



RT 204: Systemic Security and the Role of Heterarchical Design in Cyber-Physical Systems

Technical Report SERC-2019-TR-007

June 07 2019

Principal Investigator: Mr. Tom McDermott, Stevens Institute of Technology

Co-Principal Investigator: Dr. Valerie Sitterle, Georgia Tech Research Institute

Research Team:

Georgia Tech Research Institute:

Ms. Erika Brimhall

Dr. Dane Freeman

Mr. Nicholas Bollweg

Dr. Zachary Welz

Stevens Institute of Technology:

Ms. Megan Clifford

Sponsor: DASD(SE)

Copyright © 2019 Stevens Institute of Technology, Systems Engineering Research Center

The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract [HQ0034-13-D-0004, TO#0388].

Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense nor ASD(R&E).

No Warranty.

This Stevens Institute of Technology and Systems Engineering Research Center Material is furnished on an “as-is” basis. Stevens Institute of Technology makes no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose or merchantability, exclusivity, or results obtained from use of the material. Stevens Institute of Technology does not make any warranty of any kind with respect to freedom from patent, trademark, or copyright infringement.

This material has been approved for public release and unlimited distribution.

TABLE OF CONTENTS

Table of Contents	iii
List of Figures	iv
List of Tables.....	v
Executive Summary	1
Introduction	3
A Functional Ecosystem View for CPS Analysis	4
Overview of RT-204 Approach.....	6
Benefits of Approach	7
MBSE Foundations, Background, and Rationale.....	9
MBSE in the Systems Design Process and its Relation to Assurance	9
MBSE for Modeling Security – Summary of Previous Work.....	11
Functional Representation in MBSE – A SysML Overview	14
Expressing System Function through SysML Activity Diagrams	15
Task 1: MBSE to Discovery and Labeling of a Functional Graph.....	20
Terminology for Functional Graph Data Extraction from an Activity Diagram	21
Technologies for Functional Graph Data Extraction from an Activity Diagram	22
Implementation.....	24
SysML Data Model to RDF	24
System Model in RDF to Directed Functional Graph.....	26
Graph Compression.....	27
Task 1 Results and Discussion	28
Task 2: Graph State Simulation and Analysis	31
Directed Graphs as a Model to Capture Functional Architecture and Flow	32
Scalability – Auto-Attribution of Graph Properties	34
Implementation.....	34
Node State Dependencies	36
Simulation Data Structure and Flow	36
Node Functional State Relation	40
State Propagation Patterns	44
Threat Addition	45
Variation on Approach Using Probabilistic Graphical Models	47
Task 2 Results and Discussion	49
Task 3: Threat Characterization and Modeling.....	51
CPS Attack and Countermeasure Patterns.....	51
Tools to evaluate threat and countermeasure patterns.....	56
Mission/operational resilience analysis	58
System-Theoretic Accident Model and Process (STAMP)	60
Fault and attack tree analysis.....	61
Cyber Security Requirements Methodology (CSRM)	62
Limitations of Cyber Assurance Requirements Activities	63
Task 3 Results and Discussion	65

Discussion and Recommendations	66
Conclusion	69
Project Timeline & Transition Plan	71
Appendix A: List of Publications Resulted	72
Appendix B: Cited and Related References	73

LIST OF FIGURES

Figure 1. Cyber Resilience Analysis Model.	5
Figure 2. High-Level Overview of RT-204's Phase 1 Approach	7
Figure 3. SysML Diagram Types (http://www.omg.sysml.org/what-is-sysml.htm)	14
Figure 4. A SysML Activity Diagram Containing a Call Behavior Action (Action XYZ) that Invokes another Activity Diagram	16
Figure 5. The Activity Diagram Invoked by Action XYZ in Figure 3	16
Figure 6. The Flattened Version of the Activity Diagrams Shown in Figure 3 and Figure 4	16
Figure 7. SysML Control Flow and Object Flow	17
Figure 8. Parameter Nodes 1 and 2 are Inputs into the Activity Diagram and Parameter Nodes 3 and 4 are Outputs	17
Figure 9. Removal of Dangling Edges.....	18
Figure 10. A Single Dummy Function for Preserving Edges.....	18
Figure 11. Dummy Functions for Each Parameter Node	18
Figure 12. Activity Diagram with Actions, Pins, and Object Flow	21
Figure 13. High-level Flow of Information from SysML Model to Model Graph	24
Figure 14. High-Level View of RT-204 Process for Converting a SysML Data Model to RDF Format	25
Figure 15. Activity Diagram as a Complex Functional Graph.....	26
Figure 16. Activity Diagram as a Simplified Functional Graph.....	26
Figure 17. Example of a SPARQL query for SysML Activities	27
Figure 18. An Activity Diagram from SysML (left) with its In-Memory Graph Representation in NetworkX (right, circled).....	27
Figure 19. Activity Graph in NetworkX Before (left) and After (right) Applying the Compression Algorithm.	28
Figure 20. Abstract Representation of a CPS Configuration showing Cyclic Nature	33

Figure 21. Illustration of Difference Between (a) Traditional, Component-Based Representation of Functional Flow and (b) the MBSE Functional Architecture Representation.....	33
Figure 22. A High-Level View of the RT-204 Simulation Concept.....	35
Figure 23. Illustration of State Dependency Assumptions	36
Figure 24. Example Functional Graph Template.....	37
Figure 25. Expanded Example Functional Graph.....	37
Figure 26. Example Feedback Functional Graph Template	38
Figure 27. Expanded Example Functional Graph with Feedback	38
Figure 28. Example Functional Graph ($\eta=0.5$, $n_1=1$, $n_4=0$)	39
Figure 29. Example Functional Graph Node Capacities overtime ($\eta=0.5$, $n_1@0=0$, $n_4@1=0$, $n_4@24=1$).....	39
Figure 30. Example Functional Graph Node Capacities overtime ($\eta=0.5$, $n_1@0=0$, $n_4@1=0$, $n_4@24=1$).....	40
Figure 31. Node Functional State Behavior using Parametric Variation to Capture Functional Degradation	43
Figure 32. Node Functional State Behavior using Parametric Variation to Capture Functional Recovery.....	43
Figure 33. Illustration of Threat Functional Node Addition to an Existing Functional Graph of a Blue System.....	47
Figure 34. Example Factor Graph.....	48
Figure 35. Expanded Factor Graph	48
Figure 36. Need for Convergence of Safety/Reliability and Security Assurance.....	57
Figure 37. Functional Dependency Graph linking Mission Objectives to System Assets [adapted from Goldman, 2010].....	59

LIST OF TABLES

Table 1. Taxonomy of CPS Attack Types.	572
Table 2. Generalized Control System Attack Models.	573
Table 3. Generalized Control System Countermeasure Patterns [Wan, Canedo, Al Faruque, 2015b].	574
Table 4. Specific Countermeasure Patterns for CPS.	575

EXECUTIVE SUMMARY

Background:

This effort constituted Phase 1 of RT-204, *Systemic Security and the Role of Heterarchical Design in Cyber-Physical Systems*. Previous work, conducted in the pilot investigation of RT-180, envisioned a multi-year progression of research and associated development to support analysis of security protection strategies for cyber-physical systems (CPS). RT-180 proposed an approach based on the abstraction and subsequent simulation of dynamic processes on functional graph models. Extending RT-180 into RT-204, the ultimate goal is to produce systems engineering methods, processes, and tools that enable co-development of system models and threat attacker models to inform requirements and design of complex engineered systems, specifically CPS.

Purpose:

The work performed under Phase 1 of RT-204 takes the first steps toward developing and gradually maturing a capability to define and analyze security threats and counter-threat design patterns for cyber-physical systems. The envisioned approach aims to enable the greater community to rationally compare and select security implementations (a) *in the early stages of design*, ‘designing in security,’ and is (b) *also applicable to already designed systems* where security solution are needed.

Technical Scope:

Phase 1 of RT-2014 developed the initial foundation for a holistic approach to integrating the CPS, attack vector(s), and security implementation(s) into a unified ecosystem model. The model is an abstraction of the greater system functional behavior, able to show how different types of failures may propagate through the system and what observable conditions these failures may manifest. The initial efforts of Phase 1 focused on two concurrent lines of approach: 1) methods to support generation of dynamic graph models from information extracted from existing MBSE formalisms, and 2) augmenting these structures with appropriate abstraction of functional characteristics (including from threats and protection patterns) and a means through which the dynamics associated with these structures and characteristics can be evaluated. The approach seeks to reveal how well security design choices preserve critical system functionality necessary for mission success.

Findings:

A primary discovery revealed through this Phase 1 effort is need to look more deeply into how to model system functionality to develop simulations of cyber-physical systems. Several fundamental questions vital to modeling CPS in a formal MBSE sense as well as make these models amenable to a dynamic analysis of security design evaluation must be addressed. Specifically, how should a functional architecture be defined in the form of an activity diagram that brings cyber and physical function types together with threat attack patterns in a meaningful and representative way and at what level of decomposition? Maturing future work under RT-204 from this focal point will better position the research to answer the bidirectional problem of formal MBSE specification and simulation of the specified functional architecture. As part of the

maturation of this work task, a pilot library for cyber-physical systems, their potential threat attack patterns, and associated security protection patterns should be specified. This would enable systems engineers to partially automate the assembly of a new meta-model from existing component libraries containing key CPS components and/or associated functions, common threat vectors and associated attack libraries, and libraries of security design patterns. Additionally, future work will still need to mature the threat characterization and modeling space. This is a monumental gap in current understanding and practice. Specifically, how can various threat types be best expressed as functional patterns themselves? Research is still needed for a consistent, repeatable way to extract relationships between threat vectors and functional assets common to cyber-physical systems.

Conclusions:

RT-204 established that a functional model extracted from a formal system description, augmented with attack graphs, and a library of protective functional patterns, may provide an effective path toward earlier-stage design and analysis of security for cyber-physical systems. Further, the general approach developed in this work may serve as the basis for a repeatable, yet flexible approach. It is abstract enough to scale with increased model size, especially if the notion of a function library for CPS is established for community maturation. Dynamic simulation of an ecosystem view – comprised of the original (unprotected) cyber system, the threat functional capabilities and attack vectors revealing the critical cyber assets they will target – can provide insight concerning the health of the functional state space at a level of abstraction that should prove meaningful for design.

The framework and foundations developed in Phase 1 of RT-204 are extensible to future maturation and use with different “truths,” or views that comprise the MBSE process and practice. Together, they provide a path forward for simultaneous exploitation of MBSE, Digital Engineering, and Model-Based Design. Specifying an activity diagram view – a directed graph model of a functional architecture depicting functional elements and the resource, logical, or causal flow between them – as done in this work has tremendous implications for future practice. This approach may lead to a definition of best practices for transforming functional architectures into true, active analytical tools and not merely reference design templates.

INTRODUCTION

Defense systems are increasingly composed of heterogeneous elements, both cyber and physical in nature. While this coupling can produce capabilities beyond those achievable before, it also makes these systems vulnerable to classes of threats previously not relevant for many physical control and computational systems. The Defense community needs methods to evaluate the ability of CPS to maintain mission-effective capabilities under threat as well as approaches to help design an effective control structure that reduces adverse events.

This effort constituted Phase 1 of RT-204, Systemic Security and the Role of Heterarchical Design in Cyber-Physical Systems. Prior work, conducted in the pilot investigation of RT-180 [Sitterle and McDermott, 2017], envisioned a multi-year progression of research and associated development to support analysis of security protection strategies for cyber-physical systems (CPS). RT-180 proposed an approach based on the abstraction and subsequent simulation of dynamic processes on functional graph models. *Extending RT-180 into RT-204, the ultimate goal is to produce systems engineering methods, processes, and tools that enable co-development of system models and threat attacker models to inform requirements and design of complex engineered systems, specifically CPS.*

The work performed under Phase 1 of RT-204 takes the first steps toward developing and gradually maturing a capability to define and analyze security threats and counter-threat design patterns for cyber-physical systems (CPS). The envisioned approach aims to enable the greater community to rationally compare and select security implementations (a) in the early stages of design, ‘designing in security’, and is (b) also applicable to already designed systems for which a security solution is needed.

Initial work in this phase was used to inform and was informed by, a general set of approaches for functional design of cyber-physical systems. These approaches are published in *Design Automation of Cyber-Physical Systems* [Al Faruque and Canedo, 2019]. The research team coauthored a chapter in the book, “System Assurance in the Design of Resilient Cyber-physical Systems,” capturing much of RT-204’s early research on functional design patterns. It is important to note that this research extends previous SERC research in System-Aware Cyber Security effort [Jones and Horowitz, 2012] towards a goal of automation. To this end, two aspects of this work are re-emphasized. First that the manual analysis of potential threat attacks will always be only partially successful due to the complexity of these systems and must move toward standard libraries of threat attack and countermeasure patterns; and second that the evaluation of the resulting system functional design must be supported by automated simulation and analysis tools.

This document constitutes the final technical report for RT-2014, Phase 1. It describes the background and rationale for the approaches taken, the results, and lessons learned from these approaches. The report also discusses implications with respect to best practices and future development in order to make this entire process scalable and able to be captured into standard practices and tool libraries.

A FUNCTIONAL ECOSYSTEM VIEW FOR CPS ANALYSIS

Cybersecurity may be considered in the broader concept of resilience in that security concentrates on protecting defense systems from sentient adversaries. CPS are generally designed by initially specifying critical and other necessary functionality. The high-level functionality is decomposed into specific functional capabilities, and system requirements derive from these functional needs. Boehm and Kukreja [Boehm and Kukreja, 2015] distinguish between functional and non-functional requirements as what the system does and how well it does those things, respectively. Security, a non-functional requirement, is assessed on how well a given security design pattern protects the system as intended without adversely impacting the intended functional operational and performance capabilities.

Systems theory has traditionally distinguished structural constitution from how a system behaves and deduced the latter from the former. This notion is the foundation underlying hierarchical construction of systems with defined input and output interfaces across multiple modular components and, to date, has resulted in an extensive focus on structural system representations in current cybersecurity protection methods. In CPS, however, structure and function are intrinsically linked. The interactions across the engineered network of their physical, computational, and communications components are precisely what provide new functionality and capability. [Al Faruque and Canedo 2019] provide extensive discussion of the importance of functional modeling and simulation in the design of CPS.

Graph-based methods, a core analysis approach highlighted in the RT-180 report [Sitterle and McDermott, 2017] and in this document, commonly evaluate the resilience of complex systems based on structural properties. Metrics of various static graph characteristics to measure node “importance,” connected components, shortest paths, etc. are ubiquitous in literature studies, even if the systems in question are dynamic. The literature is also replete with studies on the dynamics of networks, meaning how a network changes its structure over time. This is distinct from studies of dynamical processes on networks, which are necessary to capture functional preservation of CPS effectively.

Structural bias overtly conflates the broader sense of resilience with robustness or reliability, resulting in a focus on the perimeter or component-specific protection methods to prevent threat intrusion. Attacks are viewed the same way one would view failure modes. In practice, however, cyber threats attack both structural components of a system and its functions. These attacks may consequently produce effects in ways that do not exhibit characteristic system failures.

Figure 1 [Goldman 2010], adapted from MITRE’s Mission Assured Engineering process, depicts the importance of the combined functional and structural views. System functional decomposition is a process to derive system functions from mission objectives and associated operational tasks (or use cases). This determines the design of the system in terms of physical and information-based structural components, or “assets.” Threat attack analysis attempts to determine vulnerabilities by tracing the relationships between system structural assets back up to mission functional objectives from the threat’s point of view. Traditional perimeter-based

security strategies only focus on threats and vulnerabilities in the structural components – system and information assets. Cyber threat intrusion detection in a control system is dependent on monitoring for compromised behavior in the control system functions themselves. It is also vital to consider missions and operations in the process as “maximum reasonable assurance” must be evaluated for the mission.

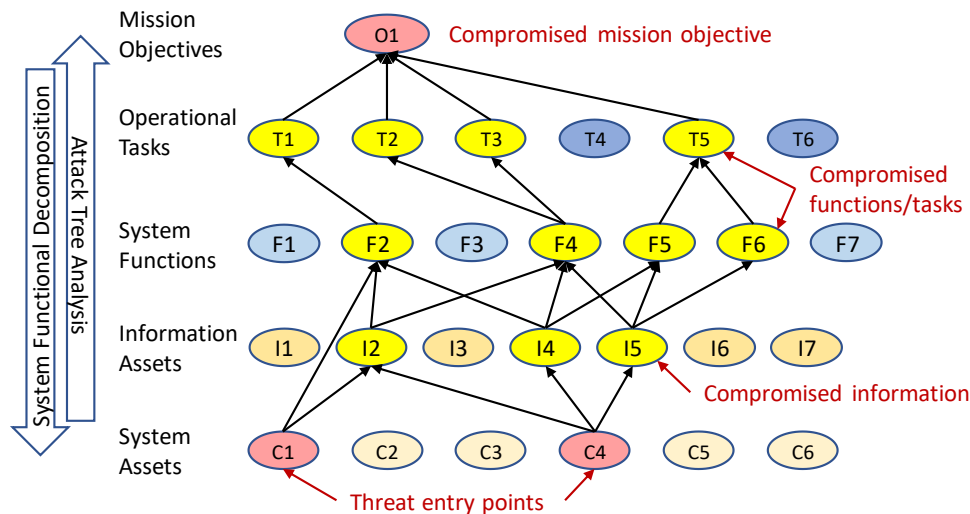


Figure 1. Cyber Resilience Analysis Model.

Exemplary work to date in model-based system assurance (MBSA) models of the threat impact to system function and structure separately, with system threat countermeasures focused primarily on the structural aspects of the modeling artifacts as an output of the model. This is evident in both the SERC System-Aware Cyber Security effort and separately in DARPA’s High-Assurance Cyber Military Systems (HACMS) and Cyber Assured Systems Engineering (CASE) programs. Although the functional characteristics of the systems are analyzed in the security design process, the model-based design automation and evaluation approaches in efforts to date remain focused on either structural correctness or on threat/countermeasure functions selected via structural asset analysis. Model-based approaches that retain an accurate functional characterization of assurance threats and countermeasures are still needed.

Relatedly, current model-based system design paradigms, in general, are also system-centric, meaning they contain no notion of external threats. While natural threats may loosely be addressed via measures of reliability, sentient threats (i.e., security) are external to the system in question. Security threats are consequently not usually captured in SysML or UML system views – the threats are evaluated separately or manually annotated in the models.

This produces a gap in current MBSE-driven analyses because of the heterarchical nature of CPS. They are comprised of numerous, heterogeneous elements acting both independently and interdependently. Unlike traditional defense systems, they are spatial and logical in scale and complex in their behavior dynamics. Because of this complexity, traditional decomposition and

predictive methods are insufficient, and the interdependency makes a threat to a critical system function inseparable from the original system.

As described in the SERC System-Aware Cyber Security effort [Jones and Horowitz, 2012], cyber system protection may be achieved through any number of implementable techniques that provide capabilities to respond to certain types of threats (e.g., detect when a system asset has been compromised, isolate a compromised asset, restore an asset to an original state, etc.) that are similarly applicable to CPS. These techniques may be represented as design patterns, each contextually relevant to a given threat type, and are ideally reusable from one system to another. Security is realized by incorporating these design patterns into the system, thereby generating an adapted system architecture with new or improved, threat-specific capabilities. The set of selected security design patterns form the security architecture. Together, the initial system, threat patterns, and protection patterns placed to mitigate the threat change the overall system structure and hence may alter functional behavior.

A system's structural characteristics and what processes and behaviors are possible both within and as produced by that system are not separable. A functional architecture (the topology of functional flow and relationships) therefore provides the basis for revealing functional behavior (how the dynamics associated with these processes and flows propagate through that topology). System security analysis via heterarchical models that unify these functional topological-behavioral dependencies may better inform CPS protection strategies than hierarchical WBS structures and component lists.

A functional viewpoint, as undertaken in this work, formulates a dynamic representation of system behavior at a level that is understandable, relevant to system design and analysis, and scalable. A functional view does not replace a structural, component-based view. Instead, it is a complement, capturing the system at a different level of abstraction.

OVERVIEW OF RT-204 APPROACH

Understanding the structure-function inseparability and relation of threats described above, RT-204 strives for a holistic approach integrating a CPS, attack vector(s), and security implementation(s) into a unified, functional, ecosystem model. The resultant model is an abstraction of the greater system functional behavior, manifested as a directed graph where each vertex (node) is a functional element and edges show functional flow. Functional flow may be data, energy, control, etc. Through simulation, this type of structure will be able to show how different types of failures may propagate through the system and what observable conditions these failures may manifest.

An underlying assumption of this research is that functional abstractions of attacks and counter-measures (whereby attacks reduce or disable desired system functionality and protection methods increase or enable functionality) can be captured in a directed graph construct augmented from the functional graph of the original system.

Phase 1 of RT-204 focused on two concurrent lines of approach: 1) development and evaluation of methods to support generation of dynamic graph models from information extracted from existing MBSE formalisms, and 2) augmenting these functional architectures with appropriate abstraction of functional characteristics (including from threats and protection patterns) alongside a means through which the dynamics associated with these structures and characteristics can be evaluated.

The first task above sought to effectively extract necessary system information from a relevant MBSE structure presumed to exist already. The effort sought to do so in a way that would enable repeatable, automated generation of a graph structure with the correct functional specification and relationships. Capturing the “original” system is a critical first step necessary prior to augmenting a graph with attack vectors (i.e., threat functional patterns) and envisioned security protection patterns.

The second task sought to develop the foundations necessary to simulate dynamic processes that could occur on the functional graph topology. This effort investigated functional relationships and what approaches, at what levels of abstraction, could represent the functional states possible for each functional element in the architecture. As before, this is a critical first step before to addition of threat and security protection patterns.

The overall approach for Phase 1 of RT-204 is depicted in Figure 2.

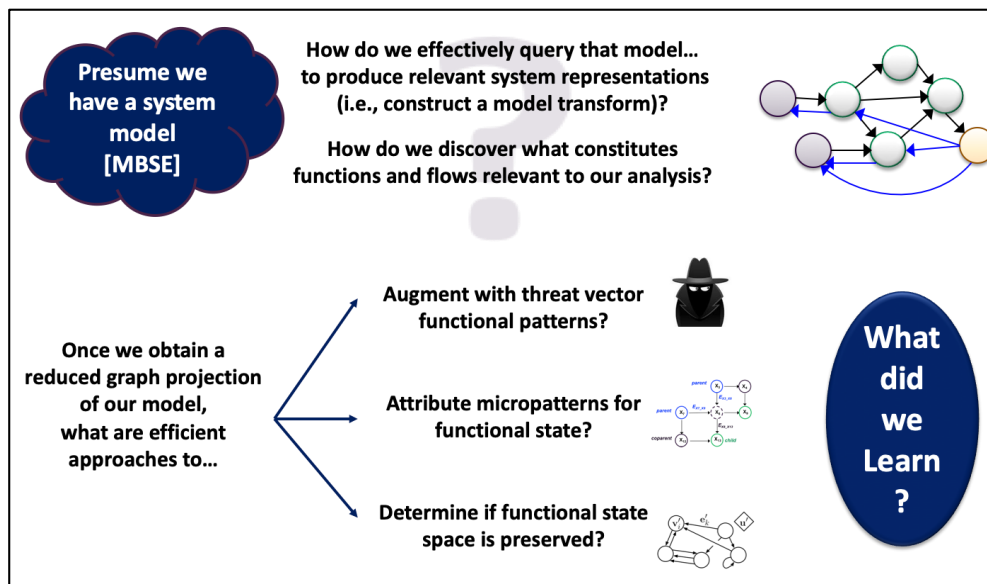


Figure 2. High-Level Overview of RT-204's Phase 1 Approach

BENEFITS OF APPROACH

This research specifically focuses on the complex interactions between threats, well-intended protection patterns, and the resultant system behaviors that may or may not be what was intended. As the approach taken for RT-204 matures, systems engineers may ascertain how well

the original functional capabilities of the system being designed are preserved in the face of the threat(s) given the augmentation with the security design pattern(s). The methods this work proposes apply to both early stage and more mature designs for which a security analysis is needed and hence has the potential to answer critical questions not adequately addressed through the currently disparate processes.

MBSE FOUNDATIONS, BACKGROUND, AND RATIONALE

Two primary research questions serve as the guideposts for this effort:

1. Can a model-based process be built in concert with existing MBSE practices to produce an evidentiary case that a system is trustworthy with respect to the properties its stakeholders legitimately rely upon within acceptable levels of risk?
2. What model-based approaches capture relevant and representative levels of abstraction sufficient to help validate the integrity of the system requirements and the integrity of the design?

The push toward a digital engineering thread has moved MBSE from a document-centric practice to one that emphasizes the role of models as the documenting entities, i.e., is model-centric. The emphasis is on digital (i.e., computer-based) models that may offer integrated views across the various structural, functional, and requirement layers; the specification is a product of the modeling process. RT-204 focusses on the functional information that can be extracted from, analyzed, and fed back into the MBSE design process.

The guiding principle for RT-204 is that the work should define a methodology and associated set of processes that are repeatable across different, unique cyber-physical system representations. Overall, this research task focuses on determining, recommending, and piloting the initial frameworks and methods to serve as a direct compliment to existing model-based systems engineering (MBSE) processes and tools. The ultimate goal is to extend the envisioned executable ecosystem model to assurance test framework and patterns, thereby enabling the community to maintain explicit knowledge of vulnerabilities and corrective patterns in design models.

As presented in the *Overview of RT-204 Approach* section, Phase 1 of RT-204 focused on two concurrent tasks: 1) development and evaluation of methods to support generation of dynamic graph models from information extracted from existing MBSE formalisms, and 2) augmenting these functional structures with appropriate abstraction of functional characteristics (including from threats and protection patterns) alongside a means through which the dynamics associated with these structures and characteristics can be evaluated.

The remainder of this section will review current MBSE formalisms with respect to capturing system functions and behavior. Specifically, this section will focus on which aspects of this diverse practice serve as starting points for RT-204 and the rationale behind the approach.

MBSE IN THE SYSTEMS DESIGN PROCESS AND ITS RELATION TO ASSURANCE

As a conceptual approach, Model-Based Design (MBD) is an approach that has been embraced and undertaken by engineers. Every model is an abstraction, and engineers and scientists all rely on different tools and an array of modeling approaches to support design and analysis. In engineering practice, this is often more formally defined as Model-Based Engineering (MBE), “an approach to engineering that uses models as an integral part of the technical baseline that

includes the requirements, analysis, design, implementation, and verification of a capability, system, and/or product through the acquisition lifecycle” [NDIA, 2011].

At a higher level, Model-Based Systems Engineering (MBSE) is described as “the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases” [INCOSE, 2007]. A system model represents structure, behavior, requirements, and parametrics (equations). In formal MBSE, these aspects are coupled.

Model-Based Assurance (MBA), in turn, aims to serve system verification and validation requirements. MBA requirements are typically determined via the development of formal assurance cases. Assurance case design for cybersecurity cases is addressed in the DARPA HACMS and CASE programs but still has limited acceptance across the community.¹ Assurance cases specify primary verification and validation requirements for the system design process. Verification and validation of mission-critical systems through test and evaluation has historically been the gold standard for assurance but is significantly expensive and increasingly fraught with difficulty as systems – CPS in particular – become more complex, more expansive, and more inter-dependent on other systems to realize their intended capabilities.

The use of models to support analysis of assurance, while promising and necessary, still faces an uphill challenge to establish the best practices and systems engineering foundations required to produce what can be counted as evidence to support assurance judgments. This is especially true for CPS, often employed in system-of-systems operational configurations, increasingly connected and increasingly complex when considered in the context of their higher order dynamics with other systems in the environment and facing increasingly diverse and sophisticated threats. MBA is a set of SE activities using a model or group of models as a basis of understanding to produce evidence that a given system will perform as intended in various potential environments, operational conditions, arrangements with other systems, etc. Part of MBA includes ensuring that a model allows determination of whether a system design meets functional requirements. To do so requires ensuring that the model, with all of its levels of abstraction, represents accurate system functional performance and characteristics.

CPS create new challenges to the concept of MBA. Specifically, what model-based approaches capture relevant and representative levels of abstraction sufficient to help validate the integrity of the system requirements and the integrity of the design? While most models may not exist at a level of specification such that a system may be completely built from their template, many of these models can specify diverse categories of functional or other performance requirements a system will need to be safe and secure within a reasonable level of uncertainty. To satisfy analysis for assurance, a model that is an abstraction of a system’s functional behavior should represent failure effects in the system, how failures propagate through the system, and observable conditions those failures manifest.

¹ Standardization of cyber assurance case processes and metrics is being addressed in a separate SERC research project.

New methods and tools are needed for the design and analysis of CPS dependability and security. Tools and techniques exist for design and analysis of fault-tolerant systems, and these tools may be adapted for application in the CPS domain. However, traditional tools are insufficient for resilient CPS due to the complexity of these systems and the potential for cyber-attack. Engineers traditionally view threats and uncertainties as stability and/or robustness issues instead of security concerns, and current MBSE methods consequently evolved to support these perceptions.

In 2012, Burmester et al., described a framework for modeling security of a CPS in which the behavior of the adversary is controlled by a threat model that captures the cyber and physical aspects – the cyber with discrete values and the physical with continuous. The framework addresses combined, dependent vector attacks and synchronization/localized issues, and can be used for formal proofs of the security of CPS. The authors distinguish that traditional computer and network security approaches do not address how systems can both outlive and recover malicious attacks (survivability and recoverability). The authors also discuss that securing a CPS goes beyond securing the individual system components, illustrating that a highly skilled attacker may use a multi-vector attack exploiting weaknesses – while individually the attack may not pose a severe threat, the more significant, dependent attack vectors may be disastrous. The paper attempts to address the gap by extending the traditional Byzantine fault model for cryptographic applications to CPS – the Byzantine faults paradigm has the CPS represented as a set of linked abstract machines, and the messages between the CPS components are represented as formal expression [Burmester, 2012]. The threat model that controls the framework has the vulnerabilities identified by the system specifications. The threat model does not address security aspects that are not part of the security specifications.

The analysis of vulnerabilities of CPS to external attacks continues to garner attention, and regardless, the general approach has been to study the effect of specific attacks against particular systems components and not the holistic, overall system. Pasqualetti et al. describe a unified modeling framework for CPS and attacks. While motivated by existing attack scenarios, the CPS was modeled as a descriptor system subject to unknown inputs affecting the state and measurements. The research showed fundamental limitations of a static, dynamic, and active monitors – illustrating attacks can be done without knowledge since systems are often built to an “as-known” basis. The authors then provided a graph-theoretic characterization of undetectable attacks through the theory of structured systems [Pasqualetti, 2013]. This still, however, is based on linear causal models that sequence a series of events over time.

Game-theoretic methods for resilient control design have been researched to help develop a framework that studies the tradeoff between robustness, security, and resilience. The hybrid dynamic game model provides holistic and cross-layer viewpoints in decision-making and design for CPS. Zhu presented a general class of system models (where nonlinear ordinary differential equations describe the physical system and the Markov models capture the cyber system), and deliberated the need for the framework to further extend to other classes of systems including

sample data, systems with delayed measurements, and model predictive control systems. The author states that the optimal design of new classes of systems can follow the same games-in-games, but future research should be pursued on stronger coupling, where control and defense strategies depend on both cyber states and physical states with better tools to compute the strategies [Zhu, 2015]. While the research explores the game-theoretic approach to security, it is done solely through the hierarchical lens, and still needs stronger coupling between cyber and physical system components as well as threat/attack strategies.

Since 2015, modeling security in CPS, specifically the threats, has increased. Kalinina et al. explored the detection of threats in CPS based on deep learning methods using multidimensional time series. The method is based on the use of neural network technologies to predict the values of the time series of system data. It then identified deviations between predicted and current data. The authors used SCADA-solution for visualization and collection of data on the state of the technological process. The use of a recurrent multilayer network allowed predicting the expected values from the objects of the system, but additional research is needed for the optimization of the model and increased accuracy. The methods also do not adequately represent threat/attack models [Kalinina, 2018].

The DARPA HACMS program focused on creation of technology for the construction of high-assurance cyber-physical systems, where high assurance is defined to mean functionally correct and satisfying appropriate safety and security properties. Key HACMS technologies include interactive software synthesis systems, verification tools such as theorem provers and model checkers, and specification languages. The HACMS effort used a combination of formal assurance case languages and system architecture modeling to produce a system that could be validated for secure construction via a combination of formal proof and code generation. A combination of SysML and the Architecture Analysis and Design Language (AADL) were used to model the system. SysML was primarily used to define the structural characteristics of the target system. AADL is a design specific language focused on modeling the computing hardware and software components of real-time embedded systems. It represents the dynamics of process scheduling and control flows but is limited to modeling behavior as state machines does not specifically model functions as activities. This work is ideal in transition from top-level functional design to system construction. It does not include features that aid in functional simulation of the combined system and threat as envisioned in this research.

Wan, Canedo, and Al Farugue have extensively explored functional modeling as a means to express both CPS design and functional behaviors of CPS threats. This work has focused on the use of the National Institute for Standards and Technologies (NIST) Functional Basis language as a baseline for functional decomposition into low-level base functions [Hirtz, 2001]. This work proposes that both the CPS system and threat/countermeasures can be described functionally using standard base function patterns [Wan, Canedo, Al Farugue, 2014]. This work is perhaps closest to the RT-204 view of the modeling problem and will be further explored in future phases of the research as a tool development baseline.

The SERC's System Aware Security work has explored the representation of both system behaviors and threat behaviors in formal MBSE representations, specifically in SysML. The RT-204 effort wholly adopts the threat analysis process described in this research will reuse the process in this and the next research phase. The limitation of the system aware work today is that the dynamic behaviors of the system and the threat are still modeled separately, and the resulting analysis still derives a primarily structural model of the full system. Work continues to explore better methods to dynamically model and simulate the combined system/threat functions.

Moving laterally to the way other researchers are looking at attack models, Rezaei and Liu discuss adversarial and evasion attacks on developed deep neural network models. Deep learning, in industry, has been used in numerous applications from speech recognition, image classification, and more. Training these deep models can be time and cost consuming, so often transfer learning is often adopted. Transfer learning is transferring a part of the network that has been trained on a similar task, adding layers, perhaps more critical to the task, and then retrained. However, with transfer learning, the models that are pre-trained are normally either open source or the code is available to everyone. Google Cloud ML tutorial suggests using Google's Inception V3 model as a pre-trained model, and Microsoft Cognitive Toolkit suggests using ResNet18 as a pre-trained model. Therefore, it is known to potential attackers. Attackers can launch target-agnostic attacks to fool the entire network when only the pre-trained model is available. Therefore, the research team uses the system model with the transfer model input, simulating the attacker that does not know the retrained model but has already infiltrated aspects of the pre-trained since it was publicly available. The attack design, due to the lack of sufficient non-linearity, can iteratively generate a set of inputs each of which triggers only a single neuron at the final transferred layer and quickly craft adversarial sampled that trigger one class with high confidence [Razaei and Liu, 2019]. The paper is a direct emphasis on the need to approach the development of models concerning threats and attacks. Open source, or at least the ability to view the code, of an iterated base model, provides insight into the development for an adversary. However, open source can be an excellent way to build systems of massive, intricate design

As a community, a modeling paradigm is needed that holistically unifies traditional MBSE with outputs of attack tree analyses and design alterations resulting from security implementation. Complimenting these existing processes with methods that embrace the inherent complexity or resultant ecosystem may reveal emergent behaviors, economies and diseconomies of scale, and consequences not otherwise seen. Additionally, augmenting the design and analysis process to elucidate structure-function relationships in a formal modeling paradigm will traceably capture how the functional dynamics of a system are changed by considering the entire ecosystem. In turn, RT-204's formal model will enable engineers to explore the consequences of attacks that penetrate perimeter protection in un-anticipated ways, moving security analyses beyond the limits of approaches described above. Finally, the model can inform and be augmented with system assurance test models that capture the context of the attack for formal test.

FUNCTIONAL REPRESENTATION IN MBSE – A SysML OVERVIEW

The Systems Modeling Language (SysML), a dialectal extension of the Unified Modeling Language (UML) 2, has emerged as the de facto standard system architecture modeling language MBSE applications. SysML is “a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems” (<http://www.omg.sysml.org/what-is-sysml.htm>). It was created for the specific purpose of modeling systems – the behaviors, requirements, and structure, as illustrated in Figure 3.

SysML is considered an enabling technology for the practice of MBSE and, for this very reason, RT-204 started with how system functions and behaviors are expressed in SysML as an initial point from which to draw information to support an abstract simulation of system behavior.

A model can contain as few or as many of the diagrams shown in Figure 3 as needed to convey the concepts relevant to the particular system. The various types of behavior diagrams all convey something about what the system *does* – the system’s functionality. Requirement diagrams indicate what the system must satisfy. The structural diagrams describe the physical or logical components of a system, their interfaces, their groupings, and their relevant parametric equations. Relationships can be defined between elements on different types of diagrams to provide an integrated understanding of the system.

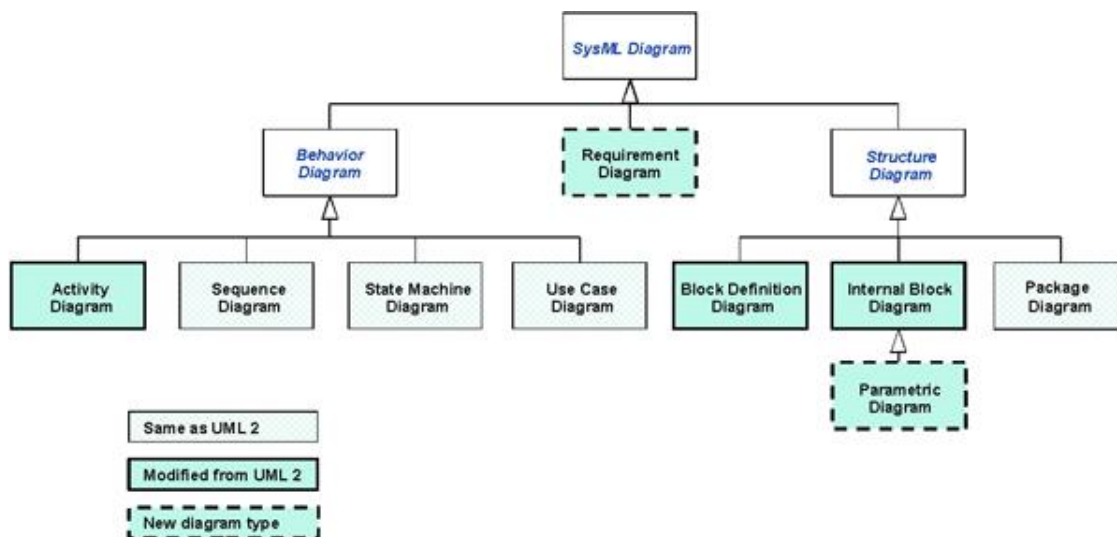


Figure 3. SysML Diagram Types (<http://www.omg.sysml.org/what-is-sysml.htm>)

Finally, the different types of SysML behavioral diagrams each provide a different perspective of how the system functions:

- Activity Diagrams - Represent a set of actions and the flow connecting them.
- Sequence Diagrams - Represent how messages are passed between parts of a system.

- State Machine Diagrams - Represent event-based states of a system.
- Use Case Diagrams - Represent the basic usage of the system

Sequence diagrams relate to the passing of messages between structural aspects of the system, not functions. State machines could be converted into a graph, but the nodes would be the states of the system rather than the functions of the system. Use case diagrams provide a very simple look at what a system can do without much of an indication of how things work together, so they are not well suited for generating functional graphs.

Activity diagrams, however, are particularly well suited for being converted to functional directed graphs since the actions can be directly converted to functional nodes and the flow can be converted to edges. In an activity diagram, flow from one action to another can represent the flow of information, energy, material, etc.

RT-204 consequently uses activity diagrams as the MBSE basis of functional behavior representation.

EXPRESSING SYSTEM FUNCTION THROUGH SysML ACTIVITY DIAGRAMS

Understanding how directed graphs representative of system functionality may be extracted from SysML first requires understanding how these behaviors are represented in the MBSE formalism. The specifics associated with the semantic rules governing the expression of allowable elements that compose an activity diagram also highlight characteristics that pose challenges or may necessitate additional processing before arriving at a graph model amenable to simulation as a dynamic construct.

SysML activity diagrams are “similar to traditional functional flow block diagrams but with many additional features to precisely specify behavior” [Friedenthal, Moore, and Steiner, 2015, p205]. They represent a set of actions (functions) and the flow connecting them. Various types of actions exist in SysML, including actions that invoke other behaviors (e.g., they can trigger another activity complete with other actions and flows, allowing nested activities – see Action XYZ in Figure 4, expanded in **Error! Reference source not found.**) and primitive actions that simply perform a standalone function. Each action can be converted to a single node in a functional graph, but including a flattened version of the nested activities as shown in **Error! Reference source not found.** leads to a more comprehensive conversion of the model, allowing each nested node and edge to be affected individually.

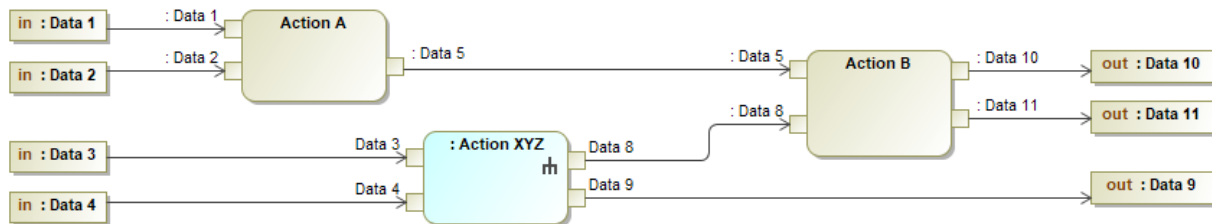


Figure 4. A SysML Activity Diagram Containing a Call Behavior Action (Action XYZ) that Invokes another Activity Diagram

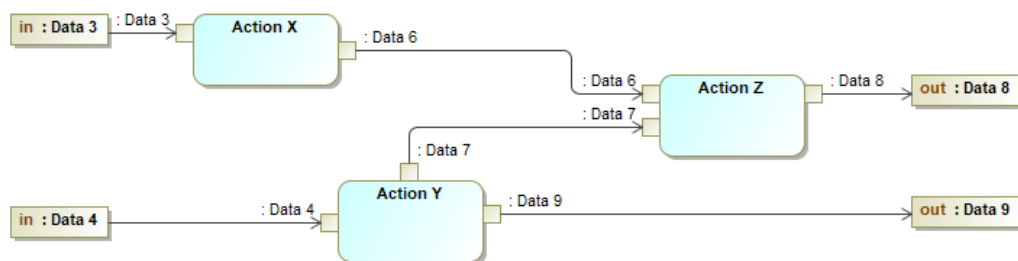


Figure 5. The Activity Diagram Invoked by Action XYZ in Figure 4

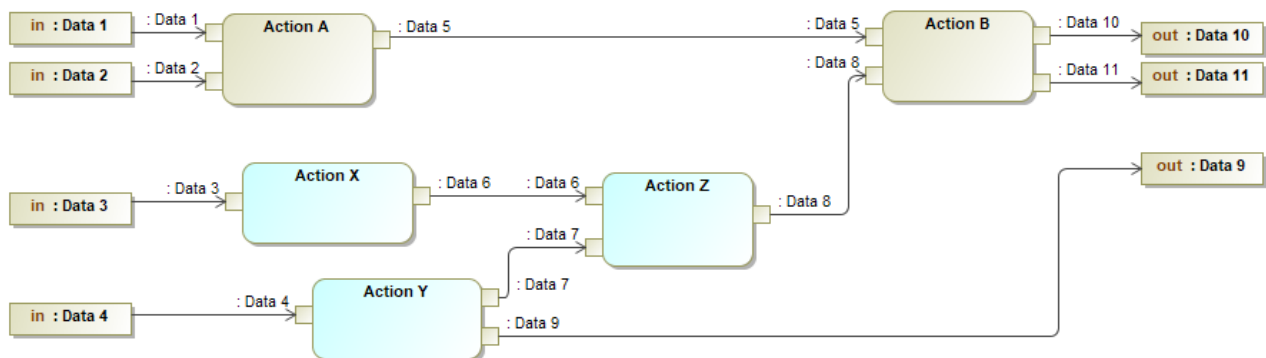


Figure 6. The Flattened Version of the Activity Diagrams Shown in Figure 4 and Figure 5

In SysML activity diagrams, there are two types of flow: control flow indicates when one action ends and triggers another, while object flow passes objects, material, or data in addition to triggering the next action. SysML flow is “based on token-flow semantics related to Petri-Nets.” [Friedenthal, Moore, and Steiner, 2015, p208] Either type of flow, illustrated in Figure 7, can be used with action nodes to form a directed functional graph, but object flow allows the edges to contain information about what exactly is being passed between functions. This type of attribute is helpful in cases where different types of edges should be treated differently, such as in the

case where a threat affects only certain types of edges (e.g., RF signals) and not others (e.g., power).

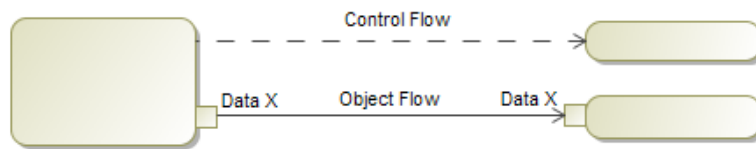


Figure 7. SysML Control Flow and Object Flow

Another notable element within SysML activity diagrams is the use of parameter nodes. Inputs can enter into an activity via parameter nodes, and outputs can exit the activity in the same way as depicted in Figure 8. This is helpful from a modeling perspective as it shows what inputs are needed to trigger each initial action within the functional boundary of the activity, and it shows what outputs are provided to outside the system boundary.

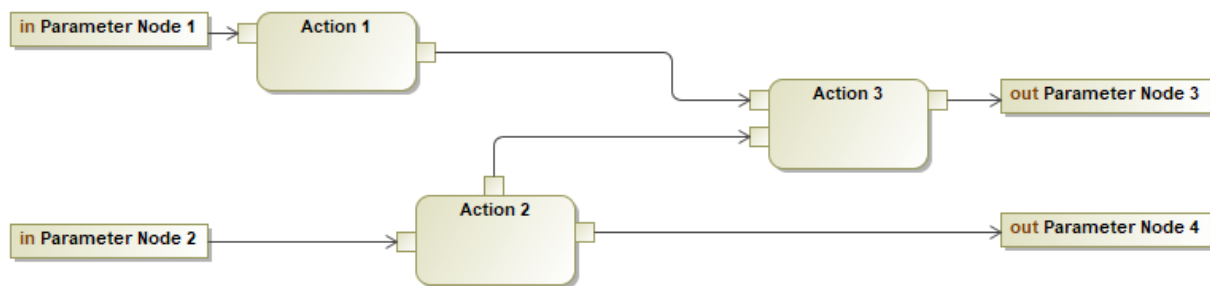


Figure 8. Parameter Nodes 1 and 2 are Inputs into the Activity Diagram and Parameter Nodes 3 and 4 are Outputs

However, from a graph-drawing perspective, parameter nodes produce “dangling edges” that only have a node on one side. That is, a graph has a functional resource as an input or output edge to a functional node, but with no functional node on the other end of that edge.

Consequently, when converting a SysML activity diagram with parameter nodes to a functional graph, there are three options:

1. Simply remove the dangling edges (Figure 9),
2. Add in a single dummy node to act as the source of all parameter node inputs and the sync for all parameter node outputs (Figure 10), or
3. Add in multiple dummy nodes for each dangling edge or each relevant grouping of dangling edges (Figure 11).

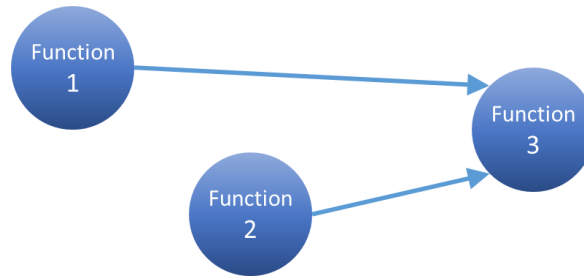


Figure 9. Removal of Dangling Edges

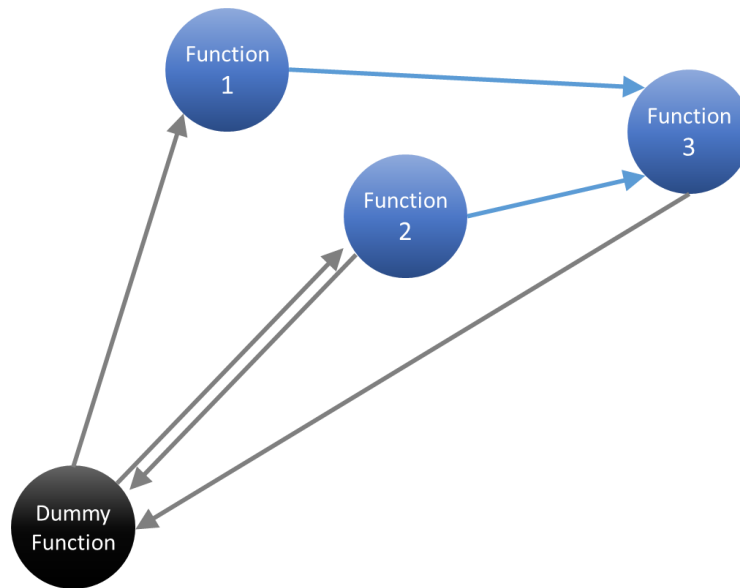


Figure 10. A Single Dummy Function for Preserving Edges

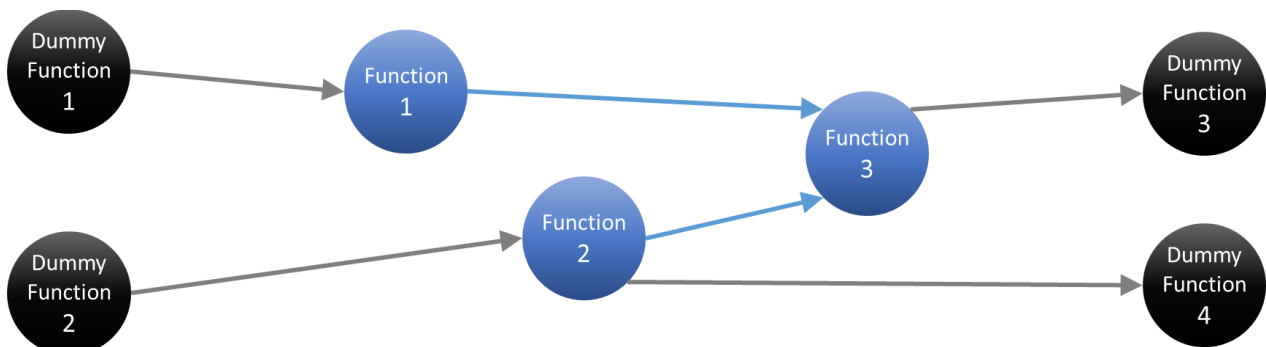


Figure 11. Dummy Functions for Each Parameter Node

Each option has advantages and disadvantages. Removing dangling edges causes a loss of edge data but keeps the functional model limited to the scope of the functional boundary set by the

original activity diagram. Using a single dummy node (or one node for inputs and one for outputs) preserves the edge data while adding minimal extraneous nodes to the graph. Adding a node for each dangling edge or group of dangling edges allows processes that act upon nodes to affect different edges in different ways. Though potentially burdensome due to the potential number of added nodes, this can provide insight into data sources and targets.

Which of these approaches constitute best practices for the functional modeling and dynamic analysis of security designs for CPS will be determined as the work matures and experience is gained with relevant examples.

TASK 1: MBSE TO DISCOVERY AND LABELING OF A FUNCTIONAL GRAPH

The first task under RT-204, Graph Structure Discovery, addressed one goal: *presuming a model exists in an MBSE formalism such as SysML, develop a way to effectively query that model to produce relevant (here, functional) system representations*. Capturing the “original” system is a critical first step necessary prior to augmenting a graph with attack vectors (i.e., threat functional patterns) and envisioned security protection patterns.

When starting with a SysML or UML activity diagram, there are a variety of approaches to convert it into a format ingestible by graph software for analysis or simulation:

1. **Manual Transcription into Tables** – This involves manually transcribing actions in an activity diagram and their inputs and outputs into a table, then converting the information to a format such as CSV. This approach is the most time consuming and prone to human error, so it is only recommended in cases where the system model is not available, and the only content available is a diagram. Additionally, some critical information may not translate well into a concise normalized table structure due to the inherent graph nature of the system being modeled.
2. **Autogenerated Tables** – This involves using the system modeling tool to automatically generate tables of relevant information about activity diagrams and then saving those tables in a format such as CSV. The efficiency and precise process for this approach is tool-specific and may be much easier in some tools than others. With this approach, there is added metadata overhead to maintain the semantic meaning of the column relationships between the various tables.
3. **XMI Parsing** – XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML). As such, XMI is a specific application of XML. XMI parsing involves the automated extraction of relevant content from a system model’s XMI file and then putting it into a more convenient format such as a database, graph store, or series of CSVs. Even though it is an OMG standard, parsing XMI requires detailed knowledge of a specific modeling tool’s XMI formatting, which may change between versions, and can therefore result in brittle implementations.
4. **Modeling Tool Plugin** – This involves creating a plugin to automatically identify relevant content and then either generate a file in a format such as CSV or directly feed the information into the graph analysis software. This requires knowledge of the specific system modeling tool’s API, which could change between versions, and it is based on vendor implementation. It is not necessarily based directly on an OMG standard.

At a minimum, the relevant system model content to extract would include each action with its outputs and the action to which that output is an input. More information can be included as needed to enable a richer understanding of the model.

Phase 1 of RT-204 started with evaluating the XMI parsing approach described in (3) above. This approach was selected due to its potential for automation and deep flexibility to what content about activities and related information from other model views could be extracted and correlated.

This effort sought to do so in a way that would enable repeatable, automated generation of a graph structure with the correct functional specification and relationships. Specifically, the task sought to automatically extract semantic descriptions of system functional elements that would include entity type, relationships with other entities of any type, and type of relationship (i.e., logical flow, information flow, or causal dependency). This information, once extracted, will serve as the template to create a directed graph of the system functional architecture.

Once the system functional information is extracted from the relevant activity diagrams into a form that may be used to create a directed graph, functional capabilities will be captured as graph vertices (nodes). Graph edges (links between nodes) will represent logical flow, information flow, or causal connectivity between those functional elements. Both direct and indirect connections can be consequential, together producing cascading effects in the event of a disruption.

TERMINOLOGY FOR FUNCTIONAL GRAPH DATA EXTRACTION FROM AN ACTIVITY DIAGRAM

A **system architecture** is a representation of a system appropriate to the system's current stage of development. Early in the design process, the system architecture is conceptual and simplified; later in the design process, it is more fully developed with details. A system architecture often includes multiple perspectives of the system, including a structural perspective that includes a hierarchy of components and a behavioral perspective (often called a **functional architecture**) that includes the system's functionality. This behavioral view can take many forms. For example, it can show functional interactions, message timing, states and transitions, or other behavioral information.

An **activity diagram**, simply illustrated in Figure 12, is one particular type of behavioral diagram. It specifically shows a series of actions (functions) and the flow between those actions. SysML and UML activity diagrams include many types of elements, including actions of various types, control flow, object flow, parameter nodes, interruptible node regions, partitions, structured activity nodes, merge nodes, decision nodes, join nodes, fork nodes, initial nodes, final nodes, central buffer nodes, datastore nodes, accept time event nodes, etc. Some of these are critical to conveying system functionality, while others provide supplementary implementation details.

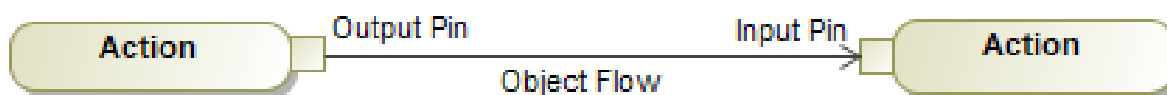


Figure 12. Activity Diagram with Actions, Pins, and Object Flow

A **functional graph** is a directed graph where each vertex (node) represents a function. This is quite distinct from graphs representing system structure, where each node represents a component. The difference in interpretation of a physical element (i.e., component) versus functional view of abstraction cannot be overstated. The functional architecture, the organization of functional relationships that determine functional flow and hence overall system behavior, is best represented through a directed functional graph.

TECHNOLOGIES FOR FUNCTIONAL GRAPH DATA EXTRACTION FROM AN ACTIVITY DIAGRAM

Reaching a queryable representation from a system model expressed in SysML through an XMI parsing approach requires additional technologies (i.e., more than simply SysML) to develop an extraction process that can be repeatable, automated, and scalable. This section will discuss those technologies and how they relate to one another in the path from transforming single source of truth models (e.g. SysML) into a complete graph, then reducing the graph to a content level appropriate for visualization and analysis. This process will support the application of higher-order abstractions to system models for functional execution discussed in

Task 2: Graph State Simulation and Analysis.

SysML, discussed in *MBSE Foundations, Background, and Rationale*, provides a standards-based data model and notation format for capturing system models, but vendor implementations are focused on the interactive editing experience by a single user in a dedicated tool. The standard SysML exchange format is the XML Metadata Interchange (XMI), which is neither a convenient nor efficient form for database storage due to its verbosity and difficulty to interpret as important model semantics encoded using tags, attributes, and nested elements. There are technologies, however, such as OpenMBEE (discussed below), which attempt to normalize the representation of the data model though interfacing with SysML via vendor tool plugins.

The **Open Model Based Engineering Environment (OpenMBEE)**, (<http://www.openmbee.org>) is not a single technology rather an open source ecosystem built to support model-based engineering (MBE). Developed and maintained by NASA JPL, OpenMBEE is an environment comprised of an amalgam of technologies that provides for modeling, reporting from a model, versioning, branching, and merging activities between heterogeneous data sources relevant to MBE. System models are constructed, queried, and rendered following the view and viewpoint paradigm. (OpenMBEE uses the SysML version of viewpoint and view. A viewpoint is a model of rules for describing, analyzing, and communicating a model of a system. A view is an interactive visualization of the formal model that conforms to a specific viewpoint.)

OpenMBEE includes plugins for commercial tools, including SysML. Presently, the Model Development Kit (MDK) is OpenMBEE's plugin for MagicDraw and is its most developed and supported plugin for formal MBSE tools. The MDK synchronizes the SysML model with OpenMBEE's Model Management Systems (MMS). Generally speaking, the MMS can be accessed by rich SysML desktop clients (like MagicDraw, now matured into Cameo) or any other tools that use REST (REpresentation State Transfer) web services. The MMS serves as model data versioning control, placing the underlying semantic model in (flat) JSON, and exposing the model elements via REST APIs.

JavaScript Object Notation for Linked Data (JSON-LD), (<https://json-ld.org>) provides a way to capture semantic meaning of the attributes in JSON, as used on OpenMBEE, which is simply an attribute-value pair data structure. This added layer of context in JSON-LD around the basic attribute-value concept is very flexible and assists with capturing the graph representation inherent in the SysML model. It allows elements to be disambiguated and object properties to be ontologically linked [W3C, 2018]. A linked data version of the original system model expressed using SysML may then be used to create a Rich Data Format (RDF) cache of the model, enabling semantic querying for relationships between objects.

The **Resource Description Framework (RDF)**, the technology behind the now popular "Semantic Web", is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. At a basic level, RDF describes resources (which could be anything) and their relationships. RDF is increasingly used for conceptual description or modeling of information implemented in web resources, using a variety of syntax notations and data serialization formats, and for knowledge management applications.

RDF content is stored as sets of triples, containing the (1) subject, the (2) object, and the (3) predicate (relationship). This format provides the linking structure to derive a directed graph representation where nodes represent subjects or objects and the directed edges indicate relationships.

The **SPARQL Protocol And RDF Query Language (SPARQL)**, <https://www.w3.org/TR/rdf-sparql-query/> is an RDF query language to retrieve and manipulate relationship-structured data. It provides full relational algebra capabilities (i.e., a full set of analytic query operations) whose schema is intrinsically part of the data. Collections of queries (called projections) allow the extraction of specific information from the original system model description that may then be analyzed or used in a different tool/ process. From the W3C report [W3C, 2008]:

“SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.”

For this application, SPARQL provides a convenient format for specifying patterns to extract targeted information from the RDF graph. RDF, in effect, produces a set of relationships that are intrinsically mini-directed graph structures; SPARQL can create customized queries of the RDF data store to produce a reduced set of related data structures, leading to a reduced graph representative of the queried model view.

IMPLEMENTATION

This portion of RT-204 sought to design and develop a process for extracting formal graph representations of SysML *Activity Diagrams*.

The high-level workflow involves the transformation of a SysML model into a graph representation that can be modified and executing based on the specific needs of the analyst. The process in Figure 13 is a depiction of the process and represents only one potential implementation for achieving the project goals.

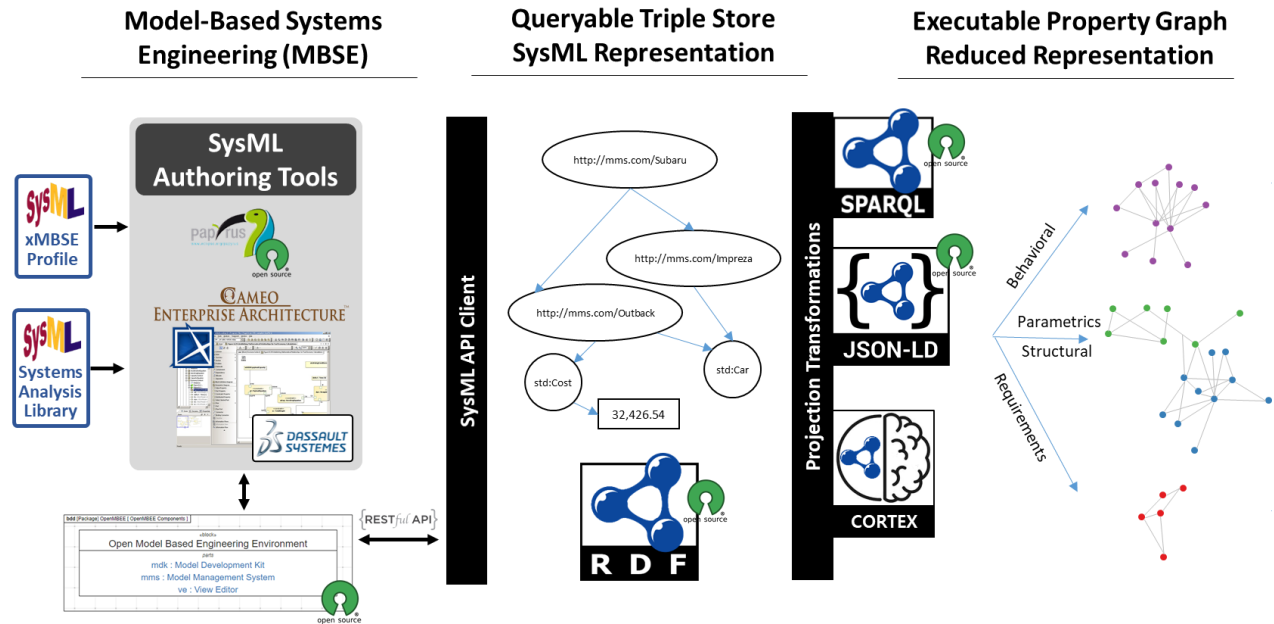


Figure 13. High-level Flow of Information from SysML Model to Model Graph

The workflow uses a collection of open source tools to parse SysML models (emitted by a proprietary application but in a relatively open format) exposed as a REST API into an in-memory graph, query the graph based on functional patterns, and compress the resulting graph into abstract representations. The deployment of this approach empowers a single analyst, and relies on either a hosted SysML API or a self-hosted instance. The following sections cover each stage of the process.

SysML DATA MODEL TO RDF

The implementation shown in Figure 13 uses SysML as the single source of truth model, which is implemented in the proprietary application *Cameo Systems Modeler*. This tool is well documented, and is a common application for working with SysML models [No Magic, 2014]. SysML/UML provides a standards-based data model and notation format for capturing system models, but is focused on the interactive editing experience by a single user in a dedicated tool. While Cameo is a powerful tool for viewing and editing these SysML models, it does not currently provide an API for performing actions from external tools. To expose the model as a REST API, the NASA JPL developed OpenMBEE, which provides standardized model serialization.

RT-204 started with very small, exemplary system models described in SysML using MagicDraw that were openly available (<https://www.nomagic.com/mbse/resources/examples.html>) as teaching tools. These models were then brought into OpenMBEE, which currently only has a plugin to read MagicDraw (Cameo) system model files. (Vendor tools do not have the same implementation of XMI standards, frequently even across different versions of the same tool, so the MDK plugin is not cross-compatible with other tools.)

Bringing the SysML system model into OpenMBEE exposes the model data as (flat) JSON through a REST API. Model data exposed via the REST API needs to be parsed into subject-verb-object triples (a standard linked-data structure), in order to perform tool-agnostic queries. This allows queries to be defined that isolate information within a model independent of the specific design implementation. To produce these triples, the JSON data exposed via the API is expanded into JSON Linked Data (JSON-LD) and parsed into a Resource Description Framework (RDF) graph.

To expand legacy JSON to JSON-LD, a context must be defined that captures inter-element relationships and facilitates easy integration within deployed systems [W3C, 2018]. GTRI converted the JSON to JSON-LD by defining a JSON-LD context (i.e., in effect a JSON file used to give other JSON data semantic meaning) in order to give data elements attributes that would correspond to their interpretation in the SysML views. The context defines, at a most basic level, “based on understanding SysML, this is what data attributes, and values mean and how they relate to one another in the MBSE sense.” Once the data is expanded, JSON-LD can be parsed into an RDF graph containing the model triples, which are then able to be queried via SPARQL.

The overall process, depicted in Figure 14, has several layers of transformation. First, the plugin must accurately convert the system model information from MagicDraw into the OpenMBEE format. OpenMBEE has its own transformation and viewpoint nuances. Among them, the JSON exposed by OpenMBEE is not a strict, universal standard. Consequently, the JSON-LD context is an interpretation of OpenMBEE JSON and the meaning of data elements and their relationships expressed in SysML to support of MBSE formalisms. Once in JSON-LD form, however, the system model information as RDF triples is lossless.

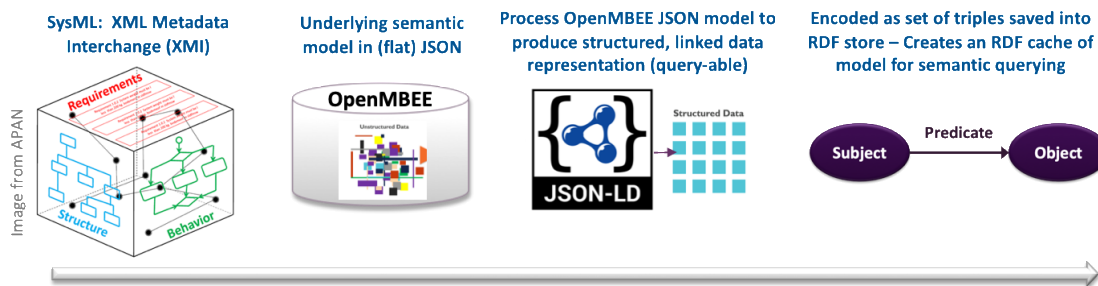


Figure 14. High-Level View of RT-204 Process for Converting a SysML Data Model to RDF Format

SYSTEM MODEL IN RDF TO DIRECTED FUNCTIONAL GRAPH

An activity diagram can be viewed as a type of functional graph that includes pin elements to represent function input and output resources. An object flow is specified to connect source and target pins, thereby linking associated actions in the diagram. Every element on an activity can be treated as a node to form a complex functional graph, depicted in Figure 15. Alternatively, the activity diagram can be collapsed to its core content, such that actions are nodes and object flows are edges as shown in Figure 16; this greatly simplifies analysis and reduces computational time.

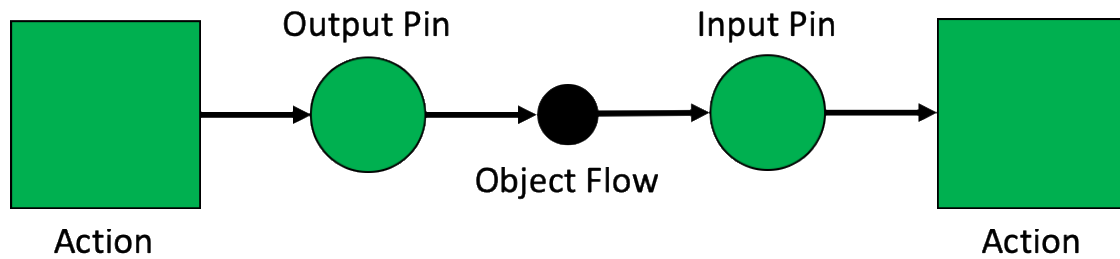


Figure 15. Activity Diagram as a Complex Functional Graph

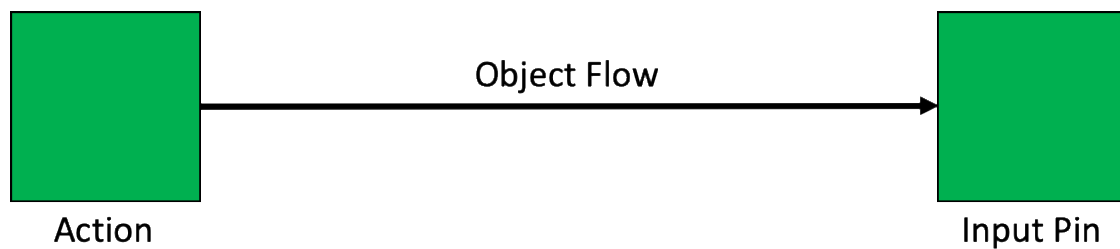


Figure 16. Activity Diagram as a Simplified Functional Graph

RDF content is stored as sets of triples, in the form of $\langle \text{subject}, \text{predicate}, \text{object} \rangle$. As a collection of multiple directed graphs (i.e., each triple), the edge is always directed from the subject to the object. The entire system model in RDF graph format contains valuable information, but must be simplified via reduction techniques to a manageable size for scalability and in-memory execution. To accomplish this, collections of SPARQL queries called *projections* are constructed, which enable the extraction of specific information related to an analysis or execution of the model. From Figure 13, the high-level projections for SysML models are *Requirements*, *Behavioral*, *Parametrics*, and *Structural*. For RT-204, the system model in RDF format was queried for action names, specifically looking for verb-object pairs using SPARQL. An example of a simple SPARQL query is given in Figure 17.

```

1  SELECT DISTINCT ?activity_uri
2  WHERE {
3      ?activity_uri rdf:type ns:Activity .
4  }

```

Figure 17. Example of a SPARQL query for SysML Activities

In RDF, everything is a node. This means that each 'arrow' for ObjectFlow in an activity diagram in SysML is a node in RDF, and the attribute that links something (i.e., a resource) to the ObjectFlow is the edge.

Consequently, conducting a query of an activity view does not derive from a simple $\langle action_1, functional\ flow, action_2 \rangle$ representation. It instead derives from the SysML semantics of how the elements comprising an activity diagram are specified. The functional flow from one action to another is therefore described by a set of related RDF triples $\langle subject, predicate, object \rangle$: $\langle action_1, relation, output\ pin \rangle$, $\langle output\ pin, relation, ObjectFlow \rangle$, $\langle ObjectFlow, relation, input\ pin \rangle$, $\langle input\ pin, relation, action_2 \rangle$. This process results in the in-memory graph representation shown in Figure 18.

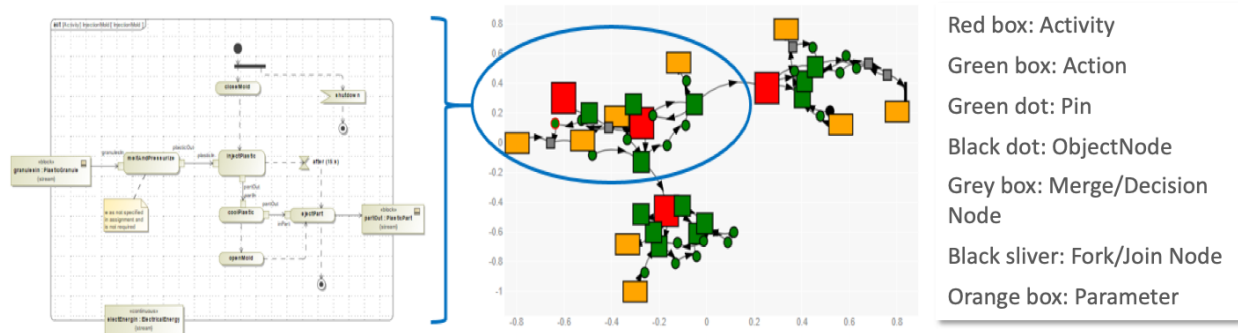


Figure 18. An Activity Diagram from SysML (left) with its In-Memory Graph Representation in NetworkX (right, circled)

The extracted in-memory graph representing the activity (functional) view is implemented with **NetworkX**, which is a common python property graph. The process for state abstraction and dynamic simulation is discussed in section

Task 2: Graph State Simulation and Analysis.

GRAPH COMPRESSION

While the NetworkX graph is a subset of model data in the RDF graph, there are cases that require additional reduction to minimal complexity representations. The preliminary functional analysis research conducted as the other main part of RT-204 involves abstract simulation/execution on system functional behavior. To support this research, a graph compression algorithm was constructed to transform the RDF-derived NetworkX graph into a simple Element-to-Element graph (i.e., action-to-action). A visual example is provided in Figure 19.

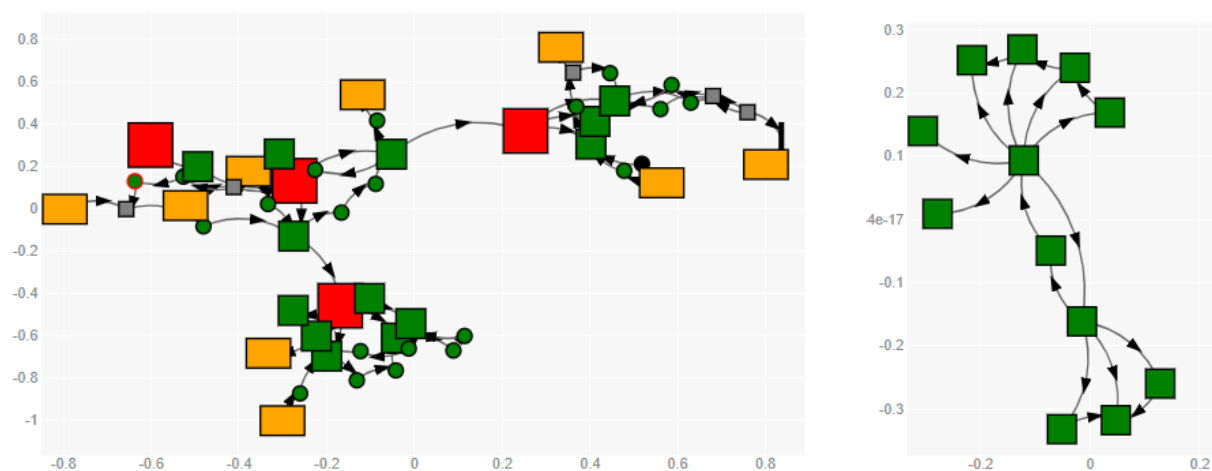


Figure 19. Activity Graph in NetworkX Before (left) and After (right) Applying the Compression Algorithm.

In Figure 19, actions are represented using green squares. In the left panel, Actions are connected to one another via paths between non-action elements. The compression algorithm collapses these paths into a single edge to produce the action-to-action graph in the right panel. The algorithm checks all outgoing edges and recursively calls checks on objects until either there are no more edges to traverse or an action is found. If an action is found, the edge from the original action to the discovered action is added to the reduced graph. This is repeated for every action.

This compressed graph produces a lower resolution action-to-action graph where immediate edges associated with other object (e.g., pins, ObjectFlow) are collapsed. This reduced view supports the other RT-204 preliminary research involving abstract simulation/execution on system behavior.

TASK 1 RESULTS AND DISCUSSION

This line of effort under RT-204 sought to evaluate (1) how a formal system model could be effectively queried to produce relevant system representations (i.e., how to construct a model transform) and (2) how the model-to-graph approach based on an RDF query could discover what constitutes functions and flows relevant to dynamic functional analysis of a CPS.

The overall implementation for this portion developed a process whereby SysML model data could be parsed into subject-predicate-object triples amenable to defined tool-agnostic queries. By developing methods that allow SysML models to be transparently surfaced next to other forms of captured knowledge, systems may be viewed as instances of more abstract or general patterns via their linkages. These patterns help describe the computationally interesting characteristics of a system design in a way not supported without a custom implementation with a SysML authoring tool, and make this knowledge usable by other methods, processes, and tools.

Firstly, there are several SysML elements associated with activity diagrams not yet being converted using the process outlined in this section:

- Activity Partitions: For a purely functional representation, activity partitions (“swimlanes”) are irrelevant because the physical allocation is irrelevant.
- Time Events: The timing of events is not needed for developing the functional graph representation.
- Initial, Flow Final, and Activity Final Nodes: The graph has no concept of Tokens; the entire graph executes sequentially to completion.
- Interruptible Activity Region: The graph has no concept of Tokens; the entire graph executes sequentially to completion.
- Merge, Fork, Decision, and Join Nodes: These are initially brought into the NetworkX graph, but they are collapsed during graph compression so that only function nodes (actions) and edges (object flow or control flow) remain.

There are also several process-related limitations associated with the approach in its current level of maturity. Firstly, currently implemented graph processing methods support activity diagram extraction via ObjectFlow only. The tool is designed to make projections based on portions of the model graph. The current implementation performs a behavioral projection that is only designed to extract out behavioral elements from Activity diagrams. These diagrams contain two categories of resource flow between elements: ObjectFlow and ControlFlow. The tool currently supports only ObjectFlow. This means that any elements that are separated from other elements via ControlFlow edges will not yet be present in the projection graph.

Similarly, combination of pin and ObjectNode notations will result in lost edges. As mentioned in previous sections, paths between actions typically follow pin or ObjectNode notation. In the event that the modeler chooses to design a path using both a pin and ObjectNode, no valid edge is generated between them at the API level. This prevents accurate extraction of ObjectFlow and will cause unexpected results in the graph.

Also, partition and group information is not extracted. While partition and group information is useful for logical grouping within SysML models, the current implementation does not extract or utilize this information. Users should not rely on these elements to provide information to help typify actions. However, future capabilities may support using this information if their use in system models is helpful for this class of problems.

Both of the concepts above – ObjectNode notations and partition/ group information – relate to best practices for modeling cyber-physical systems. There is not yet an established understanding for what best practices and associated levels of decomposition should be for CPS in functional architectures. This is discussed further in the *Discussion and* section.

Only certain predefined attributes are added to the RDF graph. In order to expand the model from JSON to linked data (JSON-LD), a context that describes each JSON attribute must be defined. Attributes that are not defined are not expanded or parsed into the RDF graph. Helper functions are included in the current tool to generate the full context, but the current implementation, as a proof-of-concept, covers less than 15% of all attributes. However, the context is designed for easy expansion as necessary, and will grow with the complexity of developed graph projections.

Relatedly, only certain predefined attributes are presently added to the property (NetworkX) graph. SysML element types are often more specific than is currently necessary in the context of the model graph. SysML classes (element types) are transformed into custom types within the graph projection algorithms. During this transformation, only specified attributes of interest are added to the property graph nodes. In many instances of the activity behavior projection, this is only the name and documentation for each node. Additional attributes can be added on an “as-needed” basis.

Additionally, most SPARQL queries are static. SPARQL is a powerful language for defining queries on RDF graphs. Even so, while query files can be written to support dynamic queries that are extensible to future model needs, many of the queries included in the current tool implementation are static and susceptible to edge case failure as the system models being analyzed grow in complexity. Future efforts will improve queries to support more general patterns and avoid scalability issues.

The current graph compression algorithm is limited to action-to-action elements and may result in lost edges. The graph compression algorithm was developed to support execution at high levels of functional abstraction. The current implementation starts at actions and traverses edges until another action is found, or until all outgoing edges are exhausted. Potential outgoing edges are also limited to attributes defined in the JSON-LD context. As a result, there may be edge cases that occur resulting in lost edges between actions. If the model data extraction approach described in this section continues to mature, this step in the process will be revisited as well.

Finally, current efforts utilize the Cameo Systems Modeler application to develop and visualize SysML models, and interface to OpenMBEE’s MMS API. When elements are created in SysML that are not visualized in the diagrams, these elements will still end up in the resulting model graph.

One common example is hidden pins, which exist on an action and are not visible. As a result, modelers will often create visible pins that are duplicates of their respective hidden counterpart. This causes instances of extraneous pins in the graph. Cameo has taken steps to highlight when this occurs, and modelers should take care to ensure no hidden pins exist before performing graph processing.

TASK 2: GRAPH STATE SIMULATION AND ANALYSIS

In CPS, many essential system properties such as stability, safety, performance are expressed in terms of physical behavior (i.e., how the system functions). This is important because the very interactions between cyber and physical functions that are necessary to produce new capabilities lead to more inherent system complexity while still requiring safety, security, and agile capability guarantees for actual use. These new types of interactions across what is now a highly extended spatio-temporal dimension produce the possibility that physical behavior of systems may be manipulated by adversaries capable of compromising a system. There is a lack of composition theories for these heterogeneous systems and an increase in unforeseen (and consequently called “emergent”) behaviors. Verification, predictability, and consequently assurance faces fundamentally new challenges.

Cyber systems are generally designed by initially specifying critical and other necessary functionality. The high-level functionality is decomposed into specific functional capabilities, and system requirements derive from these functional needs. Security is assessed based on how well a given security design pattern protects the system as intended – without adversely impacting the critical functional capabilities.

The aim of Phase 1 is to develop the methodological foundations whereby the functional efficacy of security protection patterns may be evaluated in the face of different threats to a level sufficient to guide design decisions and eventually support MBA activities.

The research approach in this second line of effort under RT-204 makes three central assumptions. Firstly, system functional behavior may be well represented to a level of abstraction suitable for simulation of functional flow

Secondly, functional abstractions of attacks and countermeasures, whereby attacks reduce or disable desired system functionality and protection methods increase or enable functionality, may be created. Thirdly, modeling the original system together with these abstractions as a directed graph supportive of simulation will reveal important behavioral dynamics across the CPS, threat, and protection elements. Namely, by adding consideration of the threat(s), and protection(s), a new system functional structure with new functional dynamics is produced.

The primary premise is that these system, threat, and security pattern functions can be represented using a directed graph structure. This structure will represent a functional topology, in turn allowing functional flow to propagate or be interrupted depending on the functional types and their relationships. Change the functional topology and the dynamic processes that may occur on that topology may similarly change.

This effort investigated how to capture notions of functional state for each functional element in the directed graph at levels of abstraction. Simultaneously, the work evaluated ways to simulate the functional flow, using the functional state abstractions, to represent what behavioral dynamics could occur on the architecture. As before, this is a critical first step prior to addition of threat and security protection patterns.

As discussed in [Sitterle and McDermott, 2017], graph theoretic measures based on aggregate statistics alone do not adequately characterize a system in terms of its throughput, performance, and vulnerabilities. Networks may have identical degree distributions, for example, and yet fundamentally different structures and functional performance [Li, 2007]. Similarly, graphs are commonly analyzed with respect to how structural statistical properties of a network topology will be affected by additions (growth or augmentation) and removals (failures or attacks) of nodes and edges, and particularly how networks can be more robust against the latter [Geard, 2010]. Those additions and removals are typically assumed as perturbations coming from external sources, not incorporated into the dynamics of the network itself. Moreover, the pilot research in [Sitterle and McDermott, 2017] found functional graph models of CPS are not random or statistically evolved but purposefully designed and often exhibit correlated patterns. This is very unlike most studies investigating attacks in uncorrelated, scale-free graphs that assume all nodes are identical [Newman, Barabasi, and Watts, 2011; Barrat, Barthélemy, and Vespignani, 2008; Boccaletti, Latora, Moreno, et al., 2006].

Key to the ability of a directed functional graph to elucidate relationships and performance behaviors will be its amenability to dynamic simulation (i.e., to reveal dynamic processes *on* graphs). As stated previously, nodes in the directed functional graph will represent the functional elements at whatever level of decomposition employed by the modeler, while edges will represent a logical flow, information flow, or causal connectivity.

The original notion was to cast CPS graphs as directed acyclic graphs (DAGs), a common approach for flow-based functional models. CPS differ very much from the simple-pairwise relationships traditionally modeled for many systems as their flow and logic relationships are directional. DAG representations were also used in [Luckett, 2014] to investigate the potential for graph models to serve as a structural framework for system-aware cyber security architecture selection. CPS, however, are not acyclic.

Figure 20 illustrates the cyclic nature of an individual CPS and group configuration. Higher-level functional capabilities are often achieved through local network structures that, while still directed, also contain cycles. While a cyclic structure may be transformed into an acyclic one by duplicating node types and adding them in appropriate places, this will not produce a functional topology representative of the CPS problem. In fact, cycles in CPS are often important sources of vulnerability (e.g., an attacker tricks the system into staying locked into a given functional state or cycle). Any simulation of dynamics must consequently be able to handle graphs with cycles.

Many instances of “functional representation” found in practice actually try to model functional or resource flow between system components. In formal MBSE, specifically in the Open Systems Architectures domain, functional architectures are not component-based but strictly function-based. How this manifested depends heavily on the modeler and level of decomposition used for the problem. The difference in these two views is illustrated in Figure 21, with various potential

threat vectors added. For RT-204, the latter view is used based solely on functions and does not include components.

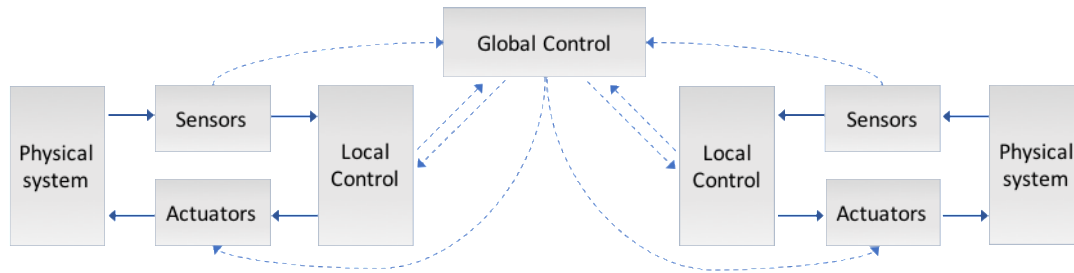
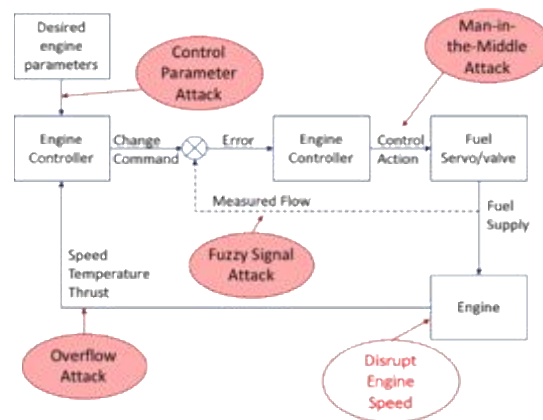
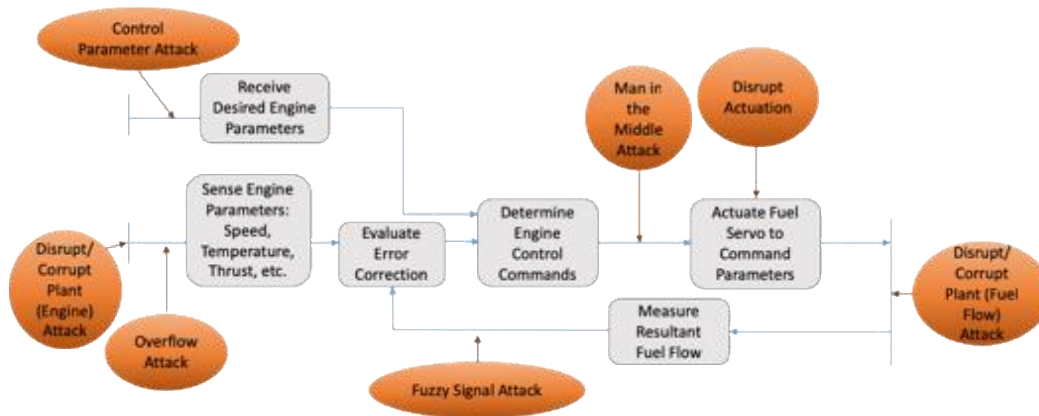


Figure 20. Abstract Representation of a CPS Configuration showing Cyclic Nature



(a) Traditionally used, component-based view of functional flow



(b) MBSE expression of a Functional architecture

Figure 21. Illustration of Difference Between (a) Traditional, Component-Based Representation of Functional Flow and (b) the MBSE Functional Architecture Representation

SCALABILITY – AUTO-ATTRIBUTION OF GRAPH PROPERTIES

Non-trivial CPS functional graphs are expected to have a significant number of functional nodes as well as an assortment of node types corresponding to different functional classes. Hand-evaluating each node and assigning it properties to represent its functional state would take an excessive amount of time and produce too much potential variation from case to case. To make simulation of CPS functional graphs scalable, motifs – algorithmic expressions that represent functional states of each distinct functional class – are needed. This will make the assignment of different properties to different nodes in a graph scalable while still enabling variation that keeps the abstraction of functional flow representative of the system being modeled.

The functional state associated with each node may be viewed as a node weight. How the weight is described in terms of its bounds and what other aspects, edges, and/or nodes of the graph that weight is dependent on will, in turn, determine the simulation dynamics. These functional state motifs should be generalizable enough by type to be reusable and allow attribution of similarly executable graphs at much greater scale.

The vision is that a library of functional element types will provide a customizable look-up from which different nodes may obtain class-appropriate motifs for functional state behavior. Through this approach, a directed graph derived from a system activity diagram may be specified in a table form, and each functional element auto-assigned representative expressions of functional state prior to simulation.

Key to this approach will be to simplify the relational state rules as much as possible to still reflect an abstraction of the contextual cyber system behavior over time. An advantage is that this work may begin to generate insights from a more restricted understanding of state expression and assignment of state variation by node type. As the community understanding of the problem matures, the approach is easily modifiable and extensible to reflect the lessons learned that will best capture CPS security analysis. These techniques will, together with the graph's functional structure, govern the spreading dynamics of the impact of the threat(s) and security implementation(s) and, consequently, the final state of the cyber system's functional capability.

IMPLEMENTATION

Unlike the random or statistically generated graphs studies in other fields, developed techniques must be best suited and modifiable for simulating the impacts of highly correlated threats and protection implementations on functional capabilities of a system. Graph parameters should produce behavioral consequences meaningful to the problem space and yet efficiently propagate changes in functional state across graph elements. Importantly, graphs will be directed and may also have cycles. This has significant implications that make traditional graph metrics and propagation techniques unsuitable for this space.

Some SysML modeling tools, including Cameo, include plugins that will enable direct “execution” of activity diagrams using a method directly analogous to Petri nets where “tokens” are passed

from one activity to another. Whether or not a given activity “fires” depends on whether or not it has met its required token condition (i.e., does it have the required number of tokens from across its inputs to execute). This approach can illustrate functional timing and dependencies for the system modeler. The tool is not, however, suitable for adding and exploring the notion of threats. It is very limited in terms of how sequencing is handled and not flexible with respect to how threats and actions could interact.

The work in RT-204 developed a framework specifically for the purpose of analyzing how failure due to security risk (whether through degraded functionality, system compromise, or system corruption) could propagate through a functional architecture. Moreover, that framework would be extensible to the analysis of security protection pattern efficacy. A tremendously significant consequence of the RT-204 approach is that functional states may recover, not simply degrade.

The simulation uses a propagation process similar to a feed-forward neural net. Functional nodes feed their state information forward to each of their children; there is no need for back-propagation in this process. Using a feed-forward process analogous to how water flows through channels enables graphs to be either acyclic or have cycles without creating degenerate calculation points. The Markovian nature of the propagation comes in through the state relation that depends not just on the state information fed forward from parent nodes but also self-state information, the current state of the node undergoing a state value update. A high-level depiction of this concept is shown in Figure 22.

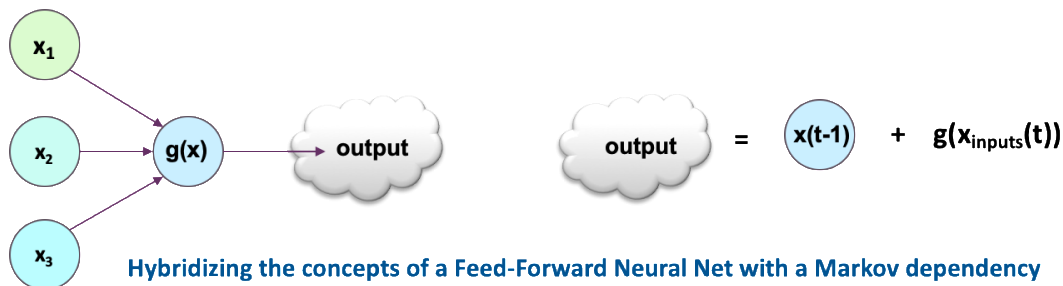


Figure 22. A High-Level View of the RT-204 Simulation Concept

Time is treated as an abstract quantity. Time steps are whatever the systems engineer views them to be, with the understanding that time steps should be considered equal for each state update. This is exactly how time is treated in finite state analysis.

Graph structures themselves are not dynamic entities during the simulation. The entire framework allows the functional graph structures to be easily modified, but the structures on which state changes propagate are static for the propagation simulation. Parent nodes and rules associated with parent nodes (e.g., max/min target states, derived target states, etc.) may consequently be determined up front.

NODE STATE DEPENDENCIES

Considering how function would flow across different functional elements, both cyber and physical, guiding assumptions were defined:

- Edges represent functional flow. This may be data, a resource, control, etc. that is produced by one functional element as an output and sent to another functional element (or elements) as an input or dependency.
 - A given edge state is assumed to be a function of its parent node state only (i.e., is the parent node functioning and to what degree).
 - By definition in a graph, an edge has a single parent node.
- Vertices (nodes) represent functional elements, whether cyber or physical in nature, at the level of decomposition present in the MBSE functional model view (i.e., the activity diagram).
 - Node states represent the functional state for each element.
 - A given node state is assumed to be a function that node's inputs and functional type.
 - By extension from the assumptions above, a given node state may be modeled as a function of the states of its parent nodes and its own functional type.

Future extensions may support a given node state to also be a function of the functional type of its parents. Presently, the work assumes that functional specification will be specific enough in the MBSE decomposition that the degree of dependency for a given node on its parents may be abstracted and parametrically controlled without this additional complexity. The specific node state relation serving as a starting point for RT-204 is discussed in the following section.

These guiding assumptions are illustrated in Figure 23.

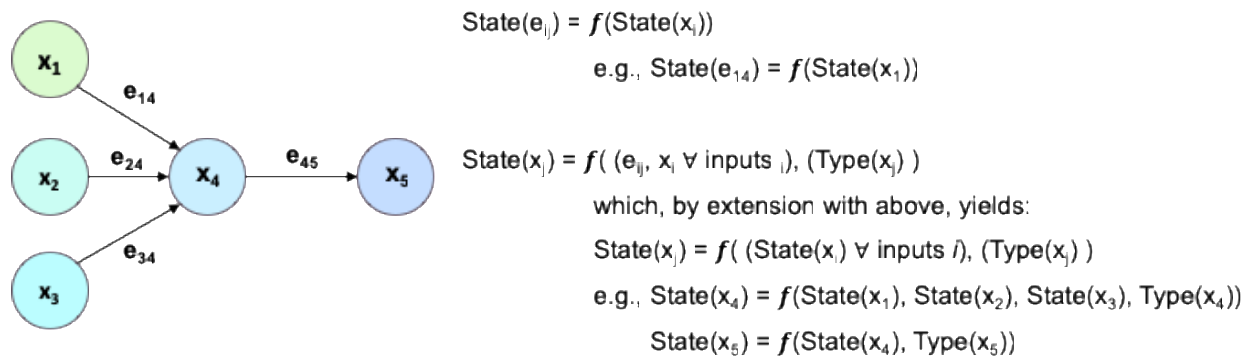


Figure 23. Illustration of State Dependency Assumptions

SIMULATION DATA STRUCTURE AND FLOW

At the core of the implementation is the graph template data structure. Figure 24 shows an example of one such functional graph where there are five unique nodes with edges representing a dependence relationship. From this figure, n0 and n4 are independent and both flow into n2.

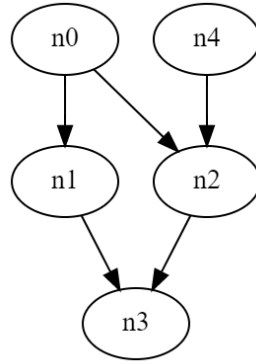


Figure 24. Example Functional Graph Template

This structure is treated as a template because the goal of this research is to start building formalisms to understand dynamics inside these functional graphs. For example, in Figure 25, the template has been expanded for three time slices. Now, in addition to intra-time slice dependencies captured in the structure of the template, there are added inter-time dependencies. These inter-time dependencies flow in an autoregressive sense whereby the future state of a node depends on the previous value.

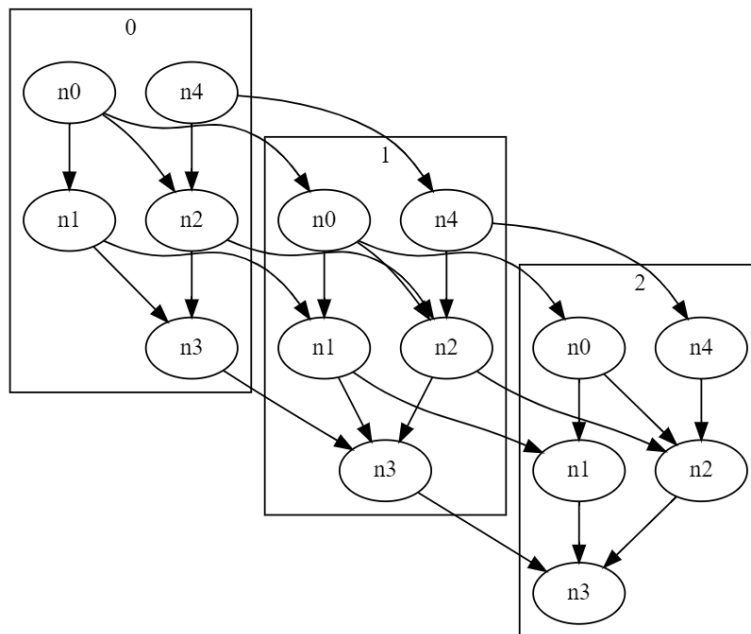


Figure 25. Expanded Example Functional Graph

In addition to the autoregressive connections, inter-time dependencies are introduced to capture feedback loops in the graph. Figure 26 demonstrates a slight variation on the simple example from above in that n2 informs both n3 and n0 at the next time step denoted using the offset time in brackets. Although there is a feedback loop it is clear how states evolve and what is specified as the initial conditions.

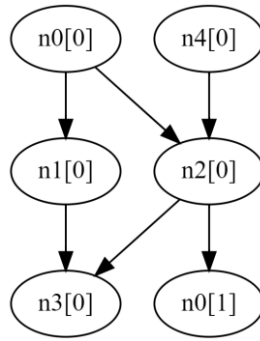


Figure 26. Example Feedback Functional Graph Template

Because of the flexible nature of this graph templating data structure, expanding the graph in the same manner as above results in Figure 27. Importantly, while there is a feedback loop, the temporal nature is clearly captured, resulting in a nice directed acyclic graph when expanded.

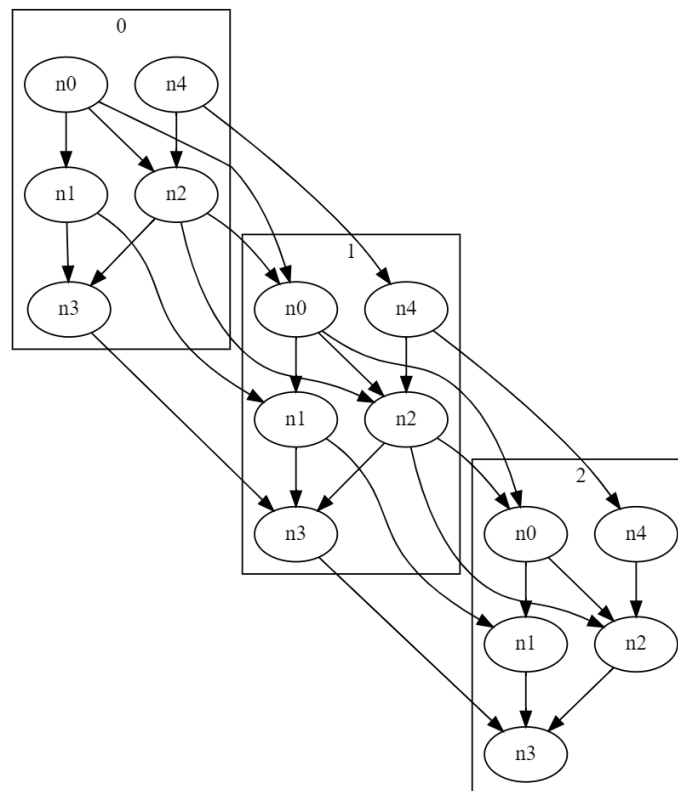


Figure 27. Expanded Example Functional Graph with Feedback

Directed Acyclic Graphs (DAGs) have a useful property in that the nodes can be sorted topologically. This topological sort results in a nice linear sequencing of the nodes where all subsequent nodes are guaranteed to be dependent on the previous set in the sequence. In other words, this order describes how the capacity computation is to be executed for some arbitrary expanded functional graph.

Returning to the first simple example graph, the capacity model was run with the conditions that n1 was at full capacity (1) and n4 had failed (0) and that all other nodes started at full capacity. The results, shown in Figure 28, are predictable in the sense that n0's and n4's capacities are constant throughout the simulation and n2's and n3's capacity gradually decreases. The response rate of each node is adjustable with a hyperparameter and with the goal to eventually establish a library based on properties originating from the full SysML model.

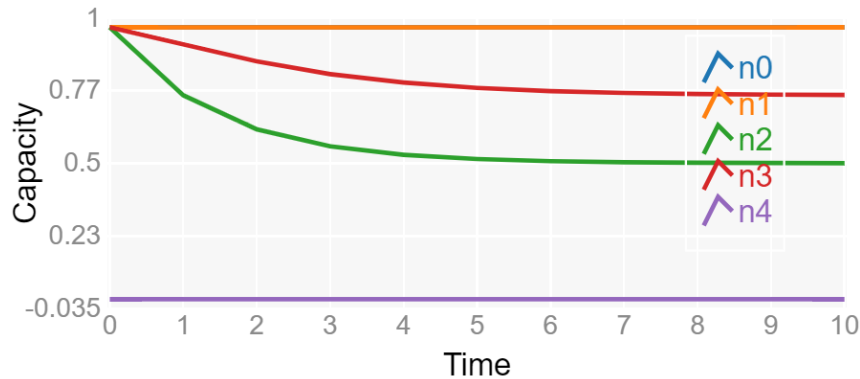


Figure 28. Example Functional Graph (eta=0.5, n1=1, n4=0)

These capacity simulations are not limited to only starting conditions with nodes in a full or failed state. Figure 29Error! Reference source not found., shows the recovery capacity dynamics for the simple function graph model where at time 0 only n0 and n4 are at full capacity and n1, n2, and n3 at zero. Furthermore, n4 is set to fail at time 1, then later return to full capacity. Figure 30 shows this progression via coloration changes on an image of the directed graph. (In the implementation, this progression may be viewed dynamically. This feature is shown in the progression bar at the top of each graph image.)

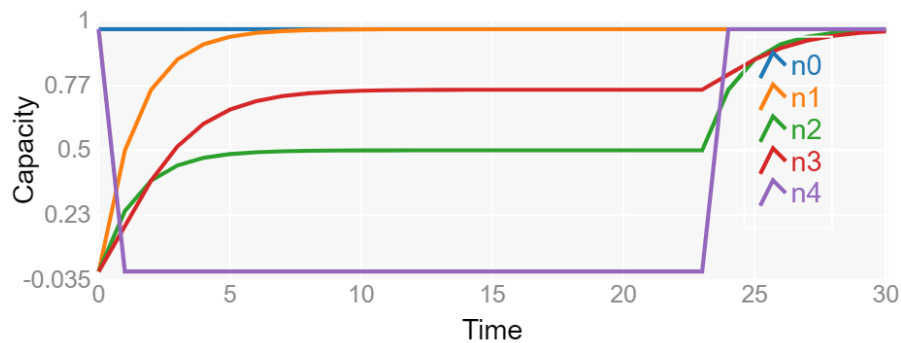


Figure 29. Example Functional Graph Node Capacities overtime (eta=0.5, n1@0=0, n4@1=0, n4@24=1)

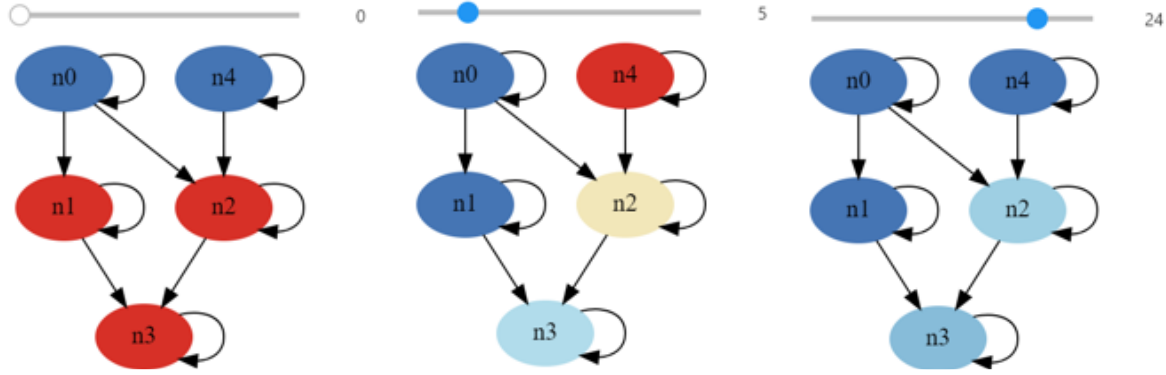


Figure 30. Example Functional Graph Node Capacities overtime ($\eta=0.5$, $n1@0=0$, $n4@1=0$, $n4@24=1$)

These fundamental building blocks of template graphs, feedback loops handled with inter-time slice dependencies, topological sorting, tune-able hyperparameters based on a function library, and time-variant node state manipulation can be used to model a variety of different functional capacity scenarios.

NODE FUNCTIONAL STATE RELATION

There are three primary node states to consider for the CPS security problem: functional state (i.e., is the node functioning and to what degree of its intended capabilities), whether or not a node has been compromised, and whether or not a node has been corrupted.

Node state is therefore a three-state vector, with each state $\langle \text{Functionality}, \text{Compromise}, \text{Corruption} \rangle$ independent of one another. The propagation of changes in these values will expose how functionality is preserved or degraded, potential pathways through which a system is exposed following compromise, and how corrupt functional flow (e.g., corrupted data or control signal) may propagate unwanted behavior through the system.

Functionality

$\langle \text{Functionality} \rangle$ is treated as a state ranging from $[0, 1]$: 0 signifying that a functional element is not functioning, $0 < \text{Functionality} < 1$ signifying some level of degraded functionality, and 1 signifying that it is functioning as intended with full capability intact.

A good functional state relation should have several requirements:

- It should be simple enough to scale in assignment (i.e., easily assigned to functional elements through an auto-attribution lookup as described above).
- It should be efficient in terms of the computational effort required to evaluate the relation.
- It should be flexible enough to be parametrically varied, where that variation reflects different abstractions of behavior possible from different types of functional elements.
 - Capacity/ Buffering – Some functional element types may behave with a built-in level of capacity or buffering so that if an input fails, degradation of that element's functional state is not immediate.

- State Sensitivity – Functional element types will have different levels of sensitivity with respect to inputs. Some elements may fail when one of many inputs fails; other functions may continue to operate, albeit at a degraded level of capability.
- Functional Type Definition – Customization of rules with respect to how different graph elements may preserve or degrade their functional state should be simply codifiable. These rules should be captured in the look-up “library” used of auto-attribution, resulting in efficient parametric variation that, in turn, changes the dynamics of the functional state relation.

Several potential variations for a functional state relation were explored. Exponential relations were identified early in the investigation as the best depiction of functional state that could reflect aspects of capacity and for which the rate of decay (reflecting state sensitivity as well) could easily be controlled through equation parameters.

The functional node states were assumed to be a function of:

1. The given node’s present state,
2. A target node state value derived from the functional states of the parent nodes (flexibly controllable across a range of scenarios), and
3. Shape parameters to control the shape of the exponential curve across time.

The reliance of the next state value on (1) that node’s present state and (2) the values incoming from its parent nodes produces a hybrid Markov/ neural net type of propagation process.

The general form for exponential growth or decay equations expresses a value as a function of time and a rate parameter as shown in $x(t) = a(1 \pm r)^t$

(1):

$$x(t) = a(1 \pm r)^t \quad (1)$$

Where:

x :	Value undergoing exponential growth (+) or decay (-)
a :	Initial value of x
r :	The rate associated with growth (+) or decay (-)
t :	Time in arbitrary units

The general form is unsuitable for the current problem, however, because (a) the state changes at a specified rate but does not aim toward any specified target value, (b) there is no dependency on current state (self-dependency), and (c) there is no dependency on parent node states.

A next iteration used a customized, but general form specific the functional state space dynamics being abstractly modeled, shown in Equation $x_{self}(t) = x_{self}(t - 1) + f[x_i(t) \forall parents i]$

(2):

$$x_{self}(t) = x_{self}(t - 1) + f[x_i(t) \forall parents i] \quad (2)$$

Where:

$x_{self}(t)$: Updated value of functional state of current node, [0, 1]
 $x_{self}(t - 1)$: Initial value of functional state of current node, [0, 1]
 $f[x_i(t) \forall parents i]$: Some function of the value of the functional state for each parent node

Here, each new node state depends on its previous state but in an additive and not multiplicative sense. This prevents a state from staying at a value of 0 for all future times. Several variants of the last term, the function of the parents, were explored. In one, a weighting term based on the number of parents was explored. However, this form could never bring a node's new state to its target value in a single time step if there were multiple parent nodes. Even with other shape parameters, the number of parents dominantly determined the rate of convergence.

The problem needed a better way to control convergence of a functional state value to its target value in a way that makes sense for an analyst and is commensurate with the dynamics of functional elements being abstracted by the relation.

The final formulation developed for Phase 1 proved to be the most flexible, reflective of the exponentially changing state dynamics for degradation or recovery with or without a level of intrinsic capacity, and exhibits convergence rates readily controlled by a single parameter that can be varied according to type of functional element, importance of parents, etc. There are other variations of this relation that, while they were effective, they were not as intuitive and simple to control. Importantly, the relation needed to capture an exponential form of state change, not necessarily be formulated explicitly with an exponential itself. (This is not unlike the general form for this behavior shown in $x(t) = a(1 \pm r)^t$

(1).

The functional state relation used for Phase 1 exploration is given in Equation $x_{self}(t) = x_{self}(t - 1) + \text{sgn}(\Delta) * \eta * |\Delta|$ (3).

$$x_{self}(t) = x_{self}(t - 1) + \text{sgn}(\Delta) * \eta * |\Delta| \quad (3)$$

Where:

$x_{self}(t)$: Updated value of functional state of current node, [0, 1]
 $x_{self}(t - 1)$: Initial value of functional state of current node, [0, 1]
 Δ : The difference in the current node state and the target state, $(x_{self}(t - 1) - x_{target})$
 $\text{sgn}(\Delta)$: The sign, or signum, function of Δ .
 η : Eta, the shaping parameter controlling the rate of change in $x_{self}(t)$
 $|\Delta|$: The absolute value of Δ (also expressible as $\Delta * \text{sgn}(\Delta)$).

Eta (η) is an exponentially sensitive parameter; equally spaced changes in eta do not result in equally altered curve shapes. This notion is illustrated in the functional state degradation and recovery parametric graphs in Figure 31 and Figure 32, respectively. Future work will develop an intuitive mapping for eta as it is used as a shaping parameter such that eta produces the intended, instinctively believed effect when altered from one node type to another by a systems engineer.

Derivation of target state value is discussed in the *State Propagation Patterns* section.

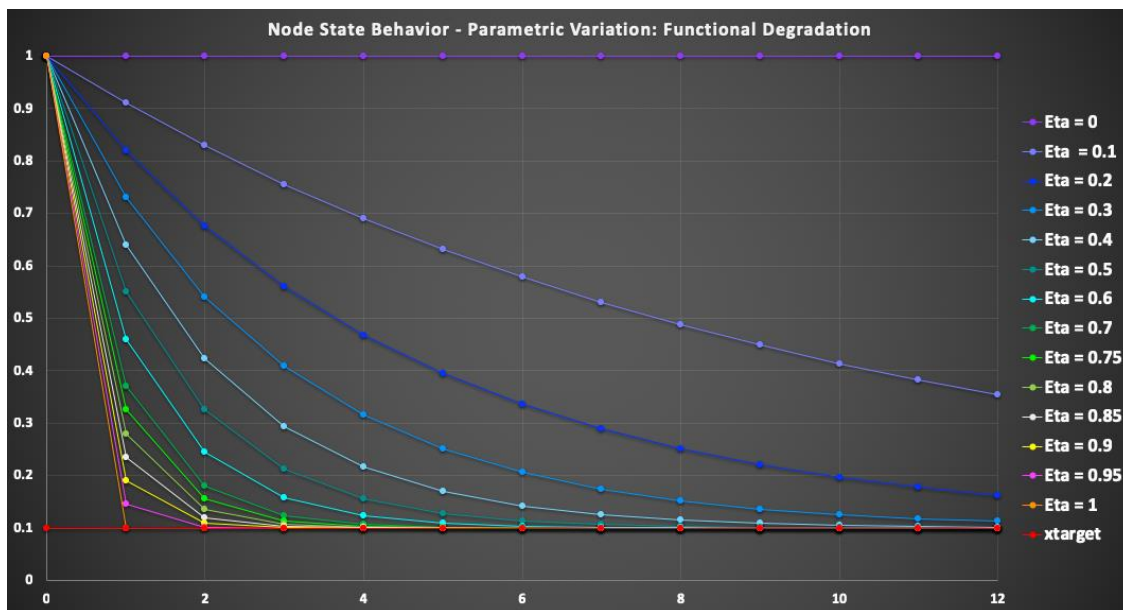


Figure 31. Node Functional State Behavior using Parametric Variation to Capture Functional Degradation

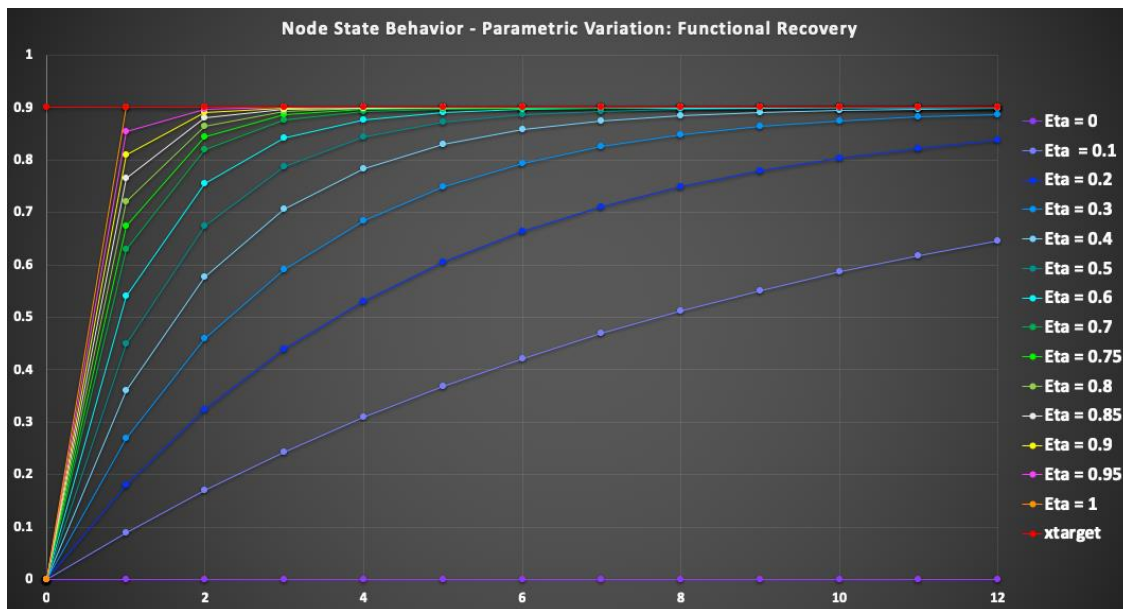


Figure 32. Node Functional State Behavior using Parametric Variation to Capture Functional Recovery

Compromise

<Compromise> is treated as a binary state: 0 signifying that a functional element is not compromised and 1 signifying that it has been compromised.

Children nodes are not assumed to be knowledgeable of a compromised state. A functional node on the graph will only change its <Compromise> state if (a) it is directly attacked by a threat functional pattern possessing a capability to compromise its target or (b) if its parent node is compromised. These likelihoods may be deterministic or probabilistic, depending on the significance attributed to the threat, the type of functional node under attack or receiving compromised input, or whether or not security protection patterns are in place specific to compromise protection in that region of the graph.

Specific implementation of these concepts will be developed as the conceptual development of threat patterns matures. This is a large gap in current understanding and practice and is discussed further in the section on

Task 3: Threat Characterization and Modeling.

Corruption

<Corruption> is treated as a binary state: 0 signifying that outputs from a functional element are not corrupted and 1 signifying that the outputs have been corrupted.

<Corruption> is treated similarly to <Compromise>. Children nodes are not assumed to be knowledgeable of a corrupted state. A functional node on the graph will only change its <Corruption> state if (a) it is directly attacked by a threat functional pattern possessing a capability to corrupt the outputs of its target or (b) if a parent node (and hence input flow) is corrupted. These likelihoods may be deterministic or probabilistic, depending on the significance attributed to the threat, the type of functional node under attack or receiving corrupted input, or whether or not security protection patterns are in place specific to corruption protection in that region of the graph.

As before, specific implementation of these concepts will be developed as the conceptual development of threat patterns matures. This is a large gap in current understanding and practice and is discussed further in the section on

STATE PROPAGATION PATTERNS

There are several possible variations for state propagation:

- Deterministic, Min/Max, Immediate State Change: Set the target state for a given node to the minimum or maximum functional state of its parents, signifying degradation or recovery respectively. Set the relation parameters to fail (or recover) in one time step.
- Deterministic, Min/Max, Exponential State Change: Set the target state for a given node to the minimum or maximum functional state of its parents, signifying degradation or recovery respectively. Set the relation parameters to fail (or recover) exponentially in accordance with the given function type.
- Deterministic, Derived Target, Immediate State Change: Set the target state for a given node to some function of the state(s) of its parent(s). This target may be an average, weighted average, or priority-based (i.e., based on the importance of a parent relative to others) calculation. Set the relation parameters to meet the target value in one time step. This may signify degradation or recovery based on whether the target value is less than or greater than the current value of the node, respectively.
- Deterministic, Derived Target, Exponential State Change: Set the target state for a given node to some function of the state(s) of its parent(s). This target may be an average, weighted average, or priority-based (i.e., based on the importance of a parent relative to others) calculation. Set the relation parameters to meet the target value exponentially in accordance with the given function type. This may signify degradation or recovery based on whether the target value is less than or greater than the current value of the node, respectively.
- Probabilistic State Changes: A probabilistic version of each deterministic case above is possible. This has not yet been developed, however, because a rational basis for when this type of behavior would apply and for what types of nodes in association with what types of threats has not yet been developed. This reiterates the challenge that many foundations of MBSE functional specification for systems with cyber and physical elements have yet to be developed. This notion is addressed further in section *Variation on Approach Using Probabilistic Graphical Models*.

As briefly described above, the target value for a node's functional state may be specified through a flexible, rules-based approach dependent on either the functional type of the node or which level of severity (best case versus worst case) a systems engineer is interested in revealing. One option is to specify the worst (corresponding to $<\>$) or best ($>\>$) value in the set of incoming state vectors from parent nodes. This simulates critical failure; one input fails, the entire receiving function fails. Another option is to derive a target value through some average, weighted, or prioritized parent valuation. An example method to make this scalable and also customizable through library entries is to use a rank exponent weighting method to establish weighting for parent values. The rank exponent method is a generalization of the rank sum approach, and is easily controlled using a single parameter [Roszkowska, 2013]. This would allow easily modifiable changes in parent node prioritization through value weighting. Again, probabilistic rules are also

possible within the framework developed under RT-204. There is just not a rational basis for their selection in the current state of understanding.

It is envisioned that which approach to use will eventually be ascribed to certain functional types based on experience and subject matter expertise and captured in a library for future use by the community.

THREAT ADDITION

Security is concerned with coordinated attacks by an intelligent, adaptive, and rational adversary. CPS may be vulnerable if they are reachable logically, not just physically. Attack tree analysis is the most widely accepted method to assess how an attacker can gain access via the cyber aspect of these assets and subsequently exploit system functions. An attack tree represents adversary perspectives and decomposes a high-level objective (i.e., “compromise the system”) into possible paths through which an attacker could exploit a specific system feature. By themselves, however, attack trees – even with human-supported feasibility assessment and prioritization – do not merge likely threat vectors with the system in question.

Fundamentally, the objective of a cyber-attack is to gain unauthorized access to, modify, or disable a system. CPS attacks are distinguished by a primary focus on regulating (controlling) the state of physical processes. This notion is not normally exploited and has only recently been considered by most security analyses.

Most security efforts for cyber-physical systems have followed an analogous paradigm to that of cyber systems, focusing on protecting the perimeter of supervisory networks controlling the embedded processors. Security measures include typical IT security firewalls, anti-virus packages, data encryption, access control, and user authentication. In a component-based analysis, specific interfaces and hardware components are assessed against the extensive threat databases already in development for known vulnerabilities [Beling, Horowitz, Flemming, et al., 2019].

These threat databases, all maintained by MITRE, include several efforts:

- Structured Threat Information eXpression (STIX™) (<https://stixproject.github.io>) archive, a collaborative community-driven effort to define and develop a language to represent structured threat information
- Common Vulnerabilities and Exposures (CVE®) (<https://cve.mitre.org>), a list of entries for publicly known cybersecurity vulnerabilities with each entry containing an identification number, a description, and at least one public reference
- Common Attack Pattern Enumeration and Classification (CAPEC™) (<https://capec.mitre.org>) dictionary and classification taxonomy of known attacks, envisioned to form a standard mechanism for identifying, collecting, refining, and sharing attack patterns among the cybersecurity community.

These databases, however, focus on threats and vulnerabilities for cyber systems. The emphasis is on the cyber, similar to strictly IT systems. Cyber-physical systems for Defense will have some of the same vulnerabilities if similar cyber components and interfaces are used. Critically,

however, use of CPS in theater environments exposes these systems to new classes of threats. Some of these threats combine cyber concepts of entry and corruption or inserting a control signal into a CPS just as for the strictly cyber systems, yet the means of entry may be quite different or even unforeseen. For example, a new generation of software-defined communication technologies may capture, modify, and send a corrupt signal back to a system with a vulnerable communication interface. This has tremendous implications for the “Mosaic Warfare” concepts being explored at present by the DoD.

This means that while these databases and the component-based view are highly valuable and necessary to security analyses for CPS, they are insufficient to address the full nature of the problem. An approach like that developed in RT-204, which bases an analysis on (a) a functional architecture view, and (b) enables that functional architecture to be attacked by adding threat vectors at any point in the architecture, allows both known and unknown-but-possible threats to be evaluated as part of a model-based analysis. Once an attacker penetrates the supervisory network, the adversary often finds little to no security or validation schemes on the embedded process controller. This allows the attacker to modify the behavior of the embedded processor and consequently the physical process [Lyn, Lerner, et al., 2015]. The RT-204 approach enables exploration of these consequences even when the true penetration vulnerability is not yet known. This, in turn, creates a process whereby systems engineers may “design in” security.

With this understanding, threats may seek to attack either a functional element directly or the flow between functional elements, depending on the flow type and therefore what may be exposed to an adversary. These threats may be described as functional patterns themselves, with differing intent and abilities to achieve that intent. This will be discussed further in section

Task 3: Threat Characterization and Modeling.

To add these threats to the functional graph of the baseline CPS, a threat will be added as a node (or set of nodes) and a corresponding edge to the functional element being attacked (at left in Figure 33) or inserted as a node between two CPS functional elements if the threat is attacking the functional flow between them (at right in Figure 33).

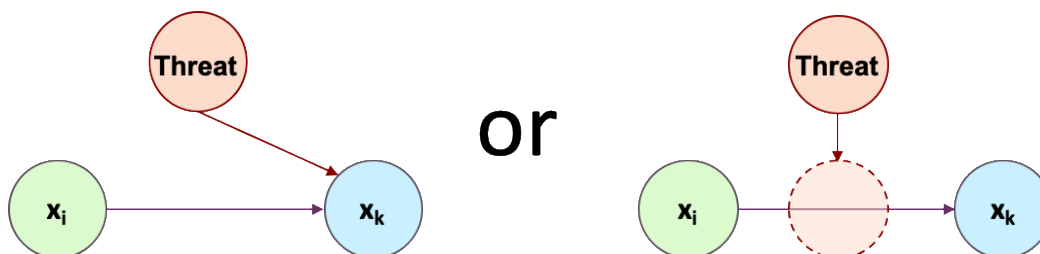


Figure 33. Illustration of Threat Functional Node Addition to an Existing Functional Graph of a Blue System

Future implementation will require timing of simultaneous threats. The means to address that in the simulation is discussed in the *Discussion and Recommendations* section.

VARIATION ON APPROACH USING PROBABILISTIC GRAPHICAL MODELS

Extending the ideas from the previous capacity simulation the template graphs can be used as building blocks for probabilistic graphical models. Probabilistic graphical models are representations used for encoding some complex joint probability distribution. They accomplish this by representing conditional distributions among various random variables. Figure 34, parallels the ideas from Figure 24 but here the x , y , and z nodes represent random variables and the simple node represents the conditional probability of z given x and y .

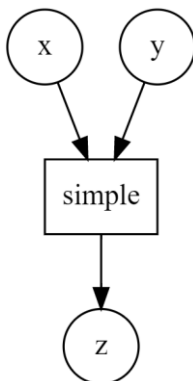


Figure 34. Example Factor Graph

Analogous to the template expansion from above, Dynamic Bayesian Networks are a form of chaining slices of a particular time slice, Figure 35. By chaining the fragment factor graphs together with inter-time-slice variables allows the sampling algorithms to capture the evolution

of the state space. The two example graphs below show how a simple Bayesian Network representation looks for states X, Y, Z and simple function. The dynamic Bayesian Network extends that single time slice example by introducing two more time slices and a temporal dependency of the previous state of X.

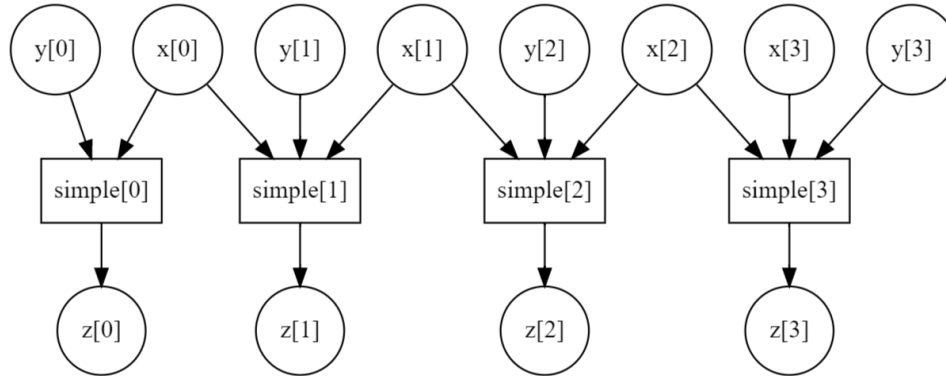


Figure 35. Expanded Factor Graph

This probabilistic modeling approach is computationally expensive but has the potential to capture the temporal evolution of the state space for the activity model. Because of the computational requirements, the implementation needs to be careful to use frameworks know to scale. For this research, the sampling of the model is done with Tensorflow. Tensorflow is an open source deep learning framework from Google that has been scaled to very large data sets and distributed computing power. Tensorflow isn't directly designed to be a tool to model Dynamic Bayesian Networks so the implementation and experimental effort has been focused on creating a flexible representation of the network that can be composed and expanded as necessary. Finally, by having a transformation between this flexible factor graph representation to Tensorflow, the researcher gets an easy to use representation with the ability to scale up sampling power as required.

Next steps are to investigate the scale-ability of this probabilistic approach. These samples can be drawn independently (while respecting the topological sorting) so distributed computing or GPUs could be leveraged to partially mitigate the time to generate a sufficient number of samples. More work needs to be done to understand the kinds of conditional probability functions that reflex the real world functions being represented. For instance a brittle function should be more likely to fail given it's inputs are also failing while a robust function should be more resilient. Finally, more work needs to be done to understand and visualize the resulting state space sampled to make digesting the captured dynamics more tractable.

TASK 2 RESULTS AND DISCUSSION

Complex systems such as CPS ecosystems are not explicitly predictable, even when using first principles approaches. This effort is not trying to produce quantitatively accurate models in terms of expressing voltages, gains, inductances, etc. Rather the aim is to create functional abstractions

of a system, using final node and edge states after graph simulation, that can accurately reveal whether critical system functions are preserved or not in the face of defined threats.

The current framework and implementation supports:

- Failure path propagation, using different state change conditions.
- Failure dynamics, using functional-type parametric control of state dynamics.
- Rules-based, type-determined customization of state dynamics.
- Initial micropatterning for customization of functional state dynamics.
- 3-State vector, using a feed-forward approach: *<Functionality, Compromise, Corruption>* where each state is independent of one another

Graphs offer a powerful approach through which system functional architectures may be captured and understood. As constructs, they do have limitations, however. For example, event-triggered flow and other forms of conditional iteration are not straightforward to represent using graph structures, even when using them to capture a functional level of abstraction. Such characteristics will likely require additional assumptions and/or adjustment to a resultant functional graph structure prior to dynamic evaluation.

For example, some systems contain parallel functional paths not otherwise connected. Fire and intrusion detection, for instance, are typically parallel activities within a security system. Future work needs to develop a foundation for how are parallel functional concepts should be modeled and what this type of behavior means for functional graph discovery and subsequent simulation.

In addition, vertices (nodes) represent functional elements, whether cyber or physical in nature, at the level of decomposition used by the system modeler. There is not yet a well-established foundation for what level of decomposition in the MBSE of functional architectures is most appropriate for cyber-physical systems.

Similarly, a system's behavior may have continuous, discrete, or even event-triggered interactions. These different concepts need to be better understood in terms of simulating system functionality at a level of abstraction that preserves relevance to the notion of security evaluation.

While the node state relation derived and evaluated under Phase 1 accomplished everything intended, the implementation in the framework is flexible such that the relation may be modified (even according to type of functional element, if necessary) as the community understanding of the CPS problem matures. Even so, the aim of this work is an abstract depiction of dynamic behavior answering the question *"Is the functional state space of the CPS system being modeled preserved or not under threat(s) and with designed security protection patterns in place?"* This approach is flexible and will scale. The goal is not to produce a detailed, nearly first-principles model of the system.

At present, state changes are propagated deterministically, dependent on parametric control for rate of change and rules-based derivation of a target state. The framework is extensible to implement a probabilistic propagation of state. However, a rational basis for selecting probability

distribution shapes and for what types of nodes in association with what types of threats has not yet been developed. This reiterates the challenge that many foundations of MBSE functional specification for systems with cyber and physical elements have yet to be developed.

TASK 3: THREAT CHARACTERIZATION AND MODELING

The RT-204 approach also considers the relationships between security, dependability, and trust highlighted in the NIST *Framework for Cyber-Physical Systems* [Griffor et al., 2017]. The NIST framework highlights the concern of resilience in terms of trustworthiness, related to the ability of the CPS to withstand instability, unexpected conditions, and gracefully return to predictable, but possibly degraded, performance. Trustworthy CPS architectures must be based on a detailed understanding of the physical properties and constraints of the system in the context of threats, attributes, and effects. Analysis in support of design activities must include creation and simulation of up-to-date adversary models as well as physical and functional models of the CPS.

CPS ATTACK AND COUNTERMEASURE PATTERNS

Traditionally, threat models are restrictive and do not adequately capture the security of CPS. A survey of vulnerabilities, failures, and attacks on real-time distributed control systems, and the mitigation strategy are divulged by Kisner et al. in a technical report for Oak Ridge National Laboratories. Kisner specifically mentioned that the reality of the characteristics within CPS makes many methods “inadequate, unsatisfactory, or even harmful.” This includes cyber protection methods that are utilized in the IT domain [Kisner, 2010]. Cárdenas et al. stressed that little interaction occurs between control and security practitioners, proposing research challenges within the areas of resiliently [Cárdenas, 2011]. Cárdenas also developed an informal, yet comprehensive threat model and taxonomy of attacks against sensor network in SCADA systems, helping to identify the specific challenges and gaps [Cárdenas A. A., 2009]. The threats of concern to CPS are different than IT systems – they are constrained by the nature of CPS and functionally different. Griffor summarizes the CPS specific concerns as 1) tolerating intrusion and disruption of signal/information flow, and 2) ensuring there is no insertion, fabrication, or replay of legitimate control commands [Griffor, 2016]. Table 1 enumerates a number of CPS attack types. The attacks are categorized according to what is being exploited: software vulnerabilities or flawed network implementations (e.g., leading to cyber-attacks), physical vulnerabilities, or cyber-physical combinations [McDermott and Horowitz, 2017].

Table 1. Taxonomy of CPS Attack Types

Cyber Attacks	Description
Network Denial of Service	Disrupt network operations to stop control signal/flow during the attack period
Malware	Installation of malicious software, or viruses, into the device
Software Vulnerability Exploitation	Exploitation of software vulnerabilities such as buffer overruns, numerical overflows, or backdoor applications; can be defects or intentionally created
Physical Attacks	Description
Sensor Spoofing	Providing false sensor data to the CPS, either by injecting false information into the sensor or into the communication paths
Signal Jamming	Disrupting control/signal flow by preventing or changing the signal, primarily using external signals
Hardware Vulnerability Exploitation	Exploitation of hardware vulnerabilities such as network protocol errors, or side channel attacks on power, cooling, or other flows
Physical Damage	Physically disrupting the CPS to disrupt control signal/flow, such as damaging interconnects or sensing surfaces
Cyber-physical Attacks	Description
Insider Threat	Stealing and/or modifying design or operational data for exploitation; emphasizes the importance of the CPS design environment
Identity Spoofing	Providing signals or information to the CPS that appears to be legitimate, such as false network packets; used in man-in-the-middle attacks
Supply Chain Compromise	Introducing a flawed or vulnerable hardware or software component into a CPS when it is being manufactured or configured
Information Disclosure	Monitoring CPS information to gather information needed for additional attacks or to steal private information
Social Engineering	Deception or influence on the CPS engineer or operator in order to gain information for other attacks, or possible to induce poor safety or security decisions into the design
Replay Attacks	Recording the control signal/flow over a period of time, in order to replace the actual signal/flow with the recorded data to confuse the system.
Control System Instability	Disrupting control/signal flow in order to produce control system instability

Wan, et al. further defined a set of generalized control system attack models of specific concern to CPS shown in Table 2 [Wan, Canedo, and Al Faruque, 2015b].

Table 2. Generalized Control System Attack Models.

Attack Model	Description
Interruption attack	Also called denial-of-service attack, stops the control signal/flow during the attack period
Man-in-the middle attack	Mimics the human attack behavior. When the attack happens, the control signal/flow is changed to a different manipulated signal/flow controlled by the attacker.
Replay attack	Records the control signal/flow over a period of time in a vector, and when the attack starts, it replaces the actual signal/flow with the recorded data to confuse the system.
Control parameter attack	Modifies the vulnerable control parameters of the system to the attacker's defined parameters. This changes the quality of control of the system.
Coordinated attack	Combines two or more basic attack models, for example, combining a man-in-the-middle attack with a control parameter attack

Attack trees and attack-graph-based visualization offer an intuitive, well-understood approach to capture the complexity of coordinated attack models and to view the effects of the attacks and combined countermeasures. Attack graph analysis is the most widely accepted method to assess how an attacker can gain access to cyber assets in a CPS and how that can be used to exploit the system functions. Up-front human-centered workshops capture the definition of cyber assets in terms of system functions as well as the attack graph analysis that explores the dependent functions that can be exploited by the attack. These provide a tool basis to explore these basic patterns in multiple combinations. CPS functional modeling must consider threat parameters as additional functions to be captured in the control system design decomposition process, producing a graph of both the control function and the attack.

Wan, et al. also defined a set of threat countermeasure patterns of specific concern to CPS shown in Table 3 [Wan, Canedo, Al Faruque, 2015b].

Table 3. Generalized Control System Countermeasure Patterns [Wan, Canedo, Al Faruque, 2015b]

Countermeasure	Description	Attack model countered
Isolation	Creates an isolated runtime environment (sandbox) for the critical asset that is resistant against attacks.	Escalation, interruption attacks
Redundancy	Replicates the functionality of the critical asset in order to create multiple paths for high availability and fault tolerance in the case of individual function failures	Attacks that disable individual instances of critical assets and functionality.
Diversification	Produces functionally equivalent variations of binaries running in software critical assets. This is an enhancement of the redundancy countermeasure.	Coordinated attacks, zero-day attacks effective in identical binary copies of the critical assets.
Physically Unclonable Function	Secures the integrity and privacy of the messages in the system using a Physical Unclonable Function (PUF) that is hard to predict and duplicate.	Attacks that hijack the communication channels such as man-in-the-middle attacks.
Obfuscation	Obscures the real meaning of data/signals/flows by making them difficult for an attacker to understand. It can use random sources of noise from the environment of the critical assets to increase the entropy.	Attacks that require knowledge of the inner workings of the system, its functions, and its mission.
Parameter Assurance	Compares input data to a table of values in the system to check for large, unexpected deviations.	Attacks that manipulate data files or messages that are sent to the system.
Data Consistency Checking	Verifies the source of a parameter change.	Attacks that use operator specific data entry.

These also can be captured into a functional model of the system, effectively combining intended control functions, threat functions, and countermeasure functions into a single model using selected patterns. This allows analysis of “reasonable assurance,” which is effectively a cost trade. While the effects of the attack-countermeasure interactions cannot be directly costed due to the lack of details, CPS designers can assess the effectiveness or “residual risk” associated with selected countermeasure patterns as shown in Table 3. For example, the Control Parameter Attack model requires an adversary with moderate knowledge of the system functional design, low attack specific technical ability, and generally low resources, while the countermeasure approach would have a low to medium implementation cost, and low collateral impact to the system. On the other hand, coordinated attack models that would be effective in identical binary copies of the critical assets would have higher attacker specific technical ability and medium to high resources to implement but would also require more intrusive design impacts to the CPS. Attack graph tools already have support for such cost/risk analyses.

The basic countermeasure patterns can be extended to more complex attack strategies by combining patterns. For example, diverse redundancy is a pattern that employs redundant control systems using differing hardware and/or algorithms whose outputs are voted or averaged [Rashid et al., 2017] [Wan, Canedo, Al Faruque, 2015b]. This mitigates the ability for a cyber threat to compromise all redundant elements in the system. The diversity and redundancy can be provided by components and algorithms added to the functional design of the control system. Horowitz and colleagues collected a number of these redundancy strategies as summarized in Table 4 [McDermott and Horowitz, 2017].

Table 4. Specific Countermeasure Patterns for CPS

Countermeasure	Description
Redundancy	Checks for errors by comparing outputs from multiple components at different levels of the system. This can range from logic levels (parity or cyclic codes) to structural component levels to system levels. Checking accuracy and latency of detection is improved by lower level techniques, while higher level techniques may be more cost effective.
Forward Error Recovery	Allows continued system operation in the presence of errors by compensating for the error with correction strategies. This provides redundancy beyond just error detection. It can also be implemented at multiple levels. From cyclic correction codes and lower levels to voting strategies at component or system levels.
Triple Modular Redundancy	Provides forward error recovery with masking redundancy – the system has three identical modules that perform redundant operations. The output of these modules is checked by a voting module, which forwards the output based on majority vote. An error in one of the modules is masked by the majority.
Backward Error Recovery	Tracks known good process states as recovery points. When an error is encountered the process returns to the last known good state. This is effective for temporary errors but can recur continuously (known as livelock) in the presence of permanent errors.
Dynamic Redundancy	Detects errors and reconfigures the system in response. This is often referred to as “hot standby” when the reconfiguration selects a continuously running system and “cold standby” when the standby system has to be started.
Diverse Redundancy	Schemes like triple modular redundancy can use identical or diverse modules. Use of diverse, or non-identical, modules counters errors that may occur in common designs but also complicates the voting design.
Data Consistency Checking	Verifies the source of a parameter change.

The outcome of this process is a “cyber protected system model”. The result is a set of security design patterns described as to (a) their functional capabilities, (b) the cyber assets they require to achieve their functional capabilities, (c) the critical cyber assets and/or functions they will

protect, and potentially if applicable (d) the specific threat functional capabilities and/or threat cyber assets they are designed to detect or counter through direct connective action. These patterns (actually a set of new functions) can be designed into the CPS or a monitor device that tracks the CPS functional behaviors externally. Using a general model-based approach, different security patterns and countermeasure approaches can be implemented as updates to the CPS as threats evolve. There are a number of processes and tools that support this analysis.

A systematic mapping study pulled from thousands of relevant publications, and found that from 2014-2016, the number of primary model-based security engineering practices (tackling the challenges between security by design, abstraction, and automation) increased significantly, and the use of Domain-Specific Languages (DSLs) is comparable to the use of standardized UML modeling. It is interesting that within the study, it demonstrates that most primary studies do not explicitly address specific security concerns, but focus instead on security analyses in general on threats, attacks, or vulnerabilities. There is a severe lack of engineering security solutions for CPS (Nguyena, 2016).

TOOLS TO EVALUATE THREAT AND COUNTERMEASURE PATTERNS

Tools and techniques exist for design and analysis of fault-tolerant systems, and these tools may be adapted for application in the CPS domain. However, traditional tools are insufficient for resilient CPS due to the complexity of these systems and the potential for cyber-attack.

Most failure analysis tools used today represent the process as a set of linear causal models that sequence a series of events over time. Analysis of the system anomaly is traced backwards in time from the system failure event (if it has occurred) or forward in time from the analyzed potential event causes to their effects (in a preventive design process). These analyses are essentially event chains with branching logic [Benner, 1975]. They trace preconditions and effects of errors in the system as a chain of events that progresses sequentially in time, then apply a countermeasure pattern that changes the chain of events to a more desired effect. McEvilley issued a call for convergence of traditional safety/reliability assurance tools with those focused on security as described in Figure 36 [Reed and McEvilley, 2018].



Figure 36. Need for Convergence of Safety/Reliability and Security Assurance.

Existing causal analysis tools are a useful and logical way for the system designer to think about natural and intentional causes of system disruption but ignore non-linear relationships such as feedback and intentional insertion of subversive behaviors. They also tend to ignore long timespan chains of events such as a control attack that is placed in the system long before the subversive action is planned or intended. This leads to great subjectivity in the analysis. For example, a design feature intentionally placed into a counterfeit programmable logic part can exist for years before a separate piece of malicious software is inserted into a CPS by a human attacker who gains control of the systems data through a third vulnerability in a communications protocol. Together, none of this manifest themselves as errors, and the effect might create a new system behavior that takes control of the CPS.

The design of the attack creates an additional set of control and feedback systems in the CPS, which must be countered not by tracing linear chains of events and preventing each, but by creating new design patterns that serve to limit these new control behaviors from being exploited by a determined threat. The primary shortfall of causal event models is that they limit the analysis of countermeasure designs to similar linear cause and effect relationships [Leveson, 2012].

A CPS resilience analysis will convert the concept of a failure in a CPS to one of availability, or continuity of service. A resilience analysis will ask the questions:

- What will disrupt the control operation, what are the vulnerable aspects?
- How will that affect the availability of the control function to other parts of the system or to its human user?
- What other system components is that control function dependent up-on, and to what extent?
- Can those components also be trusted?

- What is the risk (and associated cost) to ensure trust?

In a linear causal analysis, these dependencies are often omitted because they cannot be quantified, or an assumption is made subjectively to omit that dependency for convenience. Although failure analysis remains important, it must be embedded in a larger process that considers the functions that result from the CPS and its external interactions and dependencies. Due to the complexity of the systems involved, the analysis should not attempt to purely identify all interactions and dependencies but should define hierarchies of control over essential CPS functions. Resilience is a process of reducing or eliminating dysfunctional interactions. These are interactions that can disrupt the availability of critical control functions or lead to potentially hazardous states in the controlled process [Leveson, 2012].

The following analysis process is summarized based primarily on methods explored in the SERC System Aware Security research. These processes were evaluated in this phase of the research for applicability to the functional modeling and simulation approach but will be tested in later phases. For further details refer to [Beling, Horowitz, and Fleming, 2019].

MISSION/OPERATIONAL RESILIENCE ANALYSIS

The resilience analysis starts with human-centered workshops to capture the definition of cyber assets in terms of system functions as well as the threat attack analysis that explores the dependent functions that can be exploited by the attack. Besides being a manual technique, the main limitation of many resilience analysis approaches is that they are too high level and the threat parameters are not treated as additional functions during the functional decomposition process that creates the CPS design. Therefore, this does not allow the traditional systems engineering analyses to include functions and requirements that address cyber threat parameters as inherent to the system functional design.

The accepted processes for non-functional requirements derivation are generally human-driven and involve scenario analysis and modeling to derive lower level functions or requirements from higher level architectural models. These are facilitated processes using subject matter experts from the operational context of the CPS, CPS designers, and the cyber experts with knowledge of potential attacks. The facilitator starts at a mission/operational level with questions about lost or changed functionality during operations. This creates shared understanding of mission objectives and operational tasks related to the CPS. The process also creates a set of information that can be used by CPS designers and cyber threat experts to conduct the analysis of attack and countermeasure patterns.

One approach to manually derive these patterns is describing “critical cyber assets” to link functional descriptions and early views of structural components. The MITRE Mission Assured Engineering (MAE) process framework [Goldman, 2010] is often used in the context of defense assured systems and is a starting point for the development of the RT-204 approach. Refer back to Figure 1 for a visual depiction. MITRE defines three different assessment processes which can describe as mission analysis, cyber threat susceptibility assessment, and cyber risk remediation assessment. Established IT guidance links cyber requirements to “critical cyber assets” (crown

jewels). DoD guidance links cyber requirements to critical mission threads. In the CPS design one needs to assess vulnerability of both critical information assets and critical control processes against criticality of the mission segment of interest.

The three assessment processes are:

- Mission analysis or “mission-based critical asset identification”. This process takes critical mission objectives or operational tasks and associates them with the critical system functions, creating an initial linking of functions to types of cyber-physical assets.
- Creation of a functional dependency graph that hierarchically links high level mission objectives to operational tasks to information (or control) assets to sets of system assets as was shown in Figure 37. This is a traditional mission to function to structure decomposition approach, which can be captured in a model. System assets are those structural components that might be exploited by a cyber attacker.
- Creation of an equivalent failure model by tracing back up the graph. The definition model is effectively a causal event chain: “if ‘asset’ is compromised, then ‘function’ is compromised, then ‘mission objective’ is compromised.

Various analysis tools can be then used to assess the resulting functional and asset relationships. Three of particular interest are fault and attack tree analysis, goal-structure notation, and System-Theoretic Process Analysis.

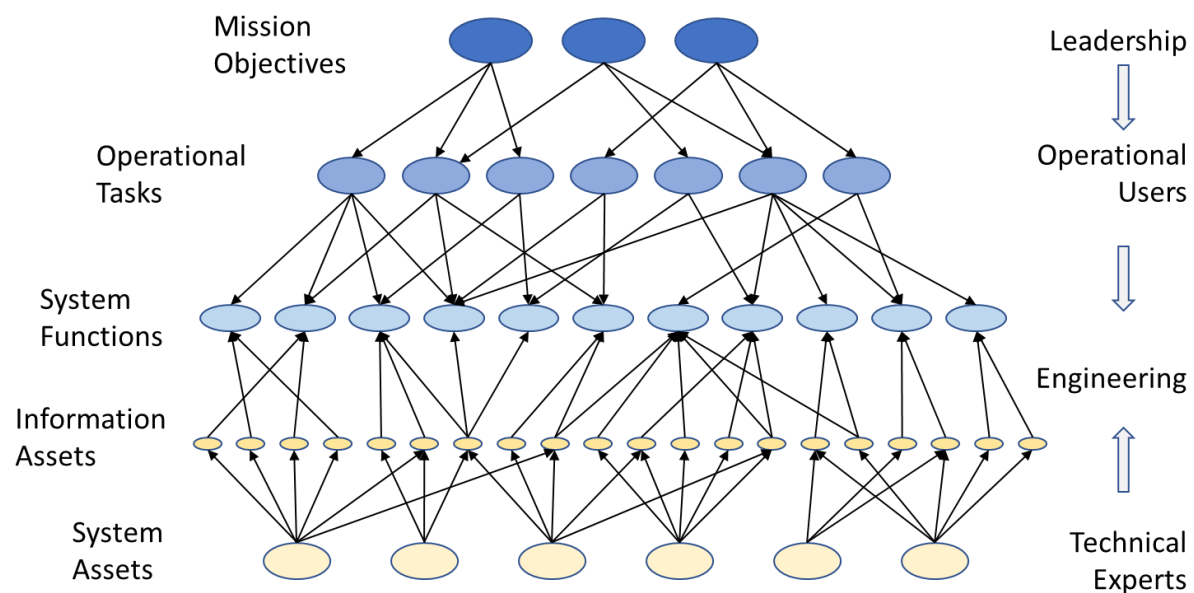


Figure 37. Functional Dependency Graph linking Mission Objectives to System Assets [adapted from Goldman, 2010].

SYSTEM-THEORETIC ACCIDENT MODEL AND PROCESS (STAMP)

To address the shortcomings of linear failure models, Leveson developed the System-Theoretic Accident Model and Process (STAMP), and System Theoretic Process Analysis (STPA). STAMP has been applied to accident analysis and prevention, and also to general dependability and security design in CPS using a security specific form of STPA called STPA-Sec [Young and Leveson, 2013]. In the STAMP framework, understanding system disruptions require the analysts to determine why the control structure was ineffective. STAMP replaces the concept of an event that results from a control failure with the concept of a constraint that enforces appropriate control. STAMP analyzes and imposes an equivalent structure of CPS information and control feedback. This feedback extends hierarchically from a central control structure to include more substantial feedback loops created by larger system dependencies. The potential interactions between dependent systems are changed from assumed trust to evidence of trust. This allows for the system to have adaptive feedback loops that either maintain or fail to maintain system characteristics of dependability and security. [Leveson, 2012]

The design of CPS resilience starts with the identifications of system hazards and vulnerabilities. This is a human process that is naturally limited by the knowledge of the human teams involved, so it should be an iterative process that evolves with the System design and use. The first step in any design for safety program should be the identification of the system hazards. To do this, accidents must be defined for the particular system being developed. An accident need not involve loss of life, but it does result in some loss that is unacceptable to the customers or users. For practical reasons, a small set of high-level hazards should be identified first. Even complex systems usually have fewer than a dozen high-level hazards. Starting with too extensive a list at the beginning, usually caused by including refinements and causes of the high-level hazards in the list, leads to disorganized and incomplete hazard identification and analysis process.

A system is a recursive concept, that is, a system at one level may be viewed as a subsystem of a larger system. Unsafe behavior (hazards) at the system level can be translated into hazardous behaviors at the component or subsystem level. Note, however, that the reverse (bottom-up) process is not possible. Hazards can also be related to the interaction between components such as the interaction between attempts by air traffic control to prevent collisions and the activities of pilots to maintain safe control over the aircraft. Due to the potential vulnerabilities (i.e., threats to a system's intended safe function) stemming from an array of complex interactions and sequences of events, STAMP views accidents – by analogy to the CPS security case, threats – as a control problem. Vulnerabilities may consequently be prevented by enforcing certain constraints on system component behaviors and their interactions. In STAMP, a process model controls the actions to help expose what are deemed unsafe control actions: control commands required for safety are not given, unsafe control commands are given, commands are given too early or too late, or the control action stops too soon or is applied for too long. [Leveson, 2012]

This view implies a modeling approach that is able to capture the functional state space of a CPS and reveal whether that state space has been compromised or preserved in the face of threats and applied protection patterns. The focus on functional modeling is consequently synergistic

with the goals of STAMP. Further, the frameworks and methods discussed in this report may serve as a direct compliment to existing model-based systems engineering processes and tools and, in turn, themselves be executable within a toolset that enables systems engineers to produce, navigate, and understand the complexity and scope of the problem.

STPA-Sec takes the mission objectives and operational tasks developed in the war-room setting and extends them down to dependability and security objectives. The first step is the identification of unacceptable losses in the mission/operational context. This list is then used to derive a set of system hazards and associated system constraints. Take, for example, a simple digital engine control loop in an aircraft. Unacceptable losses would include loss of the aircraft, loss of human life or injury, unacceptable delays in travel, and loss of trust in air travel. Three (of many) specific hazards in an engine controller would include uncontrolled changes in engine thrust, incorrect engagement of engine thrust reversers, and incorrectly reported engine failures. An example system constraint would include a requirement such as “the system shall prevent engagement of thrust reversers while the aircraft is in flight.”

This high-level descriptive model then can be used to create a functional model of the control structure, leading to a set of expected control actions and potentially hazardous control actions. The thrust reverser example is relatively simple. Two control actions likely define the function: engage and disengage reversers. This would be accompanied by several constraints defining functions such as “check for weight on wheels before engaging...” All of these would be typical system functions in a system functional model. The STPA process recommends evaluating the causes and effects associated with not providing the expected control action, providing it in a way that causes a hazard, providing it too soon or too late, and providing it out of sequence. Detailed examples of STPA and STPA-Sec can be further explored in open literature, and the process for implementation is well-described in previous SERC System Aware Security documentation as a formal Cyber Security Requirements Methodology (CSRM) [Beling, Horowitz, and Fleming, 2019].

STPA-Sec extends the analysis to identification of hazardous control actions and related security constraints, such as “thrust reversers shall not engage without direct physical indications of aircraft weight on wheels” or “weight on wheels indicators shall employ diverse redundancy.” This process leads to a set of scenarios that relate hazardous CPS control actions to security-related scenarios (and also dependability related scenarios as a set of control constraints). These can be further explored in the context of threat attacks and associated system loss of control (expressed as errors) using tree or graph models.

FAULT AND ATTACK TREE ANALYSIS

A basic approach used by designers of fault-tolerant systems is fault tree analysis. A fault tree model is a graphical representation of the various parallel and sequential combinations of errors that could reasonably contribute to the occurrence of a top loss or hazard event. Because the fault tree focuses on its top event, the tree only includes faults that contribute to this top event. This fault list is not exhaustive. However, the STPA process helps to create a holistic model of

these events and associated errors. As this chapter is focused on the cyber-attack domain, a detailed description of the system fault analysis is left to other references.

Fault tree analysis has been adapted for application in cyber physical security in the form of attack trees. Like fault trees, they begin with top events but then analyze all steps an attacker might use to cause that event. A number of commercial tools support the development of attack trees or their graphical equivalents, attack graphs. In the attack tree analysis, the attacker's primary goal is the root of the tree. In CPS this may be disruption of a control action but could be other effects such as stealing information. Although this chapter is focused on the CPS control actions, the analysis should cover all aspects of the system, including its development environment. Attack trees are built from the attacker perspective.

Most attack tree tools branch from the root node into attacker subgoals using an AND/OR logic structure where the AND nodes imply the attacker must take multiple actions to accomplish the attack and OR nodes signify one of several alternative actions. As the tree is constructed it follows the structure of Figure 37 where system control functions are decomposed into lower level system functions and information assets, then into system assets which serve as threat entry points.

After the trees are constructed, analysis the tree to the most likely attack strategies. Most attack tree analysis tools support analysis capabilities that aid in calculating cost measures and risk measures.

The attack tree analysis serves to add detail to the higher level STPA analysis so that more detailed models can be built to reflect countermeasure strategies. Actual attacks and countermeasures occur within the system, and information assets in the accomplished design. It is important to gain this level of understanding in the CPS design, but also to understand that the analysis will be limited to the knowledge and experience of the subject matter experts used at the time of the analysis. Also, the resulting attack models are static and do not capture the actual dynamic control actions that are to be protected. For this, a functional modeling language and simulation environment should be used. Thus, the analysis should be used to inform the functional model of the protected system, not as a direct analysis of system assurance.

In the dependability and security analysis, the holistic analysis process of STPA is combined with more detailed fault and attack tree analyses to gain the detailed awareness of how the CPS will remain resilient to both internal failures and external threats. These come together in a functional modeling environment viewing system functions, functional models of threat attacks, and functional models of the countermeasure designs that become part of the overall CPS design.

CYBER SECURITY REQUIREMENTS METHODOLOGY (CSRM)

Under the SERC System Aware Security programs, the research team developed a six-step Cyber Security Requirements Methodology (CSRM) [Beling, Horowitz, and Fleming, 2019]. CSRM involves a Dev/Ops-like collaboration among three collaborating teams (a systems engineering team, a system operational team, and a security focused team). In addition, the methodology

introduces how analysis and rapid prototyping/simulation tools can be used to support decision-making regarding cyber security requirements.

The six-step CSRM is intended to provide an efficient and potentially high-value mechanism to conduct a risk assessment that would lead to the desired architectural design decisions. The individual steps are listed below:

1. High level, tool-based, system description produced by system engineers, including system architecture and functional description (e.g., using SysML).
2. Blue Team consequence analysis, resulting in a prioritized list of system functional problems to be avoided.
3. SE team derivation of resilience solutions (described via use of the system description, e.g. SysML) that respond to Blue Team results.
4. Red Team—based upon experience with cyber-attack threats, COTS and GOTS cyber defense solutions, and use of analytical tools—prioritizes software engineering solutions, cyber defense solutions and resilience solutions.
5. SE team adjusts the system description to account for Red Team recommendations and rapid prototyping/simulation results related to operator requirements, for presentation to Blue Team; Initiates cost analysis effort.
6. Blue Team responds to Red Team recommendations and simulation results with revised consequence prioritization of solutions, thereby enabling SE team to provide an integrated system design discussion for requirements-related decision-makers.

The CRWS process combines the mission/operational resilience analysis with an STPA-based system assurance analysis and formal attack modeling. In future phases of this research, CRWS will be the baseline for the threat evaluation aspects of the program.

LIMITATIONS OF CYBER ASSURANCE REQUIREMENTS ACTIVITIES

Formally linking the behavioral aspects of the system and the behavioral aspects of the threat has been a modeling challenge. In early SERC System Aware Security research, Lockett conceptualized the approach of annotating system functional models with functions that reflect threat and countermeasure activities [Lockett, 2013]. Figure 20, previously discussed, provides an example conceptual annotation of system structural and functional models with threat functions. Although conceptually straightforward, implementing this in formal modeling tools has proven difficult due to the dynamic nature of the threat functions and limitations of existing modeling tools. The primary method from the SERC System Aware Security research for inserting attack behaviors into the SysML functional models has been via annotation of threat activities in the SysML activity diagrams. As these have not been directly simulatable, the research has used

separate Simulink models to capture threat dynamic behaviors. This approach makes it difficult to extract the model information into the functional graph for combined simulation.

With a good understanding of the functional model, cyber-attack patterns, and the cyber assets to be protected, the complete functional representation of these dimensions would ideally be transformed into the directed graph for further analysis. This will most likely be a model transformation process that applies predefined protection patterns – in the form of graph rewriting rules – to the attacked functional model of the system. The morphological approach can be thought of a systematic “patching” of cyber security holes in the system’s conceptual design. This will allow system engineers to “maximize attack space coverage” while “minimizing number of requirements.” It is in this transformation process that RT-204 researchers believe libraries of attack and countermeasure patterns, or metamodels, will become essential.

There are three specific challenges with respect to model transformation:

1. How to effectively connect a functional model of a given threat vector to the original system functional model representation an engineer starts with?
2. How to effectively connect the cyber security functional model associated with a protection pattern with the system-threat intermediate functional representation?
3. How to connect the functional representations across the full functional representation of attack-countermeasure interactions in a way that produces meaningful consequences within the model when executed upon – did the protection pattern implementation work or not?

A key aspect of (1) and (2) above will be to automatically extract semantic descriptions from each entity type (cyber system functional component or asset, protection pattern functional capability or asset, threat functional capability) that will be included in the graphical model with respect to entity class, relationships with other entities of any class, and type of relationship (i.e., logical flow, information flow, causal dependency). With these characteristics derived from the aforementioned functional model extracted from the system description, attack graphs (including temporal), and a library of cyber defense functional patterns, this will serve as the basis for a repeatable, directly executable model modification approach. A semantic mapping of security design patterns can be extracted with respect to (a) their functional capabilities, (b) the cyber assets they require to achieve their functional capabilities, (c) the critical cyber assets and/or functions they will protect, and potentially, if applicable, (d) the specific threat functional capabilities and/or threat cyber assets they are designed to detect or counter through direct connective action. A conceptual approach to accomplishing this was developed in this phase of the research and documented in [McDermott et al., 2019]. However, implementation will be explored in the next phase of research.

This process has the potential to define a protection strategy for any class of threat, at any level of the system. The selection of a protection function, or cyber protection pattern, will in turn determine the requirements. The classes of threat covered will be based on both the

understanding of the threat Tactics, Techniques, and Procedures (TTPs) and the availability and cost function associated with various design patterns. “Protect against many classes of threats” would be a requirement evaluated through multi-attribute tradespace analyses methods.

In addition to the purely static analysis, it is possible to achieve greater accuracy by performing an analysis of the system’s dynamic behavior. Relating to the notion of security for CPS, the dynamic problem is to determine if the functional state space of the original CPS system is preserved when attack patterns and protection patterns have combined to create a new, expanded functional structure. This requires node and edge functional/property motifs that make the ecosystem graph executable and meaningful with respect to evaluating the preservation of the functional state space. These motifs should be detailed enough to produce the needed overall system behavior but generalizable enough by type to be reusable and allow us to produce similarly executable graphs at much greater scale. Unlike the random or statistically generated graphs studies in other fields, this research has identified techniques best suited and modifiable for simulating the impacts of highly correlated threats and protection implementations on functional capabilities of a system. Further research may develop ways to specify parameters in the graphical model that, when executed via discrete simulation, produce “consequences” most relevant and meaningful to the CPS security problem.

TASK 3 RESULTS AND DISCUSSION

The process of evaluating cyber threat impact on the system and associated countermeasures is maturing over time, in large part because of the SERC System Aware Security work. However, much of the existing process is still manual and dependent on the experience of the red (threat) and blue (design) teams assessing the system and current threats. Also, because the threat changes in unpredictable ways, the analysis will always be incomplete. A move from threat reactive approaches to designs that emphasize resilience to any disruptions is necessary, which will drive the SE process toward functional libraries and metamodels. As discussed, the use of functional libraries will also support the capture of threat and countermeasure models into the functional graphs. This will be a focus in the next phase.

DISCUSSION AND RECOMMENDATIONS

A primary discovery revealed through this Phase 1 effort is the need to look more deeply into how to model system functionality for the purpose of developing simulations of cyber-physical systems.

Several fundamental questions vital to modeling CPS in a formal MBSE sense as well as make these models amenable to dynamic analysis for the purpose of security design evaluation must be addressed in the next phase of this effort:

- What is an appropriate level of functional decomposition for a system with both cyber and physical functionality?
- How should a functional architecture be defined in the form of an activity diagram that brings cyber and physical function types together in a meaningful and representative way?
- What best practices with respect to MBSE specification of activity diagrams need to be developed in order to formally model these systems? For example, are concepts such as merge, join, and fork necessary, or should these concepts be modeled differently?

Given that these answers are not yet clear, future work under RT-204 should be matured from this focal point to better position the research to answer the bidirectional problem of formal MBSE specification and simulation of the specified functional architecture.

This begins with defining what elements of the functional architecture (actions, pins, object flow, etc.) are needed for the MBSE of a functional architecture and the simulation. The set of elements for the simulation is expected to be a subset of the MBSE architecture. To do this, the development of MBSE specifically for CPS must be considered, with particular focus on security and dependability aspects to provide the ability to determine if functional behaviors are preserved. Based on the understanding of critical element types in the functional architecture, the simulation abstraction should be matured to handle these types of differences from standard graph dynamics.

For example, this may include executing function state changes in a hybrid framework with discrete-event simulation so that timing can be adjusted and calculated for propagation of functional flow. Scaling the ability to time state changes or activity of parent nodes (e.g., when threat functions attack various CPS functional elements) requires an extension of the current framework. Prior, internally funded work in GTRI developed a hybrid approach combining the finite state changes and associated propagation on directed graph structures with a discrete event simulation. This allows functional events of certain types to take longer to execute or produce an action than others. It would also allow coordinated attacks to be modeled, and functional execution to be ordered.

The goal is to produce reusable system models and specify the associated function libraries, component libraries, and standards necessary to support the semi-automated process. As part of the maturation of this work task, a pilot library for cyber-physical systems, their potential threats, and associated security protection patterns should be specified. This would enable

systems engineers to partially automate the assembly of a new meta-model from existing component libraries containing key CPS components and/or associated functions, common threat vectors and associated attack libraries, and libraries of security design patterns. The graph properties can then be auto-attributed to functional elements in the graph according to their type and associated attributes in the library. This may include rules for how sensitive a functional type is to a failed input (i.e., functional state = 0) and other factors.

CSV files can be used to convert between MBSE architectures and the simulation. When creating or updating an MBSE functional architecture, CSV files for each element type and its relationships can be imported into the systems modeling software, and after all elements are in the model, diagrams can be automatically generated from the imported content. When exporting from an MBSE functional architecture for the purpose of importing elements into a simulation, CSV files can be created with each relevant element type.

The goal of automatically extracting semantic descriptions from functional architectures in an MBSE tool using the XML parsing approach described in the *Task 1: MBSE to Discovery and Labeling of a Functional Graph* section, may still be the most scalable and consistent method for eventual use. However, accurate discovery of what constitutes functions and flows relevant to this type of analysis within a query-able graph data model depends significantly on how relationships are expressed. If the foundations for functional MBSE of cyber-physical systems are established first (i.e., what types of elements, at what level of decomposition, using what consistent ontology, with which types of MBSE implementations such as join, decision, merge, etc.), then many of the challenges encountered in Task 1 will be obviated.

Importing a functional architecture is completely dependent on what kind of content someone wants to include. An activity diagram or set of activity diagrams can constitute a functional architecture, in which case completing the activity diagram import is exactly the same as generating a functional architecture. Other functional architectures are more state-machine based, in which case an entirely different set of elements and relationships is needed. Some consider a functional architecture to simply be a functional hierarchy, in which case the functions themselves and the composition relationships are all that is required. “Functional architecture” is very open to interpretation and, depending on that interpretation, may contain entirely different information.

Future work will still need to mature the threat characterization and modeling space. This is a monumental gap in current understanding and practice. Specifically, how can various threat types be best expressed as functional patterns themselves? Research is still needed for a consistent, repeatable way to extract relationships between threat vectors and functional assets common to cyber-physical systems. Even with humans in the loop of the process of identifying attack vectors, prioritizing attacks, and selecting defensive protection patterns to implement, the constructs with which a system is represented and the corresponding analytical methods must be executable in a computational environment to produce the efficiencies at a scale that are necessary for this problem.

Eventually, the associated inputs and outputs of “designing in” security revealed through a maturation of the RT-204 methods will need to be synergized with a decision tool or process. For any given CPS, multiple security design patterns may be selected for implementation, altering the original system architecture minimally or substantially as discussed throughout this report. This cost is reflected in implementation cost, resource cost (i.e., the cost of any additional resources needed by the system for that implementation), and a more abstract cost reflecting the set of collateral impacts to the rest of the system. Whereas Information Technology systems have few functional-structural constraints, cyber-physical systems are likely to be more constrained with a more tenable set of protections. It will be imperative that threats and security implementations are prioritized through a decision tool where trades on the effectiveness, ease, and “cost” parameters associated with each may be evaluated as well. This will narrow down the system-specific threat and security implementation spaces for further analysis.

CONCLUSION

RT-204 established that a functional model extracted from a formal system description, augmented with attack graphs, and a library of protective functional patterns, may provide an effective path toward earlier-stage design and analysis of security for cyber-physical systems. Further, the general approach developed in this work may serve as the basis for a repeatable, yet flexible approach. It is abstract enough to scale with increased model size, especially if the notion of a function library for CPS is established for community maturation. Dynamic simulation of an ecosystem view – comprised of the original (unprotected) cyber system, the threat functional capabilities and attack vectors revealing the critical cyber assets they will target – can provide insight with respect to the health of the functional state space at a level of abstraction that should prove meaningful for design.

By leveraging patterns at the knowledge level, modelers can better understand how systems interact, and analysts can execute and analyze tradespace explorations in more effective, reproducible, and reusable ways. Through the efforts of RT-204, the power of a truly query-able system model has been described, which opens new opportunities for systems modeling and analysis. RT-204 also revealed key gaps in current knowledge that, when addressed, will move the entire domain of design and security evaluation for cyber-physical systems forward in a meaningful way as part of the Acquisitions process.

The prevailing wisdom in MBSE is that a virtual model of a system, expressed formally, constitutes a single point of truth. From this singular reference source, engineers may discover all system concepts, requirements, verification and validation, and all associated data. In actuality, there are multiple references sources at different levels of abstraction, different SME foundations, and different views that build even that “truth” of a system.

Current research on cyber-physical systems is siloed into subdisciplines such as the sensors, actuators, communication and data transfer, control theory, decision systems including embedded decision software, software engineering, etc. Each view highlights certain features and disregards others to make analyses tractable. Typically, a particular formalism represents either the cyber or the physical parts of the system well but not both.

Due to the increasing complexity created by the evolution of cyber-physical systems for Defense, new MBSE foundations, theories, and tools are needed to design, analyze, and verify these systems at various levels of abstraction. This includes the system and software architecture levels, subject to constraints from other levels, and both component-based and functional-based system views. There is no single solution.

The framework and foundations developed in Phase 1 of RT-204 are extensible to future maturation and use with different “truths”, or views that comprise the MBSE process and practice. Together, they provide a path forward for simultaneous exploitation of MBSE, Digital Engineering, and Model-Based Design. Specifying an activity diagram view – a directed graph model of a functional architecture depicting functional elements and the resource, logical, or

causal flow between them – as done in this work has tremendous implications for future practice. This approach may lead to a definition of best practices for transforming functional architectures into true, active analytical tools and not simply reference design templates.

The framework and process foundations developed under Phase 1 of RT-204 may also support MBSA as the work matures. The same functional model building activity developed by the previous Phases can also be used to build a test framework, essentially a separate functional “harness” that can be attached to portions of the functional model. Here, attacks are replaced by actual test functions (fuzzers) that evaluate vulnerabilities of the system. Threat patterns can be evaluated by test inputs that actually propagate through the system, and the test framework can simulate multiple attack inputs. This can offer a capability unmatched with the current system view, in which methods typically test only one software component at a time. While this specific task will not sufficiently address the inherent complexity of the system, it may provide a foundation from which the community can gradually build libraries of more complex system tests. The key is to create “safe” designs, not simply respond to known threats.

Future cyber-physical systems will require hardware and software components that are highly dependable and reconfigurable. Trustworthiness will encompass preserving functionality when under attack and is consequently inseparable from security. Innovative approaches to abstraction and architectures may help unify the currently disparate subdisciplines within CPS research and development. In turn, this may enable more rapid and effective, modular design of these systems, create opportunities for different types of supportive analyses empowered by standardized abstraction methods, and even potentially spark innovative new design solutions.

PROJECT TIMELINE & TRANSITION PLAN

This project was conducted as Phase 1 of an envisioned multi-year research program aiming goal to produce systems engineering methods, processes, and tools that enable co-development of system models and threat attacker models to inform requirements and design of complex engineered systems. This initial phase laid the foundations necessary to evaluate the feasibility of that ultimate goal as well as refine the path necessary to reach the goal.

Phase 0 of this effort (RT-180) was completed during the incubation project period and lasted 2 months from when funding was received until this report was produced. Phase 1, Phase 2, and Phase 3 are projected to last 12 months each.

Specifically, the work performed under this task took the first steps toward developing and gradually maturing a capability to define and analyze security threats and counter-threat design patterns for cyber-physical systems. Prior work conducted in the pilot investigation of RT-180 envisioned a multi-year progression of research and associated development to support analysis of security protection strategies for cyber-physical systems through the abstraction and subsequent simulation of dynamic processes on functional graph models. The approach, refined by the work in RT-204, aims to enable the greater community to rationally compare and select security implementations (a) in the early stages of design, ‘designing in security’, and (b) is also applicable to already designed systems for which a security solution is needed.

Phase 1 was intended to produce a minimal viable demonstration of the proposed approach and a comprehensive description or underlying conceptual advances for the SERC and DoD sponsors. This was accomplished.

Phase 2 is envisioned to produce and validate stronger theoretical foundations, prototype processes, and tools capable of being demonstrated on realistic problems and integrated into existing complimentary MBSE and attack modeling processes.

Phase 3 is envisioned as a transition step. After Phase 3, significant validation of both theory and tools in pilot applications at realistic scales and associated complexity is expected. By the end of Phase 3, the work aims to achieve maturity sufficient to extend and transition into support for CPS assurance.

Throughout the entire effort, progress will continue to be measured by annual technical review assessments with key stakeholders across the SERC, DASD(SE), and complimentary RT stakeholders. The findings and results as the work matures will be vetted with the stakeholder community for any necessary course-alteration and transition planning.

APPENDIX A: LIST OF PUBLICATIONS RESULTED

McDermott T.A., Canedo A., Clifford M.M., Quirós G., Sitterle V.B. (2019) System Assurance in the Design of Resilient Cyber-Physical Systems. In: Al Faruque M., Canedo A. (eds) *Design Automation of Cyber-Physical Systems* (pp. 143-178). Springer, Cham.

Sitterle V.B. (2018) Systemic Security and the Role of Hierarchical Design in Cyber-Physical Systems, presentation at the 10th Annual SERC Sponsor Research Review, Washington DC, Nov. 2018. https://sercuarc.org/wp-content/uploads/2018/11/2_Track3_SSRR_2018_Sitterle.pdf.

(Submitted) McDermott T. (2019) "A Rigorous System Engineering Process for Resilient Cyber-Physical Systems Design," 5th IEEE International Symposium on Systems Engineering, Edinburgh UK, Oct. 2019.

APPENDIX B: CITED AND RELATED REFERENCES

- Al Faruque M.A. and Canedo A., eds. (2019) *Design Automation of Cyber-Physical Systems*. Springer, Cham.
- Barrat, A., Barthelemy, M., & Vespignani, A. (2008). *Dynamical processes on complex networks*. Cambridge University Press.
- Beling, P., Horowitz, B., Fleming, C., et al. (February 2019). Model-based Engineering for Functional Risk Assessment and Design of Cyber Resilient Systems. SERC report SERC-2017-TR-002.
- Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., & Hwang, D. U. (2006). Complex networks: Structure and dynamics. *Physics reports*, 424(4), 175-308.
- Boehm, B, and Kukreja, N (2015). An Initial Ontology for System Qualities. In INCOSE International Symposium, vol. 25, no. 1, pp. 341-356.
- Burmestera, M. M. (2012). Modeling security in cyber–physical systems. *International Journal of Critical Infrastructure Protection*, 118-126.
- Cárdenas, A. A. (2009). Secure Control: Towards Survivable Cyber-Physical Systems. The 28th International Conference on Distributed Computing Systems Workshops (pp. 495-500). IEEE.
- Cárdenas, A. A. (2011). Attacks Against Process Control Systems: Risk Assessment, Detection, and Response. ASIACCS '11, 355-366.
- Friedenthal, S., Moore, A., & Steiner, R. (2015). *A practical guide to SysML: the systems modeling language, Third edition*. Morgan Kaufmann. ISBN 978-0-12-800202-5.
- Geard, N. (2010). *Adaptive Networks: Theory, Models and Applications*. T. Gross and H. Sayama (Eds.), Springer-Verlag.
- Griffor E (ed) (2016) Handbook of System Safety and Security: Cyber Risk and Risk Management, Cyber Security, Adversary Modeling, Threat Analysis, Business of Safety, Functional Safety, Software Systems, and Cyber Physical Systems. Syngress.
- Griffor, E. R., Greer, C., Wollman, D. A., & Burns, M. J. (2017). *Framework for Cyber-Physical Systems: Volume 1, Overview* (No. Special Publication (NIST SP)-1500-201).
- Hirtz, J, et al. (2001) Evolving a functional basis for engineering design. Pro-ceedings of the ASME Design Engineering Technical Conference: DETC2001, Pittsburgh, PA.

Hirtz, J, Stone, R, McAdams, D, et al. (2002) A functional basis for engineering design: Reconciling and evolving previous efforts. *Res. Eng. Design* (2002) 13: 65. doi:10.1007/s00163-001-0008-3.

INCOSE (2007). *Systems Engineering Vision 2020*. INCOSE-TP-2004-004-02 September, 2007.

Jones, R. A., & Horowitz, B. (2012). A System-Aware Cyber Security architecture. *Systems Engineering*, 15(2), 225-240.

Kalinina, M. L. (2018). Detection of Threats in Cyberphysical Systems Based on Deep Learning Methods Using Multidimensional Time Series. *Automatic Control and Computer Sciences*, 52(8), 912-917.

Kisner, R. e. (2010). *Cybersecurity through Real-Time Distributed Control Systems*. Oak Ridge National Library. Springfield: U.S. Department of Energy (DOE).

Leveson N (2012) *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press. 13.

Li, L. (2007). *Topologies of complex networks: functions and structures*. California Institute of Technology.

Luckett, B. (2013). *Integration of Graphical Modeling Techniques as a Structural Framework for System-Aware Cyber Security Architecture Selection*. Thesis from <http://libra.virginia.edu/catalog/libra-oa:3720>.

Lyn, K. G., Lerner, L. W., McCarty, C. J., & Patterson, C. D. (October 2015). The trustworthy autonomic interface guardian architecture for cyber-physical systems. In 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (pp. 1803-1810). IEEE.

NDIA Systems Engineering Division, M&S Committee (10 Feb 2011). Final Report, Model-Based Engineering Subcommittee. <http://www.ndia.org/-/media/sites/ndia/meetings-and-events/divisions/systems-engineering/modeling-and-simulation/reports/model-based-engineering.ashx>.

Newman, M., Barabasi, A. L., & Watts, D. J. (2011). *The structure and dynamics of networks*. Princeton University Press.

No Magic. (2014). Cameo Systems Modeler. <http://www.nomagic.com/products/cameo-systems-modeler.html>. Accessed 31 May 2019.

Pasqualetti, F. D. (2013, November). Attack Detection and Identification in Cyber-Physical Systems. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 58(11), 2715-2729.

Rashid N, Wan J, Quiros G, Canedo A, Al Faruque M (2017) Modeling and Simulation of Cyberattacks for Resilient Cyber-Physical Systems. 13th IEEE Conference on Automation Science and Engineering (CASE) 2017:988-993.

Reed and McEvilly, "Leveraging System Safety Improve System Security," 21st Annual National Defense Industries Association (NDIA) Systems and Mission Engineering Conference, October 25, 2018.

Rezaei, S., & Liu, X. (2019). A Target-Agnostic Attack on Deep Models: Exploiting Security Vulnerabilities of Transfer Learning. *arXiv preprint arXiv:1904.04334*.

Roszkowska, E. (2013). Rank ordering criteria weighting methods—a comparative overview. *Optimum Studia Ekonomiczne* 5 (65): 14-33.

Sitterle, V.B. and McDermott, T.A. (2017). Systemic security and the role of heterarchical design in cyber-physical systems. SERC report SERC-2017-TR-180.

Wan, J, Canedo, A, Al Faruque, M (2014) "Functional Model-based Design Methodology for Automotive Cyber-Physical Systems", IEEE Systems Journal (IEEE ISJ'14).

Wan, J, Canedo, A, Al Faruque, M (2015a) "Cyber-Physical Co-Design at the Functional-Level for Multi-Domain Automotive Systems", IEEE Systems Journal (IEEE ISJ'15) 11(4): 2949-2959.

Wan J, Canedo A, Al Faruque M (2015b) Security-Aware Functional Modeling of Cyber-Physical Systems. 2015 IEEE 20th International Conference on Emerging Technology & Factory Automation (ETFA) 2015:1-4. W3C (2008). SPARQL Query Language for RDF. <https://www.w3.org/TR/rdf-sparql-query>. Accessed 20 March 2019.

W3C (2018). JSON-LD 1.1: A JSON-based Serialization for Linked Data. <https://json-ld.org/spec/latest/json-ld/>. Accessed 2 May 2019.

Young W, and Leveson N (2013), Systems thinking for safety and security. Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC '13). ACM, New York, NY, USA, 1-8.

Zhu, Q. B. (2015, February). Game-Theoretic Methods for Robustness, Security, and Resilience of Cyberphysical Control Systems. *IEEE Control Systems Magazine*, 46-65.