Software and Systems Modeling
https://doi.org/10.1007/s10270-021-00892-z

REGULAR PAPER

An ontological metamodel for cyber-physical system safety, security, and resilience coengineering

Georgios Bakirtzis[1] · Tim Sherburne[1] · Stephen Adams[1] · Barry M. Horowitz[1] · Peter A. Beling[2] · Cody H. Fleming[1]

Received: 3 June 2020 / Revised: 22 April 2021 / Accepted: 10 May 2021

Abstract
Cyber-physical systems are complex systems that require the integration of diverse software, firmware, and hardware to be practical and useful. This increased complexity is impacting the management of models necessary for designing cyber-physical systems that are able to take into account a number of "-ilities", such that they are safe and secure and ultimately resilient to disruption of service. We propose an ontological metamodel for system design that augments an already existing industry metamodel to capture the relationships between various model elements (requirements, interfaces, physical, and functional) and safety, security, and resilient considerations. Employing this metamodel leads to more cohesive and structured modeling efforts with an overall increase in scalability, usability, and unification of already existing models. In turn, this leads to a mission-oriented perspective in designing security defenses and resilience mechanisms to combat undesirable behaviors. We illustrate this metamodel in an open-source GraphQL implementation, which can interface with a number of modeling languages. We support our proposed metamodel with a detailed demonstration using an oil and gas pipeline model.

1 Introduction

Cyber-physical systems (CPS) integrate diverse software, firmware, and hardware to control the operation of the physical components of the system based on analysis of sensor data. These largely distributed systems have already sparked innovation in a number of sectors, such as agriculture, power distribution, search and rescue, and weaponry. In the face of rapid innovation, it is important to understand and address the new challenges that arise from the deployment of CPS in environments where they are subject to cyber attacks. In particular, a characteristic of CPS is that when they fail, either because of an intrinsic fault or a cybersecurity violation, they can transition to a hazardous state and, therefore, can cause accidents or otherwise unacceptable losses. In the domain of CPS, therefore, cybersecurity and safety are intertwined [62]. However, safety and cybersecurity are often viewed disjointly in a way that considers cybersecurity mitigations without regard to safety losses [9]. The classical approach to security involves verification of components and the construction of security barriers. The most effective approaches to safety, by contrast, center on modeling losses in terms of control actions with respect to states that may reflect the condition of many components [50]. We aim to combine elements from the security and safety paradigms to develop models to support the design of resilience solutions that can mitigate against unacceptable losses. Resilience is a proactive notion where in response to anomalies the system has a way to maintain operation, even in some degraded form [38]. The application of resilience concepts to safety and security design has the potential to transform our safety considerations and security defenses from a prescriptive, perimeter-based approach to one that uses adaptable and dynamic design patterns that take into account the intended functions of the system. Such a transformation can be achieved only if we are able to unify safety, security, and resilience models.

The intrinsic complexity of CPS creates difficulties regarding how to best account for and model the intertwined nature of safety, security, and resilience [47]. Modeling safety requires thinking in terms of misbehaviors, unacceptable losses, and hazardous states. Modeling cybersecurity requires thinking in terms of confidentiality, integrity, and

Communicated by Perry Alexander.

✉ Georgios Bakirtzis
bakirtzis@virginia.edu

[1] University of Virginia, Charlottesville, United States

[2] Virginia Tech, Blacksburg, VA, United States

Published online: 01 June 2021

availability, while modeling physical security requires thinking about theft, disasters, vandalism. Modeling resilience requires thinking in terms of redundant architectures, recovery from a death state, and restricting the expected service of the system to disallow possible violations. Safety, security, and resilience have significant overlap in CPS applications. For example, both a security violation and a safety related component failure could cause an unsafe operator control action, resulting in a system response that reduces the dependability of the overall system. While this example of CPS behavior clearly shows that security and safety are coupled, the corresponding model representations are often decoupled and disparate. To combat this disjoint treatment of safety, security, and resilience, modeling efforts must be structured in terms that not only address all three qualities but also are able to relate their results to each other.

Structured modeling efforts are often captured in a metamodel, which defines the *types* and *relationships* of information allowed in a model of a system. Different metamodels apply to different applications, for example, a metamodel for geologic maps is not going to be the same as the metamodel for a software application. A strict metamodel would enforce the types and relationships, but often metamodels are used to provide a structured guidance for model creation. If automated, such a metamodel can be used to assist systems designers the same way that a sophisticated text editor does for programmers. In practice, this capability is still lacking in modeling tools, predominantly because metamodels are not developed for a particular application and, therefore, cannot enforce strict rules to users that have a diverse set of metamodel requirements. The ontological study of metamodels augments them with a refinement of types. Ontologies define what entities might exist in a particular domain application, how they might be grouped in categories, and how they relate but also can be decomposed within a hierarchy of types. The metamodel, then, is a representation of this ontology, often represented in a set of graphs.

Ontologies and metamodels can be particularly useful in the design of safety-critical CPS. For any design solution, the multitude of diverse but equally important system representations requires management of their corresponding models and relationships among them. This is challenging when attempting to manage some of the most concerning system quality attributes, such as safety and security, as early as possible in the systems lifecycle. Usage of models for this purpose can be approached in a number of ways to transition from assuring safety, security, and resilience by trial and error to addressing them early through the use of models. While other "-ilities" [74], for example dependability, survivability, and reliability have been addressed through models, safety, security, and resilience have not been addressed as one problem. However, in CPS these three qualities are perhaps the most important for the eventual deployment and use of these systems.

To achieve this transition, it requires us to learn (but avoid repeating) from highly resilient systems built over time, such as those developed with a fly–fix–fly approach in the United States Federal Aviation Administration's air traffic control system. In that case, the system modifies aircraft flight rules depending on pilot visibility through use of instrument flight rules and visual flight rules, thereby, as necessary, reducing airport capacity while increasing safety. But this raises the question about how should systems engineers model "-ilities" of the system, which are interconnected and cannot be examined in isolation, before having experienced them during deployment? As a specific example, how would we model the interconnection of a safety incident that was initiated by a security violation in the case of CPS? And further, how would one design resilience and relate such resilience solutions to losses and their prevention to assure the proper operation of complex CPS systems during deployment?

Solutions to parts of these questions have been proposed in each individual field of safety, security, and resilience but without recognizing the problems that arise when these metrics interact with each other. For example, in the field of safety, Leveson has developed systems-theoretic accident model and process (STAMP) [49], which examines safety incidents in terms of hierarchical control and unacceptable losses. In the intersection of security and resilience, there is System Aware [42,43] and its evolution Mission Aware [16,17], which see mitigation in terms of resilient modes in addition to traditional security defenses. Finally, the general field of model-based systems engineering (MBSE) has produced both methods and tools in assessing the safety and security posture of system designs and is the paradigm in which the majority of system models must conform to. An example of a fundamental systems engineering metamodel is the model openly published by Vitech Corporation in support of the company's modeling software [67].

One unifying answer is to create pragmatic ontologies based on the above capabilities developed within each individual field, which will allow for a holistic examination of those qualities in relation to the system model. Crucially, in this paper we extend an already existing and often used systems engineering metamodel to address safety, security, and resilience. For modeling purposes, this ontology is often codified in metamodels. Previous works study the theory of ontologies extensively for the purpose of understanding them at a meta-meta level [45,57,59,60]. However, there is a gap in creating domain-specific ontologies and metamodels, especially for the modeling of CPS. Creating an ontology in the field of CPS should take into account both the system and its associated "-ilities" with a particular focus on safety, security, and resilience. Assurance of these three "-ilities" combats undesirable behaviors during deployment

by focusing the design effort both on the functional and non-functional requirements of the system under design. Therefore, a concrete implementation of an ontological metamodel is one way forward in multi-paradigm modeling for CPS, which is a necessity to manage the intrinsic complexity of these systems [73].

For these reasons, we propose the use of an ontological metamodel that can be applied to the differing modeling views necessary to design CPS. We assert that this metamodel supports the structuring of modeling efforts. Additionally, this metamodel can help users see interoperability between various models and assessment methodologies, which has the potential of documenting both formal and informal requirements in one model [31] and ultimately achieve model federation. There are significant design risks if we permit reliability, safety, and security engineers to work in isolation when designing highly automated CPS. Instead, such attributes must be examined together throughout lifecycle such that appropriate tradeoffs and design decisions can also be made throughout the lifecycle. The metamodel can act as a bridge between system models and safety, security, and resilience by providing a common language that is understood by all stakeholders of the system. Thus, a metamodel can be used to support and foster collaborative design.

We recognize that a single ontology that addresses system issues associated with a large number of quality attributes would be cumbersome and likely not practical. Therefore, we do not seek to develop a universal ontology that would be applicable in all applications; different ontologies are often necessary as they provide differing views. However, we are developing our suggested ontology that integrates safety, security, and resilience considerations on top of an already tested metamodel, for the particular purpose of providing a mission-oriented perspective to system modeling and assurance.

Our suggested ontology will be one that addresses system engineering for systems where safety, security, and resilience are considered to be critical quality attributes. To make our contributions concrete in this area, we have implemented a GraphQL [33] definition of this ontological metamodel in the form of a generalized schema. GraphQL is a vendor agnostic format for schema specification augmented with a query language. Therefore, GraphQL implements the ontological metamodel as a schema that can be used by any modeling language, given minimal export capability.

In summary, our contributions are as follows:

- We derive a metamodel for structured system modeling that addresses safety and security and resilience as one problem. Specifically, we build on and adapt STAMP and mission aware cybersecurity to create general connections between safety and security concepts.
- We use the metamodel to assist with a tradespace analysis of resilience solutions to the losses associated with safety and security violations.
- We implement this metamodel in GraphQL, which encodes these relationships and attributes formally and can interface with a number of tools.

By achieving this unification, we are able to promote consistency between model views, provide one possible solution to the coordination of safety, security, and resilience within a system model, and are able to apply tradespace analysis of design solution with regard to these non-functional requirements. We illustrate the use of the metamodel in a oil and gas pipeline demonstration.

2 Literature review

In order to address the complexity of CPS design, assessment, and implementation, it is required that we keep track of how and why different entities link together [2]. The proposed metamodel is an integration of multiple technology areas into a unified ontology. This integration leverages advances in MBSE [48], safety through STAMP [49], and mission aware cybersecurity [7,17]. Mission aware is a systems engineering methodology that attempts to bridge system design with safety, security, and resilience. We consider the mission aware metamodel a specific application of the metamodel and not necessarily defining the use of the metamodel itself. Our general motivation is to construct a bridge between modeling paradigms and verification tools, which stems from the need to build tool interoperability [13]. For example, adding a contract-based framework [28] to already existing tools is largely a manual process, which could be assisted by enforcing a structured modeling framework through metamodeling. Generally, by using a consistent metamodel such model transformations become part of the system design effort, which is an increasingly desirable quality [51]. This observation is consistent with surveys of the modeling community: consistent and automated model transformations that are language agnostic are necessary to transition model based design in industrial practice [14]. The algorithmic implementation of the metamodel provides one answer to modeling tool interoperability, which has the potential to create connections between static views of the system, for example, SysML block diagrams, and dynamics views of the system, such as Mathworks Simulink. In turn, this could assist with providing evidence for the safe operation of CPS, which requires the use of several types of models [55].

2.1 Complex systems

One of the main objectives of the proposed metamodel and associated ontology is to improve the ability to manage com-

plexity. In particular, the focus is on complexity management in the context of model-based systems engineering vis-a-vis CPS and their properties of safety and security. It is therefore important to describe what we mean by complexity in this context, because there are many different notions and definitions of complexity. Rather than seeking to address computational complexity and complexity theory, we seek to address *system complexity*, which itself has several different connotations.

Attributes that lead to system complexity include nonlinearity, emergence, adaptability, multiple scales in space and time, self-organization, tight coupling between behaviors, among others [21,29,46,68]. The result of these types of attributes is that these complex systems are increasingly difficult to understand or predict in terms of behavior; costly to build, operate, or maintain [23]; and prone to migration toward riskier states [63].

This complexity relates directly to safety when the systems interact with the physical environment and control components that have kinetic, thermal, electrical energy or other types of potentially hazardous sources. Furthermore, this complexity is related to security when either a cyber attack or physical attack can lead to such hazardous behavior. It is this context of *system complexity* that gave rise, in part, to model-based systems engineering as well as the need for ontologies and metamodels [27,40]; the advent and increasing prevalence of CPS further motivates this need.

2.2 Model-based systems engineering

MBSE has gathered increasing attention both by the United States Department of Defense [70] and by standards committees, for example, RTCA DO-331 [24] and RTCA DO-333 [25], SAE AS5506C [3], and IEEE 1547 [39] to name a few. The tenant of MBSE is that models take a central role in the design and development of systems. Even further, models take the role of *living documents*, where justifications for design choices and changes are recorded and disseminated. In addition, MBSE attempts to bridge potential system design gaps by encapsulating the several layers of abstraction necessary in the design of systems. The assertion is that different views of a system's design must be contrasted and related. For example, candidate implementations of the system are modeled in relation to desired behaviors and requirements. The promise of MBSE is that within a modeling language one can diagrammatically represent any given hierarchical level of system design, from requirements to functional behaviors to architectures.

One such language is systems modeling language (SysML). SysML is a general-purpose specification of a modeling language for systems engineering which has been maintained by the object management group (OMG) since 2007 [35]. Different vendors have implemented the SysML standard, and therefore, there is an abundance of tools available in the market. As a standard, SysML allows for a variety of systems engineering approaches to be developed within what might be considered a metamodel specification for system design [79]. In 2017, the OMG issued a request for proposals for a second-generation specification known as SysMLv2 [34]. A primary requirement for SysMLv2 is to augment the flexible syntax of block diagrams with an overall model representing best systems engineering practice. This demonstrates the need for a concrete metamodel for system design that is used to control relationships between model entities and views.

Academia has already provided a number of solutions to this need [5,6,44]. However, academic solutions that attempt to view the system holistically often lack formal semantics, without which the solution cannot provide the necessary consistency between model views. On the other hand, solutions implementing formal semantics often address a small subset of the overall system design. In this paper, we attempt to build a metamodel that both views the system holistically and implements formal semantics for all fundamental entities and views of system design, including entities relating to safety, security, and resilience.

To achieve this merger, we build upon a metamodel that is already successfully used in industry. Specifically, we use the Vitech metamodel [67] as a starting point. The Vitech metamodel is publicly available and is well aligned with the SysMLv2 requirement for "precise systems engineering semantics." Our augmentations to this industry metamodel include a number of entities and interactions to address safety, security, and resilience, which are necessary for the design and deployment of CPS. This stemmed from the challenge of extending beyond the system itself and being able to capture mission-oriented information. It is possible to bring "front and center" a number of whys, hows, and whats that otherwise would be missed by just examining the system in isolation of its purpose. Often a system designer's "whats" might be a software engineers how's [76]. A metamodel explicitly considers the hierarchy within system design, where at each layer there is both behavior and architecture, an important distinction compared to modeling languages that view the two as distinct.

Specifically, what we consider to be the MBSE metamodel represents critical systems engineering concepts and their interrelationships spanning requirements, behavior, architecture, and testing (Fig. 1). This integrated model presents a high-level view of not only the ultimate specification of a system, but also the journey to that specification—concerns opened and closed, risks identified and managed. As a mechanism for managing system complexity, the metamodel supports an incremental, layered approach via the consistent

relationship across requirements, behavior, architecture, and test domains.

The domain of CPS is particularly suited for exercising MBSE because they rely on both analogue behavior—the physics—and digital—-the computation. Often modeling tools are disjoint, and they allow the designer to either allow the designer to, for example, simulate the physics of the system, on the one hand, or alternatively, verify correctness of software, on the other hand. Because CPS brings both physics and computation together in one system, it requires use of models that address both physical and computational performance as a coupled system. This consists of one of the main challenges in modeling CPS, namely the challenge to capture all interactions between those two different domains. A solution to this challenge is to enforce clear semantics between model entities, such that high-fidelity modeling is possible. This is the predominant motivation for creating a language-agnostic metamodel that contains relationships between standard modeling paradigms, for example, behavioral and architectural entities, and necessary performance metrics for assuring the correct behavior of CPS, for example, addressing both safety and security concerns.

2.3 Loss identification with STAMP

Safety assessment has a long tradition. Here, we present some of the results coming from STAMP, a safety analysis method that is based on different model of causation than many other hazard analysis techniques. Causation in STAMP is modeled through hierarchical control, which models each level of a system as a controls process, where unsafe control actions can occur. This layered approach to safety has the advantage that unsafe control actions at each level percolate upwards or downwards in the hierarchy that in turn provides a notion of consequence within the safety model. STAMP works in contrast to linear failure modes, where unsafe actions form a chain of events. Instead, in STAMP safety violation emerges from the interacting control layers governing the system. STAMP is grounded in systems theory, as evidenced by these notions of hierarchy, interactions among components, and feedback and control [4,18,53].

Specifically, STAMP [50] is a hazard analysis technique based on an extended model of accident causation. In addition to component failures, STAMP assumes that accidents can also be caused by unsafe interactions of system components, none of which may have individually failed. For this reason, STAMP further asserts that emergent properties, for example safety and security, cannot be assured by examining subsystems in isolation. STPA (System-Theoretic Process Analysis) is one flavor of STAMP modeling that is primarily used to proactively identify hazardous conditions and states. STPA-Sec is an extension on STPA with the intention of transitioning the benefits of loss-oriented safety assessment to security [80,81].

The hierarchical control notion within STAMP is a congruent idea with a number of MBSE block diagrams such as architectural or behavioral diagrams because they can be augmented to model unsafe control actions in addition to the control system that define the behavior and architecture of the system. Furthermore, MBSE is based on the same hierarchical notion, namely that systems can be modeled through different views that reside in different levels of abstraction. STAMP entities are, therefore, an important view of safety and security, which we would like to introduce in more general form within the metamodel.

Another reason to use notions from STAMP in a metamodel for CPS is that it has recently been incorporated as a possible safety and security analysis technique in several standards, for example, SAE J3187 [66], SOTIF ISO/PAS 21448 [41], RTCA DO-356A [26], ASTM WK60748 [78], and SAE AIR6913 [1], to name a few. By grounding the safety and security metamodel entities in existing and future standards, we encourage a modeling methodology that can assist with the eventual system certification and validation process. While we focus our demonstration on STAMP, the construction of the safety entities within the metamodel is general and therefore can assist or otherwise augment other hazards or safety methods.

2.4 Mission aware cybersecurity

Although the metamodel presented in this paper can be used generally for CPS safety, security, and resilience, it is important to provide context for *how* the metamodel can be used. One such methodology, termed mission aware, is presented to motivate one example of the metamodel's use (Fig. 2). In general, mission aware is a series of steps and processes that brings together expertise from three disparate disciplines: operations (blue team), systems engineering (yellow team), and cybersecurity (red team). The blue team consists of operationally oriented experts that are familiar with similar current systems and/or handling systems under duress from environmental disturbances, cyber attacks, or other disruptions. The systems engineering team consists of system analysts and modelers with technical expertise. This team is responsible for the development of initial system designs, consequence analysis based on loss scenarios, and the derivation of potential resilience design features. The red team consists of expertise in both defensive and offensive cyber capabilities.

Previous work [16] identified a six step approach for engineering in safety, security, and resilience could take the following interaction between model artifacts and teams.

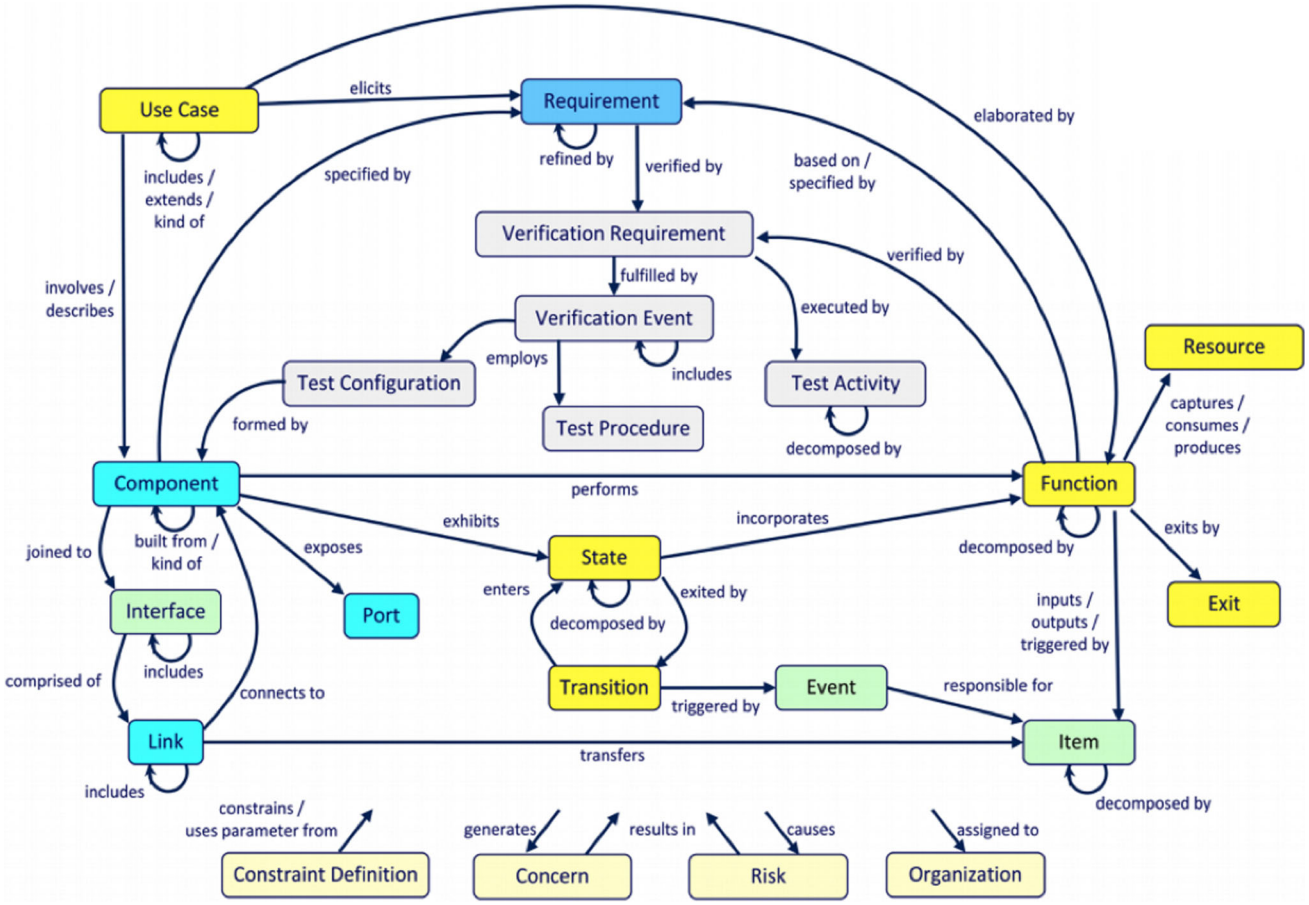


Fig. 1 The Vitech model-based systems engineering metamodel is suited to build upon a general metamodel for cyber-physical system design, safety, security, and resilience (reproduced from Scott and Long [67])

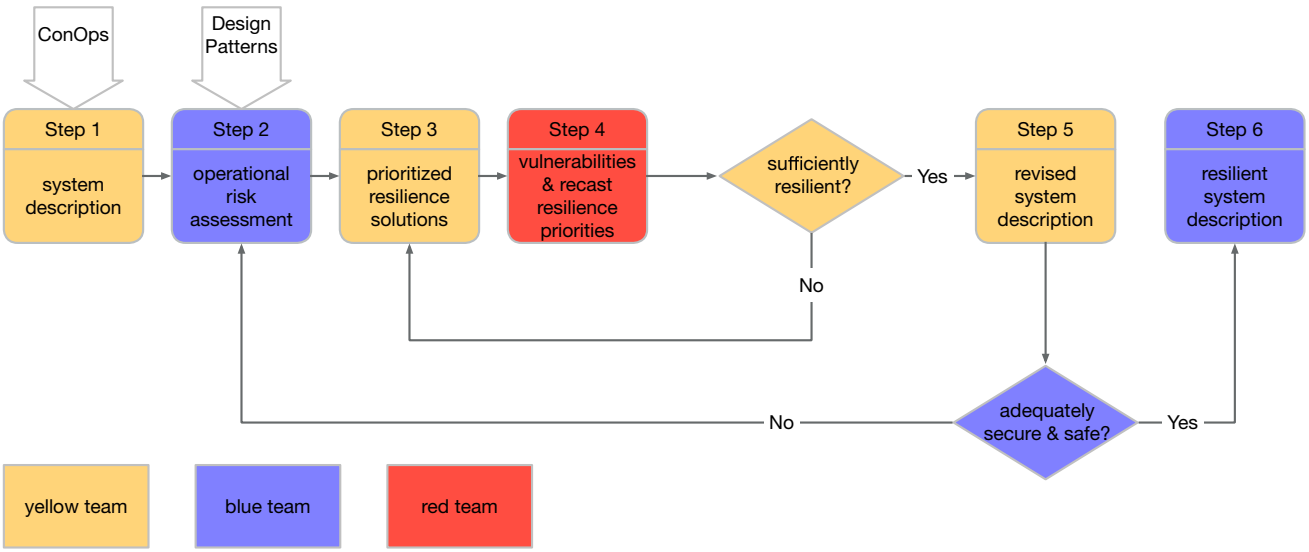


Fig. 2 In mission aware the systems engineering lifecycle is segmented by specific teams that address system design and resilience based on losses related to safety and security (adapted from Carter et al. [16])

Step (1) Generate system description produced by the systems engineering team, including objectives and goals, high-level technical requirements, system architecture, and functional or behavioral description in a tool-based medium such as SysML.

Step (2) Using those model artifacts the blue team conducts a consequence analysis, which results in a prioritized list of undesirable functional outcomes from an operational perspective, as well as a systems engineering team consequence analysis (for example by using STPA).
Step (3) The systems engineering team derives potential resilience solutions based on the results of the previous step.
Step (4) The red team identifies loss scenarios for the system and prioritizes defenses, resilience, and software engineering solutions.
Step (5) The systems engineering team refactors the system descriptions based on red team recommendations.
Step (6) Finally, the blue team responds to the refactored system descriptions.

Resilience in mission aware defines the type of mitigation patterns that allow the system to maintain state awareness, which further requires to *proactively* maintain a safe level of operational normalcy in response to anomalies, including attacks [65]. A concept familiar to resilient design pattern practitioners and necessary to achieve this operational normalcy is the *monitor* [37]. In mission aware, the monitor is termed a sentinel. The sentinel's purpose is to be as simple of a system as possible with provable guarantees on its safe and secure behavior. In terms of the overall system, the sentinel monitors the state of the system and given a number of resilient design patterns decides on mitigative actions if things are measured to be abnormal—given an initial notion of what normal behavior means in the design phase and is updated throughout the rest of the lifecycle. In the rest of the paper, we will use sentinel to indicate this design pattern.

There are many other possible methodologies and uses of the metamodel, but this section serves as a relatively concrete guide for its use. The brief familiarity with mission aware is necessary for the case study demonstration of the metamodel (Sect. 4), because the modeling artifacts assume this engineering methodology.

2.5 Metamodels for system design

The problem of creating useful metamodels for software system design and the overall need for *systems*-level metamodels is not new, and indeed, the long-term vision of using metamodels to integrate methods and tools and maintain consistency within different model views has not changed [64]. Beyond the Vitech metamodel, which was built specifically to address an industry need, there are other approaches to structured modeling that have emerged in the past two decades. The consensus and general trajectory of this line of research seems to be toward specialized ontologies for a particular application and not a general metamodel for all system design. Specialized ontologies can bridge diverse views necessary for the particular type of system, in this case CPS, leading to better management of already existing model types. In particular, by specializing, one can capture precise semantics for hierarchical decomposition, which we posit is one way to achieve multilevel modeling [22]. Any graphical language can be described or otherwise contained by the OMG meta object facility (MOF) specification [56]. However, MOF relates to the implementation of diagrammatic languages and not to the structured modeling we want to facilitate to assure a number of "-ilities". With this in mind, we see a gap in structured modeling for the domain of CPS, where we need to provide evidence of safe and secure operation before deployment.

Practical reports on modeling methods show that currently we lack effective collaboration, are error-prone in syncing models, and often metamodels are only useful if used fully [32]. Our metamodel partially overcomes these challenges by working across design artifacts, allowing checks of agreement by using a consistent metamodel outside of a particular modeling language or tool, and does not require full conformance to assist in decision-making. This means that depending on the CPS application, system modelers can use the appropriate subgraph of the overall metamodel.

In general, expertly constructing metamodels requires an iterative process [19]. Informal modeling tools must be connected to concrete semantics. These semantics often come from the relationships between metamodel entities and not necessarily from the entities themselves. Beyond the actual construction of a metamodel, there is still no deep understanding about what information a metamodel should include, how it should be included, and in what way it should be related [77]. This means that a large part of metamodeling construction happens by example.

A particular problem requires a particular *type* of entity and that entity is added to the metamodel. This can end up creating massive metamodels that are difficult to navigate and apply during modeling. This notion of type has led to a number of works in metamodeling that apply formal semantics from programming languages, which attempt to manage this process [10–12]. Composition of type entities can then lead to new types without extending the base metamodel. Composition of different metamodels can lead to modeling tool interoperability [20], which speaks to the need to create small compact metamodels for particular modeling efforts and "-ilities" associated with them. Particular examples of tools that rely on a metamodel to some extend include AADL [54], EST-ADL [75], and Mathworks Simulink [69], to name a few. Interoperability of these tools would require the composition of their respective implicit or explicit metamodels.

All these metamodels, including Vitech's, require the system engineer to accept a particular paradigm *and* model-

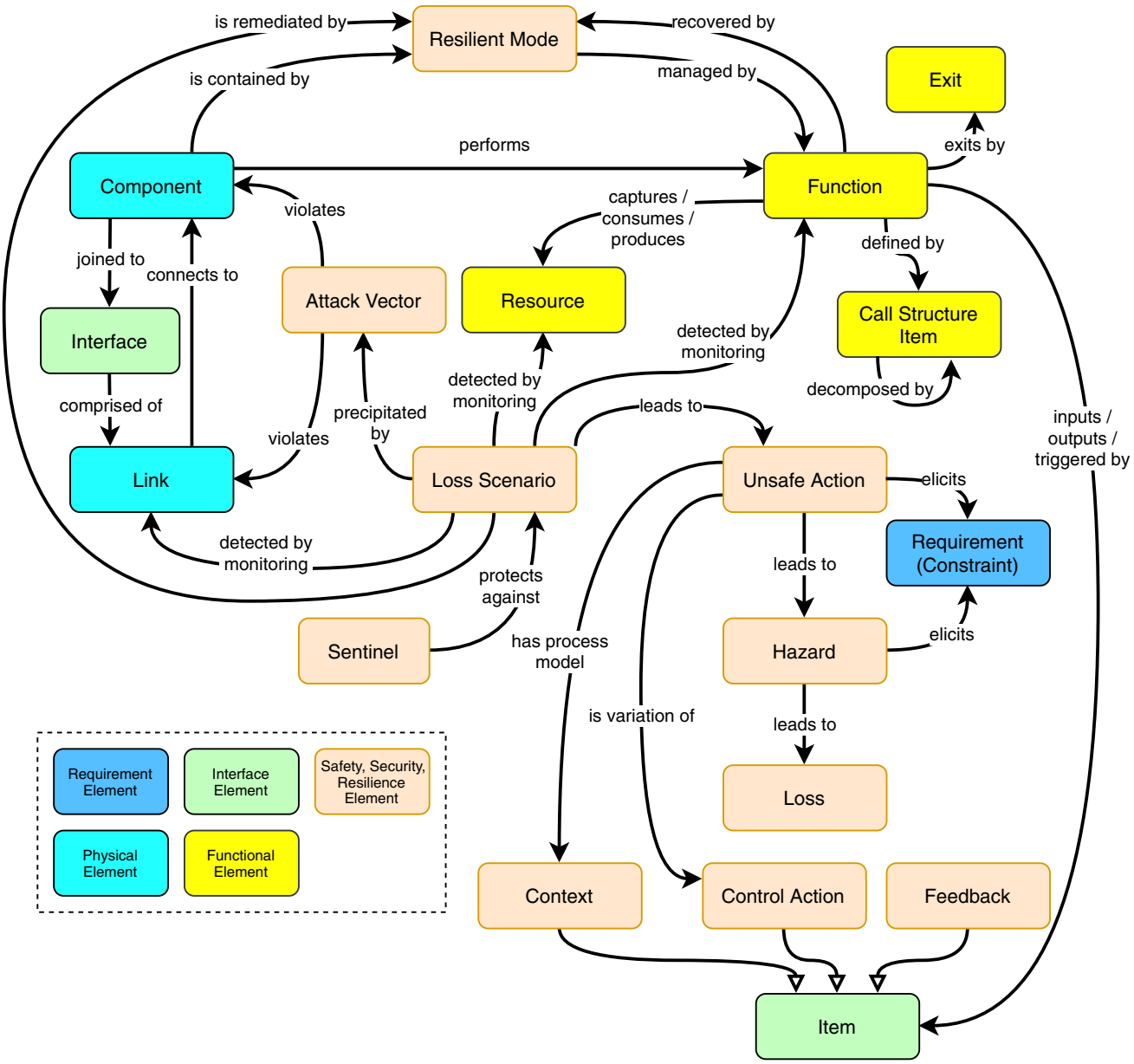


Fig. 3 *A top-level view of the metamodel*. The proposed metamodel addresses relevant performance metrics for cyber-physical systems, namely safety, security, and resilience on top of an industry model-based systems engineering metamodel

ing tool. By implementing our metamodel in GraphQL, we decouple the metamodel from a particular tool and instead enforce the method in whichever modeling language might be best equipped to model a candidate system. Even though the promise of metamodels is such that they reside on top of any particular modeling abstraction, we note that, for example, modeling a system in SysML versus in AADL has distinctly different objectives. Models in SysML typically reside at a higher level of abstraction than AADL but lose the expressiveness and guarantees versus modeling in a lower-level language such as AADL. In addition, no metamodel presented in this section addresses safety, security, and resilience as one problem, which is vital for modeling CPS.

Our motivation was also inspired by the observation that composing metamodels is a necessity in practice. This led to two requirements for the metamodel. The first is to decouple the metamodel from a particular modeling tool and create a flexible representation that is designed for extensibility. The second is to construct a compact metamodel augmentation for a specific set of "-ilities" that are necessary in the design of CPS over an already tested metamodel, which in some sense is metamodel composition. We believe that a number of advances have emerged by following this design path for metamodeling.

Table 1 The standard model-based systems engineering metamodel entities capture the fundamental components and relationships in the systems engineering lifecycle

Element	Entity	Description
Requirement	`Requirement`	A `requirement` is either an originating `requirement` extracted from source documentation for a system, a refinement of a higher-level `requirement`, a derived characteristic of the system or one of its subcomponents, or a design decision
Physical (or cyber)	`Component`	A `component` is an abstract term that represents the physical or logical entity that performs a specific `function` or `functions`
	`Link`	A `link` is the physical implementation of an `interface`
Interface	`Interface`	An `interface` describes the logical connections between parts of an architecture
	`Item`	An `item` represent flows within and between `functions`. An `item` is an input to or an output from a `function`
Functional	`Function`	A `function` is a transformation that accepts one or more inputs (`items`) and transforms them into outputs (`items`)
	`Call Structure Item`	Recursive `call structure`, for example, select, parallel, loop, for each `function`
[1em]	`Exit`	An `exit` identifies a possible path to follow when a processing unit completes
	`Resource`	A `resource` is an element, for example, power, MIPS, interceptors, that the system uses, captures, or generates while it is operating
Miscellaneous	`Constraint Definition`	A `constraint definition` captures a parametric constraint as an expression, identifying the independent variable(s), with associations to the system parameters

3 Ontological metamodel for safety, security, and resilience

During the early design phase of a systems lifecycle, various engineering disciplines—hardware, software, systems, etc.—work in parallel to refine a system design. The process is iterative in nature, with each team expressing their needs as they arise and the design morphing to incorporate the necessary modifications. The design may be documented in multiple formats to communicate and emphasize different views. Existing formats include databases for requirements, for example IBM's Rational DOORS, diagrammatic software for architecture diagrams, text documents for interface descriptions, or model-based software for covering multiple of these representations.

The proposed metamodel (Fig. 3) assists both with the ease of the aforementioned design process and with maintaining the rigor of the design. Among others, there are three anticipated benefits to using a metamodel:

1. holding the various methods of documentation in alignment, which will reduce workload in maintaining the documents and prevent errors;
2. enforcing a set structure to remain in compliance with systems engineering best practices; and
3. connecting performance metrics like safety, security, and resilience with the system design process.

The first benefit relates largely to viewing the metamodel outside of a specific modeling tool and applying the refinements everywhere in the design artifacts. The second benefit relates to producing an algorithmic interpretation of an otherwise informal metamodel, such that future development of refinement types and guarantees can be linked across modeling tool barriers. The third benefit relates to addressing the need of assuring a number of "-ilities" within the metamodel, which must take a central role in the design of CPS where undesired behaviors can lead to accidents. These three benefits summarize the difference between the Vitech MBSE metamodel and the metamodel presented in this paper and ultimately our contributions to model federation, a significant challenge facing MBSE at large [61].

Having said that, a MBSE metamodel should only be concerned with the architecting activities of system design and not with the minutiae of a particular system design. It is often the case that building on top of an existing metamodel merely leads to added complexity both to the users and tool vendors [30]. For this reason, we keep the main entities of the original metamodel as is and construct *relationships* between the main entities of system design and the entities related to safety, security, and resilience. This does not mean that we enforce which parts of the metamodel must be modeled. Modeling is as much a form of art as it is a practice of engineering and science. The metamodel exists to add extra structure *only* to the elements that a system designer needs to use.

Table 2 The metamodel augmentations to the standard model-based systems engineering entities address safety, security, and resilience

Element	Entity	Description
Control structure	`Control action`	A controller provides `control actions` to control some process and to enforce constraints on the behavior of the controlled process
	`Feedback`	Process models may be updated in part by `feedback` used to observe the controlled process
	`Context`	The set of process model variables and values
Hazard analysis	`Loss`	A `loss` involves something of value to stakeholders. Losses may include a loss of human life or human injury, property damage, environmental pollution, loss of mission, loss of reputation, loss or leak of sensitive information, or any other loss that is unacceptable to the stakeholders
	`Hazard`	A `hazard` is a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to a `loss`
	`Unsafe control action`	An `unsafe control action` is a `control action` that, in a particular `context` and worst-case environment, will lead to a `hazard`
	`Loss scenario`	A `loss scenario` describes the causal factors that can lead to the `unsafe control` and to `hazards`. Two types of `loss scenarios` must be considered: (a) Why would `unsafe control actions` occur? (b) Why would `control actions` be improperly executed or not executed, leading to `hazards`?
Mission aware	`Attack vector`	A path or means by which a hacker can gain access to a computer or network server in order to deliver a payload or malicious outcome. Attack vectors enable hackers to exploit system vulnerabilities, including the human element
	`Resilient mode`	A configuration of a target system that remediates one or more `loss scenarios`
	`Sentinel`	A highly secure subsystem responsible for monitoring and reconfiguration of `resilient modes` for a target system

3.1 Metamodel description

A key concern of any systems engineering model is an understanding of the system's architecture, including its `components`, logical `interfaces`, and physical `links` which connect them. Components may include hardware elements, software elements, external systems, and/or humans. Of equal concern is an understating of the expected behavior of the system being modeled. Behavior elements include `functions`, their input and output `items` as well as any `resources` provided or consumed. The `call structure` provides an understanding of behavior control flow including looping, parallel execution, path selection with `exit` choices, etc. Components `perform` functions thereby linking the physical architecture with the behavior model. These standard system modeling entities define the engineering process itself and provide structure to the essential design artifact of the system under design (Table 1). We emphasize the fact that the element *Physical* in Table 1 again refers to *architectural* entities in the general CPS sense; that is, these components and interfaces can include software, embedded systems, communications, as well as physical connections, actuators, etc. While there may be confusion about the term "physical" in this context of CPS, this choice was an attempt to remain consistent with the existing industrial metamodel and associated ontology and tools.

However, MBSE entities and relationships do not address "-ilities" necessary for the design of CPS. Additional entities for safety, security, and resilience that are specifically related to CPS must be added to provide evidence for the correct behavior of CPS. Such performance metrics are defined in the augmentation of the CPS metamodel and related to already standardized MBSE entities with properly defined relationships. This is an important addition to the standard metamodel provided by Vitech and defines our main contribution to metamodeling for CPS. By adding structure to performance metrics, we are better able to design CPS that provide operational assurance in the face of hazards or security violations. Moreover, we provide a framework to design in and assess the efficacy of resilience, when perimeter-based defenses are either insufficient or too costly.

Traditional system performance metrics are captured as `parameters` of links, components, and/or functions with a `constraint definition` defining the equations and relationships between individual entities. Consideration of safety, security and resilience performance metrics require

Table 3 Further refinement which captures security considerations for vulnerability assessment

Entity (Association)	Attribute	Description
Attack vector	description	The standard description of the attack, which often comes from databases that contain attack patterns, weaknesses, and vulnerabilities
	domainOfAttack	The domain of the attack; that is, software, hardware, communication, supply chain, social engineering, and physical security
	outOfScope	A binary value (true or false) flagging if the attack vector should be projected over the system
	outOfScopeJustification	A justification for not considering the attack vector within the system scope
Attack vector (violates)	description	Component and/or link specific attack vector description
	mitigationType	The attack vector system mitigation for the associated component or link, for example, defensive, diverse redundant, or hardening solution
	justification	A description of selected mitigation
Loss scenario	detectPattern	The Sentinel design pattern associated with the Loss Scenario (Changing Control Input, Data Consistency, Introspection)
	threatCategory	The category of a threat – terminology reused from STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of Privilege) [36]
Loss scenario (detected by monitoring)	constraint	Constraint (=, <, >) for associated function, link, resource monitored by sentinel

augmentation of the standard MBSE metamodel with additional concepts to capture both an operational risk perspective and an adversarial attacker perspective (Tables 2 and 3).

Safety and security often require specification of system behavior as a set of feedback control loops. As such, specializations of control action and feedback are provided as sub-types of the standard function input–output item. While this phraseology is borrowed from STAMP, it applies to a large number of safety and security methods. STAMP in some sense distills any general framework for "-ilities" at a higher abstraction level—by leveraging notions of uncontrolled actions and control hierarchy—that is suited for use in a metamodel. Specifically, losses, hazards, and unsafe actions are captured and related (by means of leads to) as part of a methodical operational risk assessment process and are recurring in a large number of safety, security, resilience, and risk management literature. Additionally, explicit associations are captured to understand an unsafe action as a variation of a specific control action with the process model system state that provides the context for the control action to become unsafe, which is borrowed from the domain of control theory and governs all CPS to some extent.

An important step in assessing any performance metric is to first identify loss scenarios which can lead to unsafe actions. These loss scenarios are the complement of the stakeholders' requirements or otherwise define the mission of the system. In the domain of CPS, unsafe behavior and security violations are intertwined, meaning that an attacker could transition the system to a hazardous state. To augment the safety loss scenarios, we use databases, for example, MITRE CAPEC [15], which contain attack vectors. Attack vectors define the steps necessary to attack a particular system component or set of components. By using this information from the databases, it is possible to apply security information to the system architecture noting loss scenarios that could be precipitated by particular attack vectors. The metamodel relates the notion of loss scenario with the notion of recovery and resilience by identifying how a sentinel could protect against the loss by first indicating how it can be detected by monitoring a link, resource or function and then how it can be remediated by a specific resilient mode.

Augmenting the meta-model for these performance metrics, in the form of losses, attack vectors, and resilient modes has the added benefit of facilitating tradespace analysis. The metamodel does not *do* the tradespace analysis per se, but instead relates the expert and operator perspectives, which are required for priority ranking of system losses, likelihood and severity determination for attack vectors in order to evaluate the effectiveness and complexity of resilient modes. Other import evalu-

Table 4 Further refinement that captures metrics for assessing resilience

Entity (Association)	Metric	Description
Loss	`priority`	A ranking based on blue team (operator) determination: 1. Unacceptable `loss` and highest priority to provide `resiliency`. 2. Avoid `loss` as long as `resiliency` solution does not over-complicate operation. 3. Would like to avoid `loss`, but solution needs to be incremental. 4. Lowest priority, low-cost, simplistic solutions should be considered
Attack vector	`likelihood`	A ranking based on the red team determination: the `likelihood` of the attack (high, medium, low)
	`severity`	Typical `severity` of this type of attack (very high, high, medium, low, very low)
Loss scenario	`detectionTime`	Time budget to detect `loss scenario`. Architecture impact and tradeoff for `sentinel` interfaces, for example, polling-based (system and link loading) or event-based
	`operatorDecisionTime`	Time budget to isolate the `loss scenario` via system and/or `component` tests
Resilient mode	`complexity`	Degree of model "contained by" associations. Indication of development cost (high, medium, low)
	`effectiveness`	Impact on remediating High `likelihood` attacks associated with High mission `priority` (high, medium, low)
	`operationalImpact`	Degree of operator training need. Degree of mission interruption (high, medium, low)
	`restoreTime`	Time budget to restore system function via `resilient mode`. Architecture Impact / tradeoff for Resilient Modes: Active/Active, Active/Standby (hot, warm, cold)
	`operatorDecisionTime`	Time budget for operator decision time to enable `resilient mode`. 0 implies automated resilient mode
Function (recovered by)	`recoveryRatio`	Calculated [per `Loss Scenario`] measured/expected where if < 1 acceptable and if > 1 not acceptable. `Recovery time` includes `detection and isolation and restoration`

ation metrics for system resilient modes are an understanding of the `operational impact` and the time budget for system recovery. Recovery time includes `detection time`, `isolation time` and `restore time` including any `operator decision time`. System simulation can evaluate the `recovery ratio` for critical system functions under various system loads and simulated attack patterns. Tradespace analysis, based on resilience metrics, enables specification of a system which responds to safety and security violations, while achieving operational priorities, within programmatic cost and time constrains (Table 4).

3.2 Algorithmic implementation

In Myers's seminal work [58] on taxonomies of visual programming, it is argued that visual programming languages have a number of issues in practice, some of which include:

- the visual representation is always significantly larger than the text representation they replace;
- visual languages lack formal specifications;
- visualizations often are poor representations of the actual data; and
- implementations lack portability of programs.

These observations are also applicable to systems modeling languages. Database representations are often more concise and compact versus their visual counterparts. Implementations of modeling languages like SysML lack formal semantics. The available diagrams often are not always the best representation of a system models abstraction. Finally, OMG produces standards for portable SysML models, but in practice this is not the case between vendor tools. These drawbacks show that a tool agnostic, text-based, and database-oriented implementation for containing the relational data defining the model is useful, even when most of the modeling takes place within the visual tool. In addition, this opens up the system model to a number of external tools to select model entities, process the model through other means, and report on the results of external analysis back into the initial visual tool.

For the above reasons, we implement the formal specification of the proposed metamodel in GraphQL. We provide this formal specification to the modeling community for scientific dissemination, augmentation, and ultimately use. The full CPS schema is published online as open-source software [71]. GraphQL is a schema specification language, a query language, and provides query results via a JSON docu-

ment [33]. While not initially created for capturing modeling language metamodels, these attributes make it an appealing language to programmatically implement metamodels. This is particularly the case because GraphQL is an open-source standard, which is implemented in a vendor agnostic format. Furthermore, the schema syntax is both human readable and machine parsable. The JSON model document provides a useful interchange format for various system and design modeling tools.

The top-level GraphQL specification of the CPS schema defines the metamodel at its highest level of abstraction (Listing 1). A single query, that is, `cpsSystemModel`, returns a complete system model including both standard MBSE entities and augmentations (Sect. 3.1).

Listing 1 The top level of the CPS schema captures the complete metamodel specification

```
schema {
 query: Query
 mutation: Mutation
}
type Query {
 cpsProjects: [Project]
 cpsSystemModel(projectId: ID!): CPSsystemModel
}
type CPSsystemModel {
 project: Project
 attackVector: [AttackVector]
 component: [Component]
 context: [Context]
 controlAction: [ControlAction]
 document: [Document]
 domainSet: [DomainSet]
 exit: [Exit]
 feedback: [Feedback]
 function: [Function]
 hazard: [Hazard]
 interface: [Interface]
 item: [Item]
 link: [Link]
 loss: [Loss]
 lossScenario: [LossScenario]
 requirement: [Requirement]
 resilientMode: [ResilientMode]
 resource: [Resource]
 unsafeAction: [UnsafeAction]
 callStructure: [CallStructure]
}
```

Listing 2 The component definition follows the standard entity pattern and naming convention

```
type Component {
 identity: ComponentID!
 attributes: ComponentATTR
 parameters: [Parameter]
 relations: ComponentREL
}
type ComponentID {
 id: ID!
 name: String!
 number: String!
}
type ComponentATTR {
 type: ComponentType
 inventory: [String]
 clin: String
 outOfScopeAttackAnalysis: Boolean
 outofScopeJustification: String
 mission: String
 operations: [String]
 puid: String
 purpose: String
 cost: Float
 receptions: [String]
 abbreviation: String
 title: String
 description: String
}
```

Each entity definition follows a pattern of:

1. identity (id, name, number)
2. attributes
3. parameters, and
4. relations

and includes a formal definition, provided in a comment block, as a mechanism to enable common understanding and usage of the modeling artefacts.

The attribute and relationship definitions are extendable and specific to the entity type. For example, the `component` entity includes a number of useful attributes (Listing 2) and relations including both refinement, for example, `specifiedBy`, and action sets, for example, `isViolatedBy` (Listing 3).

Listing 3 The component relationships show the interconnections between model types

```
type ComponentREL {
 builtFrom: [BuiltFromTarget]
 builtIn: [BuiltInTarget]
 connectedTo: [ConnectedToTarget]
 documentedBy: [DocumentedByTarget]
 enablesDetectionOf: [EnablesDetectionOfTarget]
 generalizationOf: [GeneralizationOfTarget]
 isViolatedBy: [IsViolatedByTarget]
 joinedTo: [JoinedToTarget]
 kindOf: [KindOfTarget]
 performs: [PerformsTarget]
 protectsAgainst: [ProtectsAgainstTarget]
 reportedBy: [ReportedByTarget]
 simulates: [SimulatesTarget]
 specifiedBy: [SpecifiedByTarget]
}
```

The algorithmic implementation allows us to decouple the metamodel from any specific modeling tool or language. A key advantage of GraphQL over a simple JSON or XML specification is the ability to extend the GraphQL schema with additional query types as well as *mutation* types to enable an application programming interface (API) for updates to a system model (Listing 4). The system-level mutation `cpsSystemModel()` provides a mechanism to capture the delta between variants of a system specification. These variants of functionally equivalent systems allow for a tradespace analysis between alternative approaches of addressing system safety, security and resilience.

Additionally, this API could be used to create a bidirectional link between the GraphQL implementation and a modeling tool. Developers will find an abundance of GraphQL open-source libraries across multiple language bindings as well as multiple cloud hosted GraphQL services. The algorithmic implementation also provides a formal framework upon which further analysis tools can be built. For example, a graph database could be populated with entities representing nodes in the graph and entity relationships representing the edges of the graph. One application of this transformation would be to automatically propagate security violation over the hierarchy of the model after doing model-based security assessment [8]. Another could be using standard data filtering and processing tools on the model to find particular subsystem entities, which is a significant capability in larger system models found in industry.

Listing 4 The CPS schema supports a system-level mutation as a mechanism to capture the delta between variants of a system specification

```
type Mutation {
 cpsProject(project: Project_Input): Project
 cpsSystemModel(projectId: ID!, cpsSystemModel:
 CPSsystemModel_Input): CPSsystemModel
}

input CPSsystemModel_Input {
 attackVector: [AttackVector_Input]
 component: [Component_Input]
 context: [Context_Input]
 controlAction: [ControlAction_Input]
 document: [Document_Input]
 domainSet: [DomainSet_Input]
 exit: [Exit_Input]
 feedback: [Feedback_Input]
 function: [Function_Input]
 hazard: [Hazard_Input]
 interface: [Interface_Input]
 item: [Item_Input]
 link: [Link_Input]
 loss: [Loss_Input]
 lossScenario: [LossScenario_Input]
 requirement: [Requirement_Input]
 resilientMode: [ResilientMode_Input]
 resource: [Resource_Input]
 unsafeAction: [UnsafeAction_Input]
}
input Component_Input {
 operation: MutationOperation!
 identity: ComponentID_Input!
 attributes: ComponentATTR_Input
 parameters: [Parameter_Input]
 relations: ComponentREL_Input
}
enum MutationOperation
{
 Create
 Update
 Delete
}
```

Finally, an important benefit of using the algorithmic GraphQL implementation of the metamodel is that often different vendors use different modeling tools. This means that to evaluate the diverse vendor solutions the system solicitor must acquire and be familiar with each of the tools used in the design process of each vendor. By using the GraphQL implementation (assuming that a bridge has been built from each individual tool—a one-time process), it is possible to have a

common language among modeling tools that adhere to the same set of modeling requirements based on the metamodel.

4 Demonstration: oil and gas pipeline

Oil and gas pipeline companies share the concerns of policymakers and others regarding the potential implications of a security violation on industry assets. There are ongoing activities to protect critical infrastructure, provide reliable energy for society, and to preserve public safety and the environment. Adversaries to this industry activity include nation states, criminal organizations, and unaffiliated bad actors seeking to steal intellectual property, compromise industrial control systems, and other nefarious goals.

The industry has seen the evolution of attackers and the advancement of the techniques, tactics, and procedures they use, moving from manual operations to more sophisticated and widespread machine-to-machine automated attacks with use of augmented intelligence. There are multiple other attack vectors including insider threats and attacks via supply chain tampering or disruption. We show an application of the metamodel, namely how the metamodel allows us to capture the mission-oriented information, the potential losses to the stakeholders, and the implications of security violations through this CPS example. Additionally, this is an appropriate candidate for resilience mitigation strategies because it consists of a legacy system. We are not designing from scratch, although the methodology is suited for model-based design as well, but rather show how to examine an already existing system using the metamodel and its associated methodology.

This demonstration system is decomposed and organized according to the mission aware methodology using the Vitech GENESYS MBSE modeling tool which was extended with our metamodel. The particular tool is not necessary to use the metamodel but we use the tool and its associated diagrams to visualize the different model views as defined by the metamodel. Additionally, we were able to show that our metamodel extension was straightforward and that an organizational investment in a particular modeling tool can be preserved without significant tooling or retraining costs. GENESYS provides a mechanism to export a system model to a web-based team view. In this section, we present a summary of key model artifacts but we offer the complete model as open source, which does not require to use the tool. The full oil pipeline model is available online [72]. The web-view model navigator (Fig. 4) shows a `package` view to organize the model artifacts presented in the following sections. Expanding a package folder presents a hierarchy of related entity types.

Based on the metamodel, the general overview of the system model takes the form of a system description, which contains an architecture and an associated behavior, operational risk to the system, resilience design patterns based on the operational risk, and potential threats to the system. In the following sections, we will navigate the model at each of those levels to show how the metamodel assists with adding structure to the modeling activity and how the metamodel relates these different but important views. This means that we will not focus on an exhaustive presentation of the model itself but rather show the relationship between the model and the metamodel and how it could be used to analyze and extend the system's safety, security, and resilience considerations.

As a notational choice, we denote model-specific entities with sans serif text font and metamodel entities with `typewriter` text font.

4.1 System description

The system description defines the base model for the system under examination. It is fundamentally the model in which "-ilities" act on or are otherwise contained in. Therefore, it associates strongly with the MBSE entities of the metamodel (Table 1). Specifically, artifacts of the system description include the system context, the architecture of the system, and its functional behavior.

The system context physical block diagram (Fig. 5a) defines the boundaries and external interfaces for the oil and gas pipeline being evaluated. Each node on the diagram is an instance of a `component`, while each connecting line is an instance of a `link` associated by the `connects to` relation. The system context diagram enables a common understanding among stakeholders of the scope of the system model. In our demonstration system, the pipeline end-points (drilling rigs, refineries, etc.), wide-area communications network (cellular, satellite, etc.), environment (weather conditions, geography, etc.), and security operations center, for sharing of industry related events, are all external to our demonstration system model. Additionally, to enable simulation of cyber attacks, an advanced persistent threat interface is included. Within the system context is the pipeline itself and a peer mission aware system (sentinel) which is concurrently evaluated and is responsible for resilient mode reconfiguration based on detected illogical system behavior which may indicate safety and/or security vulnerabilities.

The system context block definition diagram (Fig. 5b) shows three levels of the pipeline system decomposition. Each node on the diagram is an instance of a `component`, while each connecting line is a `built from` association between components, showing the cardinality of each subcomponent. In our demonstration system, the oil and gas pipeline contains a single human operator, one or two (if redundant) system managers and n pipeline routes. In turn, a pipeline route contains one or two (if redundant) route

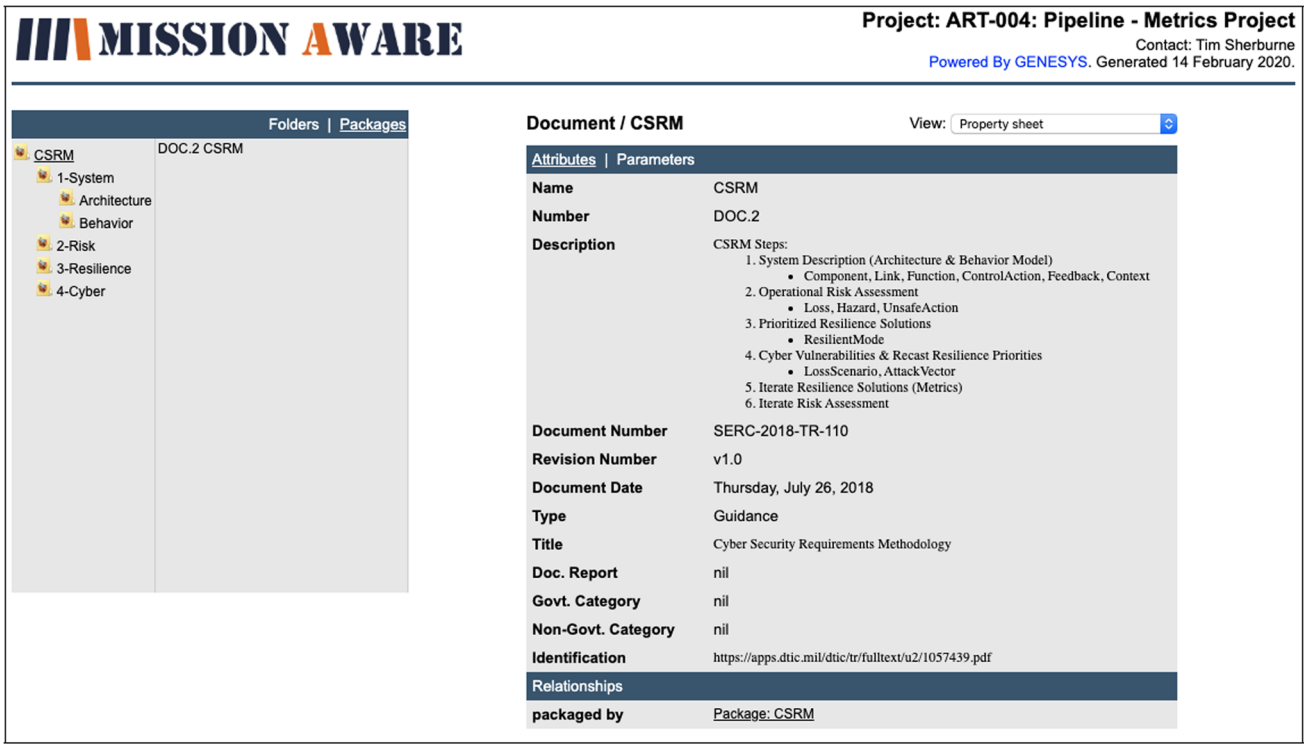


Fig. 4 The top-level web-view of oil and gas pipeline demonstration maps to the top-level concepts on metamodel

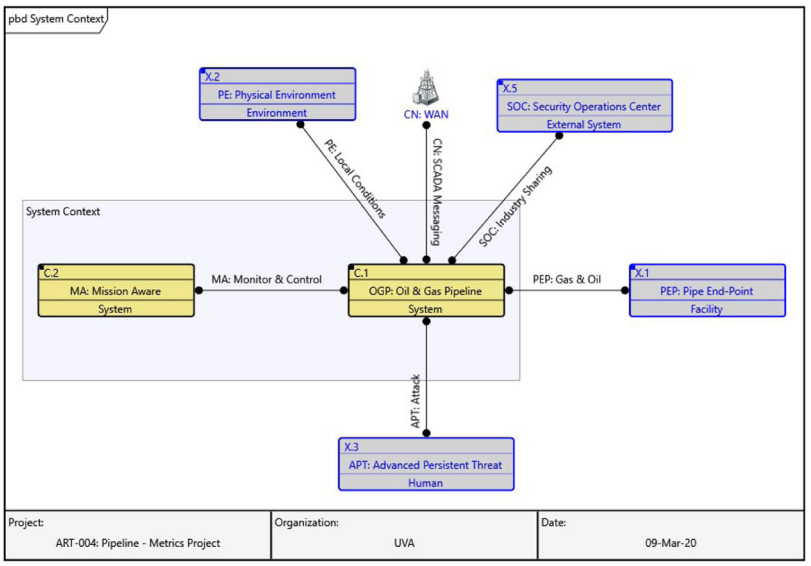
managers, one or two (if redundant) route LANs, and *n* pipeline segments.

Finally, a physical block diagram (Fig. 5c) defines the architecture of a pipeline segment. Each segment includes the physical pipe, pump, and valve that deliver oil and gas from one endpoint to another. A set of sensors (pressure, temperature, flow rate) provide feedback to the segment controller on the operational state of the segment. A segment LAN connects segment components for SCADA messaging and also provides connectivity to the higher-level route LAN. As part of the regular maintenance process for a pipeline segment (known as "pigging" within the industry), a smart PIG is periodically used to clean and inspect the pipe.

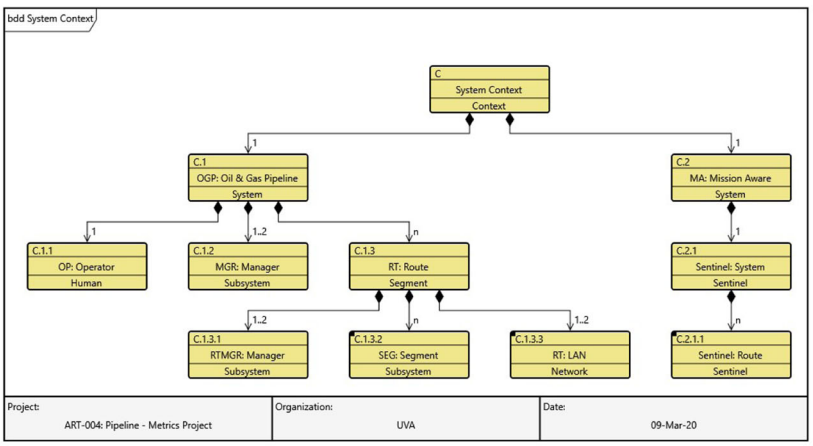
Turning to system behavior, an enhanced functional flow block diagram (EFFBD) (Fig. 6a) defines the top-level behavior for the oil and gas pipeline in the form of a feedback control structure. The diagram shows the top-to-bottom hierarchy of the control structure. The outer-level `and` block shows that each of the lanes of behavior (operator, system manager, route manager, segment controller) execute in parallel. The behavior control flow logic is captured as instances of the metamodel `call structure item`. Each of the yellow rectangles represent an instance of a metamodel `function` showing the function number, name and `performed by` associated `component`. The green rounded rectangles represent instances of a metamodel `control action` or `feedback` item. The lines between functions and control actions or feedback items are represented as `outputs` or `triggers` associations. The `replicate` blocks for the Route Manager and Segment Controller indicate that there *n* concurrent instances of these behavior blocks and are captured as call structure items that `decompose` their respective branches of the behavior model.

The perform segment control function is further `decomposed by` a second level EFFBD (Fig. 6b) with lanes for operations, control, status, and transfer. Each of these lanes operates in parallel in a continuous loop. The handle route manager request receives `control actions` from the route manager and maintains the requested state within the segment status: context. Based on segment state context (requested state and senor status), the control pipe pump and value function initiates `control actions` to the segment pump and value. The collect pipeline sensor status function monitors and maintains the sensor status within the segment state: context and then forwards that state via segment status: feedback to the route manager. Finally, the transfer gas and oil function provides the physical movement of oil and gas through the pipe segment.

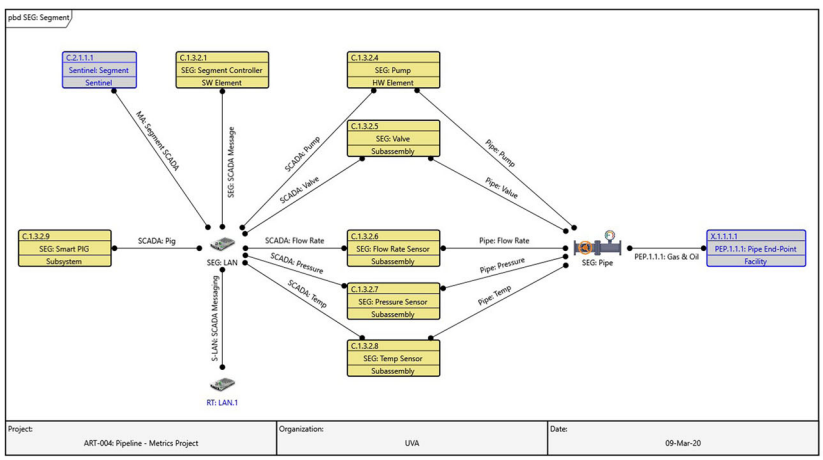
Fig. 5 The architecture of the system is hierarchically decomposed using the metamodel `component` and `link` entities and is visualized with physical block diagrams and block definition diagrams



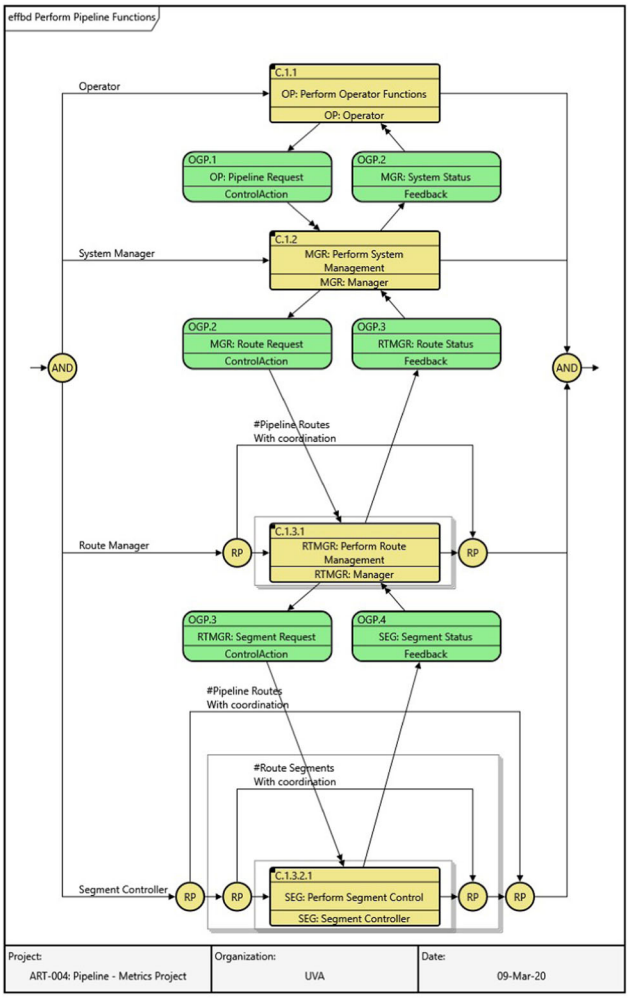
(a) The *system context* physical block diagram models the boundary of the system and its external interfaces.



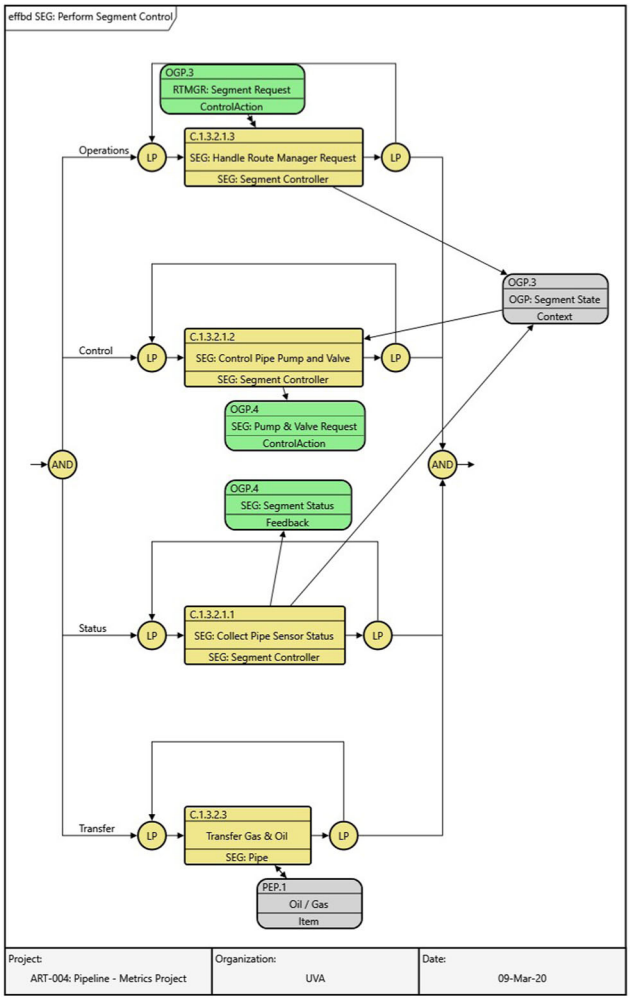
(b) The *system context* block definition diagram models the decomposition of the system including cardinality of sub-components.



(c) The *segment* physical block diagram models the physical pipe, pump, value, sensors and controller and connectivity of the SCADA LAN.



(a) The perform pipeline function EFFBD models the hierarchical feedback control structure of the oil and gas pipeline.



(b) The perform segment control EFFBD models the management of segment state to affect oil and gas transfer.

Fig. 6 The behavior of the system is hierarchically decomposed using the metamodel `function`, `control action`, `feedback`, `context` and `call structure` entities and is visualized with enhanced function flow block diagrams (EFFBD)

4.2 Operational risk assessment

After the system is described to an appropriate level of detail, attention is focused on understanding operational risks for the system from the perspective of end users. Through STPA [50] the metamodel provides a top-down process to aid in the identification of the model artifacts for the operational risk assessment including `losses`, `hazards`, and `unsafe actions` (Table 2).

The system behavior model provides an inventory of `control actions` that are methodically considered to identify potential unsafe actions. From the pipeline behavior model, control actions include Operator: Pipeline Request, Manager: Route Request, Route Manager: Segment Request, and Segment: Pump & Valve Request. As defined by STPA, there are four ways a control action can be unsafe:

- Not providing the control action leads to a hazard.
- Providing the control action leads to a hazard.
- Providing a potentially safe control action but too early, too late, or in the wrong order.
- The control action lasts too long or is stopped too soon (for continuous control actions).

As `unsafe actions` are considered and identified, the `is variation of` association is updated between the `unsafe action` and target `control action`. This association can be used by model reporting tools to ensure that all system `control actions` have been considered and that a complete set of `unsafe actions` have been defined, especially as the system behavior model is iterated and evolved over its lifetime. Additionally, each `unsafe action` is associated via the `has process model` to a

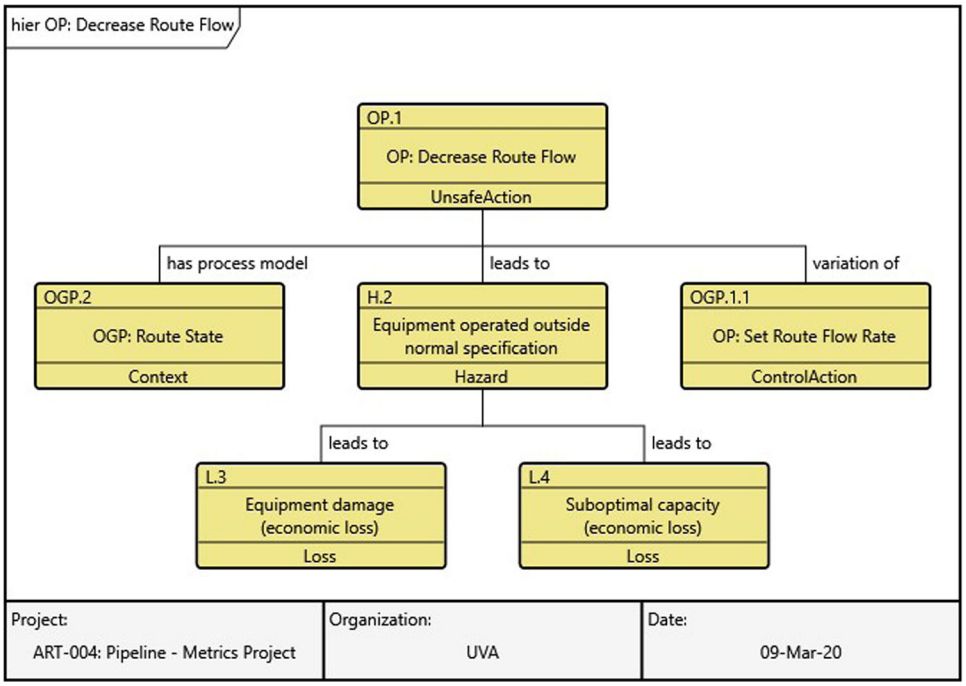


Fig. 7 The operational risk of the system is described using the metamodel `loss`, `hazard`, `unsafe action` entities with associations to the system description `context` and `control action` entities and is visualized in a hierarchy diagram

`context`, identifying the specific values that will cause the `unsafe action` to `lead to` a `hazard`.

One example of such risk is decrease route flow (Fig. 7) where flow rate for a pipeline route is decreased before achieving optimal flow as defined by the associated `process model` context route state. This unsafe action is a `variation of` the set route flow rate `control action` which can `lead to` equipment operated outside normal specification `hazard` which in turn can `lead to` either equipment damage or sub-optimal capacity `losses`. As part of the tradespace analysis, it is required that system operators *prioritize* losses (Table 4).

Failure to fully identify `unsafe actions` can lead to an incomplete specification of resilience solutions and/or an incomplete specification of loss scenarios which are considered in the next sections. The metamodel `is variation of` and `has process model` associations provide a mechanism to ensure a robust operational risk assessment.

Therefore, in this view of CPS modeling (through the metamodel presented in this paper) losses are the main criterion for operational risk. This is in alignment with much of the current literature in safety but might strike security analysts as odd. We posit that this is one of the benefits of the metamodel, namely that safety, security, or resilience extension or modification are based on the potential loss that they mitigate from, thereby providing evidence for their necessity and cost to decision makers. Further, by prioritizing losses, it is possible to compare and contrast what safety, security, or resilience considerations should be added to the system first.

4.3 Resilience solutions

Following the operational risk assessment, system resilience solutions are proposed by system design experts. These solutions are focused on segments of the system that are within a feedback control path for related `unsafe actions` that `lead to` the highest priority system `losses`. Summarizing the feedback control loop; a `link` *`connects to`* a `component`, a `component` *`performs`* a `function`, a `function` *`outputs`* or *`is triggered by`* a `control action` or by `feedback`, and `control actions` and `feedback` are *`transferred by`* a `link`. Resilient solutions are prioritized for `components` and `links` that address the highest quantity of, and most important, `unsafe actions`. The decrease route flow unsafe action is a `variation of` the set route flow rate `control action`. From the behavior model, it is shown that the feedback control loop for this `control action` includes the Operator, Manager, Route Manager, Segment Controller, Pump & Valve, and the Sensor `components`. From the architecture model, it is shown that these `components` are connected by the WAN: SCADA Messaging, Route: SCADA Messaging, and the Segment: SCADA Messaging `links`. Resilience solutions are considered for all of these `components` and `links`. Without these metamodel associations, important resilient modes could be missed or the value of proposed resilient modes may not be appreciated during the tradespace analysis leading to a less resilient system which in turn may cause `hazardous` system states `leading to` unacceptable system `losses`.

Design patterns for resilience solutions include diverse redundancy (which limits the effectiveness of insider or supply chain attacks), hardened design, perimeter defense, etc. An example of a resilience solution is diverse redundant sensors (Fig. 8) where the segment controller and sensors are `contained by` the solution. The degree of contained by associations are an indication of implementation complexity (Table 4) and provide an additional aid in tradespace analysis. The solution `recovers` the collect pipe sensor status `function` and is `managed by` (enable or disable) the perform segment control `function`. The control function could be performed by an operator or, if desired, automated by a sentinel.

4.4 Vulnerability assessment

Guided by safety and security experts, a system vulnerability assessment is performed next. Identification of `loss scenarios` is the primary metamodel artifact that captures this assessment. STPA provides a structured approach for identifying loss scenarios by analysis of the system feedback control structure, while security experts will also consult databases of `attack vectors` considering how a `loss scenarios` could be `precipitated by` these attack vectors. The loss scenario is linked to the operational risk assessment using the `leads to: unsafe action` association and to a resilience solution using the `is remediated by: resilient mode` association. To enable a `sentinel` monitor, a `detected by monitoring` association to a system `link, resource or function` is defined. The inventory of `unsafe actions` and associated `context` that lead to `hazardous` system states provide valuable insight into illogical system behavior that a `sentinel` could detect and report, as an indication of safety and/or security vulnerabilities. Identification of these monitoring needs early in the system design life cycle ensures that appropriate interfaces are incorporated in the system design and are not an afterthought. For example, as defined by the decrease route flow `unsafe action`, this `control action` is considered unsafe if received when the `process model` currentFlowRate is less than 80% of targetFlowRate (i.e., control action received too soon). One cause of this `unsafe action` could be falsified sensor `feedback`. For a sentinel to detect this situation, it would require interfaces to both the targetFlowRate and to real-time senor reports. The sentinel `data consistency` design pattern looks for inconsistencies between values as an indicator of safety and/or security vulnerabilities.

An example of a loss scenario is false sensor reports (Fig. 9) where the loss scenario is `detected by monitoring` SCADA message: link, `is precipitated by` modification during manufacture: attack vector, `is remediated by` diverse redundant sensors: resilient mode, and can `lead to` decrease route flow: unsafe action. The analysis leading to this loss scenario specification is largely the same from both the safety and security perspectives. It is the consideration of intentional actions (`attack vectors`) that differentiates the security analysis. The benefit of this overlap ensures that the identified resilient mode and detection mechanism adequately address both safety and security concerns in the most efficient way. The vulnerability assessment is also responsible for determining appropriate values for the `likelihood and severity` of `attack vectors` as well as time budgets for `detecting`, `isolating` and `restoring` the system (Table 4) which will be leveraged in the following iterative tradespace analysis.

To reason about how these loss scenarios may occur, particularly in the presence cyber security threats, this metamodel supports structured threat and vulnerability analyses. As a concrete example of vulnerability analysis, we will use instances of CAPEC entries. A potentially hazardous attack vector for control systems is described by CAPEC-175: Code Inclusion. This attack vector indicates code that has been introduced into the system either locally or remotely and may contain a payload. At some stage in the Stuxnet exploit, this was used to infect the system with the payload. Another attack vector CAPEC-583: Disabling Network Hardware can transition control systems like the pipeline to a hazardous state by obstructing safety messages and commands. Finally, we can consider attacks of the form CAPEC-441: Malicious Logic Insertion, where the attacker implants malware into the control device through remote or local means for the purpose of violating some system property. The above are illustrations of a possible vulnerability analysis as might be conducted by a security analyst. They are often significantly more comprehensive than the one shown here, and decision trees often augment the findings about the results of a possible exploit or a justification for a defensive mechanism. We have done comprehensive analyses of this form in other systems such as a UAV [7,8] and also a larger attack vector analysis for the pipeline example presented here [52].

These vulnerabilities predominantly violate the integrity and availability of system components. For example, a successful exploitation of the metering system will have the physical consequence of compromised flow in the pipeline, thereby through an integrity violation in the communication system of this device can cause an availability violation. Similar issues can be manifested in the leak detection system, distributed control system, and pipeline monitoring system.

4.5 Iterative tradespace analysis

Systems engineering is an inherently iterative process, and balancing the perspectives of operational risk and system vulnerability also require iteration to achieve an optimal system solution within programmatic budget and time con-

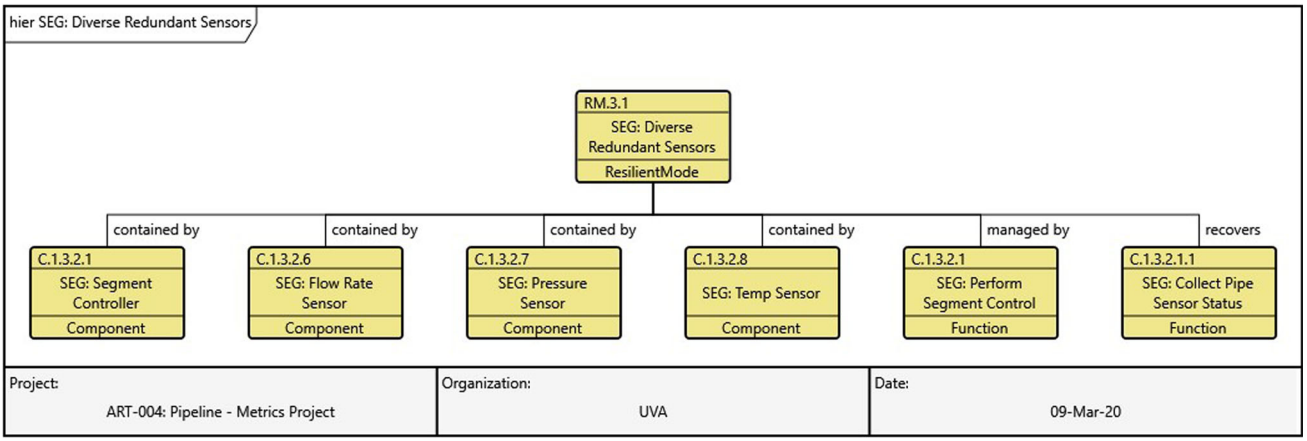


Fig. 8 Resilience solutions for the system are described using the metamodel `resilient mode` with associations to the system description `component` and `function` entities and is visualized in a hierarchy diagram

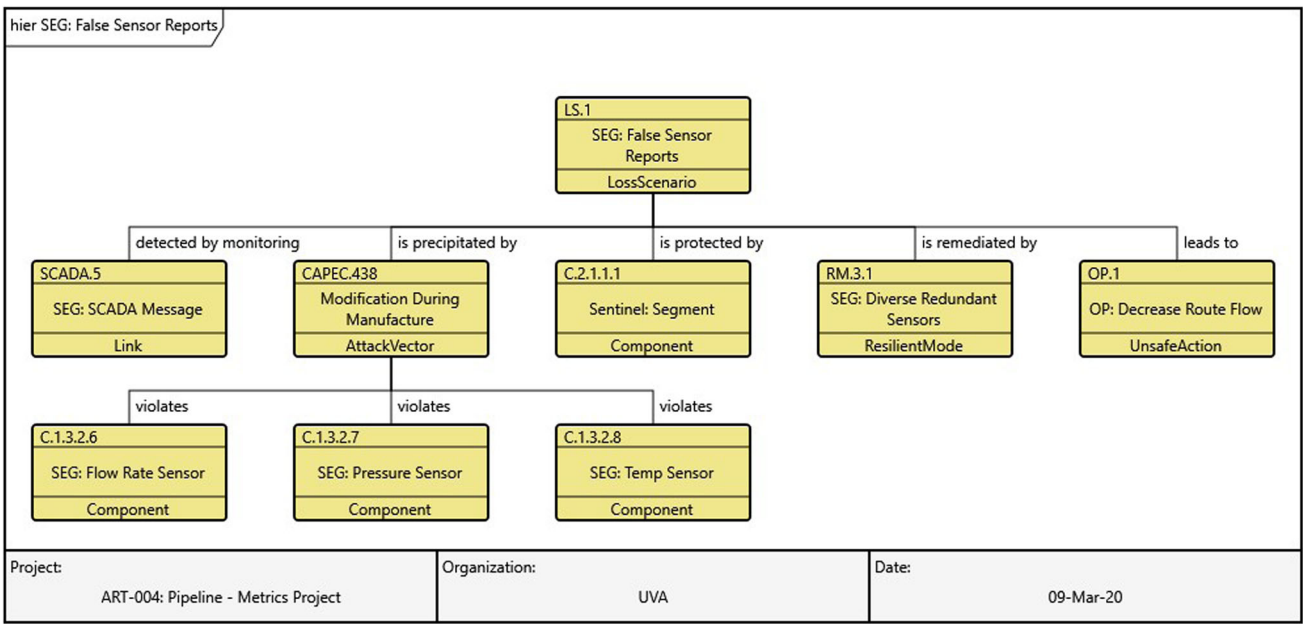


Fig. 9 The vulnerability assessment for the system is described using the metamodel `loss scenario and attack vector` with associations to the operational risk assessment (`leads to`) and resilience solutions (`is remediated by`) and is visualized in a hierarchy diagram

straints. The resilience metrics (Table 4) provide a framework for evaluating the effectiveness of resilience solutions in response to safety and security violations while achieving operational priorities. Determining an appropriate resilience solution for a critical subsystem will likely have multiple approaches, for example, deciding to design redundancy or add security hardening, with security experts preferring one approach while system operators possibly preferring another due to usability considerations. The metamodel provides a mechanism for all stakeholders to understand the trade-offs and a place to document agreements and the process used to reach consensus. This documentation is invaluable to future system enhancements, evolution and maintenance which likely involves different team members. As another example, a system architecture supporting a diverse redundant subsystem must assure that the system recovery time budgets are met. A polling-based detection mechanism may prove to be insufficient and may instead require an event-based notification solution to achieve the required detection time under various system loads.

5 Conclusion

In this paper, we extend an industry metamodel to address the safety, security, and resilience of CPS in a unified framework. The promise of a unified metamodel for all system design has proven to assist little in the assurance of different "-ilities". This is partially because different types of systems prioritize "-ilities" based on their operational needs. For the domain of CPS, it is vital that safety, security, and resilience are considered during the design phase of the lifecycle, where violation of these three "-ilities" can lead to accidents.

A unification of three technologies was needed to design the metamodel: (1) a concrete approach to MBSE, (2) a safety and security method that is grounded on system losses, and (3) a structured approach to mitigation. Additionally, we found that restricting the metamodel to a particular language or tool is insufficient given the diverse sources of generating model artifacts at the design phase. For this reason, we algorithmically implement our metamodel in GraphQL such that it is agnostic to a particular modeling language or tool.

By implementing this unification in a concrete metamodel, we facilitate consistency between CPS model views; coordination of safety, security, and resilience with system models; and tradespace analysis of these three metrics in relation to candidate design solutions. We demonstrate these results and how they are modeled in a pragmatic setting in a demonstration of an oil and gas pipeline. The GraphQL implementation of the metamodel has also allowed us to interface with other analysis methods, such as model-based security assessment outside of the particular modeling tool we use. This capability can be extended to populate other tools and methods based on one model, the results of which can then be reimported to a single modeling tool, therefore achieving bidirectionality between modeling and different types of analysis—a missing capability currently for a vast majority of tools and methods.

Acknowledgements This material is based, in part, upon work supported by the Stevens Institute of Technology through SERC under USDOD Contract HQ0034-13-D-0004. SERC is a federally funded University affiliated research center managed by Stevens Institute of Technology. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the USDOD.

References

1. AIR6913. Using STPA during development and safety assessment of civil aircraft. Standard, SAE, (2018)
2. Allgöwer, F., de Sousa, J.B., Kapinski, J., Mosterman, P., Oehlerking, J., Panciatici, P., Prandini, M., Rajhans, A., Tabuada, P., Wenzelburger, P.: Position paper on the challenges posed by modern applications to cyber-physical systems theory. Nonlinear Analysis: Hybrid Syst. (2019). https://doi.org/10.1016/j.nahs.2019.05.007
3. AS5506C. Architecture analysis and design language. Standard, SAE, (2017)
4. Ross Ashby, W: General systems theory as a new discipline. In Facets of systems science, pages 249–257. Springer, (1991)
5. Atkinson, C., Kühne, T.: Profiles in a strict metamodeling framework. Sci. Comput. Program. (2002). https://doi.org/10.1016/S0167-6423(02)00029-1
6. Atkinson, C., Gerbig, R., Kühne, T.: A unifying approach to connections for multi-level modeling. In Proceedings of the 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS). IEEE, (2015). https://doi.org/10.1109/MODELS.2015.7338252
7. Bakirtzis, G., Carter, B. T., Fleming, C. H., Elks, C. R.: MISSION AWARE: Evidence-based, mission-centric cybersecurity analysis. arXiv:1712.01448 [cs.CR], (2017)
8. Bakirtzis, G., Simon, B.J., Collins, A.G., Fleming, C.H., Elks, C.R.: Data-driven vulnerability exploration for design phase system analysis. IEEE Systems Journal (2019). https://doi.org/10.1109/JSYST.2019.2940145
9. Bakirtzis, G., Ward, G. L., Deloglos, C. J., Elks, C. R., Horowitz, B. M., Fleming, C. H.: Fundamental challenges of cyber-physical systems security modeling. In Proceedings of the 50th IFIP/IEEE International Conference on Dependable Systems and Networks (DSN). IEEE, (2020)
10. Berg, H., Møller-Pedersen, B.: Type-safe symmetric composition of metamodels using templates. In Proceedings from the International Workshop on System Analysis and Modeling. Springer, (2012). https://doi.org/10.1007/978-3-642-36757-1_10
11. Berg, H., Møller-Pedersen, B.: Specialisation of metamodels using metamodel types. In Revised Selected Papers from the Second International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2014), Communications in Computer and Information Science. Springer, (2014). https://doi.org/10.1007/978-3-319-25156-1_6
12. Berg, H., Møller-Pedersen, B.: Metamodel and model composition by integration of operational semantics. In Proceedings of the International Conference on Model-Driven Engineering and Software Development. Springer, (2015). https://doi.org/10.1007/978-3-319-27869-8_10
13. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: From isolated tools to integrated model engineering environments. Proceedings of the IEEE (2010). https://doi.org/10.1109/JPROC.2009.2037771
14. Bruel, J.-M., Combemale, B., Guerra, E., Jézéquel, J.-M., Kienzle, J., de Lara, J., Mussbacher, G., Syriani, E., Vangheluwe, H.: Comparing and classifying model transformation reuse approaches across metamodels. Software and Systems Modeling (2020). https://doi.org/10.1007/s10270-019-00762-9
15. CAPEC. Common attack pattern enumeration and classification. URL https://capec.mitre.org/, (2020)
16. Carter, B., Adams, S., Bakirtzis, G., Sherburne, T., Beling, P., Horowitz, B.M., Fleming, C.H.: A preliminary design-phase security methodology for cyber-physical systems. Systems (2019). https://doi.org/10.3390/systems7020021

17. Carter, B. T., Bakirtzis, G., Elks, C. R., Fleming, C. H.: A systems approach for eliciting mission-centric security requirements. In Proceedings of the 2018 Annual IEEE International Systems Conference (SysCon). IEEE, (2018). https://doi.org/10.1109/SYSCON.2018.8369539
18. Checkland, Peter: Systems thinking, systems practice: includes a 30-year retrospective. J. Operat. Res Soc **51**(5), 647 (2000)
19. Cho, H., Gray, J.: Design patterns for metamodels. In Proceedings of the Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH 2011). ACM, (2011). https://doi.org/10.1145/2095050.2095056
20. Combemale, B., Crégut, X., Pantel, M.: A design pattern for executable DSML. Technical report, INRIA (2010)
21. Cotsaftis, Michel: What makes a system complex?-an approach to self organization and emergence. In From System Complexity to Emergent Properties, pages 49–99. Springer, (2009)
22. de Lara, J., Guerra, E., Cuadrado, J.S.: When and how to use multilevel modelling. ACM Transac. Software Eng. Methodol. (2014). https://doi.org/10.1145/2685615
23. De Weck, Olivier L, Roos, Daniel, Magee, Christopher L: Engineering systems: meeting human needs in a complex technological world. Mit Press, (2011)
24. DO-331. Model-based development and verification supplement to DO-178C and DO-278A. Standard, RTCA, (2011)
25. DO-333. Formal methods supplement to DO-178C and DO-278A. Standard, RTCA, (2011)
26. DO-356. Airworthiness security methods and considerations. Standard, RTCA, (2018)
27. Douglass, Bruce Powel: Chapter 1 - what is model-based systems engineering? In Bruce Powel Douglass, editor, Agile Systems Engineering, pages 1–39. Morgan Kaufmann, Boston, (2016). ISBN 978-0-12-802120-0. https://doi.org/10.1016/B978-0-12-802120-0.00001-1. URL https://www.sciencedirect.com/science/article/pii/B9780128021200000011
28. Dragomir, I., Ober, I., Percebois, C.: Contract-based modeling and verification of timed safety requirements within SysML. Software Syst. Model. (2017). https://doi.org/10.1007/s10270-015-0481-1
29. Flood, Robert L, Carson, Ewart R: Dealing with complexity: an introduction to the theory and application of systems science. Springer Sci. Business Media, (2013)
30. Fondement, F., Muller, P.-A., Thiry, L., Wittmann, B., Forestier, G.: Big metamodels are evil. In Proceedings of the International Conference on Model Driven Engineering Languages and Systems. Springer, (2013). https://doi.org/10.1007/978-3-642-41533-3_9
31. Golra, F. R., Dagnat, F., Souquières, J., Sayar, I., Guerin, S.: Bridging the gap between informal requirements and formal specifications using model federation. In Proceedings of the International Conference on Software Engineering and Formal Methods. Springer, (2018). https://doi.org/10.1007/978-3-319-92970-5_4
32. Gómez, A., Mendialdua, X., Barmpis, K., Bergmann, G., Cabot, J., de Carlos, X., Debreceni, C., Garmendia, A., Kolovos, D.S., de Lara, J.: Scalable modeling technologies in the wild: an experience report on wind turbines control applications development. Software Syst. Model. (2020). https://doi.org/10.1007/s10270-020-00776-8
33. GraphQL Foundation. GraphQL specification, (2020). URL https://spec.graphql.org
34. Object Management Group. SysMLv2 RFP, (2017). URL https://www.omg.org/cgi-bin/doc.cgi?ad/2017-12-2
35. Object Management Group. SysML specification, (2019). URL https://www.omg.org/spec/SysML
36. Hernan, S., Lambert, S., Ostwald, T., Shostack, A.: Uncover security design flaws using the STRIDE approach. MSDN Magazine, (2006). https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach
37. Horowitz, B.M.: Cyberattack-resilient cyberphysical systems. IEEE Security & Privacy (2020). https://doi.org/10.1109/MSEC.2019.2947123
38. Hosseini, S., Barker, K., Ramirez-Marquez, J.E.: A review of definitions and measures of system resilience. Reliab. Eng. Syst. Safety (2016). https://doi.org/10.1016/j.ress.2015.08.006
39. IEEE 1547. Standard for interconnecting distributed resources with electric power systems. Standard, IEEE, (2003)
40. INCOSE international council on systems engineering. A World In Motion: Systems Engineering Vision 2025, (2014). https://www.incose.org/products-and-publications/se-vision-2025
41. ISO/PAS 21448. Road vehicles – Safety of the intended functionality. Standard, SOTIF, (2019)
42. Jones, R.A., Horowitz, B.: A system-aware cyber security architecture. Systems Engineering (2012). https://doi.org/10.1002/sys.21206
43. Jones, R.A., Luckett, B.A., Beling, P.A., Horowitz, B.M.: Architectural scoring framework for the creation and evaluation of system-aware cyber security solutions. Environ. Syst. Decis. (2013). https://doi.org/10.1007/s10669-013-9462-5
44. Kalnins, A., Barzdins, J.: Metamodel specialization for graphical language support. Software and Systems Modeling (2019). https://doi.org/10.1007/s10270-018-0668-3
45. Karagiannis, D., Höfferer, P.: Metamodels in action: An overview. In Proceedings of the First International Conference on Software and Data Technologies (ICSOFT 2006). INSTICC Press, (2006)
46. Kinsner, W.: System complexity and its measures: How complex is complex. In Advances in cognitive informatics and cognitive computing. Springer, (2010). https://doi.org/10.1007/978-3-642-16083-7_14
47. Lee, E.A.: Fundamental limits of cyber-physical systems modeling. ACM Transac. on Cyber-Phys. Syst. (2016). https://doi.org/10.1145/2912149
48. Leibrandt, R.: What is the INCOSE guide to the systems engineering body of knowledge (SEBoK)? In Proceedings of the INCOSE International Symposium (INCOSE 2001). Wiley, (2001). https://doi.org/10.1002/j.2334-5837.2001.tb02378.x
49. Leveson, N.: Engineering a safer world: systems thinking applied to safety. MIT press, (2011)
50. Leveson, N. G., Thomas, J. P.: STPA handbook, (2018). https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf
51. Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G.M.K., Syriani, E., Wimmer, M.: Model transformation intents and their properties. Software and Systems Modeling (2016). https://doi.org/10.1007/s10270-014-0429-x
52. McDermott, T., Fleming, C.H., Clifford, M., Sherburne, T.: Methods to evaluate cost/technical risk and opportunity decisions for security assurance in design. Technical report, SERC (2021)
53. Mesarovic, Mihajlo D, Takahara, Yasuhiko: General systems theory: mathematical foundations. Academic press, (1975)
54. Mian, Z., Bottaci, L., Papadopoulos, Y., Sharvia, S., Mahmud, N.: Model transformation for multi-objective architecture optimisation of dependable systems. In Dependability problems of complex information systems. Springer, (2015). https://doi.org/10.1007/978-3-319-08964-5_6
55. Mitra, S., Wongpiromsarn, T., Murray, R.M.: Verifying cyber-physical interactions in safety-critical systems. IEEE Secur. Privacy (2013). https://doi.org/10.1109/MSP.2013.77
56. MOF. Meta object facility core specification. Specification, OMG, (2019). https://www.omg.org/spec/MOF;jsessionid=B409E18524A8399901F9B13503715740
57. Morozov, D., Lezoche, M., Panetto, H.: Multi-paradigm modelling of cyber-physical systems. IFAC-PapersOnLine (2018). https://doi.org/10.1016/j.ifacol.2018.08.334

58. Myers, B.A.: Taxonomies of visual programming and program visualization. J. Visual Lang. Comput. (1990)
59. Obrst, L.: Ontologies for semantically interoperable systems. In Proceedings of the 12th International Conference on Information and Knowledge Management, (2003). https://doi.org/10.1145/956863.956932
60. Onggo, S.: Methods for conceptual model representation. CRC Press, In Conceptual modeling for discrete-event simulation (2010)
61. Paige, R.F., Zolotas, A., Kolovos, D.: The changing face of model-driven engineering. Present and Ulterior Software Engineering. Springer (2017). https://doi.org/10.1007/978-3-319-67425-4_7
62. Penzenstadler, B., Raturi, A., Richardson, D., Tomlinson, B.: Safety, security, now sustainability: The nonfunctional requirement for the 21st century. IEEE Software (2014). https://doi.org/10.1109/MS.2014.22
63. Perrow, Charles: Normal accidents: Living with high risk technologies-Updated edition. Princeton University Press (2011)
64. Poole, J. D.: Model-driven architecture: Vision, standards and emerging technologies. In Proceedings of the Workshop on Metamodeling and Adaptive Object Models (ECOOP 2001), (2001)
65. Rieger, C. G., Gertman, D. I., McQueen, M. A.: Resilient control systems: Next generation design research. In Proceedings of the 2009 2nd Conference on Human System Interactions. IEEE, (2009). https://doi.org/10.1109/HSI.2009.5091051
66. SAE J3187. Applying system theoretic process analysis (STPA) to automotive applications. Standard, SAE, (2018)
67. Scott, Z., Long, D.: One model, many interests, many views. Technical report, Vitech Corporation, (2018). http://www.vitechcorp.com/resources/white_papers/onemodel.pdf
68. Sheard, Sarah A, Mostashari, Ali: Principles of complex systems for systems engineering. Syst. Eng., 12(4):295–311, (2009)
69. Son, H. S., Kim, W. Y., Robert, Y., Kim, C., Min, H.-G.: Metamodel design for model transformation from Simulink to ECML in cyber physical systems. In Computer Applications for Graphics, Grid Computing, and Industrial Environment. Springer, (2012) https://doi.org/10.1007/978-3-642-35600-1_8
70. United States department of defense. Digital engineering strategy. Technical report, (2018)
71. University of Virginia. CPS metamodel. Software, (2020). https://doi.org/10.5281/zenodo.3752888
72. University of Virginia and Stevens Institute of Technology. Webview: Oil and gas pipeline CPS case study using Vitech GENESYS. Software, (2020). https://doi.org/10.5281/zenodo.3753172
73. Vangheluwe, H.: Multi-paradigm modelling of cyber-physical systems. In Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019). SciTePress, (2019)
74. Voas, J.: Software's secret sauce: the "-ilities" [software quality]. IEEE Software (2004). https://doi.org/10.1109/MS.2004.54
75. Walker, M., Reiser, M.-O., Tucci-Piergiovanni, S., Papadopoulos, Y., Lönn, H., Mraidha, C., Parker, D., Chen, D., Servat, D.: Automatic optimisation of system architectures using EAST-ADL. J. Syst. Software (2013). https://doi.org/10.1016/j.jss.2013.04.001
76. Whalen, M.W., Gacek, A., Cofer, D., Murugesan, A., Heimdahl, M.P.E., Rayadurgam, S.: Your "what" is my "how": Iteration and hierarchy in system design. IEEE Software (2012). https://doi.org/10.1109/MS.2012.173
77. Williams, J. R., Zolotas, A., Matragkas, N. D., Rose, L. M., Kolovos, D. S., Paige, R. F., Polack, F. A. C.: What do metamodels really look like? In Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013), (2013)
78. WK60748. New guide for application of systems-theoretic process analysis to aircraft. Standard, ASTM, (2020)
79. Wolny, S., Mazak, A., Carpella, C., Geist, V., Wimmer, M.: Thirteen years of SysML: a systematic mapping study. Softw. Syst. Model. (2020). https://doi.org/10.1007/s10270-019-00735-y
80. Young, W., Leveson, N. G.: Systems thinking for safety and security. In Proceedings of the Annual Computer Security Applications Conference (ACSAC 2013). ACM, (2013). https://doi.org/10.1145/2523649.2530277
81. Young, W., Leveson, N.G.: An integrated approach to safety and security based on systems theory. Communications of the ACM (2014). https://doi.org/10.1145/2556938

Georgios Bakirtzis recently finished his PhD in Computer Eningeering at the University of Virginia. Dr. Bakirtzis is interested in the intersection of formal methods and control for the safe and secure operation of cyber-physical systems.

Tim Sherburne joined the Engineering Systems and Environment staff at the University of Virginia in 2018 and is supporting Mission Aware research through rapid prototyping. Prior to joining UVA Tim worked for more than 30 years at Motorola Solutions in various Software Development and Systems Engineering roles defining and building mission critical public safety communications systems.

Stephen Adams is a Principal Scientist in the Engineering Systems and Environment department at the University of Virginia (UVA). He received a M.S. in Statistics from UVA in 2010 and a Ph.D. from UVA in Systems Engineering in December of 2015. He is a member of the Adaptive Decision Systems Laboratory, which focuses on applications of machine learning and artificial intelligence in real-world systems. He has experience developing and implementing numerous types of machine learning and artificial intelligence algorithms. His research interests

include feature selection, machine learning with cost, transfer learning, reinforcement learning, and probabilistic modeling of systems. His research has been applied to several domains including activity recognition, prognostics and health management, psychology, cybersecurity, data trustworthiness, natural language processing, and predictive modeling of destination given user geo-information data.

Barry M. Horowitz is the Munster Professor of Systems Engineering at the University of Virginia, Charlottesville. His research interests include system architecture and design.

Peter A. Beling is a professor in the Grado Department of Industrial and Systems Engineering and associate director of the Intelligent Systems Laboratory in the Hume Center for National Security and Technology at Virginia Tech. Dr. Beling's research interests lie at the intersections of systems engineering and artificial intelligence (AI) and include AI adoption, reinforcement learning, transfer learning, and digital engineering. His research has found application in a variety of domains, including mission engineering, cyber resilience of cyber-physical systems, prognostics and health management, and smart manufacturing. Prior to joining Virginia Tech in 2021, he was a professor of systems engineering at the University of Virginia. He serves on the Research Council of the Systems Engineering Research Center (SERC), a University Affiliated Research Center for the Department of Defense.

Cody H. Fleming is an associate professor in Mechanical Engineering at Iowa State University. Dr. Fleming is interested in developing theory and methods to assure the safety and security of controls systems and autonomy more broadly. He received his PhD in Aeronautics and Astronautics from MIT, and before joining Iowa State, he was on the faculty at the University of Virginia.