Architectural Concepts for Human-Rated Automation

David A. Wagner[*], Daniel L. Dvorak[†], Gregory A. Horvath[‡], Andrew H. Mishkin[§], Glenn S. Johnson[††], Grailing Jones, Jr.[§§]
Caltech/Jet Propulsion Laboratory 4800 Oak Grove Dr. Pasadena, CA 91109

Since the beginning of human spaceflight, spacecraft designers have emphasized simplicity of design to minimize failures and have relied on a large ground staff to assist flight crews. As spacecraft and missions have grown in complexity, human workload has grown, keeping operational costs high and limiting productivity. A more progressive approach to automation can address these problems if risk can be reduced. We believe that a major contributor to risk of automation is weak architecture, leading to designs that are difficult to understand, analyze, verify, and operate. This paper explores architectural considerations of human-rated automation for space systems, where safety is paramount. Improved architectural concepts for automation are analyzed. We focus on goal-oriented control, and how goal-oriented analysis and design address the concerns associated with increased automation.

I. Introduction

DECISIONS about the degree of automation in human spaceflight face a well-known dilemma: automation is desired for its benefits but feared for its risks. Automation can reduce operational costs by offloading human workload, and can improve safety by reacting swiftly to time-critical problems, but the added functionality can also increase system complexity and reduce reliability. Given a history of concerns with software automation and given the high priority to safety, human spaceflight decision-makers have generally taken a conservative position with respect to automation. The consequence of minimal automation is that crews spend less time on key objectives and missions pay for a larger operations staff.

Since automation is largely implemented in software, the risks of increased automation are sometimes equated to the risks of additional software and its complexity, but this view is an overly simplistic statement of the problem. The fundamental problem with traditional techniques for automation – scripting, sequencing, and other forms of imperative programming – is that they lack any explicit understanding of intent. Intent is only implicit in the procedure or the code.

---

[*] Senior Software Architect, Flight Software Applications Group/MS 301-240, AIAA Member.
† Principal Engineer, Engineering Development Office/MS 301-270, Senior Member
‡ Staff Software Engineer, Flight Software Applications Group/MS 301-270.
[§] Principal Engineer, Planning and Execution Systems Section/MS 301-250D
[††] Senior Systems Engineer, Mission Systems Concepts Section/MS 171-300. AIAA Member
[§§] Senior Technical Staff, Planning and Execution Systems Section/MS 301-250D. AIAA Member

Some other problems attributed to software have roots earlier in the project lifecycle, often as a result of ambiguous or incomplete requirements and weak architecture. Other problems that appear later in the lifecycle, whether during integration, testing, or operations, have roots in weak architecture and inadequate attention to quality requirements such as verifiability and operability. Perhaps the biggest problem is the gap between systems engineering, software engineering, and operations engineering, making it difficult for these tightly coupled disciplines to reach a shared understanding of the system as a whole. In order to reduce risks and gain the benefits of automation, these issues must be addressed.

Unfortunately, the dominant paradigm in automation is part of the problem. That paradigm relies on sequences, scripts, and other programmed sequential behavior to perform complex sequences of actions as a function of time and sensory inputs. As systems have grown in complexity to satisfy increasingly ambitious objectives, this paradigm has been increasingly strained. System behavior can be hard to predict, especially in off-nominal situations, leading to misbehavior and operator confusion, in part because there is no general architecture for coordinating automation in the first place. Not surprisingly, automation in human-rated systems has been limited because its trustworthiness has not kept pace with spacecraft complexity.

Human rating has been described as "the *process* of satisfying the mutual constraints of cost, schedule, performance, risk, and benefit while addressing the requirements for human safety, human performance, and human health management and care."[12] While there are minimum standards for the provision of life support and health care, and clearly defined constraints on the limits of human performance, there are few absolute standards for what is safe. Instead, safety is managed as a set of engineering decisions judging the relative costs and benefits of mission objectives and methods for achieving them. Ultimately, many safety issues amount to what implementation choices can be trusted, and how that trustworthiness is attained.

What human spaceflight programs need—as well as other safety-critical programs—are principled concepts that help systems and software engineers *repeatedly* design and deliver automation at lower risk and higher trust than conventional methods. Such human-rated automation must be transparent, predictable, fault-tolerant, verifiable, and safety-cognizant. By 'transparent' we mean that system-level requirements are clearly represented in software design such that systems engineers and hazard analysts can inspect and understand the software design. By 'predictable' we mean that automation must not be a source of unexpected behavior. By 'fault-tolerant' we mean that fault tolerance must be an integral part of design, not an afterthought or add-on. By 'verifiable' we mean that the software is structured in a way facilitates verification through analysis and testing. Finally, by 'safety-cognizant' we mean software that, by design, explicitly represents safety properties, provides mechanisms to monitor and enforce them, and automatically escalates problems when functionality at one level is unable to achieve safety properties.

This paper shows how these important properties of automation become easier to achieve in a cognizant control architecture. In such an architecture control systems are "cognizant" of objectives and not just blindly executing commands. The concept of *goal* is fundamental to cognizant control in that a goal explicitly specifies intended behavior, spawning a number of benefits: translation from requirements to design is simplified because goals are understood on both sides; system behavior is easier to verify because goals specify acceptable behavior; and operations is simpler and less error-prone because the system is commanded in terms of intended behavior rather than a sequence of actions meant to implement some implied intent.

Much of the effort in software engineering over the years has focused on software abstractions that facilitate modeling of software programs and the systems in which they operate, thus minimizing the conceptual mismatches between the code and the requirements it is intended to fulfill. A goal-oriented control system leverages this approach by modeling not only the mechanisms by which the intent is achieved, but by including the intent itself in the models in the form of goals. Thus, transparency of design is achieved through a clear and unambiguous mapping from the intent of the designers to design and implementation.

Expressing intent in the form of goals also allows for much clearer verification that the system is correctly implementing behavioral requirements because the goals directly express the intent of functional requirements. As a result, traceability to requirements can be much more explicit than it might be in a traditionally coded control system. Verification of reliable systems must also consider *all* off-nominal execution paths to ensure that the system will be able to recover from expected and unexpected problems. Again, if the control intent is explicit, the system can more easily detect when that intent is not being achieved.

Space systems in particular must be able to operate in an unpredictable environment, and must be robust to any number of predictable and unpredictable hardware faults that may occur. They must be able to reliably continue to achieve mission goals in the presence of faults, and at least ensure the safety of the crew in the event of failures that prevent completion of the mission.

Traditional safety engineering focuses on a reliability of parts, and a fail-fast control design approach where the system is highly sensitive to potential faults, in which case it transitions to a stable safe state where operators can sort out what happened. However, not all situations have an easily reachable state that is both stable and safe. A safety-cognizant system can continually enforce safety constraints by actively controlling away from a safety boundary or responding to a violation in a context-sensitive way.

This paper presents architectural concepts for human-rated automation, with emphasis on the qualities listed above. ‘Architecture’, in this case, is the fundamental principles and well-defined patterns that guide not only software design but also systems engineering and operations engineering. Although the examples used here are drawn from lunar surface operations scenarios, the principles and patterns can be applied broadly to safety-critical systems in which humans interact with automated and/or robotic systems to accomplish operational objectives. Many of the examples described here have been demonstrated in software simulations in our lab.

Finally, it is worth noting that the operations and control methods described in this paper extend beyond today’s near-Earth mission scenarios. Today, crews and ground controllers can interact in a relatively continuous and conversational manner because of the crew’s close proximity to Earth. As missions venture farther from the Earth, communications latency increases, and at the distance of Mars a signal can take more than twenty minutes to reach Earth, making interactive control impossible. Thus, as was recognized long ago for robotic missions, interactive ground control becomes untenable and a higher degree of automation is required to make missions productive and fault tolerant. As this paper describes, goal-based operations provides a foundation for automation that can serve both near-Earth and deep space missions.

We begin by reviewing the constraints and the driving need for human-rated automation, elaborate on architectural mechanisms that can achieve these qualities, and conclude with some analysis of the viability of this approach.

II. The Need for Automation and Autonomy

First, we need to clarify terms. In the human spaceflight community the term autonomy refers to the ability of the crew to operate a spacecraft without the help of ground controllers. This may or may not rely on the use of automation. In the robotic spacecraft community the two terms are generally taken as equivalent because there is no onboard crew, and thus autonomy can only be accomplished through automation.

NASA's human-rated spacecraft have traditionally been designed following some basic principles first articulated during the Apollo program. To avoid unnecessary complexity, systems are designed to be as simple as possible while still meeting the functional requirements of the mission. The prevailing view is that complexity, generally speaking, can increase risk and cost. Consequently, any onboard automation has been viewed as introducing unnecessary complexity because systems were thought to be robust and safe enough as they were, under the assumption that mission control would be able to adequately monitor operations and correct problems from the ground. This view that automation is unnecessary may be changing.

The history of robotic space exploration began the same way, except that controllers' ability to interact with their spacecraft diminished with increased distance from Earth. The laws of physics dictate that the information bandwidth that can be received for a given transmit power diminishes with distance, and the latency of that information increases. Additionally, spacecraft in orbit around other bodies are likely to be entirely out of view of Earth-based communications for some part of every orbit. Operators soon realized that there were some situations where the ability to enter a stable "safe mode" would give ground controllers the necessary time to assess a situation and plan a response. However, there are also many situations where the physics of the situation can end the mission before ground control could even know that an anomaly occurred. As spacecraft developed the ability to articulate and drive around on the surface of other planets, the latency of control became a significant impediment to simple operations. If operators had to wait 30 minutes after taking each picture or turning each wheel in order to see its outcome and decide on what to do next, the amount of science that could be achieved during the nominal mission lifetime would be severely limited.

Today, all deep-space robotic exploration depends on some amount of automated control for all critical activities such as orbit insertion; entry, descent, and landing; and critical science observations. Missions increasingly depend on the enhanced operational capability and dependability this automation enables in order to achieve optimistic mission goals in increasingly distant and unforgiving environments. Similarly, human exploration systems are becoming increasingly complex in part as a result of more complex mission objectives, but also due to a greater awareness of safety and mission risks. Increasingly complex systems have more operations to coordinate, resources to manage, and states to monitor. A failure to automate at least some of this could leave crews no time for actual exploration.

Since the Apollo days, awareness of safety risks has risen through experience and better understanding of the limitations of both human and avionics capabilities, and the risks posed by the deep-space environment. Yet, mission goals have not diminished – quite the opposite. The Augustine commission[13] clearly advocated more focus on exploration. As human-operated spacecraft venture farther from Earth, it is clear that they will need to respond to the physical constraints that dictated the need for automation in robotic spacecraft.

III. Architectural Quality Attributes for Human-Rated Systems

In their seminal text on software architecture, Bass et.al.[10] list the fundamental architectural drivers of software systems as functional requirements, technical or business constraints, and quality attributes. In our previous paper[11], many of the functional requirements specific to human-rated systems, as well as some of the constraints imposed by the Project Constellation systems architecture, were discussed. In this paper, we will focus on the last term of the architectural driver equation – quality attributes – and their importance to the process of developing control systems supporting human-rated autonomy.

When we speak of quality attributes, we are referring to the qualities or behaviors that we want the system to embody. Desired qualities are often beyond what is typically captured in the system functional requirements because they define the ideals, or dimensions in which the system design attempts to optimize, and as a consequence are often the most obviously lacking portion of systems with a weak or informal architecture. By elevating these qualities to first-class concerns, we explicitly capture their formal role in the system, as well as indicate that fulfillment of these qualities is essential to fielding a successful system.

As discussed above, there are several architectural quality attributes that are of primary importance to developing control software for human-rated systems, which must meet the highest of standards, due to the high consequence of failure. Because there are many different interpretations of the quality attributes and to frame the discussion of the goal-oriented control architecture that follows, we present a more detailed discussion of each quality attribute and comment on their importance to fulfilling the objectives presented in later sections.

- **Transparency.** Transparency refers to the ability to see through the implementation in order to understand how and why it works, and to the ability to trace implementation artifacts and behaviors back to requirements. This attribute can be particularly vexing, as it is not only a function of the final implementation, but also a feature of the requirements themselves, as well as the systems engineering processes employed in the design and construction of the system. To be clear, however, transparency implies more than just an annotation of a given code base with references to the parent requirements of each class or method; rather, it implies a holistic approach to engineering of complex systems. Transparency calls for capturing requirements in a manner that is amenable to an implementation, and an implementation that provides a clear and direct mapping to both the content *and* structure of those requirements. Transparency can help to facilitate communication between and increase the consistency of work done by the systems engineers and software engineers; this is a general theme that we will touch upon throughout the paper.
- **Predictability.** When we discuss predictability, we are referring to the behavior of the automated elements of our systems. We cannot expect systems to be completely deterministic when they operate in the real world because the real world is not that predictable. Devices can fail, and the space environment can readily deliver unanticipated situations. However, it is important for human rated space systems to react to these situations in predictable ways so that crew members and ground controllers can understand what happened, and – if necessary – respond appropriately. Systems which make some sort of predictability guarantees can make great strides in addressing the often strong resistance to adoption of new automation technologies due to a perception that systems developed using these technologies are prohibitively difficult – if not impossible – to characterize and demonstrate compliance with the

driving requirements. As discussed above, greater automation is an enabler for the goals of future exploration activities. The challenge is to introduce these automation technologies in a way that engenders the trust of those responsible for its safe and reliable operation.

- **Fault-Tolerance.** Fault-tolerance, or dependability, goes beyond reliability to define an ability to actively recover from faults[14]. It is a quality that many in the domain are familiar with, and it is a subject that has received considerable attention over the years, not just in the domain of human spaceflight, but also in the fields of autonomous spaceflight, aviation, automotive, manufacturing, and more. However, we note that many traditional systems engineering and design methodologies only consider reliability of parts, or treat fault-tolerance as an afterthought – that is, the system is designed to meet the requirements of nominal operation, while the task of ensuring that the system can tolerate off-nominal scenarios is undertaken as a separate and parallel activity. This artificial division often leads to duplication of effort in the best cases, and can have deleterious consequences in the worst cases. Therefore, we argue for an architecture that enables the design of robust systems, as well as a supporting systems engineering methodology which considers designing for fault-tolerance as a fundamental task, not separate from the 'nominal' design processes. More broadly, we argue that the term fault-tolerance should imply a measure of robustness to latent coding errors or unexpected features of system or environmental behavior, as opposed to the more traditional definition that focuses solely on robustness to hardware failures.
- **Verifiability.** Systems that must be highly complex, yet dependable, are challenged to prove their dependability. Clear requirements and a sound design are essential to trustworthiness, but not sufficient. The system design and implementation must be verified, through formal analysis, testing, or both, and a good design will facilitate that. For human-rated systems, verifiability is an essential quality, since we want to be able to conclusively prove something about the system behavior before committing to putting a human onboard. Couple this requirement with the magnitude of modern software-intensive systems – which are too large and complex to test completely even with the best tools available – and it becomes clear that a different approach is needed. Currently, the task of verifying a system is based in large part on adherence to a pre-established process or set of processes that are put in place, presumably, to help the designers to build the system in such a way as to ensure that the finished product exhibits 'correct' behavior, according to some set of metrics. The problem is that the processes are often focused more on the management of tasks, rather than *how* those tasks are performed. Therefore, when we talk about verifiability, we are really interested in architectural solutions we can employ to ensure that the system is constructed in a manner that enables one to unambiguously show that the system exhibits intended behavior. Furthermore, we are also interested in what we can do *from the start* to ensure that the system is built in such a manner as to maximize the probability that it will behave as intended.
- **Safety-Cognizance.** Safety cognizance is most closely related to fault-tolerance, but in reality it is related to all four previous quality attributes discussed thus far, as fulfillment of each quality attribute is an important part of formulating a truly safety-cognizant architecture. Safety cognizance differs from dependability in that

dependability is concerned with the successful achievement of mission goals, whereas safety-cognizance is concerned with awareness of hazards, hazardous situations, and combinations thereof. A safety-cognizant system design will formally incorporate mechanisms to recognize hazards and respond appropriately to ensure the safety of the crew in the face of those hazards.

The qualities discussed above are essential to safety and mission success. In general, there are other qualities that we are concerned with in the development of space systems, such as efficiency and reusability, which must be balanced with the above qualities over the life of the mission. Throughout the remainder of the paper, we will present a number of architectural principles and practical safeguards that we contend are essential to developing systems suitable for human exploration. In order to effectively implement these safeguards and mechanisms, we must have a stable base from which to begin. That base is a rigorous architecture that incorporates these quality attributes, and that helps us build a robust system from the start. In the following sections, we describe the salient features of a state-based, goal-oriented control architecture upon which we develop several safeguards essential for developing the next generation of exploration systems, and discuss how this architecture helps to reduce the cost of operations cost and reduce overall risk by improving the systems engineering process.

IV. Goal-Oriented Control

Given that more automation will be needed to safely conduct more complex future missions, how will it be safely and economically and safely implemented? We believe that the application of automation using a systematic, goal-oriented control approach offers the best chance of overall success. As will be discussed in subsequent sections, a systematic formal methodology is needed to ensure overall consistency and coordination of the many states of the complex systems involved in space exploration.

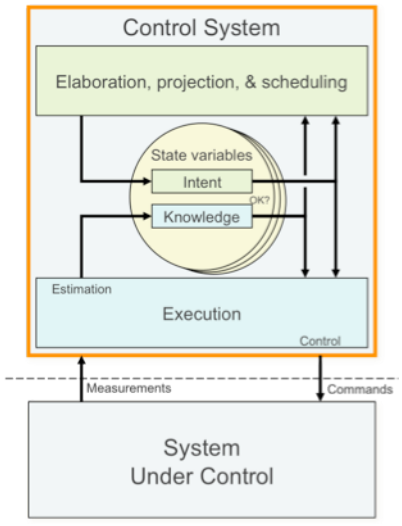

Figure 1 Key Elements of Goal-Oriented Control

The principles and mechanisms of goal-oriented control have been documented in previous work, and implemented in control frameworks such as the Mission Data System (MDS)[1-5]. The design of the MDS is driven by a formal methodology for system engineering control systems called State Analysis[6,7].

Briefly, State Analysis requires that the target states of the physical system that are to be controlled are identified and represented by state variables. Other state variables can represent other aspects of the physical system that must be known in order to exert control. User intent is expressed in the form of goals. A goal describes a constraint on the value of a single state variable over some specific interval of time. So, goals express simple intent without describing how that intent will be achieved. Goals can associate with elaboration rules that describe one or more sets of supporting goals (tactics) needed to achieve the desired state, and the control system itself can embody further static control models to perform the actuations needed to achieve expressed goals.

As shown in figure 1, a goal-oriented control system enables system coordinated control through the use of an executive layer responsible for executing goals currently in effect, and a

planning and schedule execution layer that dispatches goals to the lower executive layer according to a plan, but also verifies plan achievability against projected system states as part of the scheduling/planning process.

A key benefit of this approach is that control intent is always explicit, not only with respect to the desired states of the system, but also the duration of those effects in the time domain. And, control is always coordinated to the extent that there is a plan in effect. This not only ensures that the control system can continuously verify whether or not it is achieving this intent, but also that operators can get continuous feedback about what the system thinks it's supposed to be doing. The rest of this treatise describes other more specific ways overall safety is enhanced through the application of these goal-oriented mechanisms.

V. Safeguards

A safeguard is a safety constraint that is actively enforced by the control system as an integral part of plan execution. A safety constraint is often a unique system requirement in that the goal is to prevent the system from ever reaching a given undesirable state or condition or provide an appropriate response if it ever does. Requirements of this form are often harder to achieve than goals that specify a particular intended behavior because there may be innumerable ways the unintended situation could occur, whereas you only need to find one way to achieve a positive requirement.

While it is always better to prove that a given undesirable condition can never happen, in a sufficiently complex system this may be an impossible goal. In particular, reliability analyses based on fault probabilities are notoriously difficult to apply to software-based control systems where complex software constitutes the core of the system, potentially coupling fallible hardware devices in unpredictable ways, and introducing the possibility of latent logic defects, or random misbehavior resulting from radiation-induced upset events. Better to assume that faults will occur and be prepared to handle them. The ability of a goal-oriented system to express and execute intent over time allows the system to always know what it should be doing, and respond appropriately to anomalies.

A goal-oriented control architecture such as the Mission Data System offers system designers the opportunity to apply control logic at four distinct levels in the architecture: embedded in the physical system itself (e.g., a relief valve), in the execution layer (a goal achiever), in the planning logic that describes responses to goal failures with alternate procedural tactics for accomplishing a given goal, and externally in the operational procedures.

Physical safeguards such as relief valves or circuit breakers are commonly used as redundant backups to active controls to mitigate the risk that the active control system can itself fail, or fail to detect and respond to certain classes of faults, or when the active control system is unable to render a sufficiently immediate response. Redundant sensors, actuators, or other hardware devices can also be used by the active control system to dynamically "fail over" or otherwise take advantage of redundancy to automatically recover from failures in these devices. Although the active higher-level control system may play no part in the immediate failure response (i.e., opening a relief valve), it must be able to detect when a low-level response has been initiated to coordinate a system level response. Automatically opening a circuit breaker on a device may protect the device and the system from further harm of an over-current situation, but it may also leave the system incapable of completing its larger purpose. Responding to this condition can only be done at a higher system level.

In a goal-oriented system the achievers are control system elements responsible for achieving goals on individual system states through state variables. Achievers are responsible for actively controlling system state to keep the system in the state specified in the goal over the entire period of application (or, in the case of transition goals, to ensure that a required state transition occurs according to the time and state constraints specified in the goal). Goals can apply to very low-level system states such as the position of a switch or valve, or to very complex system states such as the representation of terrain over which a Mars Rover is driving. Goals can also apply to states that are not even controllable. As long as the system can estimate the state, it can apply a constraint and have a response to any violation of that constraint. For example, a hazardous machine might not be able to control the proximity of people to its hazard zone, but it may be able to detect when people are present and safe itself.

An achiever response to a goal violation is to try to correct the system state through the use of control actions on the target state, if possible. In cases where the target state is not controllable, or directly controllable, the situation may demand a higher-level coordinated response. The MDS delegates this response to a separate plan-level element whose sole purpose is to monitor the status of all currently executing goals and to coordinate a response when they fail. This separation enables the achievers to keep trying to achieve the goal state concurrent with any coordination needed to mount a higher-level response.

Coordinating a response requires knowledge of not only the individual goal, but the entire plan it is a part of. The plan records not only the particular sequence of goals, but also their elaboration associations. That is, when a goal requires other supporting sub-goals to achieve its purpose, it can specify those sub-goals and include them in the plan through a part of the planning process called elaboration. When a goal is failing, the system can call upon the parent of the failing goal (i.e., the goal that proposed or elaborated the failing sub-goal) to suggest a response [see 2,3 for further details]. Lacking a situation-specific response, the system will default to a coordinated safing operation. However, this mechanism also offers the possibility of delivering a response that might work around the failed device or subsystem, or other unexpected situation.

Furthermore, since goals specify intent into the future, the fault detection mechanism can use state projections to infer when a goal will not be achieved in time, allowing the system to respond proactively. Resource management is largely a task of projecting future resource usage according to a plan of activities in order to limit the activities to what can be safely completed within available resources. For example, the MER rovers on Mars can only collect and store a limited amount of solar energy each Sol. Every activity that needs to consume that resource needs to fit within the envelope of what was collected. Accordingly, plans are made based on projections of models, but these models feature varying degrees of uncertainty. During execution of the plan one does not want to wait until the battery is depleted to realize that the plan over-estimated the available energy. Goal-oriented planning allows the runtime system to continually re-evaluate the plan against the actual resource to determine at the earliest opportunity when the resource levels would prohibit plan completion, thus enabling the widest possible set of recovery options.

Responses at all levels can also include simple notifications to operators that system states are not as expected, are trending in the wrong direction, or are not likely to be achieved according to current state projections. Thus, safeguards and situation awareness are closely related functions. Safeguard constraints can help to formally specify those values of system states that are relevant to astronauts and ground controllers as an integral part of every activity.

VI. Enhanced Situation Awareness

Inadequate situation awareness has been identified as one of the primary factors in accidents attributed to human error, so it is important that any control architecture address this need directly. Situation awareness is defined as "the perception of environmental elements within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future" but can more easily be understood as "what you need to know to not be surprised"[14]. The term is most often used in more interactive systems where planning and projection extend only marginally into the future, and intent is rather immediate. The same principles generally apply to more extended planning.

Situation awareness is enhanced in a goal oriented control system through the explicit representation of state variables of the system. This is accomplished through explicit continuous monitoring of control goals and through explicit representation of time points denoting the beginning and end of goals. A user can thus be informed of the current estimated state of the system and receive an explicit indication as to whether or not those values are within context-sensitive constraints for the current activity. In this respect, goals are far more expressive than modal display parameters, because the constraints automatically change as part of the larger plan. Some of the operationally useful information a goal-oriented system can provide are listed in figure 2. Similarly, a user can be notified prior to or coincident with the initiation and/or completion of goals, thereby keeping users informed of normal progress, new activities, and important deadlines.

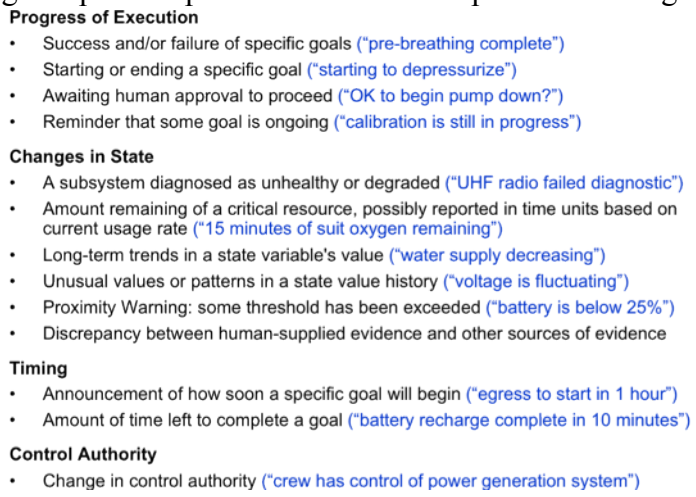

Figure 2 Notifications of System States

The Mission Data System's implementation of goals not only explains the "what" of a plan's intent, but it also explains why each goal is a part of the plan through an elaboration hierarchy. Thus, every goal retains a link to the goal or goals it supports. For example, a goal to operate an experiment can specify that it requires a certain amount of power – a supporting resource allocation goal on power – which is then elaborated into the plan. If some other goal also specifies a need for power the planner can not only detect a potential conflict, but it can then explain that it can't do the two activities at the same time because the combination would over subscribe available power. Thus, the power goals can explain why they are there, and the system can automatically infer the implications of a failure.

Fundamental to any control system's ability to keep operators' state knowledge accurate is the system's ability to express the uncertainty of that knowledge. Control systems *cannot* have absolute knowledge about the state of the system under control, which it can only sense indirectly. All electro-mechanical sensors are imperfect, and thus cannot provide a direct representation of the actual state of the system. In particular, their measurements tend to have limited precision and accuracy, which may vary over their limited sensitivity range. This is why all but the most primitive control systems use explicit state estimation to "process" measurement

information into a more calibrated and continuous representation that can then be presented to users or used in control algorithms. A common design flaw in such designs occurs when the estimator fails to express the uncertainty of the estimates it produces. An estimator that continues to produce "valid" estimates even when the sensor producing the evidence for its estimates has failed, or the value has exceeded its sensitivity range, are typical examples that could clearly lead the control system or an operator to make incorrect control decisions.

Traditionally designed control systems also rely on a common understanding of a plan, goals that extend into the future, and elaboration hierarchies, but because these are all are only implicit in the implementation they become details that operators and crew just have to keep in their heads, or look up in procedures, design manuals, and other documentation. The goal-oriented approach at least ensures that all of this information is coordinated through a common architectural approach.

One particularly tricky aspect of system state that is difficult to manage automatically has to do with who is "in control" of the system at any given time. In the next section we explore the issue of managed control authority.

VII. Managed Control Authority

Systems often have the ability to be commanded locally as well as remotely, necessitating operational rules and protocols to ensure that two different operators do not attempt to issue commands at the same time, or in a way that might conflict, leading to unintended consequences. Automated constraints enforcing control authority protocols can be implemented as a safeguard against this contingency. There are two distinct ways that these safeguards could be implemented: as a form of access control to the command interface, and as a more comprehensive resource manager.

The term "control authority" is typically used in control theory to describe the extent to which user control inputs have an effect on the desired output states, conceptually similar to control leverage. In the human spaceflight community this term has been extended to consider the situation that can arise when more than one user can simultaneously make control inputs to the same system. In situations such as this one's control authority can be diluted or otherwise changed as a function of other concurrent inputs. A simple example can help elucidate the situation.

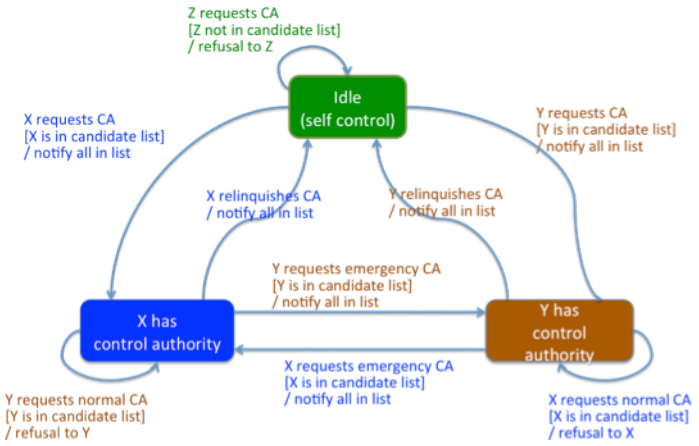

Figure 3 Access Control States

Consider the case of a thermostat that controls the temperature in a room.[†] You set the thermostat to your comfort level, expressing control intent. Later, someone else comes along

[†] The control system directly controls the heating and cooling devices that can affect the target state as a function of a control goal in the form of a desired target state. We say the system is controlling the target state using the available actuators.

and, finding the room too hot, changes it to fit their comfort level, thus establishing a new control goal. The heater blithely complies. What's the solution? Add a lock on the thermostat? If you have the only key, then you might be satisfied, but your hot colleague will be even hotter finding himself locked out. If you both have keys, you're right back where you started. Thus, access controls do not completely solve this problem, though they can certainly exclude inputs from people who don not have control privileges, and they can help to inform you about where the conflict lies. The ultimate solution is for you to *negotiate* a compromise with the other occupants, and set the thermostat to a temperature that all can accept. Figure 3 depicts a state machine describing the possible states of this mechanism for two users.

A negotiation process like this is an iterative process consisting of two repeated steps. First, each occupant proposes a goal in the form of an acceptable range of temperatures. If an intersection of the ranges exists, then that can be used as a merged goal, solving the problem. Otherwise, each proposer is informed as to how their goal differs from the others, and another round of proposals begins. The system might help advance the negotiations by offering the average of all proposed goals as a likely compromise. Note that this is not sufficient to guarantee that the process will converge. Occupants may have strong requirements for a particular temperature if they are handling food, for example, and this may lead to a stalemate, or the need for higher-level planning. A simple mechanism such as this does not require any strong authentication as long the operating environment is presumed to be cooperative. That is, as long as all participants who have physical control privileges are presumed to be trying to achieve a common mission goal, and the safeguard only needs to prevent accidental control conflicts. However, this mechanism could work in concert with a more elaborate security mechanism with strong authentication as long as the security mechanism does not prevent the ability of users to usurp control in emergency situations. That is, this mechanism should never "lock out" an appropriately authorized user from issuing commands needed to recover from an emergency situation. For example, should an astronaut holding control authority become incapacitated the system must allow someone else to take control.

A second form of control authority safeguard is a by-product of explicitly managing system states as resources. When a goal has been accepted and begins execution it expresses an explicit constraint on the target state of the system over the entire period of its execution. Should another overlapping and conflicting goal be proposed the planner would detect the conflict. This would be reported as a rejection of the second goal with an explanation that it conflicts with another goal. This mechanism does not necessarily prevent the second user from changing the plan, but doing so would require the other user to realize that the earlier plan was in effect and would have to be retracted or modified first. Access controls or priorities could also apply to this process to prevent one user from retracting the plan of another, but in a system where intent is presumed to be cooperative it is likely that just the notification of a conflict and extra steps to retract the previous intent should prove sufficient to ensure that coordinated control is maintained.

In our simple example we can consider a situation where one user has scheduled a goal to keep the temperature within a cool range. Most likely, this particular goal would be elaborated from a larger activity that explains its purpose. For example, a cool temperature might be required to support a science experiment. A second user proposing a warmer temperature goal would be informed that their goal conflicts with the previous goal, and so cannot be scheduled. Further, the user could discover that this earlier goal is associated with a science experiment, and so should not be removed. In an emergency the earlier goal could be retracted by cancelling the experiment activity of which it is an elaborated element.

Because these goals are associated as part of a coordinated plan, they cannot be removed or changed individually outside of the elaboration rules of the particular activity without changing the nature of the entire activity. Proposing ad-hoc changes to a plan, while possible, runs the risk of introducing new conflicts to a finely engineered plan. A system should discourage, but not entirely prevent this sort of ad-hoc plan modifications because even a well-engineered plan may encounter a situation its designers had not anticipated. However, when such ad-hoc changes are proposed, they should still be validated for conflicts and rejected if they directly conflict with other parts of the plan, or available resource projections, prior to execution.

Thus, the safeguard of this mechanism is that it ensures that all users can be completely informed about the intent of any plan the system is executing, and implicitly of other users who may have proposed it. This does not prevent other users from proposing conflicting goals, but it does provide a robust mechanism for detecting and resolving conflicts, and ensuring that the executed plan is achievable within the available resources.

VIII. Tunable Automation

A goal oriented control system such as MDS allows implementers and operators to select the level of automation to suit the given situation. Control logic is layered so that behavior can depend on the specific set of goals that are currently active. This feature enables operators to gradually gain confidence in the automated capabilities over time, or to lower the level of automation in response to unanticipated circumstances. By elaborating the sub-goal tactics that are used to compose larger goals, the MDS architecture enables those components to be tested separately, further facilitating incremental confidence building.

The use of automation is often seen as conflicting with qualities such as deterministic simplicity, a quality that simplifies the task of verifying correct implementation. While a goal-oriented control system may have more working parts, it is conceptually cleaner and more analyzable because it makes control intent explicit at all times. Not only does this enable more complete and comprehensive design time analysis, but it intrinsically supports continuous runtime verification.

Plan verification prior to execution is another key safety feature of a goal-oriented control system. A proposed goal is elaborated and scheduled, and then the plan is compared with the projected states of the system to ensure that every goal is actually achievable. Sometimes this can lead to plans that cannot be verified because the projection models become too uncertain, or because current state knowledge is too uncertain to project. In an emergency situation it may be worth the risk to try to execute a proposed recovery plan anyway, effectively waiving design-time safety restrictions.

A common problem in sufficiently complex control systems is that their design cannot be expected to counter the full range of situations they might meet in the real world. This is particularly true in space exploration systems, where encountering new environments is practically guaranteed. Plans must then be designed to anticipate the uncertainty of state knowledge and re-plan accordingly. If the environment is sufficiently stable and benign – for instance, a Mars rover driving across the surface – the plans need not be particularly robust, as it may be simpler and less costly to let plans fail when unexpected situations occur. The system may then respond by devising a new plan, or wait for the ground to respond using a more manual recovery procedure. In less stable situations, however, there may be no other choice than to design a robust plan.

Recovering from fault situations is often among the most complex control problems in part because these situations are likely to fall outside of normal design constraints. Of course, a robust control system has to anticipate potential device failures, but the design may only consider the most likely fault combinations. When some other fault combination occurs, operators may need the ability to work around the automation in order to affect a recovery, or to diagnose the problem. In the MDS this is accomplished by allowing users to issue commands directly to the system under control, thus bypassing the goal-oriented control system.

IX. Limitations

The ability of a goal-oriented system to verify plans prior to execution is fundamentally limited by the ability of designers to model the system so that future states can be predicted well enough to support validation. Models based on physical laws must inevitably propagate the uncertainty of initial conditions, and of the environment, including knowledge of disturbing forces, or failures that might occur. No physical model can provide perfectly accurate predictions, and uncertainty tends to accumulate over time. Thus, verification of planned goals against projected state knowledge must explicitly model and tolerate a level of knowledge uncertainty.

The most effective way to enhance the tolerance of future uncertainty is to take control system behavior and intent into account in the models. The uncertainty of most system states can only grow over time as sensors degrade and fail. However, if the control system is actively trying to control the state to keep it within a specified constraint – through the execution of a goal – the model can safely assume that the state will remain within the limits of the constraint to the extent that the control system can still estimate and affect control of the state using all available evidence. Thus, the projection models and the plan are used together to predict whether the plan is achievable. Physical models dictate whether one goal can follow another; for instance, can a robot be in one location and then another without allowing sufficient time and resources for a transition from one location to another. Even with the best rules, though, this validation cannot *guarantee* that the plan will succeed – it can only ensure that the plan is consistent with models, and at least *potentially* achievable.

A common reason stated for resistance to adopting new automation technologies is the resulting increase in complexity compared to more conservative or traditional solutions, as well as their potential to significantly increase the computational requirements of the system. It is often helpful to frame this discussion by separately addressing the *intrinsic* or *inherent complexity* of the mission and the automation it requires, and the *extrinsic* or *incidental complexity* that arises as the result of a particular choice or constraint on the implementation of said automation solutions.

The intrinsic complexity of the automation technologies that have been discussed in this paper is primarily derived from uncertainty in the physical and environmental models that form the basis of the system design, as well as the uncertainty of planning future activities based on incomplete and imperfect state knowledge. In other words, this uncertainty that compels complex solutions is intrinsic to the nature of missions of exploration. However, the issue of how to deal with uncertainty in its various manifestations is not a new challenge: traditional engineering approaches have had to overcome such challenges in the past; we propose a methodology that makes the management of these uncertainties more explicit. Much as this is not a new challenge, we note that we may make use of many of the same types of analyses that have traditionally been

applied – for instance, application of design margins or safety factors and probabilistic risk quantification and reasoning techniques – to help address this challenge.

The extrinsic complexities associated with the techniques discussed above are not dissimilar to those associated with the implementation of any large and complex software-intensive system. Choices of language, hardware, and supporting elements will always impose some constraints on the final product. The story is no different here. While it may be tempting to suggest that the explicit modeling of uncertainty, as discussed in the previous paragraph, is simply introducing new and unnecessary complexity to already complex software systems, we again suggest that our approach simply makes the uncertainties inherent in the system and its operational environment more explicit. By doing so, we broaden the space of possible solutions, allowing for informed trades of risk versus efficiency to be made – and perhaps even tailored over the course of a mission.

A common perception is that a goal-oriented control system is likely to require more software that a traditionally designed system, thus expositing it to a higher risk of latent software flaws. A goal-oriented system will involve more software than a traditional design, but several aspects of goal-oriented design will help to mitigate this risk. A comparison between a goal-oriented control system capable of executing coordinated plans and a traditionally designed system that uses command sequences or scripting, shows that a goal network, and its elaboration logic, will generally require less logic than the corresponding script because it can depend on a large amount of highly verifiable framework. The logic used to elaborate, schedule, and verify a goal-based plan is largely similar to the tools currently used to verify command sequences. Indeed, historically, they have the same roots. The notable difference between the automated planning logic and the traditional sequence verification logic is that all of the planning logic used in the traditional sequencing system is likely to be in the ground element of that system, whereas with a goal-oriented system, it would likely be onboard, though this is not a requirement. This reallocation of functionality from ground segment to flight segment, plus the associated planning engine that drives this logic, is perhaps the largest contributor to onboard code size growth. The main tradeoff is that having explicit expressions of intent through the planning process enables the planning and verification code to be much simpler than it might be in a traditional system.

While the goal oriented system may require more code to implement, the risk of failure due to software fault is mitigated by the fact that the system can know at all times what the desired outcome is, and make active corrections to continually attempt to achieve that desired state. Systems that rely on instantaneous effects from a transient command to change state and which assume that the commanded state will persist until commanded otherwise are not capable of such continual achievement and reinforcement.

While in general all safety-related conditions should be monitored continuously, it is reasonable to assume that in some cases this continuous monitoring may not be necessary. The decision to continuously monitor a given state of the system should be determined by its bearing on the overall safety of the system, and its relevance to other critical coordinated states. Importantly, we note that the solutions discussed in this paper allow for a wide range of configurations.

Lastly, goal-based systems allow for more tailored, scenario-dependent fault response and detection logic than is possible with more traditional systems. While this allows for a richer variety of fault response behaviors, it also implies that there is likely more logic to verify and test these responses. However, as the approach described in this paper integrates handling of both nominal and off-nominal behaviors into a single coherent framework – as opposed to some other

approaches which rigidly segment the handling of nominal and faulty behaviors – the ability to exploit commonality of implementation between the two is significantly increased, resulting in less additional effort, and higher-quality implementations.

X. Conclusion

The definition of human-rating for spacecraft has been refined over the years to encompass three key concerns: safety, the accommodation of human capabilities (human factors), and the provision of services for life support[9,12]. Standards have been established for the minimum set of resources and services that must be provided to support astronauts during a space mission, as well as defining limits on expectations for human performance of various tasks. Similarly, minimum standards of safety performance have been established for various types of subsystems, yet the overall methodology for human rating has been described as a process through which the actual risks and benefits of each mission are elaborated and refined in order to define a mission that is both compelling and affordable. Thus, while many specific "unsafe" risks are prohibited by various standards, there is no clearly established over-arching standard for what is sufficiently safe. Furthermore, while the reliability and safety of individual mechanisms, subsystems, and devices are important contributors to system safety they do not by themselves ensure overall system safety. System safety requires additional coordination across the states of the system, and architectural guidelines or principles can begin to address this omission.

We believe that more ambitious remote exploration missions of the future will be made possible only through the use of advanced automation and autonomy. For these systems – and, importantly, the autonomy they will embody – to be viewed as safe and reliable, they will have to be designed *from the start* to be safe and reliable.

The architectural qualities and mechanisms described in this paper begin to define the types of mechanisms that will be needed for future manned exploration missions. Although added automation is likely to increase the implementation complexity of some control systems, the intent should be to lower the *conceptual complexity* of the systems, as perceived by astronauts and other system operators, to enable increased autonomy, and reduce the number of operational constraints and rules that operators have to keep in their heads while operating these systems so that their time can be more efficiently and safely dedicated to exploration.

Acknowledgments

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

References

[1]Wagner, D., "Mission Data System," [online reference website], URL: http://mds.jpl.nasa.gov [cited 10 December 2009].

[2]Dvorak, D., et-al, "A Unifying Framework for Systems Modeling, Control Systems Design, and System Operation," *The International Conference on System, Man and Cybernetics*, ISBN: 0-7803-9298-1, Vol. 4, IEEE, Kona, HI, 2005, pp. 3648-53

[3]Dvorak, D., et-al, "Goal-Based Operations: An Overview," *Proceedings of the AIAA Infotech@Aerospace Conference*, AIAA-2007-2724 , May, 2007.

[4]Wagner, D., et-al, "An Architectural Pattern for Goal-Based Control," *Proceedings of the Aerospace Conference 2008*, ISBN: 978-1-4244-1487-1, IEEE, Big Sky, MT, 2008, pp. 1-17

[5]Dvorak, D., et-al, "Comparison of Goal-Based Operations and Command Sequencing*," SpaceOps 2008 Conference*, AIAA-2008-3335, May 2008

[6]Rasmussen, R., et-al, "Achieving Control and Interoperability through Unified Model-based Systems and Software Engineering," *Proceedings of the AIAA Infotech@Aerospace*, AIAA-2005-6918, September, 2005.

[7]Ingham, M., et-al, "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," *AIAA Journal of Aerospace Computing, Information and Communication*, Vol. 2, No. 12, AIAA, Reston, VA, 2005, pp. 507-536

[8]Dvorak., D., "NASA Study on Flight Software Complexity ," *Proceedings of AIAA Infotech@Aerospace Conference 2009*, AIAA-2009-1882, April 2009.

[9]NASA, "NASA Procedural Requirements (NPR) Human-Rating Requirements for Space Systems," NASA NPR-8705.2B, 2008.

[10]Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*. Second Edition. Addison-Wesley, Boston, MA. 2003.

[11]Wagner., D., et al. "A Control Architecture for Safe Human-Robotic Interactions During Lunar Surface Operations" *Proceedings of AIAA Infotech@Aerospace Conference 2009*, AIAA-2009-1988, April 2009.

[12]Zupp, G., A Perspective on the Human-Rating Process of US Spacecraft: Both Past And Present, NASA SP-6104, 1995.

[13]Augustine, Norman R. (chair), "Review of Human Spaceflight Plans Committee", http://www.nasa.gov/pdf/396093main_HSF_Cmte_FinalReport.pdf

[14]NASA, *NASA Software Safety Guidebook*, NASA Technical Standard, NASA-GB-8719.13, March 31, 2004