

Achieving Control and Interoperability through Unified Model-based Systems and Software Engineering

Robert Rasmussen,^{*} Michel Ingham,[†] and Daniel Dvorak[‡]

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

Control and interoperation of complex systems is one of the most difficult challenges facing NASA's Exploration Systems Mission Directorate. An integrated but diverse array of vehicles, habitats, and supporting facilities, evolving over the long course of the enterprise, must perform ever more complex tasks while moving steadily away from the sphere of ground support and intervention. Interoperability needs will grow to unprecedented levels as systems become more dependent on one another than on support from home. Accomplishing this with consistent safety and reliability calls for a long-term strategy. This paper describes the control challenge faced by future exploration systems and outlines a realistic approach to solving it, based upon a unified, principled architectural approach to both software and systems engineering. It concludes by suggesting the steps necessary to put this capability in place for exploration systems.

Nomenclature

<i>EDL</i>	=	Entry, Descent, and Landing
<i>ESMD</i>	=	Exploration Systems Mission Directorate
<i>JPL</i>	=	Jet Propulsion Laboratory
<i>MDS</i>	=	Mission Data System
<i>NASA</i>	=	National Aeronautics and Space Administration
<i>OI</i>	=	Orbital Insertion
<i>RV&D</i>	=	Rendezvous and Docking

I. Introduction

Control and interoperation of complex systems is one of the most difficult challenges facing NASA's Exploration Systems Mission Directorate (ESMD), as it embarks on its quest to send robot and humans to the Moon, Mars and beyond.¹ An integrated but diverse array of vehicles, habitats and supporting facilities, evolving over the long course of the enterprise, will need to perform ever more complex tasks while moving steadily away from the sphere of ground support and intervention. Moreover, interoperability needs will grow to unprecedented levels as systems become more dependent on one another than on support from home.

Controlling the types of complex systems that are needed to achieve the ambitious objectives of the ESMD will require solutions to multiple problems:

- *Managing a large collection of components that are tightly coupled* through shared resources, operational conflicts, and many other sources of interaction.
- *Enabling operation in uncertain or remote environments*, with no immediate appeal to ground support due to long light time lags and irregular communication.
- *Responding appropriately to anomalies*, with the help of integrated diagnosis and recovery mechanisms that may be fully automated or involve humans-in-the-loop.
- *Realizing affordable operations cost* by harnessing automation in such a way that it is considered an aid to operations, not an obstacle.

^{*} Chief Engineer, Systems and Software division, 4800 Oak Grove Dr., M/S 301-225.

[†] Senior Software Systems Engineer, Flight Software and Data Systems section, 4800 Oak Grove Dr., M/S 301-225, AIAA Member.

[‡] Principal Engineer, Systems and Software division, 4800 Oak Grove Dr., M/S 301-270.

- *Assuring system correctness and reliability* by closing the gap between how systems are specified and how they are implemented.

Similarly, integration and interoperability of these complex systems will involve additional hurdles:

- *Communicating effectively* across a distributed system of systems.
- *Collaborating with confidence, robustness, and flexibility* based on shared intent.
- *Coordinating human and robotic activities*, safely and productively.
- *Managing a large, distributed engineering effort* involving NASA, industry, academia and international partners.

While there is no single solution to all of these problems, it is possible to devise a unifying framework and long-term strategy that can effectively guide the overall effort. We believe such a framework is crucial to the success of ESMD's undertaking, and involves the following essential elements:

- *State- and model-based control architecture*: Shared information about state and models of state should be the basis for collaboration within and among systems to exchange knowledge, express intent, and accommodate new capabilities.
- *Unified systems and software engineering*: With a unified approach, the functional requirements written by systems engineers can be translated more directly and unambiguously into implementation, which is in turn more amenable to verification.
- *Design for integration, reuse, and evolution*: Software design should produce a set of control components that promote easier reconfiguration and reuse by separating common capabilities within the architecture from those specific to particular applications.
- *Processes to assure quality and timely deliveries*: The uniform structure afforded by the framework should lend itself to spiral development and other process improvements that will be necessary to assure reliable and affordable products.

Section II of this paper describes in greater detail the control challenge faced by future exploration systems. Section III outlines a framework for solving this problem, based upon a unified, principled architectural approach to both software and systems engineering. An implemented instance of this framework, the Mission Data System, is presented in Section IV, and its relevance to the ESMD initiative is discussed in Section V. Finally, Section VI suggests the next steps necessary to put this capability in place for NASA's future exploration systems.

II. The Control Challenge Facing Exploration Systems

Controlling a system of systems that can carry people to the Moon, Mars, and beyond will be well beyond present experience. The exploration initiative will impose significant demands on system capability, safety, operability, and affordability.

Capability: Although robotic systems will continue to survey distant places on their own, with complex automation to enable their tasks and preserve their safety, NASA's new exploration systems will be asked to do far more. These worlds are vast, so to study them well, our machines will need to be controlled ever more efficiently, and to leverage new capabilities that have heretofore only existed in the realm of science-fiction.

Safety: Likewise, manned vehicles and habitats will still provide safe and productive homes for their crews, but new exploration systems will need to do this in more remote dangerous environments beyond the easy reach of ground operators. Given the unprecedented level of complexity of these exploration systems, the first line of defense against hazards will have to be automated or semi-automated control functions.

Operability: We can also anticipate a continually growing dependency between astronauts and interactive robotic systems, as they work together to build and explore. To be an astronaut's dependable partner, automated systems must be watchful, responsive, and reliable. Furthermore, they must present an interface that is amenable to human interaction.

Affordability: Achieving the requisite advances in the above three areas will obviously impose a significant strain on the resources available to the exploration initiative. Enabling these advances while staying on schedule and within budget will require us to adopt streamlined processes and develop reusable and adaptable product lines.

We must ask ourselves whether present methods of system control can scale to meet these critical demands. This is not a question answered simply by an appeal to advancing technology, because other critical factors also affect how well a system works. Rather, the question requires penetrating examination of a variety of issues, and a conviction to put in place the appropriate infrastructure, develop the requisite expertise, and set the stage for a successful exploration enterprise.

A. Control of Complex Systems

As previously mentioned, the new systems required for this enterprise will be complex, many of them extraordinarily so, compared to present day systems — they will adopt a wide array of forms and functions, continually accommodating new requirements as the program develops. Time- and mission-critical control activities like orbital insertion (OI), rendezvous & docking (RV&D), and entry, descent & landing (EDL) cannot be fully scripted in advance, because of the high level of execution-time uncertainty. Things can go wrong under the best of circumstances, especially during surface and atmospheric operations. Relying on time-sequenced or human-in-the-loop control will be impractical for many situations and impossible for others. Yet reliability must not be compromised. Automated functions that step in to make critical decisions must consistently do the right things right. This is complicated by a number of difficulties.

1. Managing large numbers of tightly coupled elements

The control of complex systems would be easier if it were possible to simply divide the problem into many small independent pieces. However, space systems have never enjoyed this indulgence. Every last resource that might ever be needed must be brought along or manufactured in situ, yet designs are inevitably constrained to be highly compact and efficient. The result is a large collection of components that are tightly coupled through shared resources, operational conflicts, interference, fault propagation, environmental effects, and many other sources of interaction.²

We know already that classic methods of managing such complexity are likely to break down. For instance, conventional hierarchies can actually become an obstacle to successful control since coupling has an unruly tendency to defy normal lines of authority. System integration is largely an exercise in managing the coupling that functional decomposition fails to eliminate, and this is notoriously error prone.

2. Enabling operation in uncertain or remote environments

In one way or another, active control of most systems exploits a dependable characterization of their environment. Whenever possible, a comparatively simple and static environment is created for them, the exceptions generally requiring human intervention. Most space systems exploit these characteristics as well; otherwise proceeding only with extreme caution. For example, whereas a Mars orbiter can map the surface continuously with little intervention, a relatively straightforward Mars rover activity will often take days to unfold, as ground operators ponder each step.

New exploration systems will push the envelope of this notion to its extreme. Elaborate crewed systems will be required to operate in strange, complex environments, but with no immediate appeal to ground support due to long light time delays and irregular communication. Operation of such complex systems will be beyond what the crew alone can manage. Robotic systems will likewise be pressed to accelerate their activities, whether for more efficient independent operations, or in support of human operations, where unhurried action may be inappropriate.

When neither prescribed environments nor human involvement are options, new methods of control must be devised. It will take time for this capability to mature. Yet serious excursions into such an operational model have been studiously avoided in most programs to date.

3. Responding appropriately to anomalies

In many respects managing the behavior of a system when something goes wrong is like operating in an uncertain environment. There is a long history of fault protection in space systems, especially in launch and interplanetary vehicles where required reaction times necessitate automation. An understanding has developed over the years of how important it is to consider fault protection as an integral part of the design process.^{3,4} Nevertheless, fault protection is still considered for the most part to be a separable design activity, and typical fault protection designs reflect this division, often being implemented as adjuncts to separate base systems.

When potential anomalies and uncertain environments must be dealt with together, this separation becomes, not just unsuitable, but downright dangerous. Failing to unmask the effects of a fault or other unexpected behavior within an obfuscating environmental context has already led to mission losses.

4. Realizing affordable operations cost

Automation will be a necessity of new exploration systems, not just to address the control issues cited thus far, but to contain the cost of operations, as well. Nevertheless, designing highly automated systems without strong input from their operators has historically been a recipe for expensive and potentially unsafe operations. A common complaint is the operational complexity of systems, and ironically, operators frequently cite automation as an obstacle to success, not an aid. The essence of this criticism is that poorly executed automation can be unreliable or can rigidly obstruct activities. Unless automation is applied in a principled, reasoned fashion, it runs a high risk of reducing complexity in one area only by introducing self-defeating complexity of its own.

5. *Assuring system safety and reliability*

Safety and reliability are defining characteristics of a well-engineered system. Whether to preserve mission assets or avoid costly setbacks, safety must be a central consideration — and of course, there is no excuse for failure when human safety is involved. Automated control will be an enabling capability for exploration systems, especially in service of their safe operation, but if executed poorly it merely offers new ways for systems to fail.

Success depends on correctly specifying, implementing, verifying, and operating these control functions. Glitches in this flow of development have compounded small issues into mission-ending failures. The gap between how systems are specified and how they are implemented has historically been wide, and much of what transpires in software development remains opaque.⁵ Verification and operation both fall victim to this gulf. Prevalent techniques today are already strained and certainly no match to the challenges of new exploration systems. Marked improvement in this area is called for if we are to avoid a potentially bottomless investment sinkhole.

B. Integration and Interoperation of a System of Systems

The number and variety of systems contemplated for the exploration enterprise is very large. This will not be a scattered fleet of individual vehicles, operating in near independence. Rather, each system will work in close harmony with several others, as configurations are assembled and deployed in space and on planetary surfaces, as they communicate, rendezvous, exchange samples and crew, perform collaborative surface exploration, and so on.

Therefore, in addition to managing the complexity of individual systems, control must also deal with how systems interoperate, working together as an integrated whole. This is not merely a replay of the issues above at a larger scale. For very practical reasons, one can expect within any single system a degree of cohesion and design uniformity that will not necessarily exist within the larger system of systems. Reliable interoperation of diverse automated systems adds another layer of issues.

1. *Communicating effectively*

In a distributed system of systems, the nature of the dialogue among system elements is paramount. Communication is not merely a medium established by data links and protocols. Rather, it is a conversation, informed by mutual knowledge, motivated by mutual interests, guided by mutual conventions of interaction, and resulting in mutual understanding. Fundamentally, it is what the participants have to say and their ability to comprehend one another that really matters.

Despite the transcendence of dialogue over mere data exchange, this level of communication rarely gets the attention it deserves. The consensus required for genuine dialogue often goes unwritten and unverified. Moreover, establishing this understanding in the first place is generally viewed as something to be accomplished entirely during system design. Complex systems, cooperating in complex environments and needing to arrive at a safety-critical consensus, may not have this luxury. Thus, the ability to communicate, not just knowledge, but also the means to interpret this knowledge must be considered.

2. *Collaborating with confidence, robustness, and flexibility*

Control decisions within a system of systems cannot be purely hierarchical. There must be a shared sense of purpose. But an appeal to central authority is simply too brittle — especially when control is distributed across the solar system. For robustness, each system must be capable of making proper local decisions, informed by the intent of their principals, but with the flexibility to accomplish this intent as circumstances allow, and to choose intelligently among the objectives when not all are workable. Furthermore, systems should be able to shift responsibilities to others, as efficiency or operational exigencies demand.

As with effective communication, the key to robust collaboration is sharing. In this case it is shared intent: the ability not just to respond to direction, but to appreciate the ultimate objective and importance of that direction, in order to act on it effectively and to make correct judgments about problems in accomplishing it.

There is little experience with direction of this sort in current systems, where blind obedience is the general rule. Systems are expected either to do as they are told, or to resort to safe operation while their operators figure out how to recover. Critical scenarios (like OI, RV&D, and EDL), where this is not an option, have generally been handled by highly tailored programmed behaviors, usually unique to the situation. This has been expensive and troublesome to engineer. Extrapolated to the scope of new exploration systems, compounded by the degree of collaboration involved, and extended to continual critical operation without ground support, the approach of the past is simply not an option.

3. *Coordinating human and robotic activities*

We tend to see the relationship between robotic systems and humans in an essentially one-sided manner: machines operated by people. In certain important ways though, the interaction between them is similar to what we imagine among collaborating robotic systems themselves. Communication must be effective, exploiting mutual understanding in order to avoid a flood of data. Direction must be flexible, disencumbering the “directing” agents

from the chore of saying *how* a job is to be done, letting them concentrate instead on *what* needs to be done. In complex situations, therefore, it is best to view the relationship between robotic systems and humans, whether ground operators or astronauts, as collaboration.

To date, human and robotic activities have not generally shared this sort of relationship. The association has been more in the nature of remote control than collaboration. Changing the nature of this relationship isn't simply a matter of efficiency or comfort. Automation serves best when it relieves humans from the tasks for which they are least suited — where they are most likely to make errors in perception or judgment — but honors the human's superior command of the big picture and of complex relationships. Each should rely on the other in a complementary fashion. Thus for true collaboration, the nature of interaction between humans and the system of systems must be a dialogue too.

4. Managing a large, distributed engineering effort

Perhaps the most daunting aspect of the new exploration system of systems is its extent in so many directions. Development will be distributed across NASA, industry, and academia, as well as around the world. Operation will be distributed across the Solar System. Design evolution and extension will be distributed across several decades. And yet system elements must be integrated in a manner that spans all of these dimensions.

Whenever an engineering effort extends so far, there are bound to fundamental competing considerations. There will be competition between practicality of independent development and the need for deft interoperability. There will be competition among a broad mix of operating models in ever-changing control realms. There will be competition between permitting the graceful replacement of obsolete legacy capability and enabling the potential to adapt and evolve easily.

A reasonable solution to this predicament will not emerge on its own, even from a wisely chosen collection of virtuoso contributions. To meet this challenge, new exploration systems will need:

- a mandate for shared technical foundations that can stand the test of time,
- flexibility to grow and support continuous upgrade,
- versatility to accommodate the inevitable surprises and changes in direction along the way, and
- sustained compatibility with commercial and industrial capabilities over the course of the enterprise.

These have often been lacking in space projects, even on a smaller scale.

III. A Framework for Solutions

In order to address these challenges, it necessary to devise a unifying framework for solution that can effectively guide the overall effort. We believe such a framework is crucial to the success of ESMD's undertaking. In this section, we identify a number of essential elements of this framework, providing the necessary structure to shape solutions in an organized and disciplined manner.

Achieving the ambitious objectives of the space exploration enterprise will require more than just a technological framework. Technical aspects of the structure are important, but the framework must also address the context around the effort: in management, systems engineering, software engineering, operations, and so on. The key to any framework, therefore, is a guiding principle that unites all aspects of the problem at hand.

The issue of concern addressed here is control, but control is an overarching concept that subsumes all other aspects of automation. Other functions are not useful unless they can be operated in the manner desired, i.e., unless they can be controlled. Appropriately specifying the control functions requires an understanding, not just of what all the other functions do, but also of how those functions interact with one another, how they interact with the environment of the system, and how the environment itself can be expected to behave. Remarkably then, in the business of control lies the whole problem of system understanding. This suggests the principle we need.

System understanding involves knowing how a system changes from moment to moment, knowing why these changes have occurred, anticipating changes in the future, and knowing how to influence those changes. The fundamental way this sort of thing has been handled since the time of Newton is through the notions of state and models of state. We can apply these notions to define a multifaceted approach to the required framework.

A. State- and Model-based Control Architecture

An architecture that elevates state to a central role is key to control. More specifically, it is the state of the system under control (and models of this state) that deserves attention. And yet despite the longstanding importance of state and models to engineering, their use in systems has for decades been a fairly parochial affair within disciplines, unevenly applied in general, and often exercised without mindful regard. Many architectures fail to see it as anything but a local concern within algorithms or at ad hoc interfaces. It is given the same architectural stature as measurements, directives, computational modes, and so on. Models of system state are given even less stature, often

totally obscured within a design, if present at all. Such informality in implementation has been somewhat compensated by traditional formality in systems engineering methodology which captures notions of state and models (among other things) in the indirect style of requirements hierarchies.

An architecture that emphasizes system state corrects this oversight. The idea is to make shared information about state and models of state the lingua franca for collaboration within and among systems. State would be used to exchange knowledge, express intent, and accommodate new or altered capabilities. A few simple architectural rules can then be imposed to assure the consistency of this shared understanding. Such an architecture can:

- Give control software a more verifiable structure, informed by system models;
- Better assure that automated systems are fully cognizant of the systems they control, including all their internal couplings, their fault behaviors, and their environment;
- Express all operational intent in a form (constraints on state) that greatly improves system operability, enables continual objective assessment of all performance, and facilitates automated reasoning across the whole system; and
- Promise a more coherent response to situations within and among systems, according to the expressed intent.

B. Unified Systems and Software Engineering

State and models of state have long been a dominant concern of systems engineers. This information reflects everything the engineers understand about how the system works, the conditions under which it operates, what it is supposed to do, and how it can fail. Therefore, a central role for state and models of state in control software architecture parallels a virtually identical set of concerns in systems engineering. This provides a golden opportunity to unify these disciplines as never before. By adopting a state- and model-based control architecture, we inherit a common language for systems and software engineers to communicate, and bridge the gap between what systems engineers say they want and what software engineers try to deliver. We will finally be able to close the book on a long history of miscommunication in this area, and thus reduce the number of software errors resulting from misinterpretation of the systems engineer's intent. With a unified approach to systems and software engineering, the functional requirements that systems engineers write can be translated far more directly and unambiguously into implementation, which is in turn far more amenable to objective verification.

Furthermore, since operational intent is expressed overtly and objectively in this architectural model as constraints on state, operational requirements from the highest to the lowest levels may also be treated more directly by implementers, and the resulting system may be largely self-monitoring in its operation.

C. Design for Integration, Reuse, and Evolution

In a state- and model-based design, the structure of the software is directly informed by models of the system under control. This structure suggests the natural decomposition of control software into units of functionality that are individually specifiable by virtue of their focus on a particular aspect of system behavior (i.e., on particular states), as opposed to the traditional lines of compartmentalization (e.g., flight vs. ground vs. test, or power vs. thermal vs. propulsion, etc.). Most notably, however, the resulting arrangement goes well beyond mere functional partitioning. Exclusive responsibilities can be assigned and all relevant interactions identified according to simple rules guided by the system model. In consequence (and in contrast with the typical improvised boxes-and-lines approach to architecture), the product is a crisply defined, formal composition. This is a boon to complex system development because complex interactions are acknowledged in the decomposition and managed directly at an architectural level rather than treated as special cases.

This natural makeup also results in a set of control components that promote easier reconfiguration and reuse. Such structure is highly amenable to a design approach (commonly known as component-oriented design) that facilitates the construction, analysis, instrumentation, and execution of software by making all interactions among components overtly manageable.

Another benefit achieved by this structure is to help separate common capabilities within the architecture from those specific to the system at hand. These common capabilities (planning, scheduling, data management and transport, sequencing, time-keeping, and so on) can all be swept into generic framework components to serve as a common foundation for many applications.

The same state- and model-based architecture at work within each system can also be applied among systems. The exchange of state knowledge and intent from one system to another can be defined independent of the particular implementations within each system, thus addressing one of the major concerns of integrated development among an array of partners. Uniformity of semantics among systems can significantly ease peer-to-peer interoperability issues. Such interfaces also allow for the superposition of layered but dynamically reconfigurable control contexts, as appropriate, truly realizing the potential of controlling a system of systems.

D. Processes to Assure Quality and Timely Deliveries

The uniform structure afforded by this state- and model-based architecture lends itself well to many process improvements that will be necessary to assure excellent but affordable products. To begin with, it enables a more formal application of principles for reliable design at all levels: what can be daunting in a more ad hoc approach becomes methodical and rigorous within a highly regular organization. Moreover, in the approach outlined here the underlying model of the system under control directly informs the design process, and the models themselves are subject to straightforward comparison to reality, thus exploiting the relationships among system and control elements.

Regular structure also aids the overall management of software development. In this approach, there is an easily traceable flow from system and operational plans, to control software requirements, to software modularity and design, to component implementation, and finally to verification and validation, where results map easily back to system objectives. Thus, the whole method easily accommodates an iterative incremental process of both software and systems development, while fostering continual integration, which is essential for managing complexity.⁶ The same regular structure makes it straightforward to overlay rigorous but responsive workflow management throughout the entire life cycle, and to extract dependable performance and cost metrics that can be applied directly to subsequent iterations at all levels.

Now that we have described a framework for addressing the challenges of controlling complex systems and integrating systems of systems, we describe a specific systems and software architecture that has resulted from our efforts to instantiate such a framework and put it into practical use.

IV. The Mission Data System

In recent times there has been a substantial growth in interest in a more formal and direct introduction of state and models into system implementations. This has been largely motivated by the advantages cited in the framework for solution above, but the advances have been primarily at the component level. To realize their full promise, these developments must mature in the context of an overarching framework of the sort described above — one spanning the spectrum of needs from architecture to process, systems engineering to software engineering, design to operation, and so on.

A prototype of such a state- and model-based architecture, including associated systems engineering processes and development tools, is available today. It has been in development for several years as part of the Mission Data System (MDS) project at JPL, having been refined and demonstrated on test systems that include physical research rovers (Rocky 7 and Rocky 8) and a simulated EDL system for the Mars Science Laboratory mission, all built from a common software framework embodying the principles advocated here.

In this section, we present a brief overview of the model-based systems engineering methodology used by MDS, called “State Analysis”, followed by an introduction to the MDS software architecture.

A. State Analysis

State Analysis is a systems engineering process that improves on the current state-of-the-practice by producing requirements on system and software design in the form of explicit models of system behavior. A more complete description of State Analysis is presented in reference 5. Specifically, it provides a uniform, methodical, and rigorous approach for:

- discovering, characterizing, representing, and documenting the states of a system;
- modeling the behavior of states and relationships among them, including information about hardware interfaces and operation;
- capturing the mission objectives in detailed scenarios motivated by operator intent;
- keeping track of system constraints and operating rules; and
- describing the methods by which objectives will be achieved.

For each of these design aspects, there is a simple but strict structure within which it is defined: the state-based control architecture (also known as the “control diamond,” shown in Fig. 1). The architecture has the following key features:⁷

- *State is explicit:* The full knowledge of the state of the system under control is represented in a collection of state variables. State knowledge is updated in the form of continuous-time State Functions, to accurately reflect the fact that the system’s true state is defined at any point in time.
- *State estimation is separate from state control:* Estimation and control are coupled only through state variables. State estimation is a process of interpreting measurements and monitored commands to generate state knowledge; this process may combine multiple sources of evidence into a determination of state, supplied to a

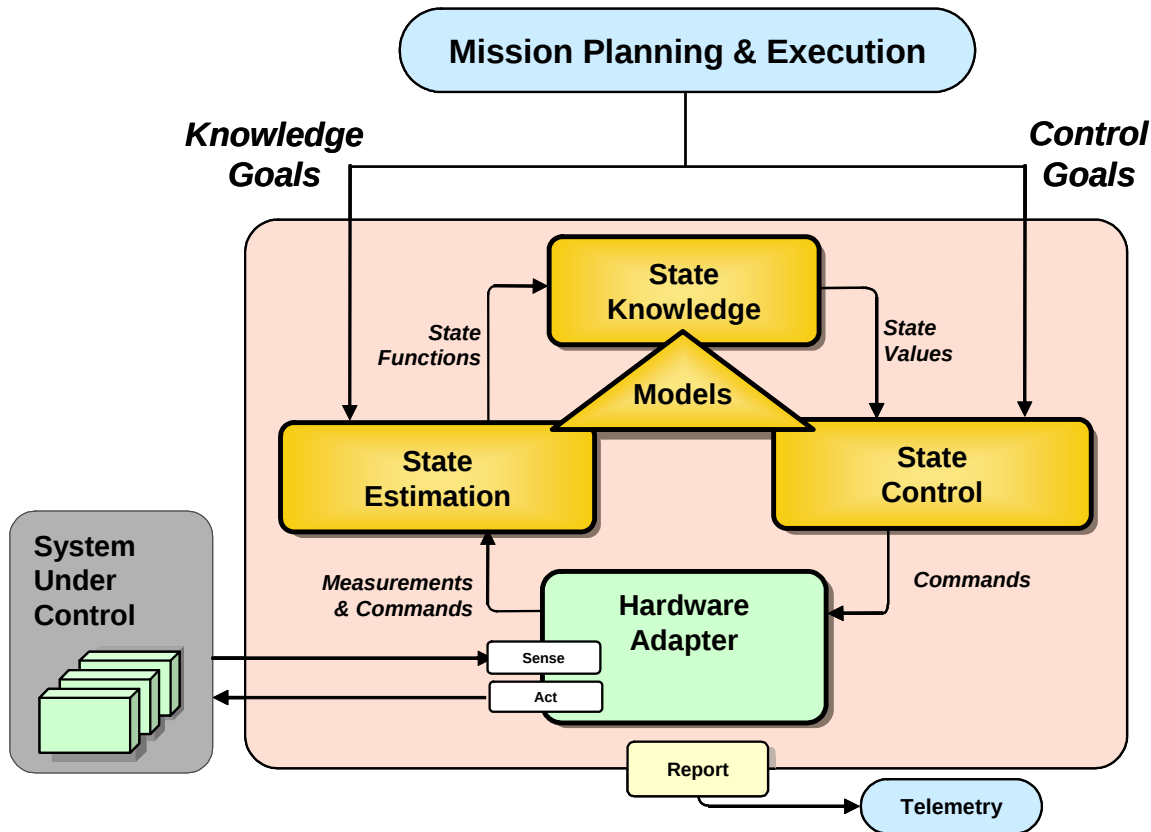


Figure 1. MDS State-based Control Architecture

state variable as an estimate. Control, in contrast, attempts to achieve objectives by issuing commands that should drive estimated state toward desired state. Keeping these two tasks separate promotes objective assessment of system state, ensures consistent use of state across the system, simplifies the design, promotes modularity, and facilitates implementation in software.

- *Hardware adapters and data collections provide the sole interfaces between the system under control and the control system:* They form the boundary of our control architecture. Hardware adapters provide all the measurement and command abstractions used for control and estimation of physical states, and are responsible for translating and managing raw hardware input and output. Measurements can be used both as evidence for estimating the state of the hardware in the system under control (e.g., accelerometer, switch position, and temperature sensor measurements), and for holding science observations (e.g., camera images and spectrometer readings). The control system can directly inspect the data collections to determine the state of the system data. Commands are directives that change the state of the system under control; these can be hardware commands (e.g., switch open/close and device operating mode commands) or data commands that are used for managing the data collections (e.g., data compression and data transport commands).
- *Models are ubiquitous throughout the architecture:* Models are used both for execution (estimating and controlling state) and higher-level planning (e.g., resource management). Whether overt and explicit, or hidden quietly in the minds of the engineers, models have always existed, since understanding and modeling are essentially the same thing. The key is that State Analysis requires that the models be documented explicitly, in whatever form is most convenient for the given application.
- *The architecture emphasizes goal-directed closed-loop operation:* Instead of specifying desired behavior in terms of low-level open-loop commands, State Analysis uses *goals*, which are constraints on state variables over a time interval. Goals are easier to specify than the actions needed to achieve them, and result in more compact specifications of desired behavior. Furthermore, goal-directed operation goes hand-in-hand with closed-loop control, because goals can be thought of as set points for onboard controllers, which are then given the latitude to decide how best to achieve the goals. In our architecture, goals are also used to specify the

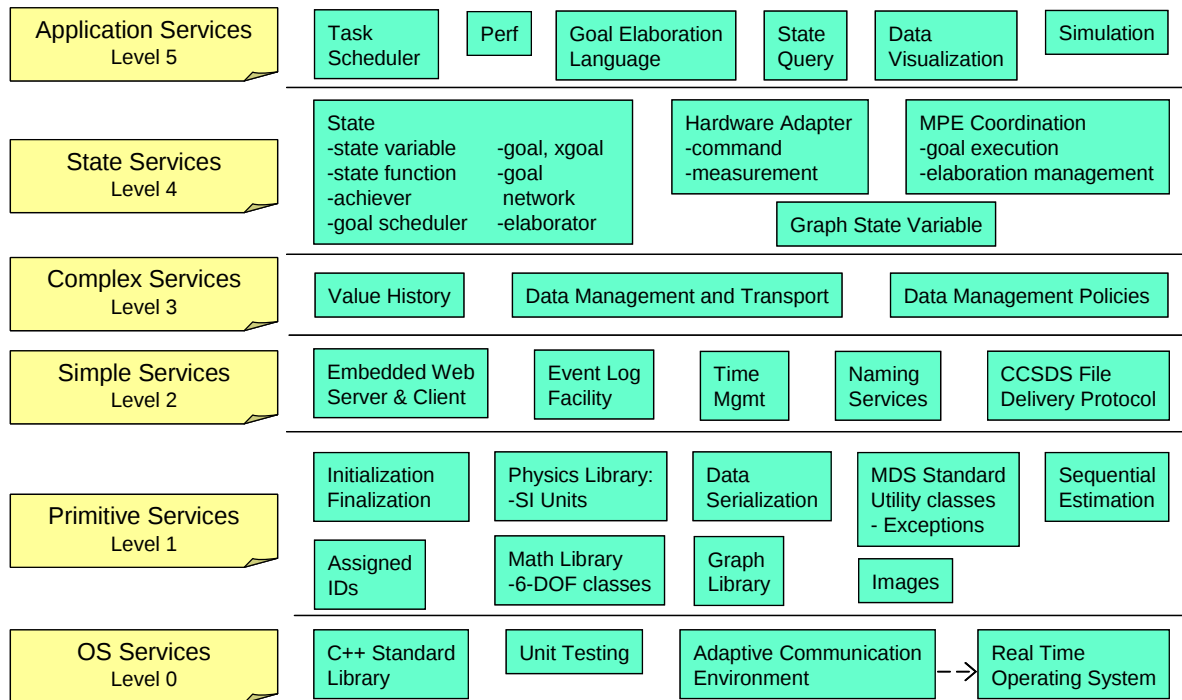


Figure 2. The MDS software framework packages.

desired quality of state knowledge to be achieved by estimators, and to express operating constraints (such as resource and safety margins) and monitored conditions (such as failure modes and external events).

- *The architecture provides a straightforward mapping into software:* The control diamond elements can be mapped directly into components in a modular embedded software architecture, such as MDS. The MDS software architecture is described below.

B. The Mission Data System Software Architecture

The overarching goal of MDS is to provide a multi-mission information and control architecture for robotic exploration spacecraft, that will be used in all aspects of a mission: from development and testing to flight and ground operations. In the process of achieving this ambitious goal, the MDS team has rethought the traditional mission software lifecycle. MDS acknowledges the intimate coupling between software and systems engineering by leveraging the State Analysis methodology. The regular structure of State Analysis is replicated in the MDS architecture, with every State Analysis product having a direct counterpart in the software implementation.

The mapping of State Analysis products to software is accomplished via a component architecture. Each state variable, estimator, controller, and hardware adapter is embodied as a component. State Analysis defines the interconnection topology among these components according to the canonical patterns and standard interfaces described in this paper; it provides the required interface details through the definition of state functions, measurements, commands, and goals; it provides the methods needed for planning, scheduling and execution; and it defines the functionality of each component to accomplish the desired intent. The component architecture helps to assure that the system is constructed in accordance with the State Analysis requirements, it aids modular reuse, and it provides support for a variety of software engineering and analysis issues.

The component architecture is part of a much larger framework of MDS software (Fig. 2), wherein each of the concepts in State Analysis is endowed with a core implementation that provides common features, interfaces, and so on. These are further constructed upon an in depth support structure of additional layered, modular frameworks supporting low-level services, extensive libraries in math and physics (e.g., units), data management and transport functions, high-level automated planning and scheduling engines, and so on. There are nearly thirty distinct core framework packages. Moreover, as various adaptations of these frameworks for particular projects occur, it is our intent to factor common elements into an additional set of engineering and science discipline framework packages for even greater reuse among projects.

The formally coherent structure shared by State Analysis and the MDS architecture has enabled an unprecedented level of coordination and control of the development process. Requirements are cleanly partitioned and traceable directly to implementation, making it easy to track and manage each step in the development process. Verification and validation exploits the same explicit structure, as well as the objective specification of each system element and the overt declaration of success criteria at all levels of operation.

We have also taken advantage of this structure in an iterative incremental development process with workflow and configuration management tools specifically built around State Analysis elements and spanning the entire development life cycle. Metrics gathered from this process are detailed and directly attributable to particular design elements, enabling far better feed-forward to future development efforts.

A C++ implementation of MDS has been demonstrated on multiple hardware platforms, including the Rocky7 and Rocky8 rovers at JPL. In addition, an MDS adaptation has been developed for the EDL stage of the Mars Science Laboratory spacecraft, scheduled for launch in 2009. This flight software prototype currently runs in a workstation environment, against a simulation of the EDL scenario. A Java implementation of MDS, called Golden Gate,⁸ has been demonstrated on the Rocky7 rover and a simulation of NASA's proposed Deep Space Network Array.⁹

MDS has already proven itself to be a versatile and amenable platform for investigation. Chosen for its regular structure and highly disciplined processes, this architecture has been the basis of a number of research efforts in the High-Dependability Computing Program sponsored by the National Science Foundation, with efforts extended across NASA, academia, and other participants. It has also been validated as a platform for infusion of state-of-the-art autonomy capabilities, such as the Titan Model-based Executive,¹⁰ which performs estimation and control via on-line reasoning through a probabilistic model of system behavior. These and other collaborations have resulted from widely held attraction to a well-structured framework within which investigators can build systems and evaluate new technologies in autonomy, verification and validation, model-based programming, real-time execution, architectural analysis, and other areas.

V. MDS in Support of NASA's Exploration Initiative

Given its wide appeal, we believe the MDS prototype framework and development processes can serve as a springboard for development of the architecture required for new exploration systems. With this in mind, we offer the following approach for establishing this framework and these processes in support of NASA's exploration program. The end objective would be twofold: (1) to establish an interoperability standard among systems, and (2) to establish examples of corresponding design frameworks for systems and software engineering, each realizing the great potential of state- and model-based architecture.

The objectives outlined here would be accomplished in a phased development approach designed to complement the larger exploration system development stages.¹ In particular, it is essential that the objectives presented here be largely in place before stage two, which culminates in human and robotic exploration on the lunar surface, and at least to some extent midway through stage one, which culminates in orbital flights of the Crew Exploration Vehicle (CEV). Otherwise, the opportunity will be lost as different systems by different vendors solidify, leaving the job of "interoperability" to be dealt with in an ad hoc fashion.

Initial phases would be concerned with the formation of an interoperability standards group, and the fielding of prototypes for demonstration and evaluation. Later phases would be concerned with narrowing the standard to a formal specification and bringing candidate technologies conforming to this standard to the level of maturity required to support wide deployment by no later than the start of stage two.

To be effective, given the expansive scope of this undertaking, products must be available early and often, with substantial feed-forward from one phase to the next, and there must also be a strong collaborative component to the effort. This suggests a parallel effort among assorted NASA, industry, and academia partners specifically demonstrating key aspects of state- and model-based architecture and interoperability conventions across a variety of system types. These efforts would be punctuated regularly (e.g., at four to six month intervals) by interim realignment exercises involving all participants to achieve convergence within two years from initiation on the formal specification. Subsequent efforts would overtly engage interoperation among elements targeted to later stage two developments.

The systems selected for early demonstration should cover the essential areas of concern. One such area is the accommodation of legacy systems. Thus, the ability to wrap existing systems for integration into a larger state- and model-based organization should be established. It will also be important to show that both flight- and ground-based assets can be addressed effectively with this approach, demonstrating reliability, operability, and cost effectiveness across a spectrum of real systems, operating in realistic circumstances. In addition, this effort should strive to

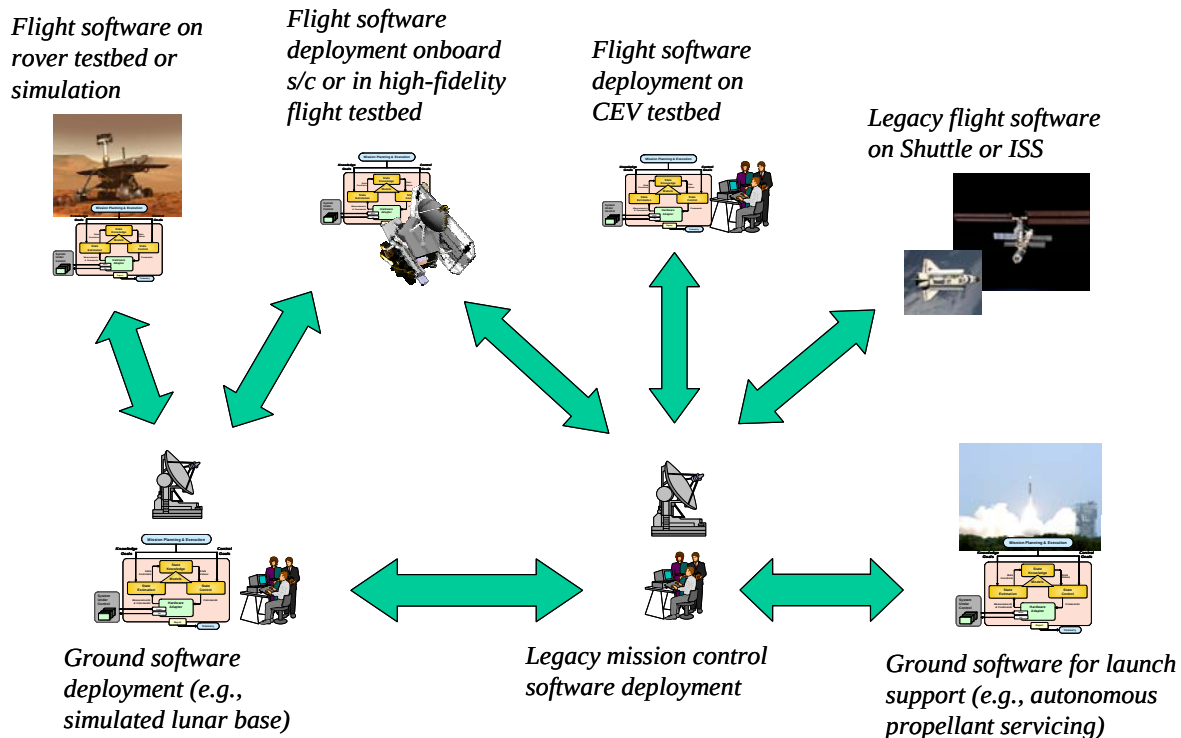


Figure 3. Analog Mission Testbed Concept

display full end-to-end life cycle development on each of the participating systems, as well as significant reuse among subsequent systems.

The nature of the tasks performed by these systems must also be compelling with regard to the issue of system complexity. Initial efforts should focus primarily on establishing fundamental principles. By 2008, however, demonstrations should be specifically targeted to the most challenging of control and interoperability examples. Specifically, in order to mature this capability in the most realistic manner available, targets from this point forward should include missions within the ESMD suite. This should include autonomous robotic systems in complex environments, human-robotic collaborations, highly interoperating systems (such as those involving RV&D), and eventually safety-critical systems.

The notion here is to prove the approach in successively more challenging steps, deliberately creating opportunities to learn from experience, rather than waiting for the ultimate challenge that forces all issues to be resolved at once. In other words, demonstrating progress in the control and interoperation of complex systems should be made an explicitly accountable obligation of ESMD projects throughout the stages of evolution.

VI. Next steps and near term opportunities

In keeping with this conviction, it is worth noting a number of near term opportunities for embarking on this agenda. For example, promising prospects may be found in extending the life of existing spacecraft, which might otherwise be abandoned after their primary missions. These could include Deep Impact, Stardust, and the Mars Exploration Rovers — each provides a platform for demonstrating some capability of direct interest to ESMD (e.g., low energy trajectory orbit transfer and maintenance at Lagrange points, robust critical sequence execution, planetary surface operations, etc.), while also addressing complex control issues.

These flight demonstrations could be a part of a broader set of *analog mission testbeds*, to be established at various NASA Centers, contractor facilities and academic institutions across the country (see Fig. 3). The primary objective of these testbeds would be to mitigate “tall tent pole” risk areas for ESMD (e.g., in the areas of distributed operations, mission-critical software, telecommunications, human-robotic interaction, etc.), mature and validate essential capabilities, and develop experience in operating these complex systems of systems.

These testbeds should be made accessible to ESMD researchers and infrastructure developers from remote sites, encouraging broader hands-on participation and engagement from the aerospace community, while further enabling the development and validation of effective distributed operations concepts. The analog mission testbed

environment would provide an excellent proving ground for validating the adequacy and completeness of the interoperability specification described in the previous section, with respect to requirements for the envisioned suite of exploration missions. This validation should encompass interoperability amongst new elements, as well as compatibility with any legacy systems.

The analog mission testbed concept illustrated in Fig. 3 is obviously very broad in scope. As discussed in the previous section, implementation of an effective analog mission program should be performed incrementally, perhaps starting with a single ground deployment (either a legacy or newly architected ground system) and a single flight deployment (e.g., CEV simulation testbed), then adding additional elements when successful validation of an increment's capabilities has been achieved. Each increment should have crisply-defined objectives tied to mitigation of a subset of the identified "tall tent pole" risks, and should follow a realistic (albeit perhaps accelerated) lifecycle process, from requirements definition through formal capability validation.

The great potential of state- and model-based architectures, the prospects for broad collaboration within the larger space community on interoperability and control of complex systems, and the inviting opportunities to explore these issues soon in a manner likely to have abiding consequences for the long-term capabilities of the exploration system enterprise all point to the need for a roadmap of development in this area. An outline for such a plan has been offered here. However, it will require concerted effort to adequately explore the opportunities suggested (or others), to set a detailed plan in place, and to begin the task of forming broad consensus on these issues. We eagerly recommend action to bring these ideas to fruition, while the time is ripe.

Acknowledgments

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We wish to thank the rest of the Mission Data System development team, for their participation in the maturation of the MDS architecture and software frameworks, as well as the State Analysis methodology and tools.

References

- ¹The White House, Office of the President, "A Renewed Spirit of Discovery: The President's Vision for U.S. Space Exploration," URL: http://www.nasa.gov/pdf/55583main_vision_space_exploration2.pdf, Released to accompany the President's NASA FY 2005 Budget, January 2004.
- ²Dvorak, D., "Challenging encapsulation in the design of high-risk control systems," *Proceedings of the 17th ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'02)*, 2002.
- ³Ong, E. and Leveson, N., "Fault Protection in a Component-Based Spacecraft Architecture," *Proceedings of the International Conference on Space Mission Challenges for Information Technology*, Pasadena, CA, July 2003.
- ⁴Rasmussen, R.D., "Goal-based fault tolerance for space systems using the Mission Data System," *Proceedings of the IEEE Aerospace Conference*, 2001.
- ⁵Ingham, M., Rasmussen, R., Bennett, M., and Moncada, A., "Engineering Complex Embedded Systems with State Analysis and the Mission Data System," *Proceedings of the 1st AIAA Intelligent Systems Technical Conference*, AIAA paper #2004-6518, 2004.
- ⁶Boehm, B., *Spiral Development: Experience, Principles, and Refinements*, edited by Wilfred J. Hansen, Special Report CMU/SEI-00-SR-08, June 2000.
- ⁷Dvorak, D., Rasmussen, R., Reeves, G., and Sacks, A., "Software Architecture Themes in JPL's Mission Data System," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, number AIAA-99-4553, 1999.
- ⁸Dvorak, D., et al., "Project Golden Gate: Towards Real-Time Java in Space Missions," *Proceedings of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2004)*, 2004.
- ⁹Dvorak, D.L., Indictor, M.B., Ingham, M.D., Rasmussen, R.D., and Stringfellow, M.V., "A Unifying Framework for Systems Modeling, Control Systems Design, and System Operation," to be presented at the *IEEE Conference on Systems, Man, and Cybernetics 2005*, Waikoloa, HI, October 2005.
- ¹⁰Williams, B., Ingham, M., Chung, S., and Elliott, P., "Model-based programming of intelligent embedded systems and robotic space explorers," *Proceedings of the IEEE*, 91(1):212-237, 2003.