SAFE and Secure: Deeply Integrating Security in a New Hazard Analysis

Sam Procter
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, Pennsylvania 15213
sprocter@sei.cmu.edu

Eugene Y. Vasserman
Kansas State University
1701D Platt Street
Manhattan, Kansas 66506
eyv@ksu.edu

John Hatcliff
Kansas State University
1701D Platt Street
Manhattan, Kansas 66506
hatcliff@ksu.edu

ABSTRACT

Safety-critical system engineering and traditional safety analyses have for decades been focused on problems caused by natural or accidental phenomena. Security analyses, on the other hand, focus on preventing intentional, malicious acts that reduce system availability, degrade user privacy, or enable unauthorized access. In the context of safety-critical systems, safety and security are intertwined, e.g., injecting malicious control commands may lead to system actuation that causes harm. Despite this intertwining, safety and security concerns have traditionally been designed and analyzed independently of one another, and examined in very different ways. In this work we examine a new hazard analysis technique—*Systematic Analysis of Faults and Errors* (SAFE)—and its deep integration of safety and security concerns. This is achieved by explicitly incorporating a semantic framework of error "effects" that unifies an adversary model long used in security contexts with a fault/error categorization that aligns with previous approaches to hazard analysis. This categorization enables analysts to separate the immediate, component-level effects of errors from their cause or precise deviation from specification.

This paper details SAFE's integrated handling of safety and security through a) a methodology grounded in—and adaptable to—different approaches from the literature, b) explicit documentation of system assumptions which are implicit in other analyses, and c) increasing the tractability of analyzing modern, complex, component-based software-driven systems. We then discuss how SAFE's approach supports the long-term goals of of increased compositionality and formalization of safety/security analysis.

CCS CONCEPTS

• **Security and privacy** → *Software security engineering*; • **Computer systems organization** → *Reliability*; • **Software and its engineering** → *Software defect analysis*;

KEYWORDS

Hazard Analysis, Security, Component-Based Systems, Systematic Analysis of Faults and Errors, System Theoretic Process Analysis

This article was originally published in the proceedings of SAW 2017 (The 4th International Workshop on Software Assurance) co-located with ARES 2017 (The 12th International Conference on Availability, Reliability and Security) by ACM.

ISBN: 978-1-4503-5257-4/17/08 DOI: 10.1145/3098954.3105823

1 INTRODUCTION

Safety-critical systems have become both more common and increasingly sophisticated in recent years, driven by the desire for additional functionality, component-based design and reuse, networking capability, and increased scale [9, 13]. Sophistication and features lead to additional complexity, and while this brings significant benefits, it also makes systems more challenging to build and more difficult to verify for safety properties. The challenge of determining whether or not a system is acceptably safe to use is sometimes addressed by performing *hazard analyses*, which are systematic ways of exploring the potential safety-related issues in a system. Problematically absent from many popular hazard analysis techniques, though, is any consideration of the *security* aspects of a system. Despite the fact that the end result—a decrease in system safety—is the same regardless of the intentionality of the cause, traditional hazard analyses consider how the safety of a system could be affected by component degradation or accidental misuse, but they do not consider possibilities of sabotage or intentionally destructive behavior.

Popular hazard analysis techniques like Failure Modes and Effects Analysis do not explicitly include security aspects, focusing instead on hardware reliability issues. Young and Leveson explain that most commonly-used techniques are several decades old, and were designed when the absence of software made systems far simpler than those that are being built now [26]. Hazard analysis techniques have matured considerably since then, but even Leveson's own technique—a systems-theory based analysis known as STPA—does not explicitly address security concerns [12]. Leveson and Young have introduced a security-focused derivative, STPA-Sec, to remedy this [25]. Unfortunately, maintaining separate analyses for safety and security means that overlaps between the efforts cannot be exploited, leading to redundant work at best and possibly missed safety issues [6].

System development techniques have also evolved in previous decades; this evolution has in some ways increased the exposure of newly-built systems to safety and security issues [12]. Not only are modern systems more complex and sophisticated than their counterparts from previous years, but they are also increasingly built as networks of cooperating, semi-independent elements; a notion nicely summarized in the term *systems-of-systems*. These distributed, networked, system topologies increase construction flexibility but also introduce a range of vectors through which errors—whether caused intentionally or accidentally—may be introduced to the running system. Additionally, well-defined interfaces enable the construction of components by multiple vendors so the provenance of an individual component may not be immediately

verifiable by users; the system itself must be able to manage the trust of its components [8].

One domain where these issues—the benefits and challenges presented by an increasing reliance on interconnectivity between heterogeneous elements—are particularly acute is in the critical-care medical space. Specifically, a vision is emerging of interoperable, connected medical devices that could automatically perform some of the mundane, though vital, tasks currently required of human clinicians by forwarding sensor readings, enabling multi-device safety interlocks, and supporting closed-loop control [7]. This vision is centered around the concept of a *Medical Application Platform* (MAP), a safety- and security-critical real-time middleware [7]. Such systems can bring significant improvements to patient safety and efficacy of care, but the requirement of network-capable medical devices presents a challenge. The promotion of patient sensors and actuators—from standalone components whose interactions are mediated by a clinician to automated elements that are potentially responsible for independently determining the veracity of their inputs—qualitatively changes the nature of safety vulnerabilities in the critical care context. Not only will medical devices now require exposing sensing and actuating functionality over a network interface—which enables new privacy and integrity threats (due to, e.g., malicious control)—but the code hosting and execution capabilities inherent in a MAP introduce a previously nonexistent adversarial path.

As a result of considering the safety of MAP applications, a heavily modified form of STPA called *Systematic Analysis of Faults and Errors*, or SAFE, was recently developed and described by Procter [16]. At a high level, SAFE is comprised of recursively performing two activities on each element in a system in order to determine how it could negatively impact the overall safety:

(1) *Considering external errors:* In this step, analysts consider the effects of erroneous (e.g., early, missing, incorrect, etc.) input on the component's declared interaction points, i.e., its ports.
(2) *Considering internal faults:* In this step, analysts consider all other problems caused either by internal issues (e.g., deterioration, compromised hardware or software, etc.) or external input that does not arrive via declared interaction points, e.g., heat, liquids, radiation, etc.

(A more extensive introduction to SAFE is provided in Section 4, but the reader should refer to [16] for full details.)

Unlike existing hazard analysis techniques, SAFE not only includes security considerations, but treats them as first-class citizens on par with more traditional safety concerns. It more tightly integrates safety and security than existing techniques, which we argue is a natural progression resulting from the realizations discussed by Young and Leveson [25], and is in the spirit of—but more flexible than—the analysis proposed by Friedberg et al. [6]. This work expands on SAFE to provide:

(1) *Introduction of a unified set of semantic error/effect concepts, based on Dolev-Yao's model [5] that address both safety and security:* On the safety side, we show how the model's semantic concepts can be given a safety-related interpretation akin to "guidewords" used in previously proposed frameworks. On the security side, embracing a widely used adversary model helps with acceptance of our proposed technique and helps relate the analysis to traditional thinking in the security space. More fundamentally, by the nature of its design, Dolev and Yao's model aims for a measure of both completeness and minimality, i.e., it attempts to succinctly cover all the ways that a component could observe its input deviating from its specification.
(2) *Parametricity with regards to adversarial capabilities:* SAFE's Activity 2 can be configured according to the capabilities described by the adversarial threat model an analyst or organization prefers to use. This gives the analysis considerable flexibility and allows it to be tuned to the specific circumstances of a system's safety needs, but it also requires that the assumptions that underlie the threat model are made explicit.
(3) *Worked Example:* We extend a portion of the analysis of a canonical MAP application from Procter [16] to demonstrate the utility of these improvements.

2 RELATED WORK

There are a number of established safety and security analyses, some of which are security-focused derivatives of safety techniques [26] as well as integrated safety and security analyses [6]. While any consideration of security problems is preferable to an exclusive focus on safety when analyzing a critical system, even having separate analyses perpetuates a divide between the two fields. This division has a detrimental impact on the system's overall avoidance of loss [6], and it is one that has persisted since the two communities typically operate independently of one another [26]. The primary difference between safety and security is the intent of whoever caused the loss [26]: security events are caused by malicious actors, while safety events are caused by random, accidental failures or incorrect component interactions. There is a growing recognition that safety and security can be designed into systems in such a way that accidents are prevented (or mitigated) regardless of the intentionality of their cause [26]. We survey three system-theoretic techniques in this section, but direct the reader to a more complete literature review by Schmittner et al. [21].

2.1 STPA: System Theoretic Process Analysis

STPA, developed by Leveson takes the fairly radical approach of basing its analysis on systems theory, which leads to a more holistic view of system safety [12]. In particular, the focus of the analysis shifts from finding root causes of safety issues to identifying safety constraints that would prevent problems regardless of their cause. Security is not an explicit focus of STPA, however. While it is applicable to security problems to the extent that safety and security issues can be mitigated by common safety constraints, a security-oriented version of STPA, known as STPA-Sec, is under development by Young and Leveson [25, 26].

2.2 STPA-Sec

STPA-Sec is a top-down process that, as a security-oriented derivative of STPA, is focused on identifying constraints that would prevent the system from being in vulnerable states [25]. Friedberg et al. identify three issues with STPA and STPA-Sec, though:

System Safety	Dolev-Yao	Network Security
None	Read	Violate Privacy
Corrupt Value	Modify Existing	Craft Arbitrary Packets
Late/Dropped Msg.	Delay/Drop	Inc. Latency/Packet Loss
Early Message	Craft & Send	Impersonate, Deny Service

Table 1: Mapping between system safety, Dolev-Yao, and network security concepts

Accident (Loss)	Hazard
Overdose	Pump Runs When Unsafe
Loss of Privacy	Cleartext Data
Unmanaged Pain	Pump Fails to Run When Necessary

Table 2: Example accidents and associated hazards derived from SAFE Activity 0 for the PCA Interlock

a) The two are not designed to be used as a single, unified analysis, b) There's no guidance on how to integrate safety and security into an analysis process as equals, and c) There's no guidance on how to secure critical elements of the system once they've been identified [6]. Thus, even though the authors of STPA are aware of the need for security and safety to be considered simultaneously, the primary goal of an *integrated* safety and security analysis has not been achieved.

2.3 STPA-SafeSec

STPA-SafeSec is a derivative of STPA that deeply integrates safety and security [6]. It extends and refines the core modeling technique of STPA to better consider security of network-based systems, and introduces lists of "general integrity" and "general availability" threats including notions of message injection, drop, manipulation, and delay. However, the technique does not address STPA's need for increased rigor in analysis. Additionally, there is no discussion of how the identified threats were developed, or the assumptions that underlie the identified threats/how to make those assumptions explicit. Procter identifies the sources and benefits of SAFE's increased rigor (as compared to STPA, from which STPA-SafeSec is derived), including: a) a firm theoretical grounding, b) derivation and refinement of guidewords from existing literature (rather than ad-hoc terminology lists), c) deep integration with a system's architecture, d) component-orientation / partial compositionality, e) partial parallelizability, and f) tool support [16].

Ultimately, of the approaches discussed in this section, STPA-SafeSec appears most promising due to its recognition of the need for deep integration between system safety and security. While its safety and security integration is an improvement over STPA-Sec, its analysis lacks the rigor and functionality of SAFE, and would benefit from a more careful consideration of how it identifies threats and their effects.

3 DOLEV AND YAO'S ADVERSARY MODEL

Dolev and Yao developed a model that enables analysis of messaging protocols on a network which has been compromised by an adversary [5]. This adversary can obtain messages, send messages to any other user, and receive messages sent by any user—regardless of their intended recipient. These capabilities let the adversary read, modify, delay/drop, or send a new message crafted from already known information. One of the key observations of this paper is that whether or not communication is perturbed due to a malicious actor (i.e., a security issue) or due to a fault in the communications infrastructure (i.e., a safety problem), what a receiving component *observes* is the same regardless of the origin of the problem. Because communication attacks and faults have the same effect from a receiving component's viewpoint, we can unify safety and security "effects" in a hazard analysis to a small set of semantic concepts. Table 1 summarizes how the semantic concepts provided by the Dolev-Yao model can be given both a communication safety and a network security interpretation; note that the concept of privacy violation does not exist in system safety.

By allowing the attacker to control the network with no restrictions [10], Dolev and Yao's model gives rise to a notion of completeness: intuitively, every way that something could go wrong with communication is covered. Moreover, the model also has a notion of minimality, in that no concept can be described as a combination of the others. Thus, a second key observation of this paper is that even though the Dolev-Yao semantic concepts were originally introduced for security reasoning, one can leverage the completeness and minimality properties in safety analysis as well. In particular, the Dolev-Yao work provides a semantic foundation for a complete and minimal set of "guidewords" for safety reasoning about communication errors—which is especially important for our focus on analysis of interoperability via networked components. While SAFE is parametric with regards to the set of concepts used to guide analysis, mapping the guidewords an analyst chooses to use to the Dolev-Yao model is a valuable first step in enabling the benefits discussed in Section 5: a) deep safety and security integration, b) reduced analysis space, c) partial compositionality/parallelizability, and d) bridging the formal methods and system safety communities.

4 BACKGROUND

In this section we discuss two topics that are integral to this effort: first, the exemplar medical application used to create SAFE that we extend in the following sections; second, an overview of SAFE.

4.1 PCA Interlock

Though a number of MAP applications have been proposed, the most studied is the *PCA Interlock*. Patients who have undergone surgery or experienced major trauma are often given devices known as patient-controlled analgesia (PCA) pumps. These devices have two primary elements: a pump with a reservoir of a strong—typically opioid—analgesic, and a "bolus trigger," which a patient can use to give himself a dose of analgesic. Though these devices have a number of safety features (i.e., maximum allowable doses, timeouts to prevent trigger "mashing," etc.), overdoses resulting in respiratory depression that can lead to serious injury or death still occur [11].

A PCA Interlock, however, would use the device-interconnectivity and control features of a MAP to disable the PCA pump when sensor

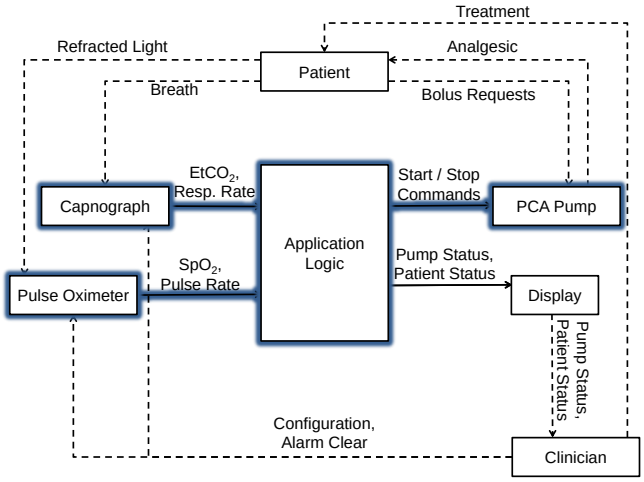

Figure 1: Developer's view of a PCA Interlock application, adapted from / Copyright Procter [16]

readings of the patient's respiratory health, e.g., his blood-oxygen saturation (SpO_2), respiratory rate (RR), and/or exhaled carbon dioxide ($ETCO_2$), showed signs of decline. This enabling/disabling of the pump would be done by having a software controller poll sensor readings, calculate windows of time where it is safe or unsafe for the PCA pump to run, and issue "tickets" with time-length values, which would begin counting towards zero once received. If the bolus trigger is activated when the pump has a positive ticket value, the pump runs normally and administers a dose. If the pump's ticket has expired (i.e., the ticket length is zero: the pump doesn't know the patient's status) or has a negative value (signifying that it is currently unsafe for the pump to run), then the pump does not respond to the trigger activation. Figure 1 shows one possible implementation of this application, in the clinical-care context, from the point of view of the developer. This application has been studied extensively within medical device-interconnectivity research [1, 2], and was explicitly considered in the construction of SAFE [16]. The application logic is simple, but application construction requires dealing with a number of complexities, including:

(1) *Heterogeneous Components:* The MAP vision expects that applications can be constructed using medical devices from different manufacturers, if they provide appropriate sensing and actuating functionality at required levels of quality of service. This is in contrast to current practices, where medical devices that interoperate are typically designed and produced by a single organization and certified for safe use as a single unit.

(2) *Variability:* To achieve interoperability goals, the exact combination of components that will define the MAP system may vary: one device may be replaced by another from the same class as long as the functionality and performance are sufficient for supporting the overall system assurance. In contrast, current state-of-the-art practices in safety-critical system engineering typically define a "system integrator" role, where an expert or group of experts guides the composition of the final system of a fixed set of components.

(3) *Network-Enablement:* Because the devices need to be integrated with one another over a common network, they are in turn exposed to network-based attacks. Network connectivity is central to the MAP vision, so the typical defense of making a device "standalone" is no longer applicable.

We have identified three potential accidents associated with the closed-loop control aspects of the PCA Interlock[1], shown on the left side of Table 2. The first, titled "Overdose," describes the situation where the PCA pump runs when it should not, and provides more analgesic than the patient can safely tolerate. The second, "Loss of Privacy," is used to denote a situation where private patient data is readable by an adversary. The third, "Unmanaged Pain," describes a patient who is in pain (and is activating the PCA pump's bolus trigger) but receives no medication. These accidents, and their associated hazards, are used as a running example throughout this work.

4.2 Systematic Analysis of Faults and Errors

In 2016, Procter introduced the *Systematic Analysis of Faults and Errors*, or SAFE [16]. It is a highly-modified form of Leveson's *System Theoretic Process Analysis* (STPA), applicable to the hardware- and software-based subsystems used in a larger socio-technical system. SAFE does not apply to the social elements of a system.

4.2.1 Activity 0: Fundamentals and Modeling. Like STPA, the first activity in a SAFE analysis of a system is to define what are known as a system's "fundamentals"—essentially a top-down view of the possible losses (or accidents) of concern. These losses are caused by *hazards*, which Leveson carefully defines as a system state and its worst-case environmental conditions [12]. Put another way, system states or actions are rarely *always* unsafe, but are more typically unsafe given some particular environmental conditions. In the PCA Interlock, for example, running the PCA pump is not always unsafe, but doing so when the patient is showing signs of respiratory failure will clearly cause serious injury. The fundamentals for the PCA interlock are summarized in Table 2.

Once these fundamentals have been defined in the first step of Activity 0, the system's control structure is modeled in the second step. In STPA, this modeling is often done using box-and-line diagrams, similar to Figure 1 [12], which will suffice for SAFE, though SAFE can also be used with the more rigorous modeling supported by a semiformal architecture description language like the Society of Automotive Engineer's *Architecture Analysis and Design Language* (AADL) [16]. Once the system has been modeled, work begins with Activity 1, Step 1 analysis of the system element closest[2] to the controlled process.

4.2.2 Activity 1: Externally Caused Dangers. Activity 1 is performed from the "point of view" of an individual component in its system context. Prior to performing this step, the analyst will know the outputs that the current component under analysis should avoid; these can be thought of as the safety constraints that the component must uphold, or alternatively, its guarantees. In SAFE, violations of these constraints are referred to as *successor dangers*; this term

[1] Other accidents involving, e.g., the clinician and/or display are elided for space.
[2] "Closest" here is shorthand for "shortest path through the control structure," or closest in the control-theoretic ordering of the system

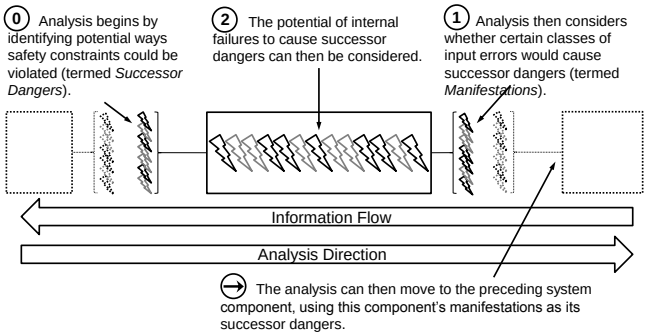

Figure 2: A high-level overview of the component-level activities in SAFE. The rectangles represent components, the lines between them connections, and the lightning bolts are potential safety issues.

is used to emphasize the fact that the outputs to be avoided are dangerous because of how they are consumed by the component's successor in the control-theoretic ordering of the system. As part of the Activity 1 analysis of a component, the safety constraints of the component(s) that provide its input are derived. These can be thought of as the assumptions this component relies on to uphold its safety constraints, or the successor dangers of the predecessor component(s). Note that analysis of the component closest to the controlled process focuses on upholding the system-level safety constraints identified in Activity 0—as it does not control any other components within the system boundary—while all subsequently analyzed components must avoid causing problems in the components they provide input to, i.e., their successors.

It is this aspect of SAFE—the derivation of component-specific input errors and their subsequent use in analysis of the component's predecessors—that gives rise to its backward-chaining structure. An analyst cannot know what particular outputs a component must avoid without having first considered how those outputs are used by the component that they are sent to. That is, in the PCA Interlock example, we do not know a priori that sending a "run" command to the PCA pump when the patient is showing signs of respiratory distress is unsafe (though our engineering expertise might suggest as much). Rather, we derive this knowledge by first considering the behavior of the pump. In doing so, we learn that the pump running while the patient is distressed should be avoided, and then that "run" commands from the application logic cause the pump to engage.

Consider Figure 2, which shows the Activity 1 and 2 analysis of a single component. The initial state—known successor dangers from the predecessor component—is labeled with a 0. Activity 1 then considers the ways that input errors would *manifest* in the current component, and those erroneous inputs are used as the outputs to be avoided for the preceding component.

Table 3 shows a highly-condensed version of the relationship between the successor dangers and derived input requirements for the PCA Pump in Figure 1. The analyst will determine that the pump needs good[3] commands from the application logic in order to provide all necessary analgesic doses to the patient as

Successor Danger	Input Requirement
Unsafe Run-Command	Good Controller Commands
Cleartext	Encrypted Controller Commands
Missed Run-Command	Good Controller Commands

Table 3: Example input requirements derived from SAFE Activity 1 for the interlock's Pump Component (cf. Table 2).

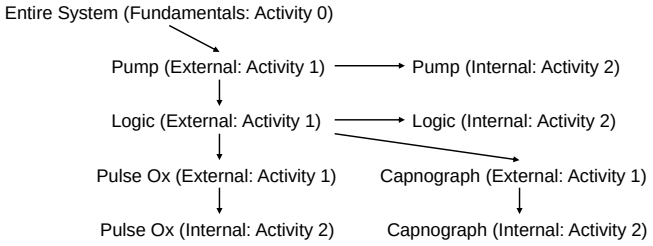

Figure 3: The steps in a SAFE analysis of the PCA Interlock system from Figure 1. Arrows represent viable next steps.

well as avoid overdoses.[4] The analyst will also determine that the pump's inputs—which are patient medical data—should not be sent in cleartext. This example is admittedly simple, but it gives a taste for the structure of SAFE's analysis.

SAFE's Activity 1 is broken down into three steps, which we summarize here. Full descriptions, which are beyond the scope of this work, are available in [16]. Note that after this activity is complete, analysis can proceed either "into" the current component (via Activity 2), or "backwards" into the component(s) that provides the current component's input via Activity 1 analysis of that preceding component. The first option is labeled with a 2 and the latter with a → in Figure 2. Thus, after performing Activity 1's first two steps on a component, analysis can either focus on internal problems, external problems in the component's predecessor, or—if multiple analysts are available—both simultaneously. Figure 3 shows the forking analysis paths when analyzing the highlighted elements of the PCA Interlock control loop from Figure 1. The end result of applying Activity 1 to a component is a collection of input requirements and the effects if input fails to meet those requirements.

Step 1: Process Models. The first task in analyzing a component is simply to document what is known as its *process model*, which is the model the component has of the controlled process. This model (or *model condition*) is what Procter summarizes as the expected relation between a system's internal actions and its external effects [16].

Step 2: Deriving Dangerous Inputs. The next task is to consider the various ways that inputs to the component can manifest as errors, and if those errors could cause the component to produce any of its successor dangers. There are various schemes that can be used to classify these possible manifestations. We emphasize classifications derived from Dolev and Yao's model in this work, but Procter used Avžienis et al.'s service failure domains [3, 16] in the original SAFE text.

[3] We will expand on what "good" means in Section 5

[4] In Procter's formalisms, the analyst is deriving $\mathbb{I}^{PCAPump}_{Overdose,Underdose}$ [16]

Step 3: Documenting External Interactions. The work of a system analyst/designer does not stop once a problem has been identified, but rather only once the system design has been updated to include compensatory measures. This final step in Activity 1 involves not only documenting how the dangerous inputs could cause a particular successor danger, but also how the system might be designed to prevent or mitigate the problem. It is these design changes—which SAFE splits into detection and compensatory measures—that are the final outputs of the system analysis.

4.2.3 Activity 2: Internal Faults. SAFE's Activity 2, which like Activity 1 is also performed from the point of view of an individual component, considers sources of component error other than the arrival of input on known interaction points. It is labeled with a 2 in Figure 2. A similar constraint-informed-requirement relationship exists here as in Activity 1, except that the requirements generated in this activity constrain the component's design and runtime environment rather than its input specification.

Step 1: Fault Class Elimination. As with the error classifications used in Activity 1, SAFE's Activity 2 uses standardized fault classifications. The activity is designed to exploit the hierarchy often found in these classifications by eliminating entire classes of faults that are not relevant to the system under analysis. For example, if the PCA pump will be physically inaccessible to non-authenticated personnel, then we do not ask the analyst to consider the class of faults involving malicious actors gaining access at runtime. The results of this approach are discussed in Section 6.

Step 2: Documenting Internal Faults. Like Activity 1's Step 3, the final step of Activity 2 consists of documenting how internal faults can cause the successor dangers in the current component, and how the faults might be detected or prevented at either design- or run-time.

5 ACTIVITY 1: INPUT SPACE COMPRESSION

One of SAFE's primary objectives is to make the analysis of modern systems—which are often component-based, distributed, and software-driven—more tractable. As systems grow more complex, analysis of them takes more time as there are more interactions and more ways for accidents to occur [22]. Secondary to this objective is the desire to reduce the cost (measured in time, analyst effort, etc.) of safety/security analysis. One key technique used by SAFE to support both objectives is *input space compression*, where the possible erroneous inputs to a component are categorized based on their effect on the receiving component itself, rather than either their source or precise deviation from the specification. This is also important when automating analysis steps using model-checking techniques: reducing the cardinality of the set of errors considered at each communication link can reduce the combinatorial explosion of states when reasoning about all possible error scenarios of the system. We note that this technique is not unprecedented within hazard analyses, but this paper's explicit recognition and discussion of its impacts is, to the best of our knowledge, novel. The technique and its benefits are discussed in Section 5.1. Further, SAFE is agnostic to the categorization technique that an analyst uses; a discussion of possible candidates is presented in Section 5.2.

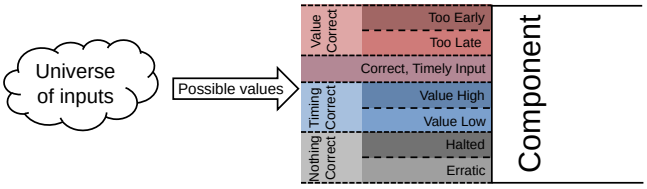

Figure 4: Compression of a component's input space into the service failure domains identified by Avižienis et al., adapted from / Copyright Procter [3, 16].

5.1 Input Space Compression in SAFE

Input space compression is the categorization of erroneous inputs based not on their source or exact deviation from a component's input specification, but instead their effects on the receiving component. As an example, rather than focus on a particular erroneous value such as an SpO_2 reading of 97% when the actual value is 90%, the input would be categorized as incorrectly high. Similarly, rather than analyze the case where input arrives 104ms after dispatch when the input specification requires sub-100ms latency, the input is simply classified as "late." This is similar to the approach used by Walter and Suri in their *Customizable Fault / Error Model* [24].

The technique has three effects on an analysis: a) it enables the simultaneous consideration of errors that are traditionally considered strictly "safety" or "security" problems, b) it compresses the space of possible erroneous component inputs into a handful of possible values, and c) it severs the dependency of a particular component's analysis from the analysis of the components that generate its input. Additionally, this technique also represents a first step in bridging the system safety and formal analysis communities.

5.1.1 Merging Safety and Security Concerns. Rather than determining the cause and/or source of errors, analysts using SAFE can focus strictly on the effects of input errors. Indeed, in a distributed system, developers of a component often cannot know what other component(s) will provide its input. Since specific causal scenarios are unknowable, analysts are forced to disregard the "root cause" of any potential input errors. An approach that focuses on the *effects* of input errors rather than their causes leads naturally to a combined view of safety and security, since the intentionality (or lack thereof) underlying unsafe input can be disregarded. For example, from the point of view of analyzing impacts on the application logic, it doesn't matter if an incoming SpO_2 value is corrupted by a networking fault or if the value has been maliciously altered by an attacker. Leveson and Young come to the same realization,[5] and suggest that the focus must instead be on broadly-applicable compensatory actions [26]. The specific mechanisms for classifying input errors are discussed in Section 5.2.

5.1.2 Analysis Space Reduction. The compression of possible inputs also reduces the impact of what those in the model-checking community would recognize as the state-space explosion problem.[6] That is, rather than consider the impacts of each possible input value outside of a component's input specification, only representative

[5] Young and Leveson write "The key understanding is that from a strategy perspective, the physical (or proximate) cause of a disruption does not really matter." [26, p. 35]
[6] In the language of Pelánek, we use a *State Based Reduction* technique [14].

values need to be analyzed. As there are a number of ways to classify input errors, SAFE has been designed to be agnostic to the exact classification technique used. Procter used Avižienis et al.'s *service failure domains* in his dissertation, and Figure 4 shows the resulting compression graphically [3, 16]. In the figure, two failure "dimensions" are considered: input arrival time and value. These dimensions assume both a real-time system where messages to a component can be either too early or too late, and that the messages are carrying payloads of an ordered type that can be either too high or too low. The six resulting domains stem from the presence of one or both of these failure dimensions.

5.1.3 Independence of "Effects-Based" Analyses. An additional benefit to an approach where the cause of input error is disregarded is that it significantly increases the independence of component analysis. Full independence of analysis would have a number of positive effects; chief among them is the enabling of truly compositional analysis where a system's global properties can be completely determined via local reasoning. While SAFE is does not fully meet this objective—and, in fact, such completely compositional safety/security reasoning may not even be possible except in limited cases—component-basis and process compositionality are not binary. Rather, they are a spectrum: SAFE is *more* component-focused, and thus more compositional, than other system-theoretic techniques, like STPA [16]. Specifically, a component can be analyzed independently of the component(s) that provide its input (i.e., those that precede them in the control-theoretic ordering discussed in Section 4.2.2), but it is fully dependent on analysis of the component(s) it provides input to. Consider Figure 1: analysis of the application logic requires that the PCA pump has been fully analyzed, but does not require anything to be known about the capnograph or pulse oximeter.

A risk—or what might be called a "process hazard"—in using informal guide words is that different organizations (e.g., component vendors) might interpret them differently. If the results of hazard analyses performed by different vendors who used different interpretations for guidewords are then brought together, there is a potential for a loss of soundness, i.e., a loss of the integrity of the analysis. Thus, the benefit of using the Dolev-Yao-derived guidewords is potentially two-pronged: a) we conjecture that since the number of guidewords used is smaller than in other presentations (meaning than efforts in explaining/comprehending the methodology are simpler) they can be treated with greater precision and clarity, and b) Dolev-Yao and subsequent authors provided a rigorous semantic interpretation in terms of a computational model which can be used to help "normalize" the interpretation/understanding across multiple vendors. These benefits support distributed risk management which is required for multi-vendor interoperability.

5.1.4 System Safety and Formal Methods. The fourth, and perhaps most significant benefit is that categorization of a component's inputs based on how they are used in the component itself provides a notion of completeness that is unavailable when considering only concrete error values derived from bottom-up causal chains. Because the error categories are considered independently of their potential sources or actual values, unknown sources or potential deviations are not left unconsidered. For example, suppose there was an undiscovered issue resulting in delayed message output from the PCA Interlock's sensors. Analysis of the application logic—which depends on those sensors for its input—will have already considered the issue (due to, e.g., the "Late Arrival" failure domain) and if necessary designed appropriate safeguards.

The line of reasoning here is similar to the justifications for the use of symbolic values rather than an exhaustive set of concrete values when performing model checking: rather than explicitly enumerating and considering all possible inputs, analysts can instead derive representative values that cover paths through a component. Indeed, Rushby's *Assumption Synthesis* [18] style of system development can be seen as this approach taken to its extreme. Assumption synthesis describes a style where a component's postconditions are first specified, and then the component's behavioral specification is used to check, using some automated tool, if those postconditions can be met. At first this is unlikely, but by adding additional preconditions (i.e., assumptions), a minimal set of assumptions required for the postconditions can be derived [18]. Thus we argue that our work here represents a step in bridging system safety analysis and the more formal style proposed by Rushby. Though we do not believe the box-and-line diagrams used in hazard analyses like SAFE and STPA are readily model-checkable,[7] this work is a recognition of the common ground between the communities.

5.2 Aligning with Existing Concepts

Recall from Section 4.2.2 that SAFE's Activity 1, Step 1 involves considering how the input to a component could be erroneous according to some set of guidewords. The exact set of guidewords used is up to the analyst: Procter used the service failure domains from Avižienis et al. when describing SAFE, but a number of techniques may be appropriate. As an example of the use of these guidewords, consider the SpO_2 input to the application logic from Figure 1. One of the failure domains is "Value High;" applying that guideword to the SpO_2 input would require the analyst to consider the effects of an incorrectly high SpO_2 value. Once those effects had been documented, the analyst would consider the same guideword applied to the other inputs (in the case of the application logic, this would be respiratory rate, pulse rate, and $ETCO_2$). Then, the process would repeat for the other guidewords (timing errors, halted service, content value errors, etc.).

As discussed in Section 3, the input space compression used in SAFE's Activity 1 has precedents in the security domain as well. One of the justifications for the original use of Avižienis et al.'s taxonomy was that it was targeted at the intersection of the safety and security communities [3]. This makes for a fairly straightforward mapping from those failure domains and concepts to notions from both security and safety; an informal version of this mapping is given in the top part of Table 4.[8]

By repeatedly analyzing the PCA pump component of the PCA Interlock, we derived a number of possible design improvements (listed in the central part of Table 4) and the ease with which each set of guidewords—when used with SAFE—detected problems which led to those improvements (listed in the lower part of Table 4).

[7] Tooling supports the execution of SAFE directly on top of architectural models specified in AADL; these models enable a considerable amount of automated analysis.
[8] We note that the informal nature of Avižienis et al.'s failure domains prevents a formal mapping. However, the Dolev-Yao model is formal, and other formal taxonomies (not discussed here) like AADL's EMv2 Error Library [19] exist.

	Dolev-Yao [5]	**Avižienis et al.** [3]	**STPA-SafeSec** [6]	**STPA/STPA-Sec** [12, 25]
		Guideword Mapping		
Early Message	Craft New & Send	Early Arrival	Injection	Providing
Late Message	Delay	Late Arrival	Delay	Late
High Value	Modify Existing	Value High	Manipulation	None
Low Value	Modify Existing	Value Low	Manipulation	None
Service Stopped	Drop	Halted	Drop	None
Babbling Idiot	Craft New & Send	Erratic	Injection	Providing
Confidentiality Violation	Read	None	None	None
		Design Improvements		
Alarming	The pump should be able to issue an alarm to notify a clinician of a problem it cannot independently resolve.			
Timeouts	The pump shouldn't listen for new tickets before the current ticket value reaches 0 to prevent message "flooding."			
Timestamps	The pump gets a ticket after significant network delay, meaning it is enabled after the safe time window.			
Negative Ticket Values	Tickets should specify unsafe time windows as negative values to prevent enabling the pump when unsafe.			
Hashed/Signed Messages	Messages should be signed and hashed. This prevents forged tickets.			
Encrypted Tickets	Messages should be encrypted. This prevents snooping on ticket values.			
		Likelihood of Suggesting Design Improvements		
Alarms	✓	✓	✓	✓
Timeouts	?	✓	?	✓
Timestamps	✓	✓	✓	✓
Negative Ticket Values	?	?	?	?
Hashed/Signed Messages	✓	✗	✓	?
Encrypted Tickets	✓	✗	✗	✗

Table 4: Informal mapping of, design improvements by, and evaluation of error concepts from the literature.

In our evaluation, if a technique would clearly suggest a problem that would lead to a particular design improvement we gave it a "✓," if a skilled/experienced analyst might discover the impetus for an improvement we marked it with a "**?**," and if a technique was unlikely to prompt an issue's discovery we used a "✗." [9]

5.2.1 Avižienis et al.'s Taxonomy of Dependable and Secure Computing. The left two columns of the top of Table 4 map the Dolev-Yao adversary's capabilities to the Avižienis failure domains. The concept of "early" message arrival is not clearly describable in the Dolev-Yao model, but the ability to broadcast arbitrary messages can be used to simulate this threat. Additionally, the Dolev-Yao model is the only one of the four considered in Table 4 that includes privacy violations; Avižienis et al. did not list an equivalent failure.

In Section 4.2.2 we discussed how an analyst would derive the need for the PCA Interlock's application logic to receive "good" sensor data (see Table 3). "Good" is, of course, quite general—so an analyst would want to enhance the specificity of her analysis by incorporating an adversary model that can classify ways that input can be in error. The running example in this paper assumes a) complete trust of hardware and software development, b) physical protection of the application logic's hardware and software at runtime, but c) no protection of the network itself. This may seem like a significant assumption, but it is the result of adopting the Dolev-Yao model: we allow our adversary full access to the network, but not to the application logic or system hardware. The lower portion of Table 4 summarizes the success of Avižienis et al.'s taxonomy's concepts in suggesting system design improvements. SAFE analysis based on the Avižienis guidewords is likely to recommend timestamped tickets and minimum inter-arrival periods (derived from the absolute value of the ticket length) between message arrival. This leaves out more security-centric design improvements suggested by the Dolev-Yao model and STPA-SafeSec.

5.2.2 STPA-SafeSec's General Integrity Threats. A second potential alternative to the Avižienis et al. service failure domains originally used by SAFE is given by Friedberg et al. in their work on STPA-SafeSec [6], discussed previously in Section 2.3. The terms are shown in the middle-right column of Table 4. The authors describe how, starting from STPA's set of causal factors, they derived additional guidewords that address integrity and availability concerns [6]. As a security-focused set of guidewords, these terms align well with the concepts provided by Dolev and Yao though they do not address confidentiality concerns. They guide the analyst to consider the impacts of command and/or measurement messages being injected, dropped, manipulated, or delayed by an adversary.

5.2.3 STPA and STPA Sec. STPA and its security-focused derivative, STPA-Sec, (introduced in Sections 2.1 and 2.2) also contain a number of hazard/threat identification guidewords. As the underlying system model used by STPA is derived from control theory, it assumes that component interactions are binary signals (i.e., no value high/low errors) and non-discrete (i.e., errors of "too long" and "too short"). Extensions to the technique exist, however, which add new guidewords to support additional types of problems such

[9] The full evaluation is available online at http://santoslab.org/pub/safe/Safe-SecureExample.pdf.

No.	Guideword	Possible Compensation	Description
3	Compromised Software	TPM-Like + Chain-of-Trust	Adversary tampers with software in development
4	Compromised Hardware	"Exotic"	Adversary tampers with hardware in development
12	Adversary Accesses Hardware	Physical Security, "Exotic"	Adversary tampers with hardware at runtime
13	Adversary Accesses Software	Access Control, Physical Security	Adversary tampers with software at runtime

Table 5: The four security related fault classes used by SAFE and their suggested compensatory actions [16].[10]

Successor Danger	Design Requirement
Unsafe Run-Command	TPM-Like + Chain-of-Trust
Cleartext	TPM-Like + Chain-of-Trust
Missed Run-Command	TPM-Like + Chain-of-Trust

Table 6: Example design req's from SAFE's guideword 3

Successor Danger	Environment Requirement
Unsafe Run-Command	Physical Security, Access Control
Cleartext	Physical Security, Access Control
Missed Run-Command	—″—, Alarm Monitoring

Table 7: Example environment req's derived from Dolev-Yao

as value errors [23]. The rightmost column of Table 4 shows the mapping (top portion) and evaluation (bottom portion) of STPA and STPA-Sec. As STPA is a very abstract analysis, it relies more heavily on analyst skill/experience than the other techniques. This is advantageous when analyzing less well-defined aspects of socio-technical systems (as the analyst has more flexibility) but is a disadvantage in more well-defined, technical systems like the PCA Interlock.

6 ACTIVITY 2: ADVERSARY MODEL PARAMETRICITY

A significant part of the challenge in safety-critical system development comes not from implementing the required functionality but from ensuring the system's functioning within its environment. Different systems not only have a unique set of objectives but they also must perform their tasks given different resources: different hardware might be in use, different software installations present, and varying levels of access might be granted to users. These environmental realities can be included in different models of a system: its deployment model might take into account available hardware and software resources, and user capabilities might be incorporated into a threat model.

SAFE's Activity 2 was designed with these models in mind, and we refer to the resulting adaptability as *adversary model parametricity*. Recall from Section 4.2.3 that this activity considers either a) faults internal to a component or b) external issues other than those arriving via declared interaction points (i.e., ports) that cause safety or security problems. The exact set of issues is therefore partially derived from the adversary model used by an analyst.

6.1 Parameterization and Fault Classes

SAFE's Activity 2 guidewords, which are partially derived from the fault classes[11] proposed by Avižienis et al., are shown in Table 5 [3]. At a high level, faults can be introduced by adversaries into a component's hardware or software at either design/build time or at runtime; the four security-related guidewords are the cross product of those concepts. Individual terms or sets of guidewords can be eliminated from consideration if the adversary model used by the analyst includes no capabilities relating to a particular time/element. Exactly what a system must be designed to avoid should be dictated by an explicit adversary model—either a general purpose model such as Dolev-Yao's, or a domain specific model akin to the one proposed by Ponikwar et al. [15]—rather than being left unstated, as in most hazard analysis techniques. For example, if the system's software development organization can be completely trusted (i.e., secure facility, no network access, etc.) and the developed artifacts can be delivered via a secure channel, then guideword 3 can be eliminated and its compensatory actions need not be taken.

In the real world, system development (including that of MAP applications like the PCA Interlock) will likely take place in a more standard environment. Thus, we should consider SAFE's guideword 3, which asks the analyst to consider the case where an adversary gains access to the component's software either while it is being built or delivered to the end user. In this case, that would mean that its application logic is compromised, which could lead to nearly any type of problem, including the three hazards originally identified in Table 2. Table 6 shows the new design requirements. SAFE's suggested compensations would direct the analyst to ensure that there is a chain-of-trust—relying on cryptographic trust mechanisms like a Trusted Platform Module (TPM) [4]—in place to ensure that the delivered software is the same as what was developed and/or certified for use. This particular compensation stems from work by Salazar, who explained the need for this approach and provided a prototype implementation [20].

As a second example, guidewords 12 and 13 ask the analyst to consider an adversary who gains access to the system's hardware or software at runtime. If we adopt the Dolev-Yao model, we should also incorporate its environmental assumptions into our analysis—meaning we should disallow the actions the adversary cannot take. Example environmental requirements are shown in Table 7; note that these are requirements that cannot be discharged to the application software itself. They include the need for physical security

[10] In some cases, compensation is difficult or relies on currently-unavailable implementations of theoretical solutions, these are referred to as "exotic" compensations

[11] Note that these are distinct from Avižienis et al.'s service failure domains, which are used as guidewords in SAFE's Activity 1.

(to prevent tampering with the software by non-users), access control (to prevent tampering with the software by underprivileged users), and alarm monitoring (to notify clinicians of problems that can't be automatically compensated for by the system).

Recall that SAFE's Activity 2 is designed to work with—and is dependent on—the results of Activity 1: the outputs of this activity are only potential *causes* of previously identified unsafe component outputs. The effects of the identified faults as well as any causal links are determined as part of the repeated performance of SAFE's Activity 1 that forms its backwards-chaining analysis process. To that end, the lack of network-based guidewords (e.g., "Adversary Accesses Network") is not an oversight: network-based inputs would arrive via a component's interaction points, and thus be analyzed as part of Activity 1.

6.2 Expanding Analysis Scope

The primary impact of this approach is a careful expansion of the system engineering activities treated by a hazard analysis. While many hazard analysis techniques involve creating an environment model, explicitly incorporating an attacker model, as described in this section, builds on the notion of completeness discussed in Section 3. Not only does an explicit adversary model tell the analyst what an attacker can and cannot do, it also makes clear what actions have not even been considered. Thus, a component analyzed using an exclusively run-time, network-based adversary model like Dolev and Yao's must still be protected against design-time and non-network-based threats.

7 CONCLUSION AND FUTURE WORK

In this work, we have discussed how a new hazard analysis technique known as SAFE enables an integrated approach to safety and security analysis. A key contribution of this work was the argument that the Dolev-Yao model provides a unified view of safety and security "effects" that has important completeness and minimality properties as well as a rigorous semantic interpretation. We then justified the usefulness of SAFE's approach (i.e., its breadth of applicability) by showing how other causal taxonomies (guideword sets) [3, 6, 25] could be mapped to the Dolev-Yao concepts. In addition to our ongoing desire to apply the technique to new systems from varying domains, there are two exciting directions for next steps.

First, there are a large number of models and guidewords available in the academic, system engineering, and standardization communities. Ultimately, which models and concepts a particular analysis should use will be system and domain specific, though we believe academia, industry, and government/regulatory/standardization authorities can provide guidance. This guidance would ideally be deeply integrated with the tooling and techniques used, an approach outlined by Procter and Hatcliff [17].

Second, we are very interested in exploring ways to move the system safety and security community closer to the formal methods community. Though both groups have at a high level the same goals, they use different definitions for common terms (e.g., safety as freedom from loss versus avoidance of a particular state) and operate at different levels of abstraction. We believe that potential connections between the communities, such as assumption synthesis (Section 5.1.4), are important opportunities for collaboration.

ACKNOWLEDGMENTS

This work was partially supported by NSF grants 1441170 and 1565544 and the US FDA Scholar-in-Residence Program. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. DM-0004639.

REFERENCES

[1] D. Arney, S. Fischmeister, J. M. Goldman, I. Lee, and R. Trausmuth. Plug-and-play for medical devices: Experiences from a case study. *Biomedical Instrumentation & Technology*, 43(4):313–317, 2009.
[2] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *ICCPS*, 2010.
[3] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 1(1):11–33, 2004.
[4] D. Challener. Trusted platform module. In *Encyclopedia of Cryptography and Security*, pages 1308–1310. Springer US, 2011.
[5] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
[6] I. Friedberg, K. McLaughlin, P. Smith, D. Laverty, and S. Sezer. STPA-SafeSec: Safety and security analysis for cyber-physical systems. *Journal of Information Security and Applications*, 2016.
[7] J. Hatcliff, A. King, I. Lee, A. MacDonald, A. Fernando, M. Robkin, E. Vasserman, S. Weininger, and J. M. Goldman. Rationale and architecture principles for medical application platforms. In *ICCPS*, 2012.
[8] J. Hatcliff, E. Vasserman, S. Weininger, and J. Goldman. An overview of regulatory and trust issues for the integrated clinical environment. In *HCMDSS*, 2011.
[9] J. Hatcliff, A. Wassyng, T. Kelly, C. Comar, and P. Jones. Certifiably safe software-dependent systems: challenges and directions. In *Future of Software Engineering Conference*, 2014.
[10] J. Herzog. A computational interpretation of Dolev-Yao adversaries. *Theoretical Computer Science*, 340(1):57–81, 2005.
[11] R. W. Hicks, V. Sikirica, W. Nelson, J. R. Schein, and D. D. Cousins. Medication errors involving patient-controlled analgesia. *American Journal of Health-System Pharmacy*, 65(5):429–440, 2008.
[12] N. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety.* MIT Press, 2011.
[13] N. G. Leveson. *Safeware: System safety and Computers.* Addison-Wesley Publishing Company, Inc., 1995.
[14] R. Pelánek. Fighting state space explosion: Review and evaluation. In *FMICS*, 2009.
[15] C. Ponikwar, H. Hof, S. Gopinath, and L. Wischhof. Beyond the Dolev-Yao model: Realistic application-specific attacker models for applications using vehicular communication. In *SECURWARE*, 2016.
[16] S. Procter. *A Development and Assurance Process for Medical Application Platform Apps.* PhD thesis, Kansas State University, 2016.
[17] S. Procter, J. Hatcliff, S. Weininger, and A. Fernando. Error type refinement for assurance of families of platform-based systems. In *SAFECOMP*, 2015.
[18] J. Rushby. Composing safe systems. In *FACS*, 2012.
[19] SAE AS-2C Architecture Description Language Subcommittee. SAE Architecture Analysis and Design Language (AADL) Annex Volume 3: Annex E: Error Model Language. Technical report, SAE Aerospace, June 2014.
[20] C. Salazar. A security architecture for medical application platforms. Master's thesis, Kansas State University, 2014.
[21] C. Schmittner, Z. Ma, and P. Puschner. Limitation and improvement of STPA-Sec for safety and security co-analysis. In *SAFECOMP Workshops*, 2016.
[22] S. Sheard, M. Konrad, C. Weinstock, and W. Nichols. Definition and measurement of complexity in the context of safety assurance. Technical Report CMU/SEI-2016-TR-013, Software Engineering Institute, Carnegie Mellon University, 2016.
[23] J. Thomas and N. Leveson. Performing hazard analysis on complex, software-and human-intensive systems. In *ISSC Conference about System Safety*, 2011.
[24] C. J. Walter and N. Suri. The customizable fault/error model for dependable distributed systems. *Theoretical Computer Science*, 290(2):1223–1251, 2003.
[25] W. Young and N. Leveson. Systems thinking for safety and security. In *Annual Computer Security Applications Conference*, 2013.
[26] W. Young and N. G. Leveson. An integrated approach to safety and security based on systems theory. *Communications of the ACM*, 57(2):31–35, 2014.