

# Active Experimentation and Computational Reflection for Design and Testing of Cyber-Physical Systems

Kirstie L. Bellman<sup>1</sup>, Phyllis R. Nelson<sup>2</sup>, and Christopher Landauer<sup>1</sup>

<sup>1</sup> Topy House Consulting, Thousand Oaks, CA USA

<sup>2</sup> California State Polytechnic University Pomona, Pomona, CA USA

**Abstract.** Cyber-physical systems are being deployed in a wide variety of applications, creating a highly-capable infrastructure of networked “smart” systems that utilize coordinated computational and physical resources to perform complicated tasks either autonomously or in co-operation with humans. The design and testing of these systems using current methods, while time-consuming and costly, is not necessarily sufficient to guarantee appropriate and trustworthy behavior, especially under unanticipated operational conditions. Biological systems offer possible examples of strategies for autonomous self-improvement, of which we explore one: active experimentation. The combined use of active experimentation driven by internal processes in the system itself and computational reflection (examining and modifying behavior and structure during operation) is proposed as an approach for developing trustworthy and adaptable complex systems. Examples are provided of implementation of these approaches in our CARS testbed. The potential for applying these approaches to improve the performance and trustworthiness of mission-critical systems of systems is explored.

## 1 Introduction

Complex systems of systems (SoS), especially those that include cyber-physical systems (CPS), are now being deployed in critical infrastructure applications such as the electrical grid, health care, manufacturing, transportation, commerce, law enforcement and defense. We bet our lives, or at least our livelihoods, that these systems will function as anticipated. Yet, as they become increasingly complex and interconnected (networked), developing the systems engineering methods to ensure that these SoS will be trustworthy has become its own technical challenge.

For example, space systems (which term includes not only the satellites, but also the ground control and dissemination systems and the launch systems that put them up there) are simply the most complex engineered systems that humans build that work (and they do work almost always and often far beyond their projected design life). They typically involve hundreds of organizations, thousands of people, tens of thousands of components, millions of pages of documentation,

and they are expected to last sometimes for decades. (The development process does usually last for decades even when the satellites are not expected to). It has been clear for some time that these systems exceed our ability to understand them, and that they only work by dint of what we have heard called “heroic engineering”, but even that approach is now regularly exceeded by current and planned systems.

Successfully planning and implementing the integration of such complex constructs would, in principle, require detailed knowledge of hundreds of thousands (or more) of components, how they are connected into subsystems, and all of the possible interactions between components, subsystems, and the environment. From a practical perspective, it is exactly the lack of this detailed knowledge that leads us to characterize a system as complex. [22] Various approaches based on formal compositional methods [10, 11, 21, 24, 25] or brokering of mutual requirements (service-oriented architectures) [2, 23] have had some success, but these approaches do not adequately address the central problem: precise descriptions of all of the components to be integrated, and especially all of their possible interactions, are not fully known, and therefore not available for use in the design and integration processes. Existing approaches do not enable the discovery of the new knowledge that is needed to guarantee appropriate functioning of the integrated SoS.

Systems of systems are built from systems that themselves have been developed and tested, often for a different application. The integration challenge, then, concerns most importantly the necessity of reconciling the multiple and sometimes conflicting operation and control strategies of these systems with respect to a new SoS purpose or goal. [6] Conflicts in which a component system continues to operate in accordance with its own best interests given the previous application may no longer allow the full SoS to operate as needed, but these conflicts are difficult to discover without testing the full operating SoS. Therefore, the testing required for verification and validation of the operation of the full SoS is potentially damaging to the SoS itself, and also risks interruption of the services it supplies. This paper proposes a strategy by which active experimentation coupled with computational reflection can refine or even discover the knowledge needed to ensure appropriate functioning of the overall SoS.

## 2 Systems Engineering Challenges

Complex systems of systems challenge established systems engineering practices in several ways.

- Managing the complexity is a fundamental technical challenge in itself, independent of the particular system or application.
- Updates and upgrades mean that the SoS evolves during its operational life.
- The capability and value of a system / component / device leads us to repurpose it for applications that were never envisioned by its original designers rather than developing a completely new device.

- Instances of the system are often unique, although there may be other, similar instances (i.e. Amtrak’s reservation system, a segment of the electrical power grid, a space system including all ground and launch resources).
- Ubiquitous wired- and wireless communications networks mean that the boundaries of the SoS and its possible states are probably not completely definable.
- Self-x capabilities mean that the system is never fully designed.

Component, subsystem, and system design and test currently utilize a variety of models at differing levels of detail, together with a set of “goodness” measures linked to *a priori* requirements, as inputs to computational processes (often optimization) that evaluate candidate strategies. However, as complexity increases, “emergent” behaviors become increasingly likely. Such self-organized, coherent actions that were not planned or anticipated by the designers often occur through interactions that are not present in the models of components, processes and interactions used in design, integration and test. CPS SoS design and testing is further complicated by the re-purposing of legacy hardware and software which may not have been designed in accordance with current procedures, standards and interfaces, thus requiring specialized adaptations. New approaches are needed for design, verification and validation of complex cyber-physical SoS to better ensure their trustworthiness. The most desirable of these approaches will also address the spiraling cost of implementing and testing these complex SoS.

### 3 Biologically Inspired Control Strategies

Biological systems provide a rich source of inspiration for engineering complex SoS both because, in spite of their obvious complexity, they achieve remarkable robustness, and also because of the extreme degree of interconnection of their various components and subsystems. [5, 9] A central lesson that we have taken from biology is that both robustness and controllability can result when each component or process interacts strongly with many other components and processes in a monitored and regulated system. (A recent example comes from work on the immune response of the mammalian gut microbiome, [1, 20] but there are many others.)

Control in biological systems occurs through the combined operation of many processes and actions, with desired behaviors being achieved by small changes in relative strengths. A web of overlapping monitoring and regulatory processes that maintain appropriate conditions at all levels of complexity is critical to the success of this paradigm. For example, the actions and processes used to achieve the top-level goal of walking over rough terrain are achieved by many instances of humans in spite of significant differences in their structure, strength and ability. That is, biological systems rely not on uniformity of structure, but on the ability to adjust similar structures and generic patterns of actions based on a high degree of monitoring of local conditions in order to accomplish a behavior that is adequate for the current context and goal.

Controlling a SoS with a complex web of balanced interactions is strikingly different from the traditional block-diagram approach to engineering design that focuses on building a few strong and well-understood interactions between components while striving to nullify all other interactions. We suggest that the assumption that small interactions can be neglected, together with implementation of this assumption throughout the modeling process, is one important reason that emergent behaviors are often not predicted by simulations. In contrast, the biological style does not deprecate interactions, but instead achieves a “balance of forces” form of control based on extensive overlapping webs of monitoring and regulation at all levels of the hierarchy of complexity. We propose that implementing this style in strategic portions of engineered systems could mitigate the challenges posed by unmodeled interactions.

The biological design approach leads to a “permissive” style in which, while actions, states, conditions, and processes may vary from one instance to another, overall performance goals are achieved by adjustments in their relative intensities. This permissive style is in clear contrast to the restrictive control approach of traditional engineered systems, in which adjustments of a few inputs achieve all of the desired actions or processes through clearly defined pathways. However, there is promising similarity between the dense web of interactions in biological systems and the challenge of managing the many unknown or unmodeled interactions in a complex SoS, again suggesting that a more biological approach to design, operation, and integration may be useful provided that the appropriate information about actual interactions can be discovered.

The admirable robustness of biological systems is due in part to their ability to learn to accomplish the same goal using a variety of strategies, although not necessarily equally efficiently. For example, if you break your right arm you are still able to accomplish most of the tasks of daily life by substituting your left arm or accommodating to the reduced motion allowed by a cast. This broad ability to find a way to accomplish a goal in spite of changes in capability or configuration is exactly the type of robustness and reliability that we would like to have in engineered SoS, and to understand and utilize during design and integration. New ways of acting can take place through the recruitment of existing structures and processes in new combinations to address a new context, purpose or goal. [3] Thus, a large space of possible responses can result from small departures from previous conditions. Since this style of operation differs significantly from the usual engineering approach with narrowly defined and targeted control pathways, new tools and methods are required in order to exploit it effectively.

## 4 Active Experimentation and Computational Reflection

The success of the biological control-through-balance style rests on experience with the available processes, structures and patterns, as well as of their limits of capability and their applicability to situations similar to the present one, either through evolutionary selection or from the experience of a specific individual.

This knowledge is not necessarily innate in a biological system, just as there is important knowledge lacking in models of SoS.

Biological systems use excess resources to actively experiment. By doing so, they discover and refine models of their capabilities, limitations, and possible interactions with their surroundings that include consideration of both internal state (hungry, cold, tired, etc.) and external conditions. Significantly, such experimentation also enables the grouping of collections of useful resources, processes and capabilities into generic pre-patterned templates with simplified control mechanisms. Such templates can be easily shaped to fit a specific current context. [7,8]

The biological analogy suggests that, if it were possible for a complex SoS or some of its components and subsystems to engage in active experimentation, the existence of conflicts between the existing operation and control strategies of a repurposed subsystem and the overall SoS purpose or goal could be identified and modeled before such a conflict gives rise to failure or to disruption of the service provided by the overall SoS. In addition, efficient strategies for accomplishing common purposes and goals could be discovered and collected into templates for accomplishing similar operations. Such templates could then be reviewed for correctness either by the system itself using further active experimentation, or supplied to designers, integrators and operators for evaluation.

Our existing systems have not been built with the capabilities required in order to engage in active experimentation. Thus, an important research challenge is to implement such processes while preventing the resulting experiments from damaging some part of the SoS or compromising the service it provides. Implementation is particularly challenging when, as with space systems or the electrical grid, there is only one operating instance of the entire SoS.

In following sections we discuss several possible approaches for introducing active experimentation into engineered complex cyber-physical SoS. However, first we list the additional capabilities required of such implementations. They are:

- instrumentation at all levels of the hierarchy of complexity to measure what happens.
- models that relate what is measured to properties or symbols that are local but have meaning that can be communicated to other parts of the SoS.
- models that relate what is measured locally to higher-level purposes, goals and constraints.
- the capability to retain the information produced by these measurements and models.
- a hypothesis-generating engine that can propose possible actions (experiments).
- a predictive capability to project and analyze the potential consequences of a proposed future action.
- the ability to engage in a proposed action.

Taken together, these resources and capabilities would create an engineered SoS able to reason about itself (its resources, capabilities, and limitations) in the

context of its current environment, purposes and goals, and also to both propose and implement a course of action based on that reasoning rather than on pre-programmed control strategies. [13, 15]

These capabilities required for achieving active experimentation, taken together, constitute computational reflection. [18, 19] That is, the SoS is able to retain meta-information, reason about itself, and implement modifications to its behavior. Computational reflection is more nuanced than feedback control, but certainly less than consciousness. Importantly, we do not conceive that computational reflection will be implemented as one top-level control strategy, but will rather be distributed throughout the hierarchy of complexity of the SoS in keeping with the lessons learned from biological systems.

## 5 Approaches to Implementation in Mission-Critical SoS

The crucial question is, of course, how to implement this biologically-inspired approach of active experimentation coupled with computational reflection to improve and extend existing SoS, as well as to design, develop, integrate and test new ones. We suggest two complementary strategies, both of which leverage the capabilities we have listed above. One approach, which we are following in our own work, is to build testbeds [7, 12, 17] to refine our understanding of the methodologies and tools required to incorporate active experimentation and computational reflection in a cyber-physical SoS.

The other, and more advanced, strategy is to implement portions of the required capabilities locally in an already-operating system and monitor the proposed courses of action for compatibility with known “concepts of operations” (CONOPS), which are the different styles of use intended for the system. Since the cases of most interest are also SoS providing important services that cannot be interrupted, we suggest that, after testing at the subsystem level, such modifications could be implemented during planned maintenance, update, or upgrade periods for the affected portion of the SoS. We note that all critical systems have methods for implementing such planned modifications. Addition of reflective capabilities and active experimentation could be implemented one step at a time, starting with reflection, but trapping the proposed modifications instead of implementing them. Multiple periods of testing and review could be accomplished during successive maintenance periods, carrying out all of the necessary processes for implementation except executing the proposed actions. This strategy allows a period during which the proposed actions can be compared with known CONOPS for consistency throughout the entire SoS, providing a basis for verification and validation of the expected operation of the entire SoS once the new capabilities are allowed to affect operation.

In a SoS that supplies a mission-critical service, we do not have the ability to isolate the whole system (with new incoming systems or capabilities and legacy systems) from its ongoing requirements within its true operational context. And yet, it is arguably even more critical that SoS, which are dynamic, which have many unknowns, which have constantly new combinations of legacy systems /

components and new systems / components, have some “safe” places within which to actively try out component configurations and to reason about and record / learn the impacts of such configurations in matching their requirements and operational constraints.

Most SoS are modular and utilize redundancy to achieve robustness so that sections can go down without bringing the rest of the system down, and also can be routinely taken offline for necessary check-out, maintenance, and upgrades. To leverage redundancy and maintenance periods for evaluation of the effectiveness of new capabilities such as reflection and active experimentation, one would have to devise a simulation that would mimic the current operational settings. Combining emulation / simulation and protected operation are currently done for checking out space vehicles and their subsystems and components, as well as other similarly expensive systems that require testing within very realistic operational conditions. These operational simulations could be used to test the new combinations of components, capabilities and system integrations by a human system engineer using a set of pre-designed tests. Certainly this would have great advantages over the current practices in developing and testing SoS, changing it from a certification process into one of continual verification and validation.

We now discuss our testbed, how it enables us to implement both active experimentation and computational reflection, and how we can apply what we learn to the cases of complex SoS.

## 6 The CARS Test Bed

CARS (Computational Architectures for Reflective Systems) is a testbed that we have been developing as an ongoing student project at California State Polytechnic University, Pomona. [7,8,17] This testbed is based on a set of design decisions that enable us to confront many of the challenges of implementing real SoS. It is composed of a group of robotic agents built from low-cost commercial off the shelf (COTS) hardware. Specifically, we use inexpensive toy radio-controlled cars and trucks. These vehicles are decidedly not ideal for the tasks we assign them, and they are also quite different one from another. Both of these circumstances mean that the self-modeling aspects of our reflective architectures are critical to successful system function. By adding our own sensors, computation, communication, and control, these toy vehicles become useful agents, although they have capabilities that are deliberately limited compared to the relatively complex tasks we require of them, a situation often replicated in real-world systems containing legacy hardware.

A series of benchmark tasks are utilized for evaluation of CARS that span a broad range of sometimes conflicting strategies: independent or multi-agent, co-operative or competitive, asynchronous or synchronous. Specifically, we use the “games” follow-the-leader, tag, soccer practice (bump a ball into a designated goal), and push-the-box (move a large, heavy object that cannot be moved by any individual agent to a designated goal). We use *Wrappings* to implement computational reflection and self-modeling. *Wrappings* grew out of work on conceptual

design environments for space systems, and has been in continuous development since its inception in 1989. [4, 13–15]

Some of the important characteristics of CARS are

- The cost of each robotic motion platform (< \$50) means that, unlike most deployed systems, the investment in any part of the system is relatively small. (A new agent can be prepared in less than a day from COTS hardware and the electronics of a damaged agent.)
- The robotic components are relatively crude, requiring more modeling and self-refinement of generic models than better hardware.
- The performance of the SoS for any task can be evaluated from recorded video of the “field of play.”
- The tasks and the appropriate performance measures are easy to express in everyday language.
- Use of *Wrappings* frees experimenters from many of the detailed programming tasks normally associated with adding or modifying a process, model, or sensor interface.

What Wrappings provides here is the ability for the system to have multiple alternative resources for any given problem, and to select them according to their operational context at the time of use. Because the process that make those selections are also resources, and are also selected just like any other, these systems have a very strong kind of computational reflection [15, 16]. The Wrappings approach also allows active experimentation in two ways. First, the system can create or otherwise collect new resources and try them out in a context that indicates simulation and evaluation, thus not needing to activate them in the “real” operational system until they are deemed to be ready. Second, the system can adjust the context conditions under which certain resources are selected and adapted, so that resources may be used in different ways.

We now speculate on the applicability of both the CARS testbed and the incremental approach as strategies for eventually implementing active experimentation and computational reflection in mission-critical SoS.

## 7 Prospects

In the CARS testbed, we have the luxury of allowing the system and its agents in the true operational environment to practice, make mistakes, learn its characteristics (e.g., turning ratio, speed on different surfaces etc.), and even damage an agent without dire consequences to itself or to the rest of the testbed, somewhat as children learn their capabilities and the constraints of their various environments through play. However, in addition to pre-defined test sets, we speculate that in fact the style of self-modeling, learning, and subsequent recording of new rules and constraints that we have advocated for the CARS testbed could become very useful for offline testing and progressive integration of parts of a SoS. In our approach, each component and subsystem of the CARS is constantly developing better and better rules and constraints on its behavior and its allowable

operational envelope. The result is that because the “experimentation” is being developed in parallel from the point of view of many different types of components playing their diverse roles, the system is very likely to discover much more about potential problems than a test set developed by even a knowledgeable and experienced system engineering team.

This kind of exploratory behavior is an extension of exploring the system’s external environment to exploring the space of potential behaviors. Since this space is enormous, some very powerful directive constraint mechanisms will be needed to keep the system within some reasonable expectations, and some very powerful verification and validation methods will be needed to assure us that the system will accede to any safety- and mission- critical constraints we may choose to impose.

Eventually, we can envision a situation in which the components themselves when faced with a novel component interface or configuration or operational setting can request a time out, a voluntary removal of themselves to maintenance / self-examination / hypothesis generation and testing mode in a simulation. Imagine that in addition to the meta-knowledge normally provided to a SoS broker, each component / system has strong self-models at multiple time and space resolutions that are being continually refined with interaction with other components and environments. Initially, as a new configuration of components is brought together with the top level descriptions provided to the broker in the Wrappings, there will now be a deeper process of negotiation among the components as their self-models now compare constraints, expectatins, rules for best practice, and other behavior modification and constraint conditions. If a component is now faced with either an unknown situation (a new condition for which it has no rules or constraints) or a partially violated constraint (whose priority might not be that high), it can request that the system allow it to temporarily go into maintenance mode.

Of course, to be able to entertain this type of negotiation will take more information in the self-model about that component’s expected CONOPS, in addition to its expected environment. The system will either have some type of holding action it can take or it might request the broker to provide it a new component and go on. Meanwhile the offline component now starts a set of experiments in the safe simulation, with current operational setting values and conditions and with either the other relevant software components (clones) and / or emulated hardware components. If its experiments go well enough (measured by seriousness of system use), then that component can go out of maintenance mode and back online. At that point, it tracks and records all the real results of its interactions in this new use or configuration for future rules and constraints. If the results of the experimentation are equivocal, then human intervention may be requested for further experiments.

To summarize then, we want to develop methods that allow and even encourage processes that continually improve the performance of a system through better use of its existing resources, the correct incorporation of new component

resources, and the most appropriate integration among resources given the current operational context and CONOPS.

## 8 Conclusion

A study of biological systems suggests possible strategies for creating robust adaptive responses of a complex SoS to changing or even unanticipated conditions. In this paper, we focused on one such strategy: active experimentation. We have shown that successful active experimentation requires several specialized capabilities that, taken together, amount to computational reflection. We then proposed various strategies for implementing active experimentation and computational reflection in mission-critical systems of systems.

We have suggested that having testbeds like CARS allows components, subsystems, and systems to build ever-improving self-models based on active experimentation. The active experimentation coupled with the reflective reasoning processes allows these components / systems to develop and refine rules and constraints with specific details about different operating conditions and other components or systems.

We have then speculated that some of these approaches could be applied to mission-critical SoS by taking advantage of the offline maintenance mode allowed for most SoS components / subsystems. This second best case is to have during maintenance, some way of setting up a safe operational environment (set of simulations and emulations) for the offline components to actively experiment performing new behaviors, joining in novel configurations of components, or experiencing new operational settings. This experimentation would help refine the current self-models to take into account these new conditions.

The last case is to develop a new style of negotiation where components are outfitted not only with their own constraints and behavioral rules, but also CONOPS that helps explicitly define the expectations for how this component is expected to be used under different circumstances. This negotiation would be going on in parallel across layers of systems and components, allowing many lines and types of detailed interactions to be analyzed by the self-modeling processes. In this last most speculative case, based on this negotiation, individual components would request being put into study mode (offline maintenance mode and into operational simulation mode) in order to follow up on any conflicts with current constraints or lack of information on requested behaviors.

We are hoping this paper will stimulate a community wide discussion into many different ways one could create safe places for self-modeling and experimentation resulting in better system integration, system validation, and system performance.

## References

1. Tegest Ayche and Steffen Jung. The axis of tolerance. *Science*, 343(6178):1439–1440, 2014.

2. Michael Bell. Introduction to service-oriented modeling. In *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley, 2008.
3. Kirstie Bellman, Christopher Landauer, and Phyllis Nelson. Systems engineering for organic computing: The challenge of shared design and control between organic systems and their human engineers. In Rolf Würtz, editor, *Organic Computing*, volume 21 of *Understanding Complex Systems*, pages 25–80. Springer Berlin/Heidelberg, 2008.
4. Kirstie L. Bellman, April Gillam, and Christopher Landauer. Challenges for conceptual design environments: The vehicles experience. *Revue Internationale de CFAO et d'Infographie*, September 1993.
5. Kirstie L. Bellman and Christopher Landauer. Computational embodiment: Biological considerations. *Proceedings of ISAS'97: The 1997 International Conference on Intelligent Systems and Semiotics: A Learning Perspective*, pages 422–427, 1997.
6. Kirstie L. Bellman and Christopher Landauer. Reflection processes help integrate simultaneous self-optimization processes. In *Proceedings Second International Workshop on Self-Optimization in Organic and Autonomic Computing Systems (SAOS 2014), 27th International Conference on Architecture of Computing Systems (ARCS 2014)*. Lübeck, Germany, February 2014.
7. Kirstie L. Bellman, Christopher Landauer, and Phylis R. Nelson. Managing variable and cooperative time behavior. In *First IEEE Workshop on Self-Organizing Real-Time Systems*, Carmona, Spain, May 2010.
8. Kirstie L. Bellman and Phyllis R. Nelson. Developing mechanisms for determining ‘good enough’ in sort systems. In *2nd IEEE Workshop on Self-Organizing Real Time Systems (SORT 2011)*, Newport Beach, CA, March 2011. (presentation).
9. Kirstie L. Bellman and Donald O. Walter. Biological processing. *American Journal of Physiology*, 246:R860–R867, 1984.
10. M. Kwiatkowska. Advances in quantitative verification for ubiquitous computing. In *Proc. 11th International Colloquium on Theoretical Aspects of Computing (ICTAC 2013)*, volume 8049 of *LNCS*, pages 42–58. Springer, Heidelberg, 2013.
11. M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. On incremental quantitative verification for probabilistic systems. In Andrei Voronkov and Margarita Korovina, editors, *HOWARD-60: A Festschrift on the Occasion of Howard Barringer’s 60th Birthday*, pages 245–25. 2014.
12. Christopher Landauer. Abstract infrastructure for real systems: Reflection and autonomy in real time. In *Proceedings SORT 2011: The Second IEEE Workshop on Self-Organizing Real-Time Systems*, Newport Beach, California, March 2011. (presentation).
13. Christopher Landauer. Infrastructure for studying infrastructure. In *Proceedings of ESOS 2013: Workshop on Embedded Self-Organizing Systems*, San Jose, California, June 2013. (presentation).
14. Christopher Landauer and Kirstie L. Bellman. Generic programming, partial evaluation, and a new programming paradigm. In Gene McGuire, editor, *Software Process Improvement*, chapter 8, pages 108–154. Idea Group Publishing, 1999.
15. Christopher Landauer and Kirstie L. Bellman. Self-modeling systems. In H. Shrobe R. Laddaga, editor, *Self-Adaptive Software*, volume 2614 of *Springer Lecture Notes in Computer Science*, pages 238–256. Springer Berlin/Heidelberg, 2002.
16. Christopher Landauer and Kirstie L. Bellman. Managing self-modeling systems. In H. Shrobe R. Laddaga, editor, *Proceedings of the Third International Workshop on Self-Adaptive Software*, Arlington, Virginia, June 2003.

17. Christopher Landauer, Kirstie L. Bellman, and Phyllis R. Nelson. Modeling spaces for real-time embedded systems. In *Proceedings SORT 2013: The Fourth IEEE Workshop on Self-Organizing Real-Time Systems*, Paderborn, Germany, June 2013. presentation.
18. P. Maes and D. Nardi, editors. *Meta-Level Architectures and Reflection*. North Holland, 1988.
19. Pattie Maes. Computational reflection. In Katharina Morik, editor, *GWAI-87 11th German Workshop on Artificial Intelligence*, volume 152 of *Informatik-Fachberichte*, pages 251–265. Springer Berlin Heidelberg, 1987.
20. Arthur Mortha, Aleksey Chudnovskiy, Daigo Hashimoto, Milena Bogunovic, Sean P. Spencer, Yasmine Belkaid, and Miriam Merad. Microbiota-dependent crosstalk between macrophages and ilc3 promotes intestinal homeostasis. *Science*, 343(6178):1477, 2014.
21. David J. Musliner, Timothy Woods, and John Marais. Identifying culprits when probabilistic verification fails. In *Proc. ASME Computers and Information in Engineering Conference*. August 2012.
22. M. Ryschkewitsch. Engineering of complex systems: Challenges and initiatives. In *7<sup>th</sup> Annual IEEE Systems Conference (SysCon)*, Orlando, FL, 2013.
23. W.T. Tsai, Xinyu Zhou, Yinong Chen, Bingnan Xiao, R.A. Paul, and W. Chu. Roadmap to a full service broker in service-oriented architecture. In *IEEE International Conference on e-Business Engineering (ICEBE 2007)*, pages 657–660. October 2007.
24. Serdar Uckum, Tolga Kurtoglu, Peter Bunus, Irem Tumer, Christopher Hoyle, and David Musliner. Model-based systems engineering for the design and development of complex aerospace systems. In *SAE Aerotech*. 2011.
25. Paolo Zuliani, Andr Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to stateflow/simulink verification. In *Formal Methods in System Design*, volume 43, pages 338–367. Springer, 2013.