Eliminating Inter-Domain Vulnerabilities in Cyber-Physical Systems: An Analysis Contracts Approach

Ivan Ruchkin
Inst. for Software Research
Carnegie Mellon University
Pittsburgh, PA, USA
iruchkin@cs.cmu.edu

Ashwini Rao
Inst. for Software Research
Carnegie Mellon University
Pittsburgh, PA, USA
agrao@cs.cmu.edu

Dionisio De Niz
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
dionisio@sei.cmu.edu

Sagar Chaki
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
chaki@sei.cmu.edu

David Garlan
Inst. for Software Research
Carnegie Mellon University
Pittsburgh, PA, USA
garlan@cs.cmu.edu

ABSTRACT

Designing secure cyber-physical systems (CPS) is a particularly difficult task since security vulnerabilities stem not only from traditional cybersecurity concerns, but also physical ones. Many of the standard methods for CPS design make strong and unverified assumptions about the trustworthiness of physical devices, such as sensors. When these assumptions are violated, subtle inter-domain vulnerabilities are introduced into the system model. In this paper we use formal specification of analysis contracts to expose security assumptions and guarantees of analyses from reliability, control, and sensor security domains. We show that this specification allows us to determine where these assumptions are violated, opening the door to malicious attacks. We demonstrate how this approach can help discover and prevent vulnerabilities using a self-driving car example.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*methodologies*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*real-time and embedded systems*

Keywords

Cyber-physical systems; analysis contracts; sensor; control

1. INTRODUCTION

High-quality cyber-physical systems (CPS) require the consideration of a broad range of system qualities. A substantial body of literature has proposed methods and tools to address traditional engineering qualities like performance of heterogeneous control [15] [52], correctness[11] [57], schedulability [35] [13], fault tolerance [59] [1], and safety [24] [51] [19]. This focus is partially due to the fact that control theory, theoretical computer science, and mechanical engineering have been seen as the "core" foundations for CPS research [2][40][53].

Security, however, has received relatively little attention as a systemic quality of CPS. As Lee put it in [38], "the term CPS is sometimes confused with 'cybersecurity,' which concerns the confidentiality, integrity and availability of data and has *no intrinsic connection with physical processes*."[1] Indeed, physical processes in CPS complicate reasoning because of the cross-cutting nature of security: sensors and actuators that interact with the physical world may contribute to a composite cyber-physical vulnerability. For example, security assurance for a car cannot be confined to its cyber part: the software relies on physical elements, which may be vulnerable to attacks in the physical realm, such as disabling its sensors [7]. As recent results show, a sensor failure can lead to a larger attack surface since the sensor set produces a larger proportion of compromised data [20]. These failures can lead to vulnerabilities in life-critical systems, which may be exploited with serious consequences [36].

A major barrier to achieving up-front systematic security engineering in CPS is incompatibility between traditional CPS engineering analyses and sensor security [53] [40]. One example of such analyses in reliability engineering is Failure Modes and Effects Analysis (FMEA) [58], which determines probable failure configurations ("modes") that can arise from component malfunction. For sensors, FMEA makes an implicit assumption that data received from a non-failed sensor is trustworthy, but does not model or verify this assumption [41]. Conversely, control safety analysis [20] typically considers data trustworthiness in the normal operation mode, but often ignores trustworthiness in the failure modes that are provided by FMEA. Thus, both of these forms of analysis make assumptions that may be inconsistent with, or ignore modes determined by, the other. Incompatibilities such as these invalidate the preconditions, and hence conclusions, of the logic behind analyses, resulting in a system design that is not secure across all modes.

CPS-SPC'15 October 16 2015, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2808705.2808714.

[1] Emphasis is ours.

CPS developed with incompatible analyses, such as those just mentioned, are vulnerable to attacks through what we call *inter-domain vulnerabilities* – vulnerabilities that arise on the boundary of engineering domains and analyses. State estimation methods for control implicitly assume that sensor configuration does not change over time, and that at least half of the sensors are trustworthy [20]. In practice, however, the available sensors may change during operation, e.g., a sensor can malfunction or fail entirely, become unavailable (e.g., GPS in a tunnel or lidar during rain and fog [33]), or be subverted by an attacker. In contrast, analyses like FMEA may consider scenarios in which the set of available sensors changes, breaking the sensor invariance assumption of other analyses. As a consequence, advanced control systems, such as adaptive cruise control, smart braking, and smart steering may have vulnerabilities that can be exploited.

In this paper we propose a design-time approach to eliminate inter-domain vulnerabilities by systematically embedding security analyses and assumptions into the CPS engineering process. This approach builds on the *analysis contracts framework* that has been validated on the domains of thread scheduling and electrothermal battery design [56]. An *analysis contract* is a formal specification of an analysis interface consisting of inputs, outputs, assumptions, and guarantees. Verification using contracts can detect situations in which an analysis produces an unsound result or violates an assumption of another analysis, thus introducing a bug or a vulnerability. Here we extend this prior work by demonstrating how the use of analysis contracts for sensor trustworthiness analysis, FMEA, and control safety analysis can lead to a more secure CPS design.

More specifically, this paper makes four contributions:

- A description of interactions and dependencies among three analysis domains: reliability, sensor security, and control. These interactions could lead to a system failure via successful exploitation of a vulnerability.
- A formal specification of these dependencies and interactions in the form of deterministic and probabilistic analysis contracts.
- An algorithm for verification of deterministic assumptions for sensor trustworthiness, FMEA, and control safety and the specified contracts.
- Demonstration of the feasibility and utility of the analysis contracts approach on a self-driving car system model.

The rest of the paper is organized as follows. The next section gives the necessary background on the domains and tools we use in this paper, as well as a brief overview of related work. Section 3 explains the vulnerability and multi-domain attack in detail. Then, in Section 4 we present our approach of specifying and verifying analysis contracts. Finally Sections 5 and 6 discuss respectively the limitations and implications of the analysis contracts approach, thus concluding the paper.

2. BACKGROUND AND RELATED WORK

This section describes the domains of security, reliability, and control, whose cross-domain interaction can lead to security vulnerabilities. It also presents prior work on modeling and verification via analysis contracts that we build upon, as well as related approaches that address similar inter-domain issues.

2.1 Sensor Security

Sensors enable the exchange between cyber and physical worlds: they interact with the physical world and provide inputs to the controller. By compromising sensors, an attacker can send erroneous inputs to the controller, which depends on sensors to estimate the state of the system. Malicious inputs from sensors can result in deception attacks, which compromise the integrity of the cyber-physical system [5].

Cyber-physical systems can have different types of sensors for measuring variables through physical channels [60]. For example, a car can have sensors for measuring distance (lidar or sonar [32]), velocity (a magnetic speedometer), and tire pressure. In order to handle faults and malfunctions in sensors, CPS can use different technologies to measure the same variable. For example, cars can use Sonar in addition to Lidar to measure distance, since Lidar can fail under foggy conditions [33].

By targeting different types of sensors, an attacker can cause specific types of failures. For instance, by sensing incorrect distance readings, an attacker could compromise braking functionality [36]. The placement of sensors (inside versus outside the car), communication mechanisms (physical network access versus remote access via WiFi), and other aspects, affect the ease with which an attacker can compromise the sensors [7].

In addition to malicious inputs, sensor inputs can be unreliable due to noise, packet loss, and faults [50]. CPS control algorithms either make assumptions about reliability or trustworthiness of sensor inputs, or incorporate mechanisms like filters and decoders to prevent, detect and tolerate unreliable sensor inputs [5]. For instance, consensus algorithms used require at least some number of sensor inputs to be reliable [5], and compressed sensing algorithms require approximately half of sensors to be trustworthy [20]. To evaluate trustworthiness of sensor input data one may use methods from literature on wireless sensor networks [41] [62].

2.2 Reliability and Fault Tolerance

Embedded and safety-critical systems have a long tradition of designs that survive random mechanical and hardware faults due to manufacturing imperfections and random events, such as cosmic rays [59]. This field is largely motivated by the imperfect reality of the material world, thus situated towards the physical side of CPS. A major design-time technique to achieve higher fault-tolerance is *redundancy* – adding functionally identical components in order to preserve system's operation in case one of the components fails.

One of the well-established analytic operations in reliability engineering is Failure Modes and Effects Analysis (FMEA) [58]. The goal of this analysis is to evaluate the system's reliability in terms of the impact ("effects") that failing components have on other components and the whole system. Such evaluation often presupposes random independent failures, such as mechanical malfunctions or hardware defects, in order to stochastically investigate the most likely failure states of the system (also known as *modes*). Sometimes FMEA is applied manually as a design process [6], but over the last two decades multiple tools have emerged to fully automate FMEA [28] [66].

For this work we consider a generalized version of FMEA that not only calculates failure modes and their probabilities, but also adds cost-constrained redundancy in sensors and controllers to reduce failure probability to an acceptable domain-specific value. This analysis can be seen as an abstraction of a semi-automated design process that arrives at a sufficiently redundant and acceptably cheap architecture.

2.3 CPS Control

Control engineering focuses on designing an algorithm to impose actuation on a system, state of which is being monitored, in order to bring the system to a desired state [49]. Control design is often model-based where the plant (the system and environment under control) and the controller (the algorithm) are represented as state transfer functions. For complex systems control engineering typically includes extensive simulation of the system with mixed qualitative and quantitative judgment, using tools like MATLAB/Simulink [9]. Smaller systems can be analyzed with more theoretical and stronger-guarantee approaches such as Lyapunov functions [26].

Regardless of what kind of analysis is done on a control system, this analysis needs to consider many design parameters such as the system equations, type of controller (reactive, predictive, adaptive), control gains, and control performance requirements (rise time, time-to-peak, settling time, and percent overshoot) [8]. For this paper we adopt a black-box view on these parameters, and represent them as a single control safety analysis with inputs and outputs. The goal of such analysis is to ensure that the controller meets the requirements given the system model.

Applying the classic control methods to cyber-physical systems faces a number of obstacles. The obstacles include the uncertainty of the environment [53], timing of computations (which is often abstracted out of control models) [39], and security that can be compromised through sensors and actuators [5]. Overcoming these obstacles often leads to challenging integration with other modeling approaches, such as state machines and hybrid systems [4]. This paper takes steps towards this integration with reliability and security domains.

Recent work on secure CPS control addresses sensor and actuator security for various domains (e.g., smart grids [45]) and types of attacks (e.g., replay attacks [44]). One of important results is a set of robust state estimation algorithms that have theoretical guarantees in face of sensor attacks such as spoofing and replay [20] [50]. We build upon this body of research in our paper, and specify sensor trustworthiness assumptions on top of which this work is built.

2.4 Architectural Modeling in AADL

The *Architecture Analysis and Design Language (AADL)* [21] is a Society of Automotive Engineers standard aimed at describing the software and hardware architecture of real-time embedded systems. AADL provides constructs focused on describing the runtime software architecture in terms of processes, threads, and data, and the executing platform in terms of processors, networks, memory storage units, and so on, and their relationship based on binding properties. AADL is designed to host the multiple analysis algorithms used to verify different critical properties of embedded real-time systems and CPS in general. These properties include timing requirements (e.g., inflation of an airbag within 0.1 seconds), logical correctness (e.g., absence of deadlocks), thermal safety (e.g., no overheating), fault tolerance (e.g., tolerate failure of one processor), and many others.

To support the ever-increasing number of analysis algorithms used in CPS, AADL allows the definition of sub-languages in the form of an *annex* and the corresponding compiler submodule. An annex allows the designer to add detailed descriptions to part of the model to be used in a particular analysis. For instance, the Behavioral Annex [22] allows a component's detailed discrete-state behavior to be analyzed by model checkers. Annexes are a powerful extension mechanism that allows AADL to become the lingua franca of model-based engineering research with an increasing acceptance in the industry.

Another important feature of AADL is a *mode* – a system configuration with components, connections, and values of properties. For example, a car may be in cruise control or manual mode, which determines whether or not the cruise controller actuates the accelerator. AADL modes allow specification of discrete switching behavior that is formally equivalent to timed abstract state machines [65]. Modes have been a feature of architectural languages since the MetaH language [64], and AADL unites other advanced features with modes to enable expressive and flexible system modeling. In this paper we will use modes to represent failure configurations of a system, e.g., if a sensor is malfunctioning.

2.5 Analysis Contracts Approach

The capacity of AADL to host an unlimited number of analysis algorithms with custom annexes has positioned it as a powerful tool to tackle the heterogeneity of CPS engineering. Unfortunately, these algorithms are traditionally developed within a specific scientific domain, making implicit assumptions and creating specialized abstractions that ignore potential interactions with other analyses. As a result, analyses may contradict each other's assumptions about the system, thus invalidating their own outputs. To deal with this problem we developed the *analysis contracts* verification framework [46] [56] that enables the description of the contracts between analysis and the system model in the form of inputs, outputs, assumptions, and guarantees. These specifications are described in the contracts annex with formalized syntax and semantics. The ACTIVE toolset was developed to support automated analytic integration [55].

To define an analysis contract we first need to describe the formal structure behind a set of domains, such as reliability and control. Each domain needs to capture the semantics in which the effects of the interacting analyses can be automatically verified. Our prior work incorporated a number of special verification tools: Spin for Promela language [30] and Z3 for *Satisfiability Modulo Theories* (SMT) v2 language [12]. In this work the contract language was composed of a first-order and linear temporal logic fragments. We utilize the former in this paper and explore the possible second-order and probabilistic extensions.

2.6 Related Work

There is a growing body of literature on integrating heterogeneous models and domains at runtime. For example, in [18] the authors present a model-based methodology for designing a car's control system. Such methodologies, implemented in frameworks like OpenMETA [61] and METROII [10], integrate a set of models through formal relations of abstraction, transformation, and composition, typically providing strong theoretical guarantees. However, these guarantees often do not extend beyond the traditional concerns such as correctness and safety. In particular to embed such a cross-cutting concern as security into these methodologies, one would likely have to change almost all modeling formalisms, which is not feasible or scalable.

Assume-guarantee reasoning originates in Hoare's logic [29] and is widely used today in component-based modeling for CPS [57]. Multiple methodologies and frameworks associate contracts with components and strive to demonstrate system-wide guarantees given local component contracts [3] [47] [48]. Unfortunately, most security concerns cannot be confined to a single component or subsystem, and propagate across most components' contracts [5]. Such global security specification takes away the compositional power of contracts, and often leads to the state explosion in verification [42]. In contrast, analysis contracts change the perspective to the algorithms that change and verify the model, creating opportunities to specify security concerns that cannot be associated with any particular component.

3. INTER-DOMAIN VULNERABILITIES

In this section we describe a realistic example of an inter-domain vulnerability that can occur in cyber-physical systems. We consider the example of a self-driving car equipped with sensors for braking functionality. We explain the interdependencies between analyses at design time that can result in vulnerabilities, and adversary models and attacks that can exploit such vulnerabilities at runtime.

Figure 1: An autonomous car driving behind a leading car uses its distance and velocity sensors to make a braking decision.

3.1 Scenario Description

Consider a braking scenario for self-driving cars. Figure 1 shows two cars traveling in the same direction. The follower car is equipped with adaptive cruise control. The leading car is about to stop, and the follower needs to make a decision: at what point and how hard to actuate the brakes. The decision to brake is based on a number of sensors that estimate velocity and position relative to the leading car. This decision is critical: most mistakes can endanger lives.

The autonomous car systems in Figure 1 use velocity and distance sensors for braking. Two distance sensors each use a different technology to measure distance: a lidar for laser ranging and a car-to-car (C2C) communication network [2] to exchange position information. Further, the lidar is internal to the car, and the network be accessed from the outside. There are two velocity sensors, and each uses a different technology to measure velocity: a GPS and a traditional magnetic speedometer. The speedometer is inside the car, and the GPS is outside. Table 1 shows the sensed variable, technology, and placement for the distance and velocity sensors in self-driving cars.

Sensor variable	Technology	Placement
Distance	Lidar	Internal
Distance	C2C	External
Velocity	Speedometer	Internal
Velocity	GPS	External

Table 1: Sensor type, technology and placement

The sensors send data to the braking controller through the CAN (Controller Area Network) bus. Based on this data, the controller decides the moment and power of braking at each periodic execution. Since the controller has no perception of the physical world except through the sensors, it is important to know which sensors are more trustworthy than others. This is indeed another important sensor parameter, known as *trustworthiness* [41], that indicates whether a sensor can be compromised by an attacker. A sensor's trustworthiness must be considered within the context of an adversary model.

3.2 Adversary Model

Our adversary model describes assumptions behind attackers and parts of a self-driving car that can be attacked. For example, an adversary could attack the sensors, actuators, controllers, or communication networks. In our example, we limit our adversarial model to attacks on the sensors, but not other components, thus assuming that other components are trustworthy. Note that we make this assumption to illustrate inter-domain vulnerabilities, and analyses for general component trustworthiness complement our approach, but are outside the scope of this paper.

We consider several adversary profiles. First, a powerful adversary can attack any sensor, both internal and external. One known case of such an adversary is one that has CAN bus access [36]. By forging CAN packets, the attacker can trivially cause system failures. However, full internal network access is not always a realistic assumption for a moving vehicle.

The two other profiles are more realistic: these adversaries are less powerful, but intelligently manage to exploit a vulnerability using limited resources. We make several assumptions about these adversaries. They have a technical capability to get information about the structure, properties (such as in Table 1), and operation of system components by exploring similar systems. For example, an adversary knows that a Lidar sensor does not work in the presence of fog. A realistic adversary can gain such system knowledge by either examining a target system or obtaining such information from third parties. We assume that the adversary does not have the computational capabilities to break strong cryptographic security measures, e.g., encryption. An adversary can attack sensors in any order, and we do not make any limiting assumptions about duration of attacks.

The profiles we consider are the attackers of sensors external to the car (external adversary) or sensors internal to the car (internal adversary). The external adversary can attack the sensors via physical channels, such as infrared [63] or short-range wireless [7]. In contrast, the internal adversary has access to various devices like a radio, USB reader, or speedometer.

[2] www.car-2-car.org

Sensor	Available in mode			
	nominal	fail 1 (fog)	fail 2	fail 3
Lidar	✓	✗	✓	✓
C2C	✓	✓	✗	✗
Speedometer	✓	✓	✓	✗
GPS	✓	✓	✓	✓

Table 2: Configurations output by the FMEA analysis. ✓ indicates that the sensor is functioning properly. ✗ indicates that the sensor is malfunctioning and not providing data.

3.3 Analyses

To design the braking system for a self-driving car, engineers carry out several analyses at design time. We consider three analyses: FMEA analysis, sensor trustworthiness analysis, and control safety analysis. As typical in many CPS projects, these analyses are carried out by engineers from different domains who generally work independently from each other. When analyses from different domains are applied to the same system and make different assumptions, it can be difficult for engineers to coordinate and account for such assumptions.

3.3.1 Failure Modes and Effects Analysis

The goal of FMEA is to incorporate redundancy into the design to handle random failures. To achieve this, FMEA considers the probabilities of random sensor malfunction. It further assumes that failures of different sensors are independent. For example, in our scenario with distance and velocity sensors, FMEA could output the three configurations shown in Table 2. The nominal mode indicates the default situation when all sensors function properly. Consider the example of "Fail mode 1" configuration. FMEA outputs this configuration after considering foggy conditions. Since lidar may not work under foggy and rainy conditions, the configuration indicates that the lidar sensor may not function properly. The remaining sensors function properly. The system may have several probable failure modes depending on the technologies used. FMEA may also change the sensor set if the probability of random system failure is too high. We consider this situation in detail in Sec 4.

FMEA analysis in AADL uses the *Error Annex* [14], a standardized sublanguage, that textually defines error state machines where failure modes and recovery transitions are specified for each component. For example, a wireless network error model can have two states – nominal and failed – and change between them via transitions that have particular probabilities. In addition, error and recovery propagation patterns describing, for instance, how a processor failure propagates to networks, devices, and the software components that run on them are affected. FMEA then uses these descriptions to improve the system's reliability.

3.3.2 Sensor Trustworthiness Analysis

The sensor trustworthiness analysis determines whether a sensor can be compromised by an attacker. This analysis takes the following inputs: sensor placement (internal or external to the vehicle, connections to networks and controllers), technical characteristics (technology, communication protocol, encryption, manufacturer, and component version) and an adversary model (formulated in terms of possible actions on components). Note that, unlike FMEA, it does not consider the probabilities of sensor malfunction due to random failures. Instead, it takes into account that the probability of an adversary attacking two similar sensors is interdependent.

It is not our focus to develop trustworthiness evaluation methods, and existing design-time and run-time ones can be applied [43] [41]. Design-time methods can be applied directly to a model, and run-time methods can be used in a simulation, and the produced data can be used to infer trustworthiness. Our approach assumes that there exists an appropriate trustworthiness evaluation method and does not place significant limitations on it.

Table 3 shows the output of the trustworthiness analysis for three adversary models. In the case of a powerful adversary that can attack both external and internal sensors, trustworthiness analysis would determine that all four sensors in our scenario are not trustworthy. In the case of an adversary that can attack only external or internal sensors, it outputs that respectively only the external or sensors are not trustworthy.

3.3.3 Control Safety Analysis

Control safety analysis generally decides whether control is functionally correct, stable and meets the required performance level. As discussed in Section 2.3, control analysis needs to consider various control quality metrics. In braking controllers for autonomous vehicles it is important to find a balance between a smooth response that is acceptable to passengers and a sufficiently low rise time so that braking happens in time.

Similar to FMEA and trustworthiness analysis, control analysis makes assumptions regarding sensors. In [20] Fawzi et al. introduce an algorithm that it interprets data from potentially compromised sensors in order to estimate the system state. This algorithm assumes that at least half of the sensors are trustworthy; otherwise, it cannot estimate the state properly. This is an important security assumption required by the control analysis to evaluate the safety of controllers.

3.4 Exploiting Inter-Domain Vulnerabilities

Unsatisfied assumptions behind analyses can lead to vulnerabilities, which can be exploited by an adversary. In our scenario, control safety analysis makes an assumption that at least half of the sensors are sending trustworthy data. This assumption can be broken in two ways. The first one may happen at design-time when the most error-prone sensors are also the ones that are less trustworthy. In this case at design time FMEA will try to replicate these sensors to increase reliability, and simultaneously decrease the proportion of trustworthy (and not error-prone) sensors below 50%. As a result, the system's controller can be misled by its untrustworthy sensors, which provide more data than the trustworthy ones.

The second possibility for this assumption to be broken is at run time. Even if an external attacker isn't powerful enough to compromise all sensors in the nominal mode, it is possible to exploit the system when one of sensors is not available, e.g., due to fog. In foggy conditions, trustworthy lidar sensors are not available, and the control algorithm has to rely on an untrustworthy C2C network, which can be exploited to spoof distance readings with larger values.

Sensor	Placement	Powerful Adversary	External Adversary	Internal Adversary
Lidar	Internal	✗	✓	✗
C2C	External	✗	✗	✓
Speedometer	Internal	✗	✓	✗
GPS	External	✗	✗	✓

Table 3: Sensor trustworthiness for three adversary models.

Assuming it still has time to brake, the misled controller will miss the deadline for braking and potentially cause a crash. The cause of this vulnerability is that the controller assumption doesn't hold in all likely failure modes.

Table 4 illustrates an external adversary using the unsatisfied assumption failure modes to cause system failures in two out of four modes. In the nominal mode both distance and velocity sensors have the trustworthiness proportion of 50%. In fail mode 1 distance sensing is compromised because the only distance sensor C2C is untrustworthy. Fail mode 2 has the required proportion of trustworthy sensors. Fail mode 3 violates the assumption because the only available velocity GPS sensor is compromised.

To summarize, an external attacker who can attack only external sensors and is harmless in the nominal mode, is still capable of exploiting the vulnerability that comes from not considering failure modes. This example shows that inter-domain vulnerabilities may occur if analytic assumptions are unsatisfied. In the next section we address this problem in a more general way, logically identifying situations in which the assumptions can be violated.

4. ANALYSIS CONTRACTS APPROACH

In this section we present a detailed formalization of the self-driving car and its analyses to expose and eliminate inter-domain vulnerabilities.

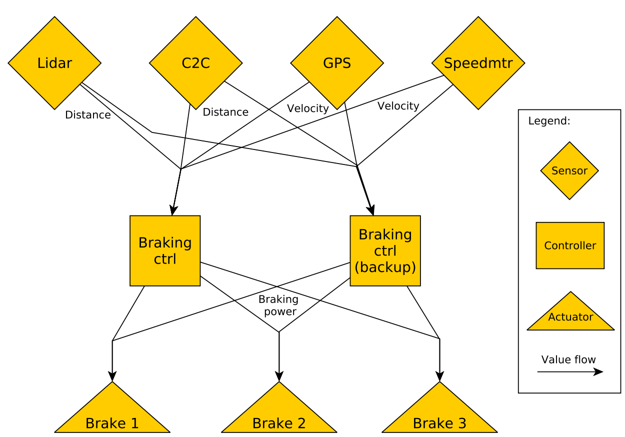

Figure 2: Braking architecture in a self-driving car.

4.1 System Model

We model the system (in Figure 2) using the AADL architecture description language (see Section 2.4 for background on AADL). We build our model upon a collision detection and avoidance model for an autonomous vehicle created by McGee et al. [16]. The original model contains a number of sensors, processing units (hardware devices and control threads), actuators, and other car components, organized into several functional subsystems: collision prediction/avoidance/response, networking, user interaction, and physical devices (various sensors, brakes, airbags, radio, and so on). We enhance this model by adding a lidar and C2C sensors for distance and a magnetic speedometer with GPS for velocity measurement [3]. Our modeling goal is to represent aspects that are relevant to inter-domain vulnerabilities.

The first step in modeling is to formalize the elements and properties of the automobile system that are relevant to the FMEA, safe control, and sensor trustworthiness analyses. AADL allows its users to define data types, component types, and new properties, and we use this flexibility to represent the aspects of the system that may lead to a vulnerability.

AADL modes encode the different configurations under the different failures of the system using state machines as described in Section 3.3.1. Mode examples are given in the rows of Table 2. Each mode m contains a full system architecture: sensors ($m.\mathbb{S}$), controllers ($m.\mathbb{R}$), and actuators [4]. In our prior work the analysis contracts methodology considered a single mode of the system [56], and we now extend it to several modes.

We specify AADL elements and properties as follows:

- Sensors $\mathbb{S}$ have the following properties:
  - Sensed variables $\mathsf{VarsS} \subseteq \mathbb{V}$: the variables for which the sensor can provide series of values. For example, a speedometer provides values for velocity. Some sensors may provide several variables, e.g., GPS values can be used to compute both the absolute position and distance to an obstacle.
  - Power status Pow (boolean values: $\mathbb{B} \equiv \{\top, \bot\}$): whether the sensor is turned on by the user or engineer.
  - Availability Avail ($\mathbb{B}$): whether the sensor is providing data. This property does not presuppose that the data is trustworthy or compromised.
  - Trustworthiness Trust ($\mathbb{B}$): whether the sensor can be compromised by the attacker and is sending untrustworthy data. We use this boolean abstraction of trust for demonstrating how a vulnerability is introduced. For sensors with $\mathsf{Trust} = \bot$ we assume that an attacker can compromise them in any quantity and at any point of time. Even with this relatively simple abstraction we showed an exploitation of vulnerability in Table 4. More sophisticated models may consider numeric

[3] Our AADL model with analysis contracts is available at `github.com/bisc/collision_detection_aadl`

[4] Actuators are critical components of the system, but we do not model them explicitly because our focus is on interaction between sensors and controllers.

Variable	Sensor	Trustworthiness	Available in mode nominal	fail 1 (fog)	fail 2	fail 3
Distance	Lidar	✓	✓	✗	✓	✓
Distance	C2C	✗	✓	✓	✗	✗
Velocity	Speedometer	✓	✓	✓	✓	✗
Velocity	GPS	✗	✓	✓	✓	✓
Control safety assumption			✓	✗	✓	✗

Table 4: External attacker exploiting inter-domain vulnerabilities.

or multidimensional trustworthiness [23] for more precise estimation of confidence in sensor data.

- Probability of mechanical failure $\mathsf{P_{fail}}$ (%): the probability of a sensor mechanically malfunctioning and remaining broken ($\mathsf{Avail} = \bot$) within a unit of operation time (e.g., an hour or a day).
- Sensor placement Place (internal or external): the sensor may be located on the outer perimeter of the car and facing outwards, or on the inside perimeter and not exposed to the outside world.

- Controllers $\mathbb{R}$ have the following properties[5]:
  - Required variables $\mathsf{VarsR} \subseteq \mathbb{V}$: the variables for which the controller should receive values from sensors. For example, the automated braking controller should receive velocity and distance to the closest obstacle on the course.
  - Power status Pow ($\mathbb{B}$): analogous to sensors, whether the controller is turned on by the user or engineer.
  - Availability Avail ($\mathbb{B}$): whether the controller is functioning and providing output to actuators. This property does not presuppose that the control is safe or uncompromised.
  - Safety of control $\mathsf{CtrlSafe}$ ($\mathbb{B}$): whether the controller meets the control performance, safety, and stability requirements.

- System modes $\mathbb{M}$ (i.e., different configurations) have the following properties:
  - Required fault-tolerance α_{fail} (%): the maximum acceptable probability of the system's random failure. The final design is expected to malfunction less or equally likely than α_{fail}.
  - Attacker model $\mathsf{AttackM}$ (internal or external): the type of the attacker considered in the system design. For simplicity, we consider only one dimension, that is, internal or external attacker. If required, we could model other dimensions such as local or remote attacker. Each attacker model contains a sensor vulnerability evaluation function $\mathsf{IsVuln} : \mathbb{S} \to \mathbb{B}$ that determines whether a particular sensor can be attacked by this attacker. This function abstracts out technical and operational aspects of attacks in order to represent the relationship between attackers and sensors. For example, the vulnerability function for a powerful adversary in Table 3 is $\mathsf{IsVuln} \equiv \top$.

Each property P is formally a function of the component set $\mathbb{S}$ that maps each component to a value in a set $\mathbb{T}$ of the property's type values, $P : \mathbb{S} \to \mathbb{T}$. Same applies to controller and mode properties. We will denote it in an object-oriented style: $Sonar.\mathsf{P_{fail}} = 0.01\%$ means that the sonar sensor has a probability of random failure equal to 0.01%.

AADL connections and ports describe how data flows between sensors $\mathbb{S}$ and actuators (located in the physical subsystem) through controllers $\mathbb{R}$ (located in other subsystems) by the means of the car's CAN bus. Although assumptions and guarantees can be formulated in terms of connections and ports, we do not use these elements in our contracts for this paper. Instead we encode the data flows between $\mathbb{S}$ and $\mathbb{R}$ in terms of sensed variables VarsS and required variables VarsR.

The described properties do not reference each other or depend on each other, so not every AADL instance is consistent: for instance, only powered sensors can provide data: $\forall s \in \mathbb{S} \cdot s.\mathsf{Avail} \implies s.\mathsf{Pow}$. Checking satisfaction of such conditions is a relatively well-explored problem and can be solved using constraint-based solving for every mode. Languages and tools for such problems had been previously developed for UML/OCL [17], Acme [25], AADL [31], and other architecture description languages.

We, on the contrary, investigate a more challenging problem: how to support analysis-driven change that preserves model consistency beyond constraints. It is important to verify each analysis operation and their order to assure that the resulting design is sound. To this end, it is essential to capture the interactions between analyses and the model, which we do in the next subsection.

4.2 Specification of Contracts

FMEA A_{fmea}. The goal of the FMEA analysis is to find a component redundancy structure [6] that is capable of withstanding the expected random failures of individual components and provide a system with a probability of failure no larger than α_{fail}. Hence one output of FMEA is the selection of sensors and controllers.

Another output of FMEA is a set of likely[7] failure modes. The output will contain failure modes (i.e., system configu-

[5]Although controllers are physical elements and can be attacked, in this paper we focus on sensor attacks and assume that direct controller attacks do not happen. Since controllers are typically not exposed to the physical world, their attacks would require an access to the internal car network, leading to a powerful attacker and trivial security analysis.

[6]This analysis is constrained by cost (in terms of funds and available space) of components: the trivial solution of replicating each sensor a large number of times would typically not be acceptable.

[7]The definition of likelihood for failure modes may differ depending on the system requirements. For example, one may consider failure modes with probabilities $\geq 0.1\alpha_{fail}$.

rations with some sensors $\mathsf{Avail} = \perp$) that need to be considered for the system to be safe.

A typical FMEA assumption is that the random mechanical failures are independent among all of the system's components. That is, a failure of one sensor does not increase the probability of another sensor failing. This assumption allows for simpler reasoning about failure propagation and failure modes during the analysis. Since the probabilities of failure are usually generalized from empirical data, we add a correlation tolerance bound $\epsilon_{fail} > 0$ to the assumption.

A guarantee of FMEA is that the controllers have all the required variable series to actuate the system. This guarantee does not ensure the full correctness of the analysis (the system may still not be fault-tolerant), but it allows to verify that the analysis has not rendered the system non-functional.

Thus we arrive at the contract for A_{fmea}:

- Inputs: $\mathsf{P_{fail}}$, α_{fail}.
- Outputs: $\mathbb{S}$, $\mathbb{R}$, $\mathbb{M}$.
- Assumption. *Component failure independence* – if one component fails, another component is not more likely to fail:

$$\forall c_1, c_2 \in \mathbb{S} \cup \mathbb{R} \cdot P(\neg c_1.\mathsf{Avail} \mid \neg c_2.\mathsf{Avail}) \leq P(\neg c_1.\mathsf{Avail}) + \epsilon_{fail}$$

- Guarantee. *Functioning controllers* – some sensor provides each variable that some controller expects:

$$\forall m \in \mathbb{M} \cdot \forall c \in m.\mathbb{R} \cdot \forall v \in c.\mathsf{VarsR} \cdot \exists s \in m.\mathbb{S} \cdot v \in s.\mathsf{VarsS}.$$

Sensor trustworthiness A_{trust}. In terms of our model this analysis determines the possibility of each sensor being compromised (which we represent with boolean Trust) given their placement, power status, availability, and the attacker model. To avoid ambiguity we assume that unpowered and unavailable sensors cannot be compromised. Therefore A_{trust} marks a sensor as untrustworthy if and only if the sensor is powered, available, and vulnerable for the given attacker model:

$$\forall s \in \mathbb{S} : \neg \mathsf{A}_{trust}(s) \iff s.\mathsf{Pow} \wedge s.\mathsf{Avail} \wedge \mathsf{AttackM.IsVuln}(s).$$

The sensor trustworthiness analysis views failures fundamentally differently from FMEA. It is expected that some sensors may go out of order together because of a coordinated physical attack or an adverse environment like fog. This leads to the failure dependence assumption with an error bound $\epsilon_{trust} > 0$. While not being a direct negation of FMEA's assumption, failure dependence makes analysis applicable in a different scope of designs. Whether the analyses can be applied together on the same system depends on calibration of the error bound parameters ϵ_{fail} and ϵ_{trust}.

The correctness of the sensor trustworthiness analysis can be expressed declaratively: untrustworthy sensors are the ones that can be attacked by the selected attacker model. We put this statement in the contract as a guarantee to create a sanity check on the analysis implementation, which may contain unknown bugs.

Given the above, we specify the contract for A_{trust}:

- Inputs: $\mathbb{S}$, Place, Pow, Avail, $\mathsf{AttackM}$.
- Output: Trust.
- Assumption. *Component failure dependence* – some components are likely to fail together:

$$\exists c_1, c_2 \in \mathbb{S} \cup \mathbb{R} : P(\neg c_1.\mathsf{Avail} \mid \neg c_2.\mathsf{Avail}) \geq P(\neg c_1.\mathsf{Avail}) - \epsilon_{trust}$$

- Guarantee. *Correct trustworthiness assignment* – a sensor is not trustworthy if and only if it is vulnerable for the considered attacker model:

$$\forall m \in \mathbb{M}, s \in m.\mathbb{S} \cdot s.\mathsf{Trust} = \perp \iff m.\mathsf{AttackM.IsVuln}(s).$$

Control safety A_{ctrl}. This analysis determines whether the control has a required performance, is stable and robust(or, in short, *safe*). We abstract away the details of this analysis and specify that it requires the control model (sensors, controllers, actuators and their variables) and outputs whether the control is safe. More details can be added as necessary for more refined contracts.

A common feedback controller architecture includes a state estimator (e.g., a Kalman filter or a decoder) and a control algorithm, such as PID. A decoder is used to estimate the genuine system state when an attacker may have falsified some sensor data. According to Propositions 2 and 3 in [20], it is required that at least half of sensors that sense the same variable are trustworthy. Otherwise a decoder cannot discover or correct an intentional sensor attack, leading to the system being compromised. Powered off and unavailable sensors are considered trustworthy, but do not contribute to the trustworthiness estimate.

We specify the assumption about a half of sensors being trustworthy by establishing a mapping function f (for each variable) between trustworthy and untrustworthy sensors. Existence and surjectivity[8] of f mean that for each untrustworthy sensor there exists at least one unique trustworthy sensor. That existence is equivalent to the proportion of trustworthy sensors being at least 50%.

We thus arrive at the following contract for A_{ctrl}:

- Inputs: $\mathbb{S}$, VarsS, $\mathbb{R}$, VarsR.
- Output: $\mathsf{CtrlSafe}$.
- Assumption. *Minimal sensor trust* – for each untrusted sensor there is at least one different trusted sensor [9]:

$$\begin{aligned} &\forall m \in \mathbb{M}\ \forall c \in m.\mathbb{R}, v \in c.\mathsf{VarsR} \cdot \\ &\quad \exists f :\ \mathbb{S} \to \mathbb{S}\ \cdot \forall s_u \in m.\mathbb{S} \cdot \\ &\qquad v \in s_u.\mathsf{VarsS} \wedge s_u.\mathsf{Trust} = \perp \implies \\ &\qquad \exists s_t \in m.\mathbb{S} \cdot v \in s_t.\mathsf{VarsS} \wedge s_t.\mathsf{Trust} = \top \wedge f(s_t) = s_u. \end{aligned}$$

- Guarantee: none.

[8] A *surjective* function covers its full range of values.

[9] This assumption can be written in a simpler form, "at least half of the sensors are trustworthy": $\forall m \in \mathbb{M} \cdot |m.S_{trustworthy}|/|m.\mathbb{S}| \geq 0.5$. Unfortunately such statements cannot be verified in classic SMT, and theories with set cardinalities have not been implemented for SMT yet.

This concludes the specification of analysis contracts. We remind the reader that the ultimate design goal is to apply the analyses in a way that guarantees that the sensors trustworthiness is adequate for the considered attacker model ($s.\mathsf{Trust} = \bot \iff \mathsf{AttackM.IsVuln}(s)$), the system's control is safe ($\mathsf{CtrlSafe} = \top$), and that the system's failure probability is not greater than α_{fail}. In the next subsection we demonstrate how we achieve this goal.

4.3 Contract verification

We first discuss the dependency resolution between analyses. After that we separately describe verification of three types of contracts: logical statements within first-order SMT, logical statements beyond first-order SMT, and probabilistic statements.

4.3.1 Dependency Resolution

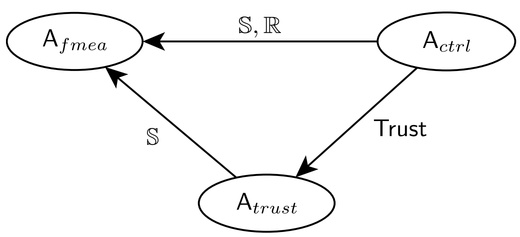

Figure 3: Dependencies of analyses.

As it follows from the contracts, the analyses under consideration have the following input-output dependencies (see Figure 3 for illustration):

- A_{fmea} does not depend on any analyses considered in this paper.
- A_{trust} depends on A_{fmea} that outputs $\mathbb{S}$ – an input for A_{trust}.
- A_{ctrl} depends on A_{fmea} that outputs $\mathbb{S}$ and $\mathbb{R}$ – inputs for A_{ctrl}.
- A_{ctrl} depends on A_{trust} that outputs Trust – part of an assumption for A_{ctrl}.

We execute the sequencing algorithm in the ACTIVE tool [55] to determine these dependencies and sequence the analyses in a way that respects the dependencies [56]. For example, if a user changes $\mathsf{AttackM}$ and tries to execute A_{ctrl}, A_{trust} is executed first so that the assumption of A_{ctrl} is verified on values of Trust that are consistent with $\mathsf{AttackM}$. Moreover, before A_{trust} is executed, A_{fmea} is executed since A_{trust} (and A_{ctrl} as well) depends on it as well.

4.3.2 Deterministic Contracts

We have verified some deterministic logical contracts in this paper with an existing algorithmic solution. Specifically, we have the capacity to automatically translate deterministic contracts written using only *first-order quantification* over variables in bounded sets into SMT programs and verify them using the Z3 SMT solver with the existing implementation of the ACTIVE tool [55]. We expect many, although not all, contracts in practice to be expressible with currently verifiable first-order statements. Quantification over unbounded (e.g., integers) or uncountable (e.g., reals) sets may lead to poorly verifiable statements, and so far we have been able to avoid such quantification.

Among this paper's contracts, the guarantees of A_{fmea} and A_{trust} are first-order statements verifiable in our toolchain because they quantify over bounded sets $\mathbb{M}$, $\mathbb{S}$, and $\mathbb{R}$. We verified them on our example system model and found no violations because the model satisfies these guarantees.

Second-order quantification means quantifying over functions, such as the sensor mapping function f in the assumption of A_{ctrl}. Such statements can also be translated directly to SMT programs. However, ACTIVE does not yet handle this translation, so we did not verify that assumption and leave this for future work. Nevertheless, if the quantified functions have bounded domain and range sets, these statements are decidable by existing SMT solvers. Thus, integrating second-order verification into ACTIVE is possible and requires three steps: (i) incorporating second-order clauses into the contract language syntax, (ii) defining these clauses' semantics, and (iii) augmenting the implementation of the ACTIVE VERIFIER – a module of ACTIVE that translates contracts into SMT and manages their verification. Once the second-order quantification is implemented in ACTIVE, we will be able to compute satisfaction of the control safety assumption in Table 4 fully automatically by invoking ACTIVE VERIFIER on the assumption and the current system model.

To summarize, for deterministic contracts in this paper we specified and verified first-order statements in contracts, specified a second-order statement, and identified a path to implementing second-order verification.

4.3.3 Probabilistic Contracts

The assumptions of A_{fmea} and A_{trust} are specified in terms of probabilities of events like a sensor being unavailable. The probabilistic specification is convenient to capture statements that go beyond boolean logic, which happens often in domains related to rare or uncertain events and behaviors. Fault tolerance, cryptography, and wireless ad hoc networks are examples of such domains.

Our analysis contracts methodology is relatively undeveloped in terms of probabilistic contracts. Although we can express them on case-by-case basis, there is no unified understanding of their syntax and semantics. In terms of semantics a big challenge is finding an appropriate model for probabilistic statements. Finding an axiomatic logical interpretation such as SMT is not practical unless a contract leads to a contradiction via theorems without considering the actual distributions, which is not a general case. Therefore we need to go beyond SMT in this verification.

One approach is mapping an AADL model into a probabilistic semantic space. This would entail firstly incorporating some probabilistic logic like PCTL (Probabilistic Computation Tree Logic) [27] into the contracts language. Secondly and most importantly, one would need to create a AADL-based probabilistic state space models with such formalisms as Markov chains or Markov rewards [54]. The role of these models would be to capture the behavior in a certain domain or subsystem, as we did with Promela models in [56]. Whether such models can be generalized beyond a single domain is another open research question. Finally, probabilistic model checking tools like PRISM [37] or MRMC (Markov Reward Model Checker) [34] needs to be integrated with the verification algorithms in ACTIVE VERIFIER.

A less general alternative is building a custom verification solution for specific domains and contracts. For example, one could implement an algorithm in a general-purpose programming language to verify the assumptions of A_{fmea} and A_{trust}. This method would not provide the guarantees and generality of model-based approaches. However, it may be more practical in case general solutions are not scalable or even feasible. To summarize, the investigation of verification for probabilistic contracts is a major direction that we envision for future work.

5. LIMITATIONS

Formal methods face several threats in terms of practical adoption. Analysis contracts capture interactions between analyses using formal logic and rely upon automated verifiers. Both require up-front effort in building the formal methods expertise and tools for their verification. However, formal specification and verification are successfully used in domains, for example avionics, where the cost of ensuring safety and security of human lives justifies the additional effort. Hence, the task of carrying out the contracts methodology can be assigned to a dedicated team of integration engineers to overcome the obstacles of practical adoption.

Another open question is the generality of our framework. Although we used analysis contracts only for a number of representative analyses, we believe that the approach is generally applicable to other analyses in security and reliability domains because it is common for reasoning to rely on fixed assumptions about how secure or reliable parts of system are. It is, however, possible that some domains do not reason in terms of analyses, and in those cases our framework would not be applicable. We have not discovered such domains yet. The practical applicability of our implementation is limited to the available verification algorithms in ACTIVE. Currently these exist for first-order logical statements, and we plan to extend the ACTIVE verification toolset.

There are also several technical challenges to the analysis contracts approach. Scalability of verification can be an issue depending on the type of contract and model involved. In our prior work, we illustrated the viability of our approach for moderate-size behavioral problems [56]. Expressiveness of the contracts relies on the logical theories and tools we employ, so absence of theories may be a roadblock. One such instance is the assumption of A_{ctrl} that could have been expressible in SMT, if not for the lack of operators for set cardinality or array counting. We could incorporate more general theories to enhance expressiveness. However, increasing expressiveness is associated with additional challenges such as decidability. For example, first-order predicate logic is decidable with quantification over bounded sets, but not over unbounded sets. Hence, we have to carefully balance expressiveness with feasibility and decidability. Lastly, as we continue to evaluate our approach on other domains, we may uncover additional challenges to the contracts methodology.

6. DISCUSSION AND CONCLUSIONS

The goal of this work was to improve multi-domain security engineering of cyber-physical systems. We presented an application of the analysis contracts methodology to representative analyses from domains of reliability, sensor security, and control. In particular, we formally specified domain interdependencies and assumptions that lead to vulnerabilities and gave a scenario of their exploitation. Towards detecting and preventing such vulnerabilities, we employed our methodology to specify and verify implicit assumptions and dependencies between analyses. We thus described how analysis contracts are used to expose and eliminate inter-domain vulnerabilities. Our next step is implementing second-order and probabilistic verification to fully automate the workflow, as discussed in Section 4.3.2 and 4.3.3. Our work has also exposed several intriguing longer-term research directions in CPS modeling and verification.

One interesting direction for future work is extending the described analyses towards richer contracts. A control assumption that we did not consider is invariance of the set of attacked nodes and the attacker model during runtime. In [20] Fawzi et al. "assume [...] that the set of attacked nodes [i.e., sensors] does not change over time." In practice, this is a fairly limiting assumption for CPS like self-driving cars that move through a highly dynamic environment. To verify this assumption, we need to model factors that change sensors over time (attacks, failures), as well as an evolutionary attacker model that may react to the system's responses.

Another future work opportunity is incorporating more analyses from the domains of this paper. So far, we have explored the control analysis for decoding potentially untrustworthy sensor readings. Other possibilities are stealth attack detection [45] or robustness for systems with noise [50]. Incorporating these analyses would allow us to move beyond the black-box approach to control analyses, thus improving the depth and quality of verification.

Finally, one can integrate other domains with the ones in this paper. For example, one may use hybrid programs [51] to analyze safety of braking behavior. But what are the theoretical guarantees if some sensors are compromised? Answering this question would require interaction between domains of sensor security, control, scheduling, and hybrid systems. For example, if the braking decision is made by voting among several controllers, it is critical to know the last moment to submit a vote, in order to not miss the braking deadline. Can compromised voters sway the decision and cause a collision? The advantage of exploring hybrid programs is that they allow modeling safety-critical behavior in continuous time, unlike many discrete approximations.

To conclude, we established that there are important yet implicit interactions between traditional CPS domains and sensor security. If not handled carefully, such vulnerabilities may be exploited with devastating consequences. The analysis contracts methodology shows promise for eliminating such vulnerabilities, and we plan to develop it further.

Acknowledgments

Copyright 2015 ACM. 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. This material has been approved for public release and unlimited distribution. DM-0002551.

This work was also supported in part by the National Science Foundation under Grant CNS-0834701, and the National Security Agency.

The authors thank Javier Camara for his help in exploring the body of research on probabilistic model checking.

7. REFERENCES

[1] P. Axer and R. Ernst. Designing an Analyzable and Resilient Embedded Operating System. In *Informatik 2012, 42. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 16.-21.09.2012, Braunschweig*, 2012.

[2] R. Baheti and H. Gill. Cyber-Physical Systems. Technical report, 2011.

[3] L. Benvenuti, A. Ferrari, L. Mangeruca, E. Mazzi, R. Passerone, and C. Sofronis. A contract-based formalism for the specification of heterogeneous systems. In *2008 Forum on Specification Verification and Design Languages*. IEEE, sep 2008.

[4] D. Broman, E. A. Lee, S. Tripakis, and M. Törngren. Viewpoints formalisms, languages, and tools for cyber-physical systems. In *Proc. of the 6th Workshop on Multi-Paradigm Modeling*, 2012.

[5] A. A. Cardenas, S. Amin, and S. Sastry. Secure control: Towards survivable cyber-physical systems. In *The 28th International Conference on Distributed Computing Systems Workshops*. IEEE, 2008.

[6] C. S. Carlson. *Effective FMEAs*. John Wiley & Sons Inc., 2012.

[7] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proceedings of the 20th USENIX Conference on Security*, Berkeley, CA, USA, 2011.

[8] D. W. S. Clair. *Controller Tuning and Control Loop Performance*. Straight-Line Control Co., Newark, second edition edition edition, 1990.

[9] J. Dabney and T. L. Harman. *Mastering SIMULINK 2*. Prentice Hall, Upper Saddle River, N.J., 1998.

[10] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu. metro II. *ACM Trans. Embed. Comput. Syst.*, 12(1s):1, 2013.

[11] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu. A Next-Generation Design Framework for Platform-based Design. In *DVCon 2007*, 2007.

[12] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Lecture Notes in Computer Science*, pages 337–340. Springer Science Business Media, 2008.

[13] D. De Niz, L. Wrage, A. Rowe, and R. R. Rajkumar. Utility-Based Resource Overbooking for Cyber-Physical Systems. *ACM Trans. Embed. Comput. Syst.*, 13(5s):162:1–162:25, Oct. 2014.

[14] J. Delange and P. Feiler. Architecture Fault Modeling with the AADL Error-Model Annex. In *40th Conference on Software Engineering and Advanced Applications*, pages 361–368, 2014.

[15] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proc. IEEE*, 91(1), 2003.

[16] Ethan McGee, Mike Kabbani, and Nicholas Guzzardo. Collision Detection AADL, 2013. https://github.com/mikekab/collision_detection_aadl.

[17] A. Evans, S. Kent, and B. Selic. *UML 2000 - The Unified Modeling Language. Advancing the Standard.* Springer, New York, 2000.

[18] E. Eyisi, Z. Zhang, X. Koutsoukos, J. Porter, G. Karsai, and J. Sztipanovits. Model-Based Control Design and Integration of Cyberphysical Systems: An Adaptive Cruise Control Case Study. *Journal of Control Science and Engineering*, 2013.

[19] J. Faber. Verification Architectures: Compositional Reasoning for Real-Time Systems. In *Integrated Formal Methods*. Springer, 2010.

[20] H. Fawzi, P. Tabuada, and S. Diggavi. Secure Estimation and Control for Cyber-Physical Systems Under Adversarial Attacks. *IEEE Transactions on Automatic Control*, 59(6), 2014.

[21] P. H. Feiler, B. Lewis, S. Vestal, and E. Colbert. An Overview of the SAE Architecture Analysis & Design Language (AADL) Standard: A Basis for Model-Based Architecture-Driven Embedded Systems Engineering. In *Architecture Description Languages*. Springer Science Business Media, 2005.

[22] R. B. Franca, J.-P. Bodeveix, M. Filali, J.-F. Rolland, D. Chemouil, and D. Thomas. The AADL behaviour annex – experiments and roadmap. In *12th International Conference on Engineering Complex Computer Systems*. IEEE, 2007.

[23] L. P. Francesco Oliviero. REFACING: An autonomic approach to network security based on multidimensional trustworthiness. *Computer Networks*, 52:2745–2763, 2008.

[24] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *Hybrid Systems: Computation and Control*. Springer, 2005.

[25] D. Garlan, R. Monroe, and D. Wile. Acme: Architectural Description of Component-Based Systems. *Foundations of component-based systems*, 2000.

[26] I. Goldhirsch, P.-L. Sulem, and S. A. Orszag. Stability and Lyapunov stability of dynamical systems: A differential approach and a numerical method. *Physica D: Nonlinear Phenomena*, 27(3), 1987.

[27] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5), 1994.

[28] M. Hecht, A. Lam, and C. Vogl. A Tool Set for Integrated Software and Hardware Dependability Analysis Using the Architecture Analysis and Design Language (AADL) and Error Model Annex. In *16th International Conference on Engineering of Complex Computer Systems*. IEEE, 2011.

[29] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. In *Programming Methodology*, pages 89–100. Springer New York, 1978.

[30] G. J. Holzmann. The Model Checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, may 1997.

[31] J. Hugues and S. Gheoghe. The AADL Constraint Annex. 2013.

[32] A. Iliaifar. LIDAR, lasers, and logic: Anatomy of an autonomous vehicle, 2013.

[33] W. Jones. Keeping cars from crashing. *IEEE Spectr.*, 38(9), 2001.

[34] J.-P. Katoen, M. Khattri, and I. Zapreevt. A Markov reward model checker. In *2nd International Conference on the Quantitative Evaluation of Systems*, 2005.

[35] M. Klein. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Springer, 1993.

[36] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental Security Analysis of a Modern Automobile. In *Symposium on Security and Privacy*. IEEE, 2010.

[37] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic Model Checking. In *Formal Methods for Performance Evaluation*. Springer, 2007.

[38] E. Lee. The Past Present and Future of Cyber-Physical Systems: A Focus on Models. *Sensors*, 15(3), 2015.

[39] E. A. Lee. Cyber Physical Systems: Design Challenges. In *11th International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. IEEE, 2008.

[40] E. A. Lee. CPS foundations. In *Proceedings of the 47th Design Automation Conference*. ACM Press, 2010.

[41] F. G. Marmol and G. M. Perez. Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems. *Computer Standards & Interfaces*, 32(4), 2010.

[42] R. Mateescu. Model Checking for Software Architectures. In *Software Architecture*, pages 219–224. Springer Science Business Media, 2004.

[43] C. Miao, L. Huang, W. Guo, and H. Xu. A Trustworthiness Evaluation Method for Wireless Sensor Nodes Based on D-S Evidence Theory. In K. Ren, X. Liu, W. Liang, M. Xu, X. Jia, and K. Xing, editors, *Wireless Algorithms, Systems, and Applications*, number 7992 in Lecture Notes in Computer Science, pages 163–174. Springer Berlin Heidelberg, 2013.

[44] F. Miao, M. Pajic, and G. J. Pappas. Stochastic game approach for replay attack detection. In *52nd IEEE Conference on Decision and Control*. IEEE, 2013.

[45] Y. Mo, T. H.-J. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig, and B. Sinopoli. Cyber Physical Security of a Smart Grid Infrastructure. *Proc. IEEE*, 100(1), 2012.

[46] M.-Y. Nam, D. de Niz, L. Wrage, and L. Sha. Resource allocation contracts for open analytic runtime models. In *Proc. of the 9th International Conference on Embedded Software*, 2011.

[47] G. Nitsche, K. Gruttner, and W. Nebel. Power contracts: A formal way towards power-closure?! In *23rd International Workshop on Power and Timing Modeling Optimization and Simulation*. IEEE, 2013.

[48] P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donze, and S. A. Seshia. A Contract-Based Methodology for Aircraft Electric Power System Design. *IEEE Access*, 2:1–25, 2014.

[49] K. Ogata and J. W. Brewer. Modern Control Engineering. *J. Dyn. Sys. Meas., Control*, 93(1), 1971.

[50] M. Pajic, J. Weimer, N. Bezzo, P. Tabuada, O. Sokolsky, I. Lee, and G. J. Pappas. Robustness of attack-resilient state estimators. In *International Conference on Cyber-Physical Systems*. IEEE, 2014.

[51] A. Platzer. *Logical Analysis of Hybrid Systems*. Springer Berlin Heidelberg, 2010.

[52] Rajhans, Akshay. *Multi-Model Heterogeneous Verification of Cyber-Physical Systems*. PhD thesis, Carnegie Mellon University, 2013.

[53] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems. In *Proceedings of the 47th Design Automation Conference*. ACM Press, 2010.

[54] J. E. Rolph and R. A. Howard. Dynamic Probabilistic Systems Volume I: Markov Models and Volume II: Semi- Markov and Decision Processes. *Journal of the American Statistical Association*, 67(340), 1972.

[55] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan. ACTIVE: A Tool for Integrating Analysis Contracts. In *The 5th Analytic Virtual Integration of Cyber-Physical Systems Workshop*, Rome, Italy, 2014.

[56] I. Ruchkin, D. D. Niz, D. Garlan, and S. Chaki. Contract-based integration of cyber-physical analyses. In *Proceedings of the 14th International Conference on Embedded Software - EMSOFT '14*. ACM Press, 2014.

[57] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems. *European Journal of Control*, 18(3), 2012.

[58] H. Schneider. Failure Mode and Effect Analysis: FMEA From Theory to Execution. *Technometrics*, 38(1), 1996.

[59] D. P. Siewiorek and P. Narasimhan. *Fault-Tolerant Architectures for Space & Avionics Applications*. 2005.

[60] V. Subramanian, S. Uluagac, H. Cam, and R. Beyah. Examining the characteristics and implications of sensor side channels. In *2013 IEEE International Conference on Communications (ICC)*, June 2013.

[61] J. Sztipanovits, T. Bapty, S. Neema, L. Howard, and E. Jackson. OpenMETA: A Model- and Component-Based Design Tool Chain for Cyber-Physical Systems. In *From Programs to Systems. The Systems perspective in Computing*. Springer Science Business Media, 2014.

[62] L.-A. Tang, X. Yu, S. Kim, Q. Gu, J. Han, A. Leung, and T. La Porta. Trustworthiness analysis of sensor data in cyber-physical systems. *Journal of Computer and System Sciences*, 79(3), 2013.

[63] A. Uluagac, V. Subramanian, and R. Beyah. Sensory channel threats to Cyber Physical Systems: A wake-up call. In *Conference on Communications and Network Security*, pages 301–309, Oct. 2014.

[64] S. Vestal. Mode changes in a real-time architecture description language. In *Proceedings of 2nd International Workshop on Configurable Distributed Systems*. IEEE, 1994.

[65] Z. Yang, K. Hu, D. Ma, L. Pi, and J.-P. Bodeveix. Formal semantics and verification of AADL modes in Timed Abstract State Machine. In *Intl Conference on Progress in Informatics and Computing*, 2010.

[66] J. Zhang and G. Li. A Novel Model-Based Method for Automatic Generation of FMEA. In *Proc. of the Symposium on Computer Communication, Control and Automation*. Atlantis, 2013.