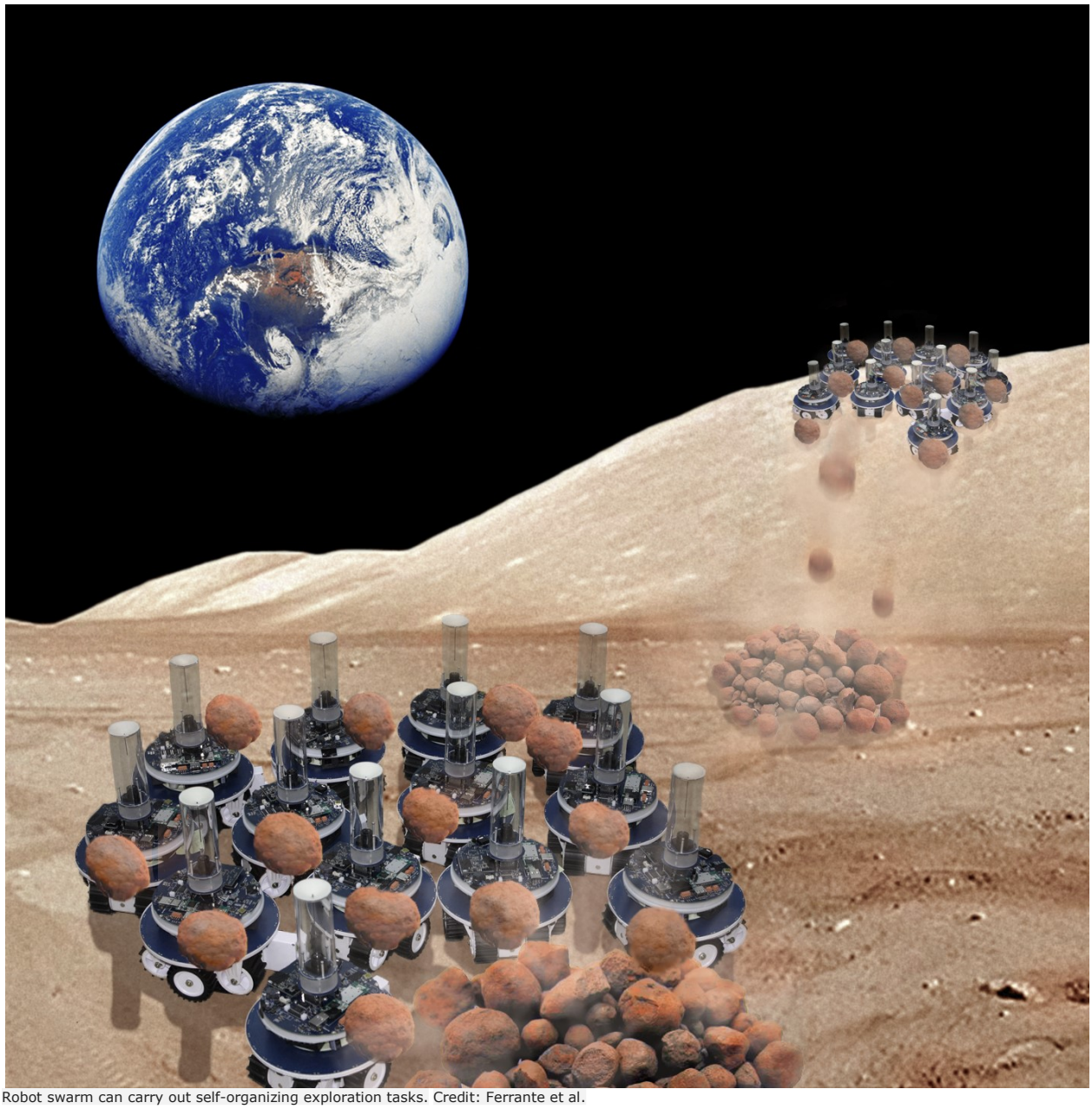


Robot swarm can carry out self-organizing exploration tasks. Credit: Ferrante et al.

Developing Robotic Swarm Coordinated Operations for a Lunar "system under control"

Ronald H. Freeman, PhD
Editor-in-Chief, Journal of Space Operations & Communicator
ronaldhoracefreeman@gmail.com

Abstract

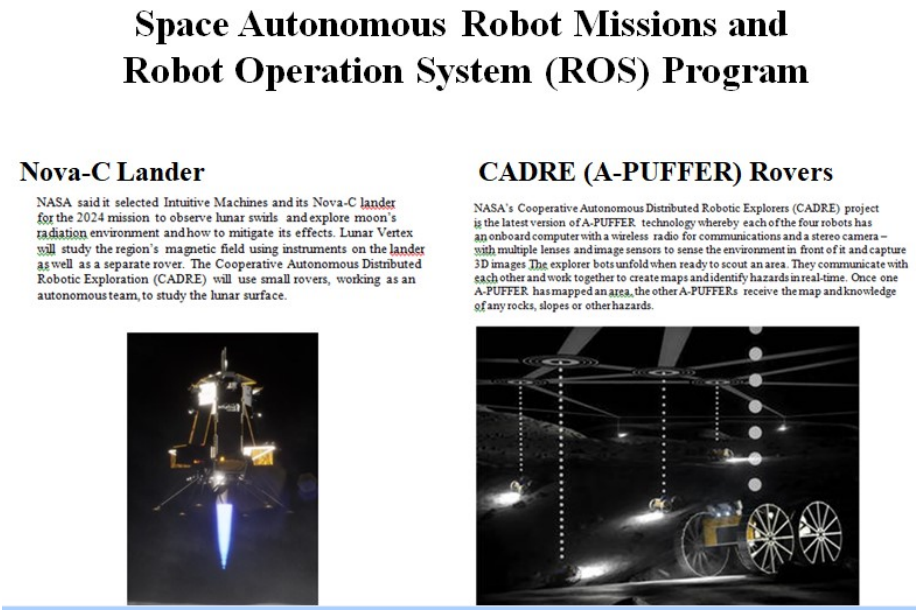
The purpose of this paper is to explore the software-defined operations of multiple, modular, re-usable multi-agent autonomous robotic systems. NASA exploration missions increasingly rely on the concepts of autonomic computing (AC), exploiting these to increase the survivability of remote missions, particularly when human tending is not feasible. AC has emerged as a promising approach to the development of large-scale self-managing complex systems. However, direct coding all the necessary rules to reliably handle coordination and uncertainty is

problematic. External correctness of the software design and how it will safely interact with a robotic software framework like ROS or GenoM requiring verification by special purpose language interpreters or compilers have not been borne out. This paper considers the role of system engineers in cooperative collaboration with software engineers to unify a framework for systems modeling, control systems design, and system operations.

1. Introduction

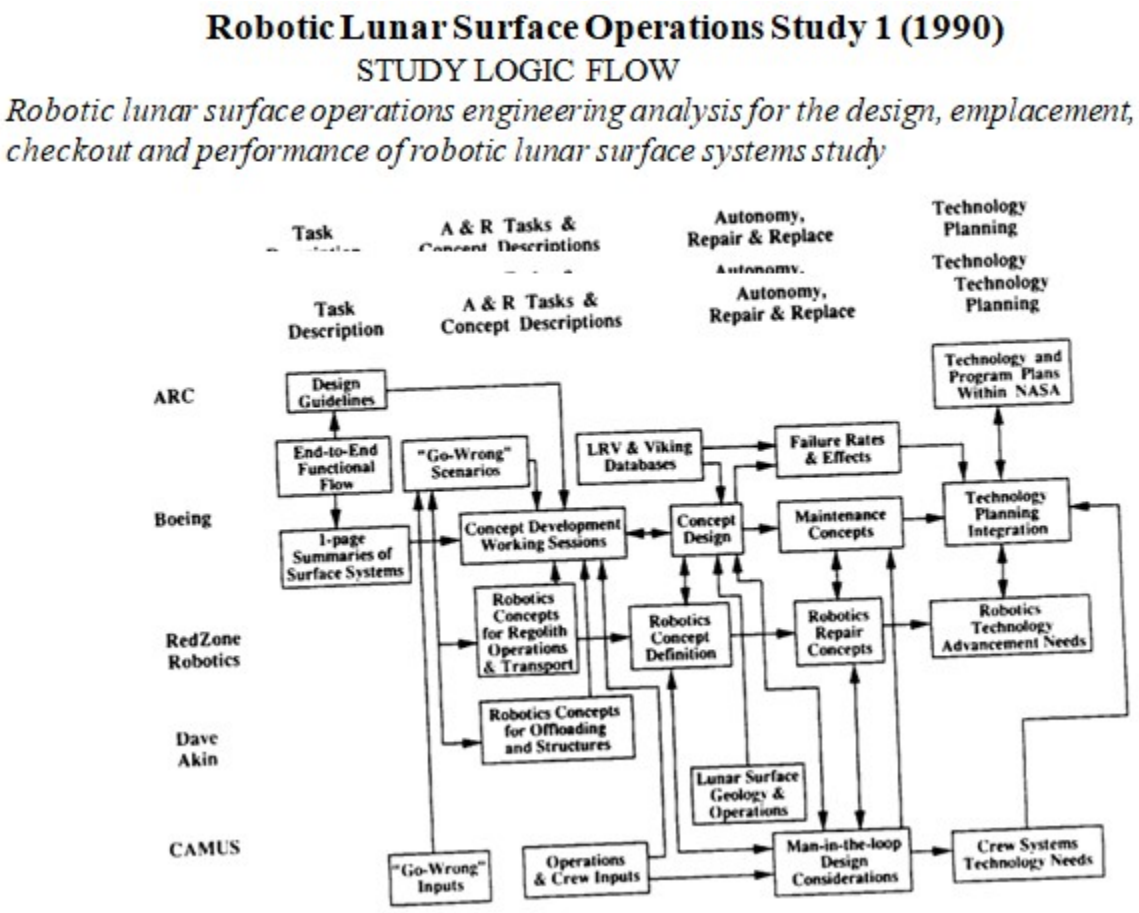
Autonomic computing promises computer systems capable of self-management, which augurs great promise for unmanned spacecraft. Such spacecraft are extremely appropriate for deep space exploration missions because the former bring onboard intelligence and less reliance on control links. Unlike traditionally engineered systems, autonomous systems are expected to behave predictably in varying environmental contexts that cannot be fully specified formally. Modeling the Moon's environment portended for Artemis Mission suggests relevancy for tackling hazard avoidance and safe rover navigation. Although several navigation algorithms exist, some cannot be used in safety-critical scenarios because they have not been verified [1].

Robotic swarms, proposed for large-scale exploration, revolve around the concept of a team of robots that work cooperatively to explore an environment, but act as a single entity aiming to accomplish a common goal.
The coordination of multiple robots within an uncertain and unsafe environment [2] challenges the objective having presence in multiple locations at once and interpreting the situation at reduced cost, increased flexibility, and increased reliability. Direct coding all the necessary rules to reliably handle coordination and uncertainty is problematic. Instead, coordinated reinforcement learning enables rovers to iteratively learn through trial and error actions. And, machine controllers can determine with millimeter accuracy the relative positions of all moving components as tasks are performed [3].



In this figure, NASA's Commercial Lunar Payload Services (CLPS) will deliver in 2024 a Nova-C lander along with four CADRE rovers to Reiner Gamma, known as a lunar swirl, in a mission to learn what lunar swirls are, how they form, and their relationship to the Moon's magnetic field. The payload manifest is designed to obtain data unique to the geographical feature of Reiner Gamma. Cooperative Autonomous Distributed Robotic Exploration (CADRE) consists of mobile robots programmed to work as an autonomous team to explore the lunar surface, collect data, and map different areas of the Moon in 3D. CADRE uses its inertial measurement unit, stereo cameras, and a Sun sensor to track the position of each robot as they explore the lunar surface [4]. The foundation for space robot missions is programmed in the Robot Operation System (ROS) framework. A single traditional monolithic autonomous robot interacts with an unstructured environment undertakes multiple functions with uncertain performance. Comparatively, a team of rovers performs a single function albeit challenged by coordinating multiple

robots within an uncertain, unsafe environment. The challenge of coordinated control of multi-agent systems almost always falls to software.



Unlike the 1990 study with the above integrated systems model, RLSO 2 included: (1) New flight systems, such as SLS and the Lunar Gateway; (2) New partnerships, including international and commercial collaboration opportunities, and modern tools, including CAD and integrated systems models. And to address today's lunar challenge of combining discrete computations and a continuous environment, a robotic system is typically separated into several layers.

2. Software Architecture for Modeling Multi-agent Systems

State Analysis is a systems engineering process for the problem domain of autonomous and semi-autonomous control systems. It is a comprehensive process in that it addresses analysis of the system to be controlled, design of the control system, and operation of the control system. Compared to more conventional systems engineering processes, it demands more up-front analysis and it represents its results in a more structured form. Its main benefit is more reliable systems due to fewer errors of omission, fewer errors of translation into software, and more easily understood operational controls [5].

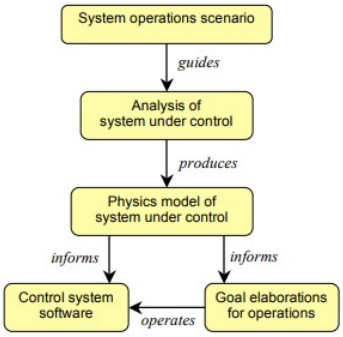


Figure. State Analysis provides a unified approach to system analysis, control system software design, and system operation [6].

At the bottom, the functional layer consists of control software for the robot's hardware. Then, the intermediate layer generally utilizes a middleware framework (such as ROS or GenenoM) that provides an interface to the hardware components. The upper layer contains the decision-making components of the robotic system, which capture its autonomous behavior [7].

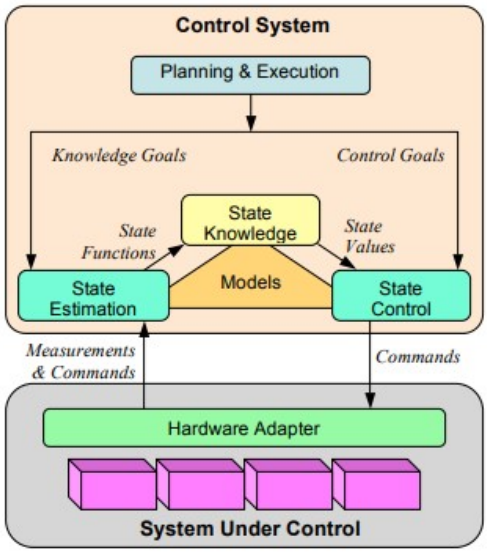


Figure. Architecturally, State Analysis emphasizes a clear distinction between the control system and the system under control, with explicit representation off State variables and behavioral models.

There is a clear distinction between the *control system* and *the system under control* because control can only be understood and exercised intelligently through models of the system under control. State variables of the system under control are explicit since it is those values that the control system is designed to control. Models of how the system under control behaves are documented explicitly because they are needed by the control system for execution (estimating and controlling state) and higher-level planning (e.g. resource management). Hardware adapters provide the sole interface between the control system and the system under control, with all interactions in the form of commands to and measurements from the system under control. ROS provides functionality for hardware abstraction, device drivers, communication between processes over multiple machines, tools for testing and visualization, and much more. The key feature of ROS is the way the software is run and the way it communicates, allowing you to design complex software without knowing how certain hardware works. ROS provides a way to connect a network of processes (nodes) with a central hub [8]. Moreover, the architecture

emphasizes goal-directed closed-loop operation because goals, as constraints on the values of state variables over time intervals, directly express operational intent. Finally, the architecture provides a straightforward mapping into software because the aforementioned architectural concepts (state variables, models, estimators, controller, measurements, commands, goals) are represented directly in software [9].

As a systems engineering process, State Analysis is a gradual discovery process, incremental in developing. The first steps of the process are: (1) identify the state variables of the system under control that must be monitored and/or controlled; (2) identify the direct effects of one state variable on another (state-to-state effects); (3) identify the measurements that provide evidence about the values of state variables (state-to-measurement effects); and (4) identify the commands that affect the state variables (command-to-state effects). All of these state variables, measurements, commands, and influences are captured in a "physics model" describing the physics of the system under control.

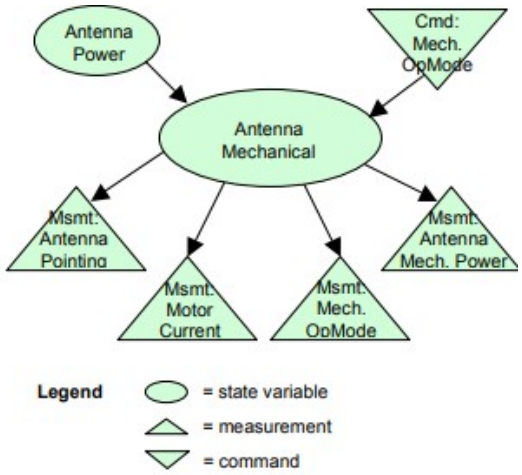


Figure. An overview of a portion of the physics model centered around a state variable for the operational mode and health of an antenna mechanical system.

A unifying framework for systems modeling, control systems design, and system operation for coordinated control of multi-agent systems almost always falls to software. Unfortunately, there is a fundamental gap between the requirements on software as specified by systems engineers, and the implementation of those requirements by software engineers. Software engineers perform the translation of requirements into software design and into code, trying to capture the systems engineer's understanding of system behavior, which is not always explicitly specified. This gap possibly leads to serious flaws in system operations.

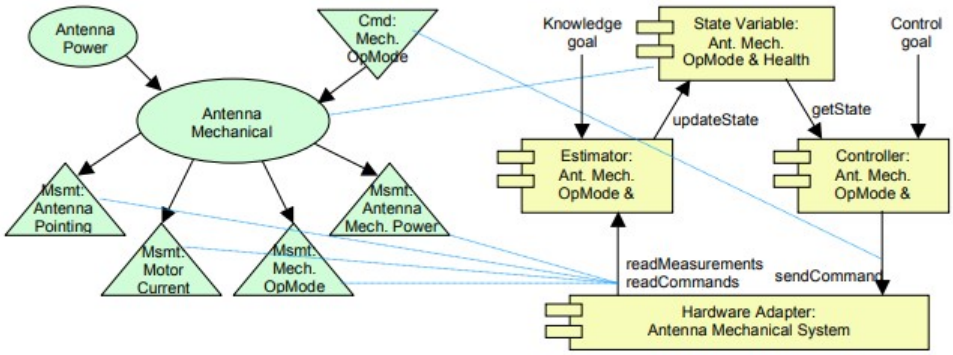
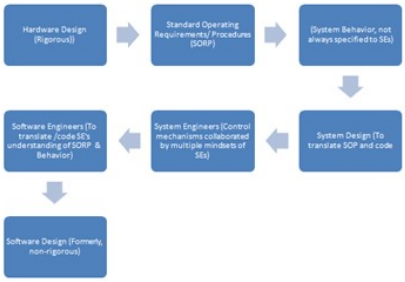


Figure. The physics model (shown on the left) directly informs the software design (shown on the right). The

physics model elements (state variables, measurements, and commands) and the models behind them (state, measurement, and command effects models) all map into the software design.

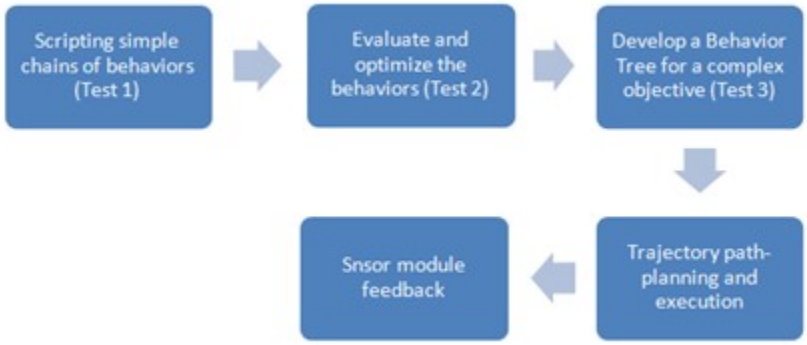
Modeling a "system under control" to assure consensus among systems engineers starts with design of the hardware system in order that standard operating procedures are performed in accordance to the requirements for the system's mission. System behavior produced of operations is reviewed for control mechanisms by multiple system engineers. Software engineers translate and code the standard operating procedures and system requirements based on system engineers' understanding, they communicated. Software engineers are tasked with software design, appearing not as rigorous as the hardware design.



Engineering of hardware systems and subsystems is rigorous architecturally designed. In contrast, the same is not generally done for software, even though software is responsible for a control-oriented software architecture based on JPL's Mission Data System [10]. Another challenge in the validation of any complex software system is that much of what transpires in software design and development is opaque to systems engineers. It is nigh impossible for a systems engineer to obtain a description of the software design that is simultaneously faithful to the software, as built and expressed, in terms directly related to the requirements.

3. Behavioral Planning

Unlike many traditionally engineered systems, autonomous systems are expected to behave predictably in varying "open world" environmental contexts that cannot be fully specified formally. Multiple sensors and communications deployed in a physical environment activate sensor inputs for autonomous decision-making components. Modeling a "system under control" System Behavior Plan: (1) informs system design, and (2) minimizes system engineers' translation errors in defining/ understanding Standard Operating Requirements and Procedures.



Formal methods offer a concrete basis for specification, verification, and synthesis in autonomous systems, but do not provide guidance for translating desired values and acceptable risks into formal models. Frameworks that explicitly allow for ambiguities in specifications and uncertainties are required for partial decisions in modeling. Software engineers in the bottom level perform the translation of requirements into software design and into code, trying to capture the systems engineer's understanding of system behavior, which is not always explicitly specified. This gap translates in the middleware framework and possibly leads to serious flaws in system operations. The twin

problems of errors of omission and errors of interpretation are exacerbated because opportunities for error grow with the number of different 'mindsets' about control mechanisms. Utilization of reusable or legacy software does not help. Robot swarms have shown both resilience and robustness. Yet, system performance and robustness are improved in the upper layer by dynamics-aware planners.

Robot swarms have shown both resilience and robustness. System performance and robustness are improved by dynamics-aware planners. Behavior planning starts with scripting simple chains of behaviors, followed by evaluation of search-based and optimization-based planning approaches to generate the behavior trees [11]. At each stage, testing will consist of trajectory planning and execution with obstacle information and state feedback provided by the sensing module. The next layer of autonomy consists of modules implementing a small set of low-level behaviors, such as moving to a goal or actuating the tool. Higher level behaviors, such as scanning the environment or picking and placing an object, will then be implemented by composing lower level behaviors into behavior trees that can represent several common decision-making data structures [12]. Forrester [13] argues that the system boundary must be drawn to include structures which enable an understanding of all the relevant dynamic behaviors. Operational control via goal networks is designed in accord with the physics model, for both nominal operations and fault responses. Complex systems are to be built more reliably with a unified approach. By adopting the State Analysis approach, elements of the physics model map directly onto software design. The system as built is no longer opaque to systems engineers because it now represents the results of their analysis in a direct, inspection manner [14].

4. Conclusion

This paper has shown how the architectural concepts of state variables, models, and goals provide a unified view of control that shapes the operational control of system analysis, software design, and operational control. Analysis is always about the system under control, not the control system, and it begins with a discovery process for state variables, commands, and measurements, plus the models of behavior that relate them. The software design indicates how the control system monitors and controls specific state variables in the system under control. To do this it depends on knowledge of how the system under control works — exactly the same knowledge of behavior captured in the models. Operational control via goal networks is designed in accord with the physics model, for both nominal operations and fault responses. Specifically, goal elaborations and their alternate tactics are employed directly during system operation for coordinated control of the whole system. The important message in this paper is that complex systems can be built more reliably if a unified corroborative approach is followed by software engineers and system engineers working and learning the operational behavior of the system under control. By adopting the State Analysis approach, elements of the physics model map directly onto software design, and state-to-state effects directly shape the goal elaborations. The system as built is no longer opaque to systems engineers because it now represents the results of their analysis in a direct, inspectable way. Modeling a "system under control" requires a System Behavior Plan so that operational performance behaves predictably in varying "open world" environmental contexts that cannot be fully specified formally.

References

[1] Bakirtzis, G., Carr, S., Danks, D., & Topcu, U. (2022). *Dynamic Certification for Autonomous Systems.*
[2] Wong, C., Yang, E., Yan, X., & Gu, D. (2017). An overview of robotics and autonomous systems for harsh environments. In *2017 23rd International Conference on Automation and Computing* (pp. 1-6).
[3] Woodcock, A., Sherwood, B., et al. (1990). *Robotic lunar surface operations engineering analysis for the design, emplacement, checkout and performance of robotic lunar surface systems study.* Contract NAS 2- 12108 Boeing Aerospace & Electronics, Huntsville AL.
[4] Fox, K., Handal, J., & Ramji, N. (2021), NASA Selects Intuitive Machines for New Lunar Science Delivery. Retrieved from www.nasa.gov]
[5]Dvorak, D., Indictor, M., Ingham, M., Rasmussen, R., & Stringfellow, M. (2005). A unifying framework for systems modeling, control systems design, and system operation. In *2005 IEEE International Conference on Systems, Man and Cybernetics* (Vol. 4, pp. 3648-3653). IEEE].
[6] Dvorak, D., Indictor, M., Ingham, M., Rasmussen, R., & Stringfellow, M. (2005). A unifying framework for systems modeling, control systems design, and system operation. In *2005 IEEE International Conference on Systems, Man and Cybernetics* (Vol. 4, pp. 3648-3653). IEEE].
[7] Luckcuck, M., Farrell, M., Dennis, L., Dixon, C., & Fisher, M. (2019). Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR), 52*(5), 1-41.
[8] [Ademovich, A. (2015). *An Introduction to Robotic Operating System: The Ultimate Robotic Application Framework.* Retrieved from www.toptal.com]
[9] Dvorak, D., Indictor, M., Ingham, M., Rasmussen, R., & Stringfellow, M. (2005). A unifying framework for systems modeling, control systems design, and system operation. In *2005 IEEE International Conference on Systems, Man and Cybernetics* (Vol. 4, pp. 3648-3653). IEEE].

[10] Dvorak, D., Indictor, M., Ingham, M., Rasmussen, R., & Stringfellow, M. (2005). A unifying framework for systems modeling, control systems design, and system operation. In *2005 IEEE International Conference on Systems, Man and Cybernetics* (Vol. 4, pp. 3648-3653). IEEE].
[11] Luckcuck, M., Farrell, M., Dennis, L., Dixon, C., & Fisher, M. (2019). Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR), 52*(5), 1-41.
[12] Leitner, J. (2009). *Multi-robot cooperation in space: A survey*.
[13] [Yu, X.;Wang, P.; & Zhang, Z. (2021). Learning-based end-to-end path planning for lunar rovers with safety constraints. *Sensors, 21*, 796]
14]Woodcock, A., Sherwood, B., et al (1990). *Robotic lunar surface operations engineering analysis for the design, emplacement, checkout and performance of robotic lunar surface systems study.* Contract NAS 2- 12108 Boeing Aerospace & Electronics, Huntsville AL].