Goal-Based Operations: An Overview

Daniel D. Dvorak[1], Michel D. Ingham[2], J. Richard Morris[3]
NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

John Gersh[4]
Applied Physics Laboratory, Johns Hopkins University, Laurel, MD, 20723

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 like 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, 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.

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 to safely and repeatedly 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.

[1] Principal Engineer, Planning & Execution Systems Section, M/S 301-270, AIAA Member.
[2] Senior Software Engineer, Flight Software Systems Engineering and Architectures Group, M/S 301-225, AIAA Senior Member.
[3] Software and Systems Engineer, Planning and Sequencing Systems Group, M/S 301-250D.
[4] Principal Engineer, System and Information Sciences Group M/S 2-236, AIAA Member.

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. 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 Figure 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. Goals also allow flexibility in the ordering and timing of activities. They facilitate adjustable levels of autonomy and a spectrum of fault responses short of simply entering "safe mode". Importantly, it enables in situ decisions needed for mission scenarios like reconnaissance, making it easier to autonomously re-task assets in response to local observations.

Goal-based operation addresses these limitations by *making operator intent explicit* and carrying it into uplink products. Goal-based operation, which focuses on what to do (goals) rather than how to do it (command sequences), provides closed-loop control of systems, from the highest levels of mission objectives and coordination of assets to supervisory control of low-level control loops. The advantages are several: simplified operator workload; flexibility in ordering and timing of activities; more robust operation in unpredictable environments and in the absence of real-time communication; and adjustable levels of autonomy.

Importantly, goal-based operation facilitates *in situ* decisions (such as in reconnaissance and exploration), making it easier to autonomously re-task assets in response to local observations. The Autonomous Sciencecraft Experiment (ASE[1,2]) on the Earth Observing 1 (EO-1) spacecraft is such an example. As a result of on-board 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 operation supports the trend of increasing functionality in flight systems, with more components designed to achieve higher-level objectives 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

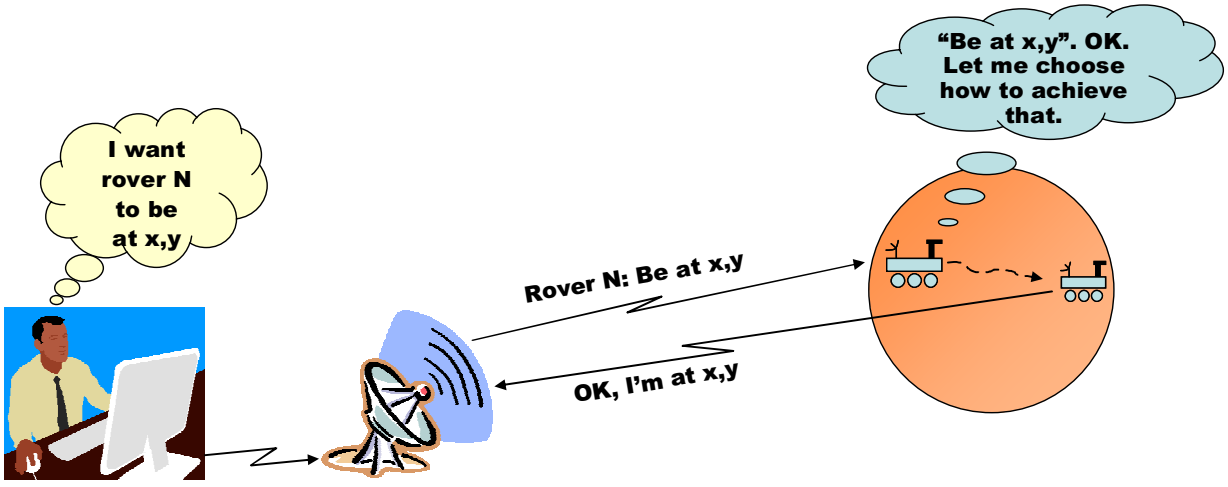

Figure 1. Goal-based operations elevate control to the level of operator intent. *A goal specifies a state to be achieved in the system under control; it specifies what to do, not how to do it, thus reducing operator workload and leaving options open for the control system.*

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 software architecture for a control system designed for goal-based operation. The paper also explores how a goal-based approach affects operations process and 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 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[3], which was flight-validated in 1999 on the Deep Space One (DS1) 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 Figure 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

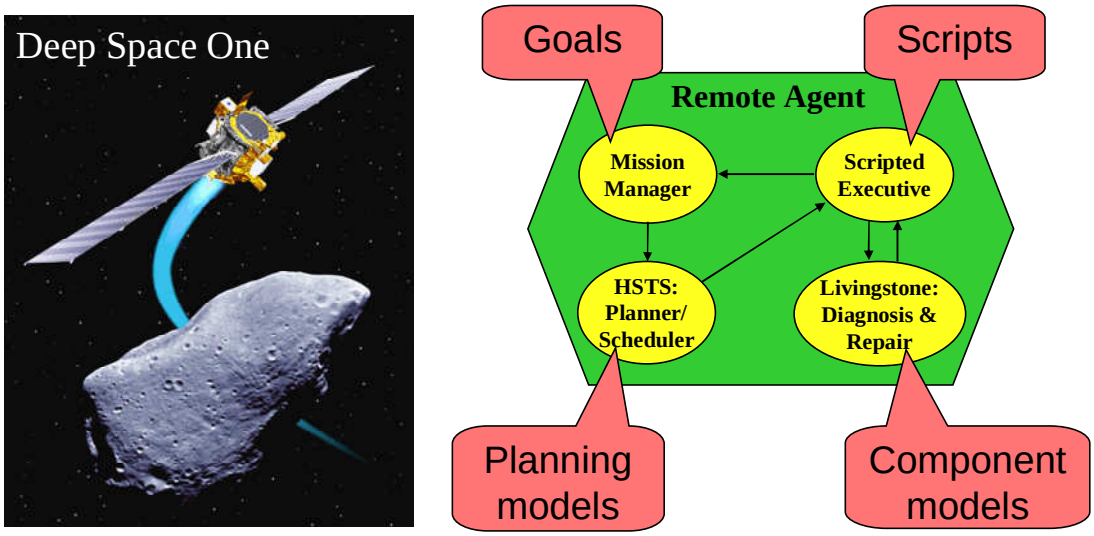

Figure 2. Remote Agent Experiment on Deep Space One. *The Remote Agent experiment demonstrated goal-based operations in 1999. In this architecture the Mission Manager sends high-level goals to the Planner/Scheduler which then generates detailed tasks (lower-level goals), and the Executive executes scripts associated with the lower-level goals, issuing commands as needed.*

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 DS1 Technology Validation Report[4].

NASA's Mars Exploration Rovers[5] (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[6]) 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 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 like 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[1,2]. 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, and has 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 control system. Through automation of the operations process, ASE has contributed operational savings of approximately $1M per year, compared to EO-1's nominal operations cost prior to ASE deployment. 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, 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 On-Board Autonomy[7] (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 on-board planning and scheduling for ExoMars[8], 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 final phase of a space mission project is termed "operations", the concept of goal-based operations shapes engineering activities much earlier. In the classical project lifecycle, as shown in Figure 3, the concept of goals plays a role in several phases, as described below.

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.

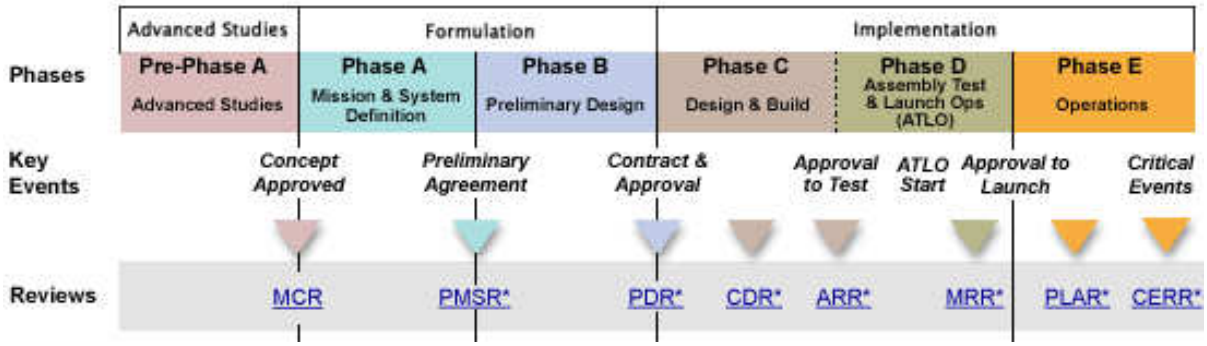

Figure 3. Project Lifecycle Phases. *The concept of a 'goal' shapes engineering activities beginning in Phase A, where high-level objectives may be stated as goals, and mission scenarios may be specified as sequences of goals, to Phase E, where goals are instantiated, elaborated, scheduled, and executed.*

Phase B (Preliminary Design):

- Identify key goal types (parametric specifications), key goal expansions or elaborations.
- Flesh out Concept of Operations, goal-based operations process.
- Identify goal-based operations tools that must be adapted for 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 consensus has emerged around a 3-layer architecture[9] 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[10] (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[11,12].

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. Since 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*[13,14].

All of this begs the question of *how* to represent intent. Since we're 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*, not the control system. For example, a goal may specify a desired spacecraft attitude since that is a constraint on a state of the system under control, but a goal must not specify a mode of an attitude controller since 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 Figure 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 or closed state or tripped state, and the control system will use evidence such as sensor measurements 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 1:00 PM to 2:00 PM. A different goal could require that the switch be closed 90% of the time between 1:00 PM and 2:00 PM, thus tolerating temporary switch 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.

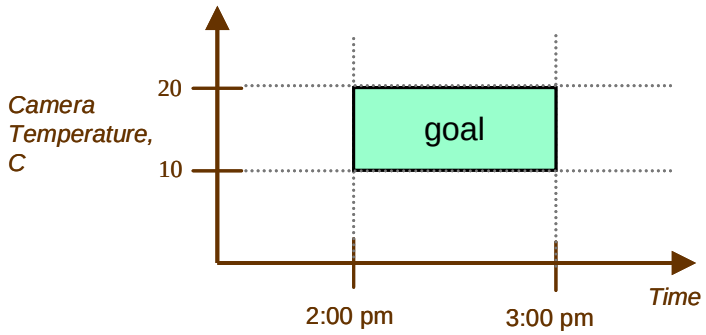

Figure 4. Goal as a constraint in value and time. *A goal represents operational intent as a constraint on the values of a state variable during a time interval, such as "Camera temperature is 10–20 °C from 1:00 PM to 2:00 PM". As such, a goal can be visualized as a region of acceptable behavior in value and time, as depicted by the box.*

Goals can also 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.

B. State Variables and Their Timelines

The preceding definition of 'goal' begs 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), dynamics (vehicle position and attitude, gimbal angles, wheel rotations),

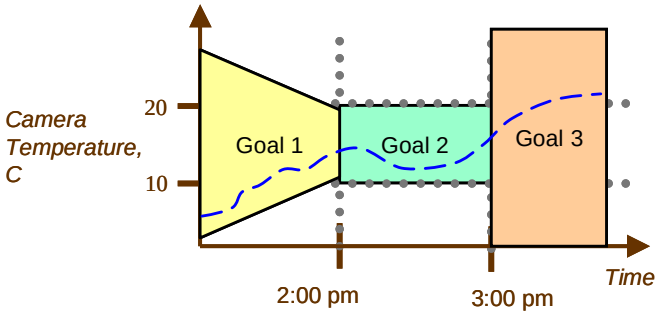

Figure 5. Intent and Knowledge Timelines. *A state variable contains two timelines. The intent timeline holds goals, ordered according to time, as depicted by the colored polygons. The knowledge timeline holds estimated state values, as depicted by the dashed blue line. A goal succeeds if its constraint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

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 we've 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 is landed on Mars within ellipse *x* by time *t*". This high-level goal will get decomposed into a myriad of supporting goals, in a manner to be described later.

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 Figure 6). 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 very important for robust control because it acknowledges that sensors and actuators are imperfect as well as 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 enables a controller to exercise caution when its control decisions depend on highly uncertain estimates. If an

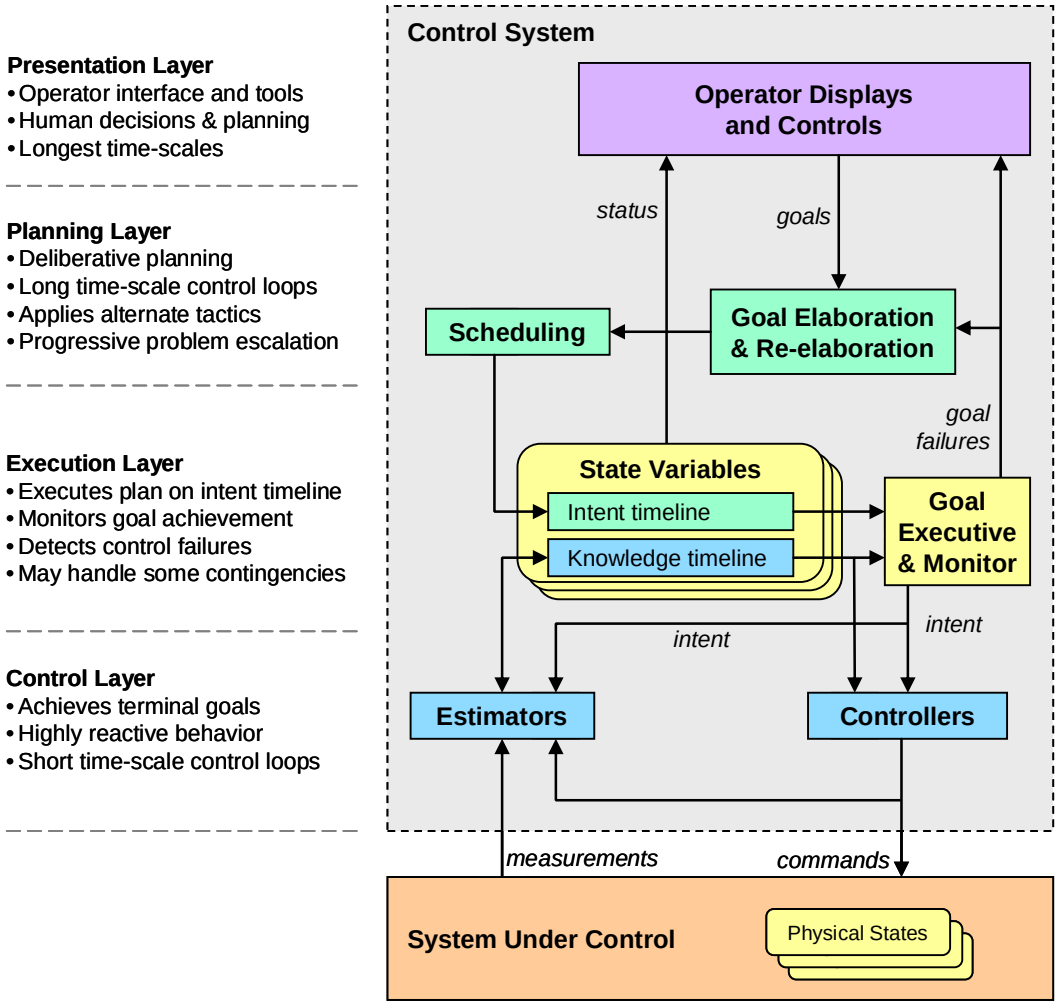

Figure 6. Software architecture layers. *All layers operate asynchronously and concurrently to achieve goals. The control layer operates on short time-scales, reacting to local observations, while the planning layer operates on longer time-scales, coordinating system-wide behavior, including system-level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

activity depends on some minimum level of certainty of state estimates, then a goal can specify such a constraint since uncertainty is part of the value representation of every state variable.

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[9]. 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, from high-level goals such as "Be landed on Mars" to low-level goals such as "Maintain switch 27 in closed state". 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 Figure 1, 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, 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 controls used by operators, where the highest level decision-making occurs.

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 status 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 through the process of re-elaboration, as described in the next section.

D. Goal Elaborations and the State Effects Diagram

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 states of the system, and many goals simply cannot be achieved without the support of other goals on other state variables that have an effect on the state of the primary goal. For example, a goal on spacecraft attitude has implications on antenna pointing and camera pointing, so it has effects that might be inconsistent with goals on those *affected* states. Likewise, a goal on spacecraft attitude can only be achieved if goals on *affecting* states are achieved or maintained, such as adequate power and propellant, warm catalyst beds for the thrusters, and healthy inertial measurement units.

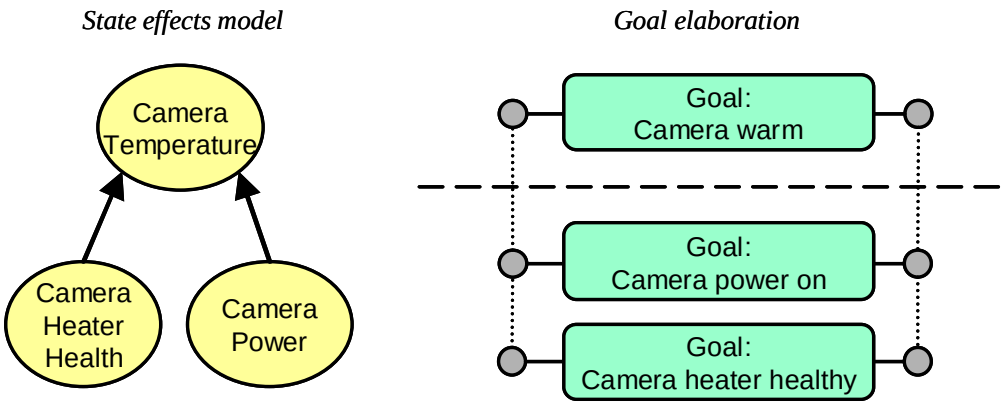

Figure 7. State effects model and goal elaboration. *A state effects model, produced during analysis of the system under control, shows what physical states affect other states. A goal elaboration, produced during operations engineering, shows a goal above the dashed line and its supporting goals below the line, reflecting the state-to-state effects that must be managed (or required) to achieve the original goal.*

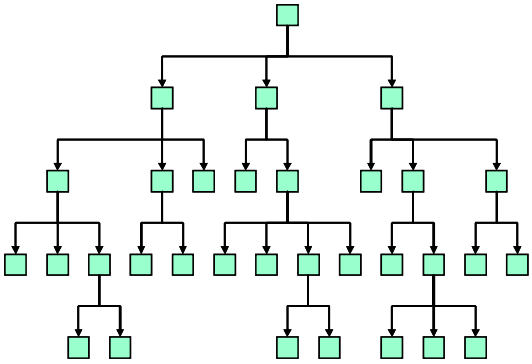

Figure 8. Hierarchical goal elaboration. *The end result of goal elaboration is a tree of supporting goals with the "leaf goals" being goals that have no further elaboration. The goals still need to be scheduled into a plan (a goal network) and then executed. All of the goals in the tree—not just the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

The engineering knowledge of what states affect other states in the system under control is represented in a *state effects diagram*, as shown in Figure 7, and that 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. The design of goal elaborations is based on the state effects diagram and the application of a few rules[12].

Since 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. 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 a determined by the temporal constraints in the network. A temporal constraint specifies minimum and maximum time duration between two time points. The time windows of time points are updated by a temporal propagation algorithm as temporal constraints are added or modified during scheduling and as time points fire during execution.

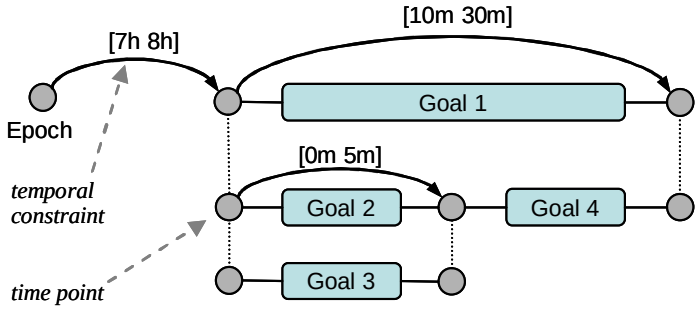

Figure 9. Example goal network. *A goal network consists of goals situated within a temporal constraint network (TCN), where the starting time of a goal is governed by the time point attached on its left side, and the ending time by the time point attached on its right side. Time points connected by a dashed line are 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.*

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 in the Control Layer informed as to what goals to execute. As time proceeds, the Goal Executive marches down the intent timeline, directing the Control Layer.

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) 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.

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 Ground could issue a single top-level goal that is transmitted to a Flight system where it is elaborated and 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 Ground and lower-level elaboration and scheduling on Flight. Also, the balance may change in either direction during the life of a mission. For example, during the earliest phase 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-hold 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 in order 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 goal/state/model-based architecture as described herein holds several benefits in system-level verification and validation. First, there is the obvious fact that the control system is self-checking since it is continuously monitoring 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 goal re-elaboration. This automatic checking is certainly preferable to any post-processing of telemetry, and is valuable not only during operations but also during testing. Second, there is a one-to-one correspondence between physical states and state variables. 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. Third, the explicit use of models—particularly the state effects model and the strongly related goal elaboration models—provide for direct inspection/validation by domain experts and also enable application of model-checking tools[19].

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. In order 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 question "why draw the line at all? 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 *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 abstraction hierarchy documents this translation in a formal, inspectable and—if we employ the MDS architecture—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 mentioned, 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 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.

The 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, due to restrictions on the duration of the process, observations are simply shed and science opportunities are lost. Goals facilitate a shift to a work-flow paradigm (Figure 10), where one product set managed by a common tool undergoes successive stages of refinement, resulting in a more reversible, collaborative, and seamless process.

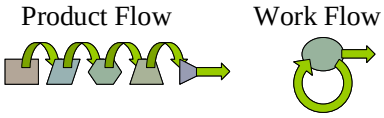

Figure 10. Goal-based operations facilitates a shift from a product-flow-oriented process to a work-flow paradigm.

VI. Operator Interaction with Goal-Based Systems

Since 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 importantly, to understand how those intentions are (or are not) being carried out *in context.* Displays 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 decomposed 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.[21] To this point, however, operational systems that deal with goals[1] 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 lower-level implementations, mission operators will not have the information necessary to decide whether planned goals are still appropriate in dynamic environments or whether the decomposition of goals into subgoals remains effective.

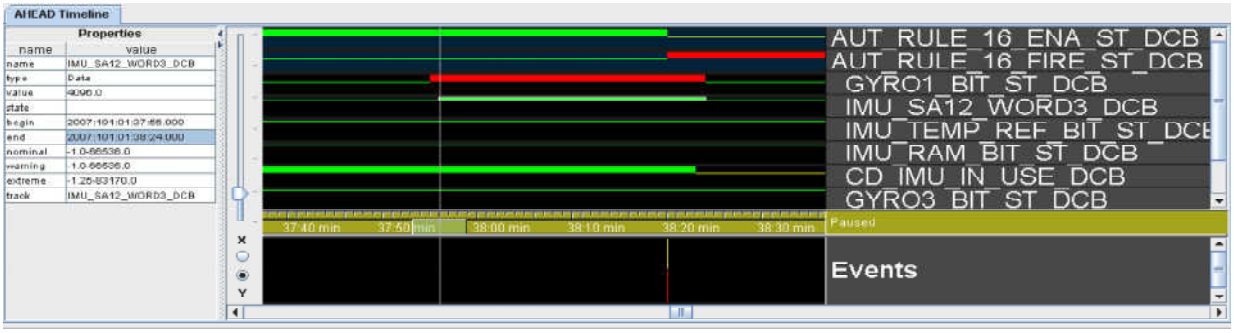

Figure 11. Temporal view of autonomous activity in a STEREO spacecraft. *A temporal view shows autonomous activities, goal accomplishment, environmental events and system responses over time, at selected levels of abstraction.*

One approach is to represent mission objectives, high-level goals, and lower-level implementations as an abstraction hierarchy[15,16]. In such a hierarchy looking upward 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 subgoals 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, from reference 15, is a temporal view of rule-based autonomy activity in a Solar-Terrestrial Relations Observatory (STEREO) spacecraft. In a goal-based system rather than the rule-based system here, goals and subgoals would be depicted in the timeline. A *structural* view depicts relationships among goals, activities, resources in hierarchies of abstraction and dependency. Figure 12 shows a concept for a structural view, indicating the complex set of relationships among goals and subgoals for an autonomous science activity. 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) on board the Mars Reconassaince Orbiter[17]. 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 Tumescence[18] (WITS) browser is another good example of

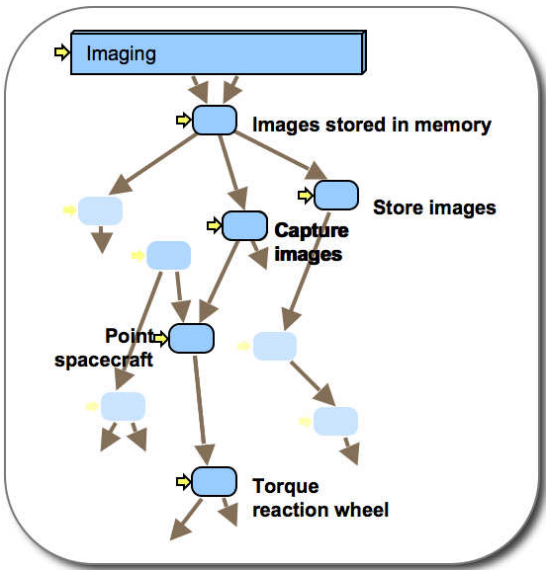

Figure 12. Structural View Concept. *This view shows relationships among goals at different levels of abstraction and dependency. From the perspective of a single goal, looking upward explains why it exists and looking downward explains how it will be accomplished.*

an operational view. All of these views should be coordinated: user selection, filtering, and manipulation in one view affects what is displayed or highlighted in the others.

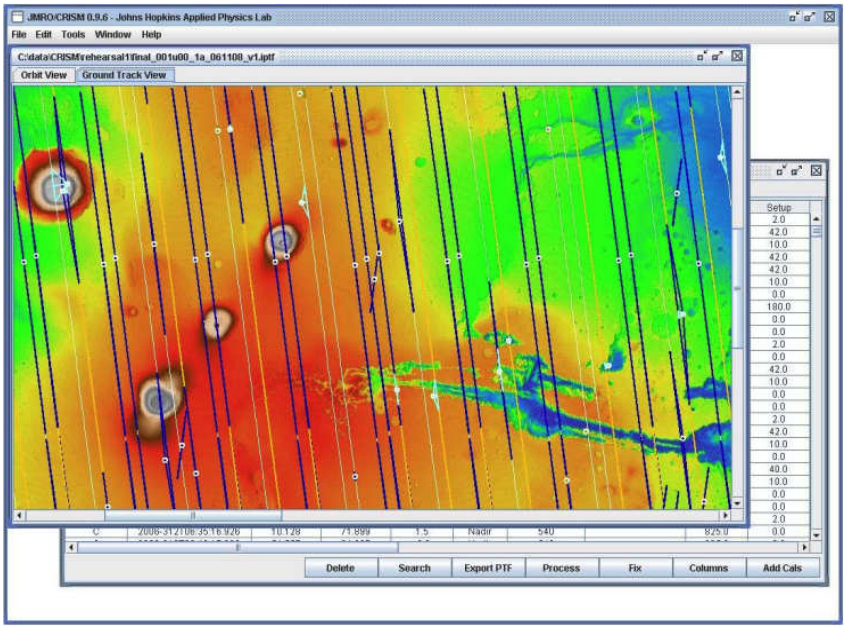

Figure 13. Operational View: CRISM Planning Tool. *An operational view depicts goals, activities, and events in an appropriate context.*

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 Section 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 decreases 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/closing switches, 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: because this approach places more emphasis on having good estimates of system state, the operator has access to more intuitive state-based telemetry, 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 to quickly 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 increase 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 (see Section V); and
- operations procedures that involve periodic confirmation of consistency between low-level telemetry and state estimates.

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 ACS will fire each thruster; it's a detail that *has* to be handled onboard since 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 ACS.

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, 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 FSW 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 Section IV has shown, a state/model/goal-based architecture provides a structure that is not only testable in traditional ways but also 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 Section 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. In order to further mitigate this concern, 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 above.

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 like 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 Section II points out how some flight projects have started tackling this problem, and how missions like 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 safety-critical real-time 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 operations (GBO) 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 standards effort exists in ESA's ECSS "Standardization in Ground Systems & Operations Domain" ECSS-E-70-11 Space Segment Operability. This standard defines three levels of autonomy including: (1) execution of pre-planned missions operations on-board, (2) execution of adaptive mission operations on-board, and (3) execution of goal-based mission operations on-board.

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 Applied Physics Laboratory, John Hopkins University.

References

[1]Chien, S. et al., "Using Autonomy Flight Software to Improve Science Return on Earth Observing One", AIAA Journal of Aerospace Computing, Information and Communication, April 2005.

[2]Chien, S., et al., "Lessons Learned from Autonomous Sciencecraft Experiment", Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2005). Utrecht, Netherlands. July 2005.

[3]Bernard, D. et al., "Design of the Remote Agent Experiment for Spacecraft Autonomy," in Proc. IEEE Aerosp. Conf., Aspen, CO, 1999.

[4]Bernard, D., et al., "Final Report on the Remote Agent Experiment", Proceedings of the New Millennium Program DS-1 Technology Validation Symposium, Pasadena, CA, February 2000.

[5]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 SpaceOps 2006 Conference, Rome, Italy, June 2006.

[6]Ai-Chang, M., et al., "MAPGEN: Mixed Initiative Activity Planning for the Mars Exploration Rover Mission", 13th International Conference on Planning & Scheduling (ICAPS '03), Trento, Italy, June 2003.

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

[8]Woods, M., et al., "On-board Planning and Scheduling for the ExoMars Mission", Proceeding of the DASIA (DAta Systems In Aerospace) Conference, Berlin, Germany, 22-25 May 2006.

[9]Watson, D. P. and Scheidt, D. H., "Autonomous Systems", Johns Hopkins APL Technical Digest, 26(4), 2005.

[10]Rasmussen, Robert D., "Goal-Based Fault Tolerance for Space Systems Using the Mission Data System", 2001 IEEE Aerospace Conference, Big Sky, Montana, March 2001.

[11]Barrett, A., Knight, R., Morris, R., Rasmussen, R., "Mission Planning and Execution Within the Mission Data System", Proceedings of the 4th International Workshop on Planning and Scheduling for Space (IWPSS 2004), Darmstadt, Germany, June 2004.

[12]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, Dec. 2005.

[13]Gat, E., "Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots," in SIGART Bulletin 2, 1991, 70-74.

[14]Gat, E., "Non-Linear Sequencing", Proceedings of the 1999 IEEE Aerospace Conference, Snowmass at Aspen, CO, March 1999.

[15]John Gersh, Kevin Cropper, William Fitzpatrick, Priscilla McKerracher, Jaime Montemayor, Daniel Ossing. " '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 Spring Symposium, Technical Report SS-05-05. American Association for Artificial Intelligence, Menlo Park, California, pp. 22-25.

[16]John Gersh, Russell Turner, George Cancro, "Visualizing Spacecraft Autonomy in Context and Across Time," Proceedings of the 2007 AIAA Infotech@Aerospace Conference, Rohnert Park, California, May 2007.

[17]Andy McGovern, "Scibox® Based Uplink Operations Planning Concepts For Responsive Space," Fourth Responsive Space Conference, 2006.

[18]Backes, P.G.; Norris, J.S.; Powell, M.W., Vona, M.A., "Multi-mission activity planning for Mars lander and rover missions," Proceedings of IEEE Aerospace Conference, 2004, Volume 2, 6-13 March 2004 Page(s):877 – 886.

[19]Feather, M., Fesq, L., Ingham, M., Klein, S., and Nelson, S., "Planning for V&V of the Mars Science Laboratory Rover Software", *2004 IEEE Aerospace Conference*, Big Sky, MT, March 2004.

[20]Sherwood, R., et al., "Enhancing Science and Automating Operations Using Onboard Autonomy," The 9th International Conference on Space Operations, Rome, Italy, June 2006.

[21]Kanna Rajan, Mark Shirley, William Taylor, Bob Kanefsky, "Ground Tools for Autonomy in the 21st Century," Aerospace Conference Proceedings, 2000 IEEE Volume 7, 18-25 March 2000 Page(s):649 - 659 vol. 7.