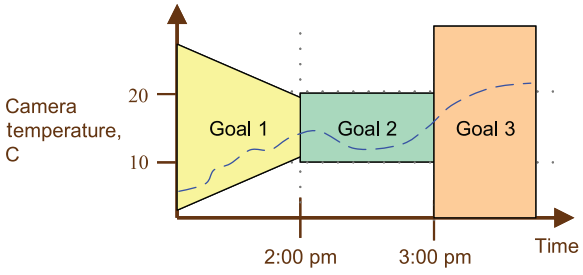

Fig. 5 Intent and knowledge timelines. A state variable contains two timelines. The intent timeline holds goals, ordered according to time, as depicted by the colored polygons. The knowledge timeline holds estimated state values, as depicted by the dashed blue line. A goal succeeds if its constraint is satisfied by the estimated state history during its time interval.

enables a controller to exercise caution when its control decisions depend on highly uncertain estimates. If an activity depends on some minimum level of certainty of state estimates, then a goal can specify such a constraint because uncertainty is part of the value representation of every state variable. We refer to such goals as *knowledge goals*.

C. Layered Control Architecture

A canonical architecture has emerged in the research community based on the notion of layered control loops that address control problems at various timescales and level of abstraction [11]. The layers run concurrently and asynchronously to provide a combination of responsiveness and robustness. Figure 6 shows the MDS architecture consisting of four layers: Control, Execution, Planning, and Presentation. Note that the Presentation layer includes human operators and their decision-making *as part of* the control system. It is here that operational intent is first specified in the form of goals and it is here that the longest control loops are closed.

Goals can exist at very different levels of abstraction, as discussed in the previous section. Goals also address control problems at various timescales, meaning that some goals have relatively short durations and/or stringent requirements on starting or ending times, while other goals have relatively long durations and/or flexible starting and ending times. As shown in Fig. 6, the lowest layer—the Control Layer—provides reactive control with real-time responsiveness while the highest automated layer—the Planning Layer—generates globally consistent plans in the form of intent timelines populated with goals, ready for execution. The Execution layer executes the current plan, determining when to advance to the next goal on each intent timeline, consistent with temporal constraints. The Presentation Layer provides the displays and control interfaces used by operators, where the highest level

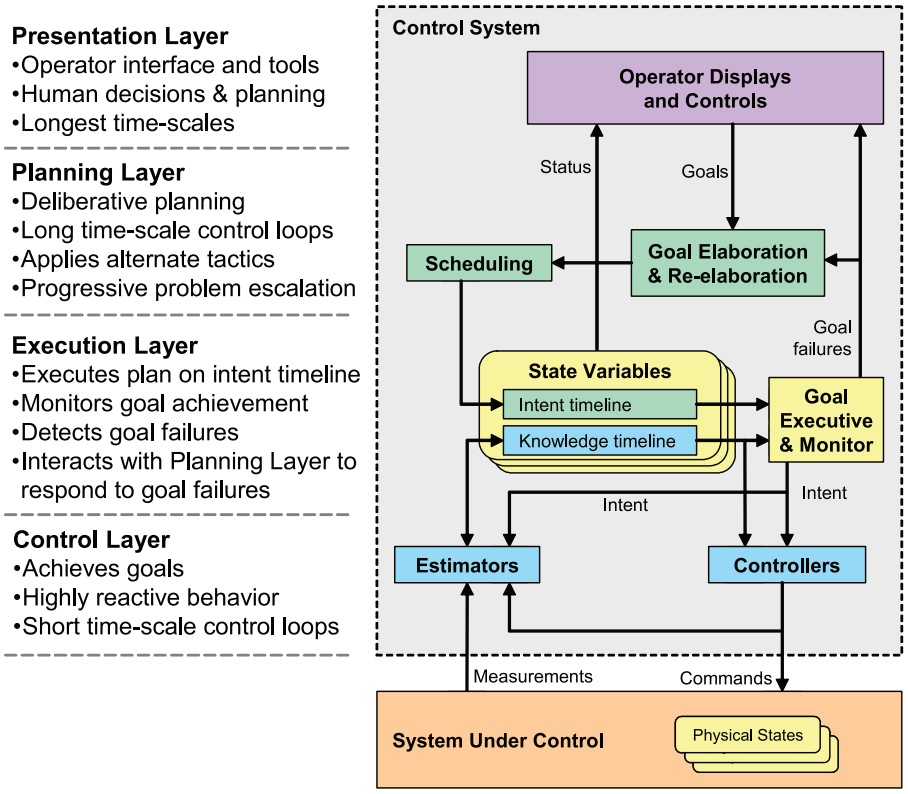

Fig. 6 Software architecture layers. All layers operate asynchronously and concurrently to achieve goals. The Control Layer operates on short time-scales, reacting to local observations, while the Planning Layer operates on longer time-scales, coordinating system-wide behavior, including system-level fault responses. State variables in the control system correspond to physical states in the system under control.

decision-making occurs—keeping in mind, however, that operators can control (i.e., specify goals) at any level of abstraction, as discussed above.

As a goal is being executed in the Control Layer, its status is independently monitored in the Execution Layer by evaluating its constraint against state estimates on the associated knowledge timeline. If the goal is still satisfiable then execution proceeds normally, but if not, the Goal Monitor reports the failed goal to the Planning Layer while the Control Layer continues to try to achieve the now-failed goal. The Planning Layer responds to the goal failure in one of several ways, as discussed below in subsection F.

D. Goal Elaborations and the State Effects Model

A goal that is directly executable by the Control Layer can simply be submitted by an operator and the Control Layer will try to achieve it. However, few goals can be achieved in isolation without considering their implications on other related state variables of the system, and many goals simply cannot be achieved without the support of additional goals on other state variables that have an effect on the state variable of the primary goal. For example, spacecraft attitude has implications on antenna pointing and camera pointing, so a goal on the spacecraft attitude state variable has effects that might be inconsistent with goals on these *affected* state variables. Likewise, a goal on spacecraft attitude can only be achieved if goals on *affecting* state variables are achieved or maintained, such as adequate power and propellant, warm catalyst beds for the thrusters, and healthy inertial measurement units.

The engineering knowledge of what state variables affect other state variables in the system under control is captured in a *state effects model*, a sketch of which is shown in Fig. 7; this knowledge guides the design of fundamental "blocks" of goals that can be assembled into plans that respect the causality among state variables in the system under control. These fundamental blocks, called goal elaborations, specify supporting goals on related state variables that need to be satisfied to achieve the original goal, or make the original goal more likely to succeed. Each type of goal has an associated elaborator that generates its supporting goals (if any), and those supporting goals may themselves have elaborations with supporting goals, so goal elaboration is a hierarchical process that finishes when no more goals have elaborations (see Fig. 8). The design of goal elaborations is based on the state effects model and the application of a few rules [14]. The resulting hierarchy is encoded in "parent/child" relationships in the goals, providing full traceability up and down the elaborated goal "tree".

Because there may be more than one way to achieve a goal, an elaborator may contain multiple tactics. These alternate tactics may be explored during initial planning and scheduling and also in response to goal failures.

E. Goal Network and Temporal Constraint Network

To achieve coordinated control, goals must be organized so that temporal dependencies are honored and that incompatible goals are not scheduled for concurrent execution. The structure in which a coordinated schedule of goals is arranged is called a *goal network*, consisting of goals situated in a temporal constraint network. The goal network is a product of fully elaborating the operator-specified goals, as described above, and scheduling the goals

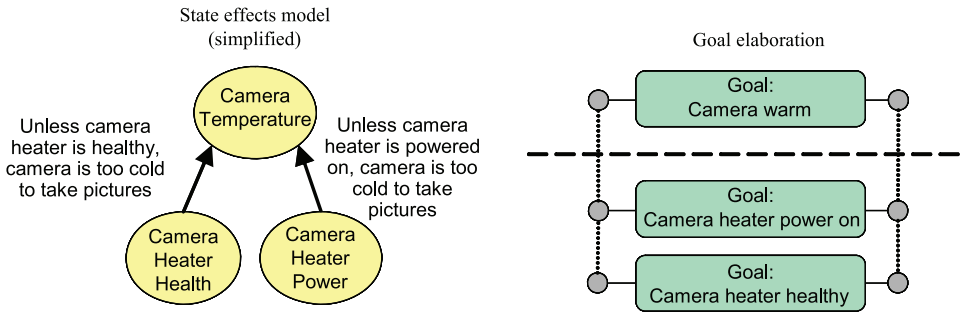

Fig. 7 State effects model and goal elaboration. A state effects model, produced during analysis of the system under control, describes effects among physical state variables. A goal elaboration, produced during operations engineering, shows a goal above the dashed line and its supporting goals below the line, reflecting the state-to-state effects that must be managed to achieve the original goal.

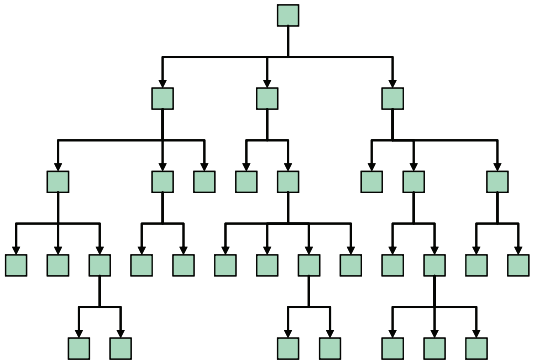

Fig. 8 Hierarchical goal elaboration. The end result of goal elaboration is a tree of supporting goals with the "leaf goals" being goals that have no further elaboration. The goals still need to be scheduled into a plan (a goal network) and then executed. All of the goals in the tree—not just the leaf goals—are monitored during execution, because each one represents intent with respect to a particular state variable.

onto the appropriate state intent timelines. Every goal has a starting and ending time point, and goals that must begin or end simultaneously share the same time point. Every time point has a time window which designates the earliest possible and latest possible times for the time point to fire during execution. This window—which may be as small as an instant of time—is determined by the temporal constraints in the network. A temporal constraint specifies minimum and maximum time duration between two time points. A temporal propagation algorithm updates the time windows of time points as temporal constraints are added or modified during scheduling and as time points fire during execution. Figure 9 shows a simple goal network.

F. Plan Execution

The Execution Layer is responsible for supervising execution of the current plan, as represented in a goal network, including monitoring the status of every active goal. The intent timelines contain the goals to be achieved, ordered according to temporal constraints, so the job of the Goal Executive is to keep the goal achievers (estimators and controllers) in the Control Layer informed as to what goals to execute. As time proceeds, the Goal Executive marches down the intent timelines, directing the Control Layer. (Some goal-based operations systems, such as the CASPER planning and execution system in the ASE software [1] on EO-1, employ a *continuous planning* paradigm, which enables better performance by defining a "planning horizon", deferring detailed planning of distant future activities, and incrementally replanning as new activities fall within this horizon. CASPER also employs a "commitment window", which precludes the scheduling of new goals in the very near term, and prevents existing goals that are

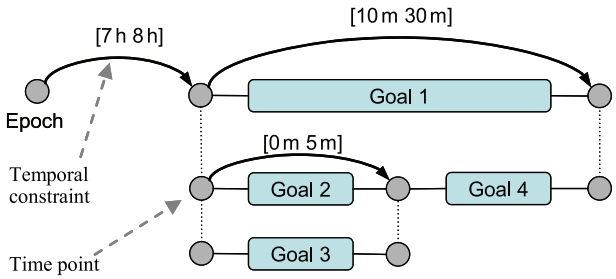

Fig. 9 Example goal network. A goal network consists of goals situated within a temporal constraint network, where the starting time of a goal is governed by the time point attached on its left side, and the ending time by the time point attached on its right side. Time points connected by a dotted line represent the same time point, meaning that multiple goals may start or end simultaneously. The Epoch is a special singleton time point designating a predefined reference instant in time. Temporal constraints of the form [Min Max] can be specified between time points.

to be executed in the near term from being modified by the scheduling process. Such features are planned to be incorporated into the MDS architecture, as well.)

At the same time, the Goal Monitor is checking each active goal to see if its constraint can still be satisfied. At any time, if the estimated state history on a knowledge timeline violates the active state constraint on the intent timeline, then the Goal Monitor reports the goal failure to the Planning Layer (to determine if a recovery is possible, e.g., by re-elaborating goals into alternate tactics) and also reports it to the Presentation Layer to keep operators informed. Of course, in the case of space missions with intermittent communication or long one-way light-time delays, such notification to operators may be delayed.

The term *goal failure* conforms to a common definition of failure as a departure from intended operation. As such, it is not a diagnosis; it simply reports that an objective has been violated without explaining why. The possible reasons for failure depend on the goal, but can be divided among four categories: hardware failure, unexpected environmental state, inaccurate model of the system under control, or software defect. These categories, of course, are independent of the control architecture and are addressed through verification, validation, and operations activities. In the case of hardware failure (or failure of the system under control), failures would be diagnosed by an appropriate estimator in the Control Layer (e.g., for the health state variable associated with the failed device), thus a goal on that health state variable would presumably fail (possibly along with any other goals on state variables related to the operation of that device). Environmental states that impact the operation of the system are considered part of the system under control, and are thus estimated in the Control Layer. Goals are used to specify constraints on such state variables; an environmental state value that violates the expectation expressed in the goal would be detected by the Goal Monitor as a goal failure. Inability of a controller effectively to achieve a goal (either owing to a software bug in the code, or an error in the model that was used to design the controller) would also result in failure of the goal. So in all cases, goal monitoring is the key mechanism for reacting to off-nominal situations.

A number of different responses to goal failure are allowed [14], including:

- simply removing the goal (and all of its supporting goals) and continuing execution of the remaining goals in the network; failed goals may be placed into a holding bin for later re-elaboration and rescheduling;
- triggering the re-elaboration and rescheduling of the "parent" goal of the failed goal;
- propagating the failure up the elaboration tree by failing the "parent" goal of the failed goal; or
- safing the spacecraft, in the case of failure of a goal that jeopardizes mission success or safety.

In addition, the failure of a goal is normally reported via telemetry.

G. Unified Flight-Ground Architecture

Although flight software and ground software operate in very different environments, they must operate together as a system, so they must share some architectural concepts, even if their respective implementations differ. Goal-based operation permits considerable freedom as to how much control is performed in flight versus ground. In one extreme, Ground could perform all the functions of goal elaboration and scheduling, and then transmit the resulting goal network to a Flight system that contains only the Execution Layer and Control Layer. In the other extreme, an operator on the Ground could issue a single top-level goal that is transmitted to a Flight system where it is elaborated, scheduled and executed without human oversight. More typically, the right balance will be somewhere in between, with a mix of high-level elaboration and scheduling on the Ground, and lower-level elaboration and scheduling on the Flight system. Also, the balance may change in either direction over the course of a mission. For example, during the earliest stages of a mission the Ground operators may choose to operate a spacecraft with goals that are entirely elaborated and scheduled on the Ground, relying on a Flight safe-mode response in the event of a goal failure. As the behavior of the spacecraft becomes better known, operators might begin to migrate some elaborations to the Flight system to reduce their workload or reduce communications requirements. In the event of a Flight anomaly that requires expert analysis, operators might then revert back to detailed control from Earth. The key point is that using the same architecture in Flight and Ground gives a mission considerable operational flexibility.

H. Verification and Validation

A state/model/goal-based architecture as described herein holds several benefits in system-level verification and validation (V&V). First, the scheduling process (described briefly above and more thoroughly in other publications [13,14]) represents an initial validation of the plan against the state effects model to determine whether the goals we

are asking the system to accomplish are achievable. Specifically, the scheduling algorithm checks for temporal inconsistencies (e.g., A before B, B before C, C before A) and state incompatibilities (e.g., resource over-commitments) in the goal network. Second, the control system is self-checking in that it continuously monitors the status of every executing goal. Thus, any behavior visible in a knowledge timeline that violates an executing goal will be detected and will trigger a goal failure response. This automatic checking is certainly preferable to any post-processing of telemetry, and is valuable not only during operations but also during testing. Third, there is a direct correspondence between physical states and state variables in the software. That means that when running the control system against a simulated system under control, it is possible to make a direct comparison between estimated state and true (simulated) state. Such comparisons simplify the validation of estimators and controllers. Fourth, an architecture such as MDS lends itself to implementation as a set of generic software frameworks, which, once validated, can be confidently reused from mission to mission. The software V&V challenge is thus reduced to validating the "adaptation" (i.e., mission-specific) code, and the "integrated" (e.g., subsystem- and system-level) software behaviors. Finally, the explicit use of models—particularly the state effects model and the strongly related goal elaborations—provide for direct inspection/validation by domain experts and also enable application of model-checking tools [17]. Of course, this does not absolve the system engineers from having to ensure the correctness and completeness of the state effects models used to design the control system.

V. Operations Process, Tools, and Workflow

Ambitious mission objectives project complexity onto operations. The complexity of mission operations can be quantified by the number of system states an operator is required to keep track of. To reduce this number, in the face of increasingly complex missions, the system must allow the operator to focus at a higher level of abstraction. This concept is not exactly new. In traditional systems, operators typically do not begin the uplink process by sitting down at a terminal and beginning to type in commands. They begin by formulating high-level objectives which are then translated into high-level activities or observations, at an appropriate level of abstraction such that operators can agree on scope and general timing. However, the uplink product is a collection of commands specified at roughly the same low level of abstraction or, at best, sequences which group commands with a similar objective. These command-based sequences are constructed with the intent that they will achieve the original higher-level objectives. One stark limitation of this approach is the decomposition of activities into commands. As complexity increases and the hierarchy grows, the question must arise "where do we draw the line between activities and their translation into commands?" This question ignores the issues inherent in translation, such as how to ensure that information is not lost. Given that concern, we are compelled to ask the more fundamental questions, "Why draw the line at all?" and "Why require a translation from one form to another?" The intent inherent in goals can be specified at any level of detail. For example, one could specify a goal for a rover to be at position (x, y, z), or for a switch to be closed. This flexibility allows a goal network, in the degenerate case, to completely mimic a command-based sequence, but with the additional benefits of an arbitrarily deep hierarchy and a seamless decomposition throughout.

Even in the extreme case where a goal-based system has been restricted such that no choices are given it with respect to the building and execution of the goal hierarchy, operators still benefit from the specification of intent. In the review portion of the typical uplink process, the reverse translation from commands to intent is an ad hoc procedure which must be performed continually by the operator and, more often than not, only inside their own mind. The goal elaboration hierarchy documents this translation in a formal, inspectable and—if we employ an architecture such as MDS—verifiable manner, because the decomposition is directly informed by models of the system under control. The plan can now be checked against an actual design document, thus creating a direct verifiable link between design and operations. Furthermore, this link between design and operations travels through the flight software, because software components are built directly from, and operate directly on, the very same models of the system under control. This enables operators to more easily understand the flight software and how the system will interpret the products that the operators uplink, thus mitigating both the need for resident flight software experts on the operations team and the increased risk associated with workforce turnover. As discussed above, goals can represent not only activities, but also resources and flight rules within the same hierarchy, thus allowing operators and tools to use the models to trace resource violations back to the source.

In addition to allowing uplink operators to inspect high-level intent with respect to low-level control objectives, goals and goal hierarchies allow downlink operators and system analysts to inspect the up-linked intent with respect

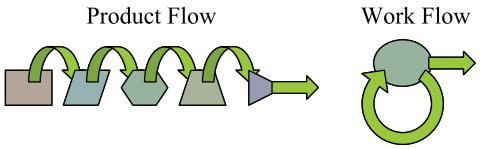

Fig. 10 Product flow vs. work flow. Goal-based operations facilitate a shift from a product flow-oriented process to a work flow paradigm.

to telemetry and actual results, in context. Furthermore, the state-based nature of telemetry in a goal-based system allows operators direct and/or automated comparison of plans to results, even if such checks are not employed in the flight system. Goals allow a system to close the loop across the entire operations process, directly relating uplink products to downlink telemetry.

Discrepancy in form at different levels in the hierarchy also places limitations on operations tools and processes. Typically, operations tools are not built to digest both activities and sequences, resulting in the use of multiple tools in the refinement of plans. This necessitates a *product flow* paradigm, where products progress from one tool to the next through the exchange of data files. Reversing the flow through this process is awkward at best, and is usually avoided. Instead, fixes are often made in place without the benefit of the functionality of preceding tools or, owing to restrictions on the duration of the process, activities are simply shed and science opportunities are lost. Goals facilitate a shift to a *work flow* paradigm (Fig. 10), where one product set managed by a common tool undergoes successive stages of refinement, resulting in a more reversible, collaborative, and seamless process.

VI. Operator Interaction with Goal-Based Systems

Because the objective of GBO is to enable control by the specification of operator intent, user interaction with the ground/flight system inherently involves mechanisms for them to express their intentions, to visualize those intentions, and, most important, to understand how those intentions are (or are not) being carried out *in context.* Displays in the Presentation Layer need to provide information about what is planned, what is happening, and what has happened at various levels of abstraction. In particular, users will specify what they want accomplished expressed as high-level goals, perhaps partially elaborated into a network of subgoals, sequenced as an overall plan for mission activity. Information depictions for mission operations will depict the goals and their state of accomplishment in the context of the operational environment that the mission is actually encountering.

It is well recognized that goal and plan depiction will be necessary for operations at levels above command sequencing [18]. To this point, however, operational systems that deal with goals [1,7] still, in general, depict only sequences of activities over time, without indication of *why* those activities are being carried out at various levels of abstraction. Without depiction of higher-level goals and of their elaborations, mission operators will not have the information necessary to decide whether planned goals are still appropriate in dynamic environments or whether the elaborated tactics of supporting goals remain effective.

One approach is to represent mission objectives, high-level goals, and lower-level implementations as an abstraction hierarchy [19,20]. Looking upward in such a hierarchy helps understand *why* something is being planned or specified; looking downward helps understand *how* it is being accomplished. Displays can be organized by relationships in the hierarchy to facilitate answering these questions. In general, goals and supporting goals form a complex network, not a strict hierarchy, which adds challenges to user interaction design.

In general, one can consider three categories of views to provide context for GBO. A *temporal* view depicts autonomous activities, goal accomplishment, environmental events and system responses over time, at various levels of abstraction. Figure 11 is a temporal view of rule-based autonomy activity in a Solar-Terrestrial Relations Observatory (STEREO) spacecraft [20]. In a goal-based system rather than the rule-based system shown here, goals and supporting goals would be depicted in the timeline. Current planning systems typically provide such a depiction, but without further structural or operational context.

A *structural* view depicts relationships among goals, activities, and resources in hierarchies of abstraction and dependency. Figure 12 shows a concept for a structural view, indicating the complex set of relationships among goals

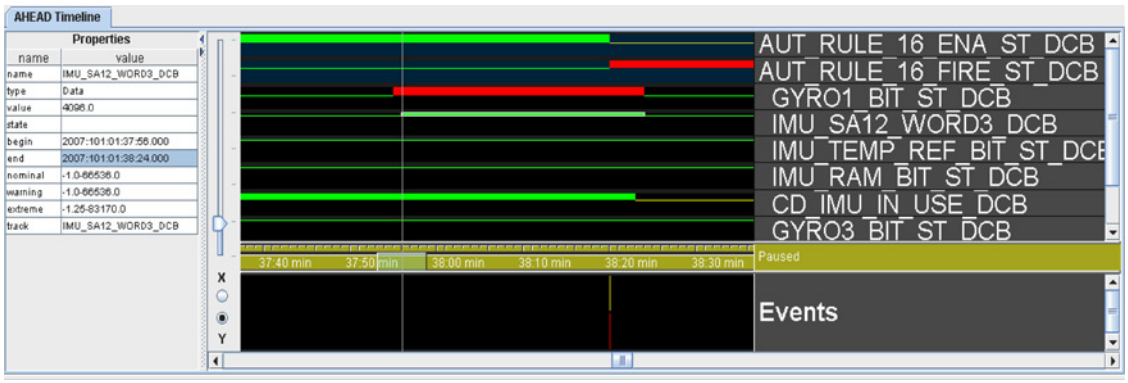

Fig. 11 Temporal view of autonomous activity in a STEREO spacecraft. A temporal view can show autonomous activities, goal accomplishment, environmental events and system responses over time, at selected levels of abstraction.

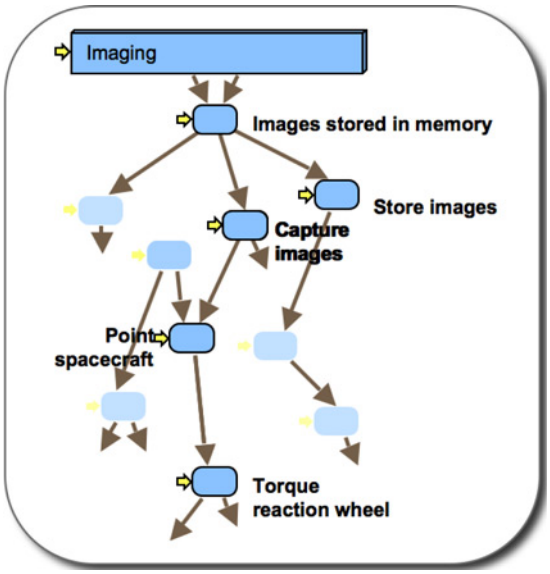

Fig. 12 Structural view concept. This view shows relationships among goals at different levels of abstraction and dependency. From the perspective of a single goal, looking upward explains why it exists and looking downward explains how it will be accomplished.

and supporting goals for an autonomous science activity. It could be based on the goal elaboration hierarchy, but may also be organized according to an abstraction hierarchy and labeled in functional terms.

An *operational* view depicts goals, activities, and events in whatever context is appropriate to the system involved: landscapes for rovers, system diagrams for spacecraft health and housekeeping, planetary surfaces for science observation, and so forth. Figure 13 shows a display from the planning tool for the CRISM (Compact Reconnaissance Imaging Spectrometer for Mars) instrument onboard the Mars Reconnaissance Orbiter [21]. The display shows orbiter ground tracks annotated with planned observations; it is used interactively together with a temporal view in the observation planning process. The Web Interface for Telescience [22] (WITS) browser is another good example of an operational view.

A key element in this concept of user interaction is that all of these views should be coordinated: user selection, filtering, and manipulation in one view affects what is displayed or highlighted in the others. Additionally, the linked

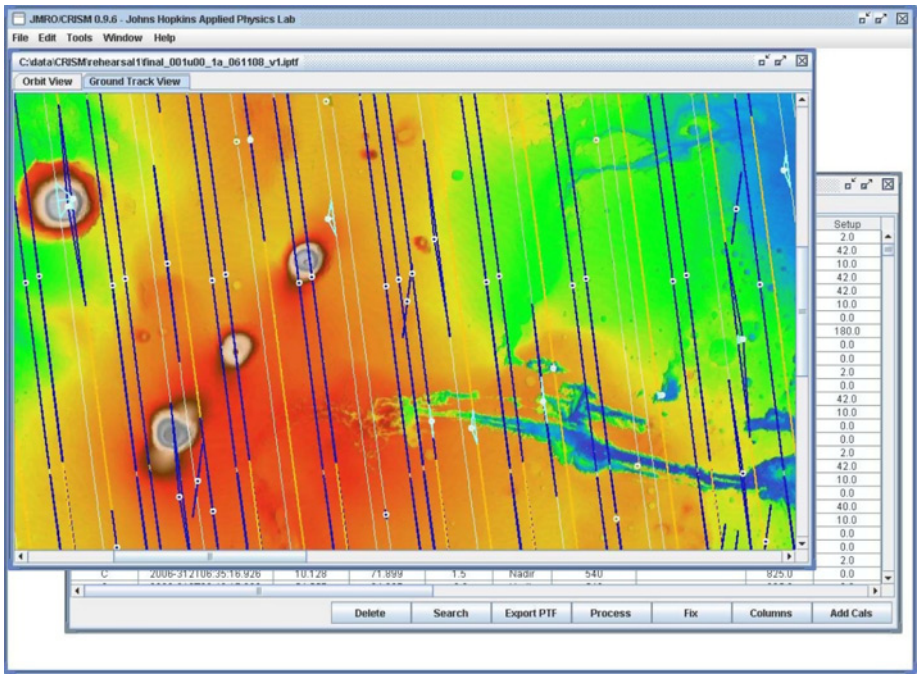

Fig. 13 Operational view in the CRISM planning tool. An operational view depicts goals, activities, and events in an appropriate context.

views are used to understand states and goals in both initial planning and in operation, driven by appropriate telemetry. For example, a goal that has failed may be indicated by a situation in the operational view (e.g., a rover not reaching an intended waypoint). That goal is simultaneously indicated in the structural view, showing how the goal was to be accomplished (look down) and why it should have been accomplished (look up). Selecting the goal slews the temporal view to the time of failure and populates it with timelines of relevant state variables. Interaction with the timeline provides causal information about the failure. The effectiveness of interactive timeline playback for understanding causal relationships leading to autonomous spacecraft actions has been demonstrated for rule-based autonomy in the STEREO mission [20], and should be even more important for GBO. If a new plan has been elaborated in response to the failure, the linked information in these views helps to provide the needed explanation [8] of the situation and the autonomous system's response to it.

VII. Challenges for Goal-Based Operations

Despite the promise of significant benefits and the impressive progress that has already been made in this area (see Sect. II), there remain a number of challenges for the wide deployment of GBO. Some of these challenges are technical in nature, while others are predominantly cultural hurdles that need to be overcome. This section provides a brief overview of some of these challenges, and suggests approaches to addressing them.

A. Observability and Controllability of the System

One commonly expressed concern about goal-based operations is that it could decrease the operator's observability into the behavior of the system, and/or the amount of controllability they have on the system. This concern stems from a common interpretation of the word "goal" as a relatively high-level objective or state to be achieved. In any mission, unusual situations may arise that require operators to take over detailed control of a vehicle, such as running very low-level sequences, even down to the level of opening and closing switches for example, and the concern is that GBO might preclude such detailed control. The fact is that GBO allows such control because goals at *any* level of detail can be specified by operators, scheduled on the ground, and uplinked to the vehicle. During normal operations, of course, low-level goals may be generated automatically from elaborations of higher-level goals, but this is not a

requirement; operators can specify low-level goals directly and *not* specify higher-level goals, if desired. This comes down to a tradeoff between desired level of ground controllability versus the amount of flexibility and robustness to be provided onboard the spacecraft.

Similarly, when properly implemented, GBO allows for improved visibility into the operation of the system. The approach places more emphasis on having good estimates of system state and the operator has access to more understandable state-based telemetry at appropriate levels of abstraction, in addition to the low-level measurements that typically make up the bulk of spacecraft engineering telemetry streams today. Furthermore, availability of the success/failure status of executed goals, and the ability to downlink this intent-related information explicitly, makes it easier for operators quickly to focus in on activities that have not executed nominally, rather than having to dig through megabytes or more of low-level telemetry, trying to identify the source of an onboard fault.

Of course, in any system with some degree of autonomy, there is a real concern that operators may become so reliant on the automation that they may become less able to intervene quickly in the case of an off-nominal situation that the onboard automation is not able to resolve. Note that this is a general issue associated with the move towards spacecraft with increased onboard autonomy, and not with GBO in particular. The solution to this problem lies in a combination of:

- proper training, including simulated failures that exercise more complex diagnostic procedures;
- development of operator interface tools that facilitate visibility into the system state and provide information about causes and purposes (see Sect. VI); and
- operations procedures that involve periodic confirmation of consistency between low-level telemetry and state estimates.

These considerations apply equally well to providing situation awareness to operators who have been called in to deal with off-nominal situations in lights-out operations or in response to a beacon call from a hibernating spacecraft.

B. Reliability/V&V (The Myth of Non-Determinism)

Another concern that has been expressed is that operators on Earth will not know in advance with certainty *what* commands are being issued and *when* they are being executed. This is another example of a concern that pertains to systems with onboard autonomy, whether goal-based or not. Although time-based sequences may allow operators to have more control over the nominal execution of commands, they are inherently brittle; given a traditional flight software architecture, where the nominal sequencing engine and the fault protection software run in parallel, any off-nominal behavior is far more likely to result in a drastic system-level response, such as safing, when fault protection takes over from nominal sequence execution. Furthermore, the fact of the matter is that operators have long accepted some degree of uncertainty in the precise timing and commanding of certain activities, such as attitude control maneuvers. Operators don't know exactly when or for how long an attitude control system will fire each thruster; this is a detail that *has* to be handled onboard because the dynamics preclude Earth-in-the-loop control. With more and more robotic vehicles interacting with unpredictable environments, it is necessary to situate more real-time decision-making in the vehicle. The challenge, then, is to design goal-based control systems that are as trustworthy as attitude control systems.

Operators might also ask, quite reasonably, "How do I know that the control system won't do something stupid?". This is a legitimate concern, just as it is for any software system, but it is sometimes based on an erroneous belief that autonomous systems are "nondeterministic," based on the arguments discussed above regarding execution uncertainty. In fact, all software is deterministic (that is, unless an algorithm has been deliberately implemented with randomization, as is sometimes the case for certain classes of optimization algorithm, for example), meaning that if it is given the same conditions, including its own internal state, it will do exactly the same thing as before. The real problem, of course, is that the state inputs to the deterministic flight software process have a certain degree of uncertainty; that is, operators cannot predict what the system is doing in real-time because they don't know what conditions it is seeing, given intermittent or time-delayed communication with Earth. The real concern is best articulated as "How well can you validate a goal-based control system?" To a large extent, the answer depends on the design of the software architecture. As Sect. IV has shown, a state/model/goal-based architecture provides a structure that is not only testable in traditional ways but also more readily analyzable by model-checking tools.

C. Software Maturity and Tool Support

A legitimate concern that people have regarding GBO systems is that the enabling software architectures and tools are not as mature and widely available as those that form the basis for existing operations approaches. However, as discussed in Sect. II, state-of-the-art goal-directed ground and flight software systems have been matured and validated to the point of sufficient TRL to justify adoption of this approach as a viable option for new missions. To mitigate this concern further, more work should be focused on the adaptation of familiar operations tools, to allow them to be used for GBO applications in the near term, and the development of intuitive, easy-to-use tools that provide operators with the observability and controllability they require, as discussed in Sect. VI.

D. Operations Processes and Training

Another valid concern is that current operations processes may not be directly compatible with GBO, and that current operations personnel are not experienced in this approach. Clearly, the transition to a new paradigm such as GBO is expected to have an impact on certain established operations engineering processes. Although a fair amount of work has been devoted to developing the technologies and software tools that enable GBO, more effort must be focused on developing and validating the processes that must go hand-in-hand with these technologies, and on training personnel to adopt these processes. The discussion in Sect. II points out how some flight projects have started tackling this problem, and how missions such as EO-1 have implemented effective operations processes built around the premise of goal-directed activities.

E. Responsiveness/Performance Concerns

Although GBO does not inherently require any particularly complex software solutions, most of the GBO systems fielded to date have leveraged deliberative planning and scheduling algorithms that exhibit worst-case exponential time performance. Clearly, this is an area of concern, to the extent that these algorithms will continue to be deployed onboard spacecraft with increasing frequency. Although progress has been made in recent years to improve the performance of these algorithms, e.g., through the use of appropriate heuristics, it is acknowledged that there is a significant level of complexity inherent in the planning and scheduling problem, which cannot be optimized away. The right way to address this concern, therefore, is through architectural and operational mitigation strategies, including:

- allocating control functionality appropriately across the flight software architecture, such that the expensive deliberation computations are performed by elements of the architecture that run at lower priority than the elements of the architecture that are responsible for real-time control and safety-critical monitoring and response behavior; and/or
- allowing the operators to specify additional constraints on the planning and scheduling problem to reduce the amount of computation required to solve it, e.g., adding temporal constraints to further restrict the number of acceptable schedules, or reducing the amount of flexibility that the planning and scheduling algorithms must reason about.

VIII. Conclusion and Outlook

Goal-based operation of robotic systems changes the semantics of operations from the imperative directives of time-based command sequencing to intentions on states of the system under control. Motivations for this change include the desire to reduce operator workload and operator error, the desire to make more effective use of expensive assets through increased automation, the necessity in some missions of making timely, in situ control decisions to respond to short-lived events, and finally the need for implementation-independent semantics for operation and interoperation of systems built by multiple vendors. In the longer term, GBO is an enabler for far-future missions consisting of multiple assets coordinating among themselves to achieve mission objectives with infrequent oversight from human operators.

It is encouraging to see the beginnings of a movement toward GBO, but if GBO is to achieve its full potential, particularly with respect to interoperability, common operational views, and trained operators, then standards will need to be developed. One such standardization effort exists in the European Cooperation for Space Standardization (ECSS) standard on space segment operability for unmanned missions [23]. This standard defines three levels of autonomy including: 1) execution of pre-planned mission operations onboard, 2) execution of adaptive mission operations onboard, and 3) execution of goal-based mission operations onboard. To our knowledge, comparable

standardization efforts specifically related to GBO have yet to be undertaken in the United States of America. This is a situation that will need to be addressed in the very near term, as attention and investment turns to large-scale "systems of systems" efforts, such as NASA's Constellation Program. To overcome the challenges of such ambitious enterprises with consistent safety and reliability, operability and interoperability standards will play a critical role in the long-term strategy [24].

Acknowledgments

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and at the Johns Hopkins University Applied Physics Laboratory, under contract with the National Aeronautics and Space Administration and as internal research.

References

[1] Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., et al., "Using Autonomy Flight Software to Improve Science Return on Earth Observing One," *AIAA Journal of Aerospace Computing, Information and Communication,* Vol. 2, No. 4, 2005, pp. 196–216.
doi: org/10.2514/1.12923

[2] Chien, S., Sherwood, S., Tran, D., Cichy, B., Rabideau, G., Castano, R., et al., "Lessons Learned from Autonomous Sciencecraft Experiment," *Proceedings of the Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2005)*, 2005.

[3] Sherwood, R., Chien, S., Tran, D., Davies, A., Castano, R., Rabideau, G., et al., "Enhancing Science and Automating Operations Using Onboard Autonomy," *Proceedings of the International Conference on Space Operations*, AIAA, Reston, VA, 2006.

[4] Bernard, D., Dorais, G., Fry, C., Gamble, E., Kanefsky, B., Kurien, J., et al., "Design of the Remote Agent Experiment for Spacecraft Autonomy," *Proceedings of the IEEE Aerospace Conference*, IEEE Publications, Piscataway, NJ, 1999.

[5] Bernard, D., Gamble, E., Rouquette, N., Smith, B., and Tung, Y., "Final Report on the Remote Agent Experiment," *Proceedings of the New Millennium Program DS-1 Technology Validation Symposium*, NASA, 2000.

[6] Morris, J. R., Ingham, M. D., Mishkin, A. H., Rasmussen, R. D., and Starbird, T. W., "Application of State Analysis and Goal-Based Operations to a MER Mission Scenario," *Proceedings of the International Conference on Space Operations*, AIAA, Reston, VA, 2006.

[7] Ai-Chang, M., Bresina, J., Charest, L., Jonsson, A., Hsu, J., Kanefsky, B., et al., "MAPGEN: Mixed Initiative Activity Planning for the Mars Exploration Rover Mission," *Proceedings of the International Conference on Automated Planning & Scheduling (ICAPS'03)*, AAAI Press, Menlo Park, CA, 2003.

[8] Bresina, J. L., Jónsson, A. K., Morris, P. H., and Rajan, K., "Mixed-Initiative Planning in MAPGEN, Capabilities and Shortcomings," *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'05), Workshop on Mixed-Initiative Planning and Scheduling*, AAAI Press, Menlo Park, CA, 2005.

[9] Proba: Observing the Earth Web site, European Space Agency, http://www.esa.int/esaMI/Proba_web_site/ index.html.

[10] Woods, M., Long, D., Baldwin, L., Aylett, R., Wilson, G., Ward, R., et al., "Onboard Planning and Scheduling for the ExoMars Mission," *Proceedings of the DASIA (DAta Systems In Aerospace) Conference*, ESA, 2006.

[11] Watson, D. P., and Scheidt, D. H., "Autonomous Systems," *Johns Hopkins APL Technical Digest*, Vol. 26, No. 4, 2005, pp. 368–376.

[12] Rasmussen, R. D., "Goal-Based Fault Tolerance for Space Systems Using the Mission Data System," *Proceedings of the IEEE Aerospace Conference*, IEEE Publications, Piscataway, NJ, 2001.

[13] Barrett, A., Knight, R., Morris, R., and Rasmussen, R., "Mission Planning and Execution Within the Mission Data System," *Proceedings of the International Workshop on Planning and Scheduling for Space (IWPSS 2004)*, ESA, 2004.

[14] Ingham, M., Rasmussen, R., Bennett, M., and Moncada, A., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," *AIAA Journal of Aerospace Computing, Information and Communication*, Vol. 2, No. 12, 2005, pp. 507–536.
doi: org/10.2514/1.15265

[15] Gat, E., "Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Mobile Robots," *SIGART Bulletin 2*, 1991, pp. 70–74.
doi: org/10.1145/122344.122357

[16] Gat, E., "Non-Linear Sequencing," *Proceedings of the IEEE Aerospace Conference*, IEEE Publications, Piscataway, NJ, 1999.

[17] Feather, M., Fesq, L., Ingham, M., Klein, S., and Nelson, S., "Planning for V&V of the Mars Science Laboratory Rover Software," *Proceedings of the IEEE Aerospace Conference*, IEEE Publications, Piscataway, NJ, 2004.

[18] Rajan, K., Shirley, M., Taylor, W., and Kanefsky, R., "Ground Tools for Autonomy in the 21st Century," *Proceedings of the IEEE Aerospace Conference*, IEEE Publications, Piscataway, NJ, 2000.

[19] Gersh, J., Cropper, K., Fitzpatrick, W., McKerracher, P., Montemayor, J., and Ossing, D., " 'And You Did That Why?'—Using an Abstraction Hierarchy to Design Interaction with Autonomous Spacecraft," in *Persistent Assistants: Living and Working with AI*, Papers from the 2005 AAAI Spring Symposium, AAAI, Menlo Park, CA, pp. 22–25.

[20] Gersh, J., Turner, R., and Cancro, G., "Visualizing Spacecraft Autonomy in Context and Across Time," *Proceedings of the AIAA Infotech@Aerospace Conference*, AIAA, Reston, VA, 2007.

[21] McGovern, A., "Scibox® Based Uplink Operations Planning Concepts For Responsive Space," *Proceedings of the* 4^{th} *Responsive Space Conference*, AIAA, Reston, VA, 2006.

[22] Backes, P. G., Norris, J. S., Powell, M. W., and Vona, M. A., "Multi-mission Activity Planning for Mars Lander and Rover Missions," *Proceedings of the IEEE Aerospace Conference*, IEEE Publications, Piscataway, NJ, 2004.

[23] Schmidt, M., Calio, E., Gessner, R., Kolster, P., Parkes, A., Pecchioli, M., et al., "The ECSS Standard on Space Segment Operability," *Proceedings of the International Conference on Space Operations*, AIAA, Reston, VA, 2004.

[24] Rasmussen, R., Ingham, M., and Dvorak, D., "Achieving Control and Interoperability Through Unified Model-Based Engineering and Software Engineering," *Proceedings of the AIAA Infotech@Aerospace Conference*, AIAA, Reston, VA, 2005.

Fernando Figueroa
Guest Editor