Analytic Dependency Loops in Architectural Models of Cyber-Physical Systems

Ivan Ruchkin, Bradley Schmerl, David Garlan
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{iruchkin, schmerl, garlan}@cs.cmu.edu

***Abstract*—Rigorous engineering of safety-critical Cyber-Physical Systems (CPS) requires integration of heterogeneous modeling methods from different disciplines. It is often necessary to view this integration from the perspective of analyses – algorithms that read and change models. Although such analytic integration supports formal contract-based verification of model evolution, it suffers from the limitation of analytic dependency loops. Dependency loops between analyses cannot be resolved based on existing contract-based verification. This paper makes a step towards using rich architectural descriptions to resolve circular analytic dependencies. We characterize the dependency loop problem and discuss three algorithmic approaches to resolving such loops: analysis iteration, constraint solving, and genetic search. These approaches take advantage of information in multi-view architectures to resolve analytic dependency loops.**

***Keywords*—Analytical models, Component architectures, Embedded software, Systems engineering and theory**

I. Introduction

Cyber-physical systems (CPS), such as self-driving cars and autonomous drones, often operate in critical contexts and therefore require rigorous up-front engineering methods. The model-driven engineering (MDE) community has been developing formal approaches to designing and verifying systems to provide guarantees on performance, safety, and other critical qualities [1] [2]. For example, recent research on collision avoidance proposes various analysis and verification techniques to guarantee an absence of collisions [3] [4] [5].

One important aspect of CPS design is using heterogeneous types of analysis to evaluate and evolve designs. For example, reliability analysis can evolve a design so that its elements have sufficient redundancy [6]; scheduling analysis can allocate computational elements to processors [7]. In reality, there are many analyses that are applied to CPS designs and one particularly challenging aspect is how to integrate, or order and properly apply, these analyses. Such analytic integration can be done by verifying logical conditions in order to control changes made to models [8]. In particular, prior work showed that verification based on *analysis contracts* can prevent errors caused by stale or missing information by determining which analyses must be redone, and in which order. The analytic perspective is convenient when model evolution patterns are simpler than patterns for model structure and behavior.

One of the problems that arises during analytic integration are *analysis dependency loops* – circular dependencies among several analyses that make it impossible to order these analyses in a sound way. Such loops may happen when analyses from different domains are developed independently but operate on the same design aspects, such as sensor infrastructure. For example, reliability analysis may change the number of sensors based on failure probabilities, and trust analysis may adjust the number of sensors to mitigate against malicious attacks on sensors. How can we conduct these analyses and be sure that we have sufficient sensors? Such dependency loops cannot be overcome with previous work using analysis contract specifications [9] and render the methodology inapplicable.

One way to resolve analysis dependency loops is to bring in a more detailed model of the system and analyses, making the dependency description better understood. In previous work we have advocated for the use of component-based (i.e., *architectural*) models to separate engineering into independent components and to assemble the components together [4] [10]. In this paper we take the first step to combining architectural and analytic approaches to CPS design. We take advantage of rich architectural models – multi-view descriptions and component types – to provide several approaches to resolving analytic dependency loops. Specifically, this paper makes the following contributions:

- A characterization of the problem of analytic dependency loops.
- Three algorithmic approaches to resolve dependency loops automatically, and a qualitative analysis of their applicability, strengths, and weaknesses.

Specifically, we examine analysis iteration, constraint solving, and genetic search as potential approaches to resolve dependencies, showing the cases in which each approach may best apply. Iteration and search rely on multi-view mappings to produce valid architectural models, and constraint solving uses a library of architectural types to set up constraint problems on architectural models.

The paper is organized as follows. The next section reviews the related work on CPS modeling and analysis. We then describe and exemplify the problem of circular analysis dependencies in Sec. III. In Sec. IV we propose three approaches to resolve such dependencies and discuss the approaches' qualities. We wrap up the paper by describing future research directions in Sec. V.

II. Related work

Recently several research efforts in architectural modeling have achieved substantial progress in MDE. Results include

composability and provability using component interfaces and contracts [11] [12] [13], rich simulation using multiple computational models [14], platform-based verification and reuse [15] [16], graph-based mapping and consistency between cyber and physical models [17], and semantic validation using logical metamodeling [18] [19]. These methods do not enable reasoning about how designs are modified throughout the engineering lifecycle. At best, there are tools like DESERT [20] for exploring the design space, but these do not support consistent evolution of a set of models. As a result, the integration of heterogeneous CPS models has to be maintained manually, which is tedious and error-prone.

The problem of dependency loops has been considered in many contexts. For example, dataflow systems that consist of concurrent actors may deadlock due to dependency loops among actors [21]. The authors develop a specification approach called *causality interfaces* for actors that helps resolve the loops. However, whether the causality interface approach can be applied to model-based analyses remains an open research question. Another approach is using game-theoretic models to synthesize proof of contract causality [9]. This method relies on detailed game models that may be difficult to obtain for heterogeneous domains where analyses often originate.

Existing research on analyses [22] [8] and change-driven transformations [23] applies formal reasoning at the level of analysis algorithms, which is distinct from architectural models. One aspect of this reasoning is identifying dependencies between analyses and ordering their execution to respect these dependencies. So far this body of work has only considered tree-shaped analysis graphs that do not have cycles [8]. The developed tools, e.g., ACTIVE [24], would not be applicable for circularly dependent analyses, which are more likely to be discovered as more domains are incorporated into the framework of analysis contracts. Our work identifies the potential ways to deal with such circularities.

Several dependency management methods address interactions in different parts of system design. For example, Qamar has developed a cross-domain dependency management approach that keeps track of dependent model variables in their designs across disciplinary and instrumental boundaries [25]. Another example is a multi-view architecture description language with dependency links to ensure consistency among views [26]. Such approaches focus on discovery and representation of dependencies and do not deal with algorithmic cycles directly. Our work therefore can be seen as a next step in automation of change and dependency management.

III. Analytic Dependency Loops

In this section we describe the problem of analytic dependency loops in detail. First we present a car model to ground the discussion and describe two example analyses from the reliability and security domains. Then we formalize several foundational concepts that help us characterize analytic dependency loops.

A. System Example

To illustrate circular dependencies between analyses, let us consider the internal digital system of a self-driving car.

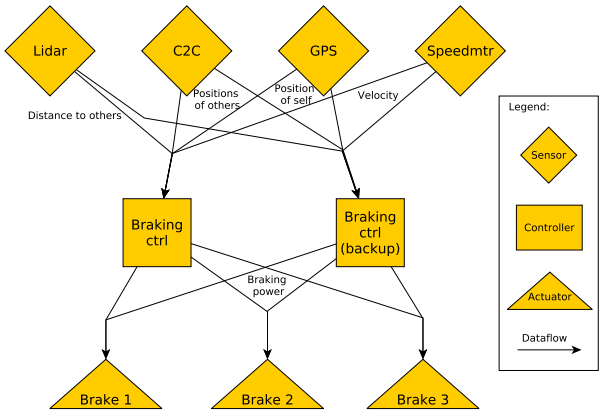

Fig. 1. Architecture of the braking subsystem of a self-driving vehicle.

Inspired by recent advances in the automotive industry [27], the car is designed to perform fully autonomous driving that includes acceleration, lane control and change, platooning, and braking to avoid collision. To inform its decisions, the car collects information about the environment through its sensors: sonar, lidar [28], speedometer, and wireless car-to-car (C2C) communication. Controllers make actuation decisions using algorithms executed in threads running on electronic control units (ECUs), and send these decisions to physical actuators such as steering, acceleration, and braking.

In this example, we focus on the braking subsystem, because it performs the safety-critical function of avoiding collision with various static and dynamic obstacles on the road. In Fig. 1 we show an example architecture of the braking subsystem. Sensors collect data about the position of the car, its speed, and the locations of obstacles. The controllers periodically make a decision about the timing and strength of braking, sending their commands to several braking actuators in the front and back of the car. Since braking control and actuation are critical functions, there is a reserve controller and redundant brakes for the case of nominal components malfunctioning. Throughout this section we will build a formal multi-view model of this architecture in order to precisely express the conditions leading to dependency loops.

One of the major quality attributes of the braking subsystem is safety, which itself depends on system *security* and *reliability*. Security needs to be considered because it may violate safety if a malicious attacker compromises sensors ($\mathbb{S}$). For instance, the braking system could be compromised internally through the CAN network of the car [29] or externally [30] by executing deception attacks on sensors [31]. Different types and placements of sensors (Place) [1] have varying capacity to be compromised by attacks, which determines their level of *trustworthiness* (Trust)[2] [32]. Sensors that output genuine data, or have a mechanism to determine genuine data, are considered trustworthy for modeling purposes. We extend this notion to controllers as well. According to [31], there exists a data decoding algorithm that is guaranteed to deliver genuine data when at least half of the sensors are trustworthy. We

[1] In the running example we consider car sensors placed internally, such as speedometer, and externally, such as car-to-car communication.

[2] For simplicity, we assume that Trust is binary – whether a sensor's output can be trusted.

model security concerns in the trustworthiness view V_{trust} (see the left half of Fig. 2) that contains components $\mathbb{C}_{trust}$ that may be compromised by a malicious attacker – sensors and controllers – and a bus connector $\mathbb{CN}_{trust}$ with data read and write operations.

Reliability of the system should be considered because safety is affected when components randomly fail (e.g., due to manufacturing defects). The reliability view V_{fmea} (see the right half of Fig. 2) contains physical components $\mathbb{C}_{fmea}$ that may fail – sensor devices, threads, electronic control units (ECUs) – and physical network connectors and buses $\mathbb{CN}_{fmea}$. View V_{fmea} focuses on such concerns as component probabilities of failure $\mathsf{P}_{\mathsf{fail}}$, failure propagation among individual components, and failure effects. For instance, if the speedometer fails in Fig. 1, the controller will not have an accurate measurement of speed. However, this can be overcome by inferring the approximate speed from values delivered by a position sensor such as GPS. If, on the other hand, both lidar and C2C fail, there is no way for the controller to obtain the locations of obstacles, which is likely to result in a critical failure. Thus, different configurations of system failure (also known as *failure modes* [6]) may have different likelihoods of effects on the system.

The views V_{trust} and V_{fmea} are related to each other through view-to-view mappings of components: $\mathsf{R}_V^V \subset \mathbb{C} \cup \mathbb{CN} \times \mathbb{C} \cup \mathbb{CN}$. Component $c_1 \in \mathbb{C}_{fmea}$ is considered mapped to $c_2 \in \mathbb{C}_{trust}$ when $(c_1, c_2) \in \mathsf{R}_V^V$, and analogously for connectors. Some components such as ECUs in V_{fmea} do not have a counterpart in V_{trust}, so the views are not full abstractions of each other as they are required to be in some approaches (e.g., structural consistency [33]). However, in our example, it is important that sensors and controllers are mapped to each other in both views: every sensor and controller considered for trust needs to be considered for failure, and vice versa. Hence we will use the following condition of consistency:

$$\begin{aligned} &\forall c_1 \in \mathbb{S}_{trust} \cup \mathbb{R}_{trust} \\ &\quad \exists c_2 \in \mathbb{S}_{fmea} \cup \mathbb{R}_{fmea} \cdot (c_1, c_2) \in \mathsf{R}_V^V \qquad (1) \\ &\quad\quad \wedge \\ &\forall c_2 \in \mathbb{S}_{fmea} \cup \mathbb{R}_{fmea} \\ &\quad \exists c_1 \in \mathbb{S}_{trust} \cup \mathbb{R}_{trust} \cdot (c_1, c_2) \in \mathsf{R}_V^V \qquad (2) \end{aligned}$$

where

$$\mathbb{S}_{trust} \cup \mathbb{R}_{trust} \subset \mathbb{C}_{trust} \wedge \mathbb{S}_{fmea} \cup \mathbb{R}_{fmea} \subset \mathbb{C}_{fmea}.$$

The views and relations constitute a full *architectural model* M of the system: $\mathsf{M} \equiv (\mathsf{V}_{trust}, \mathsf{V}_{fmea}, \mathsf{R}_V^V)$. Outside the formal boundaries of M we define component and connector types $\mathbb{T}$ to reuse common aspects of components. Types specify relevant properties such as Trust and $\mathsf{P}_{\mathsf{fail}}$, and formally are domains of these functions. For example, a component type could describe a lidar device from a particular supplier and its characteristics. Formally, types are assigned to components and connectors with a typing function $\mathsf{T} : \mathbb{C} \cup \mathbb{CN} \to \{\mathbb{T}\}$ that maps an architectural element to a subset of types. This way we can specify and reuse types separately from systems.

B. Analyses and Contracts

Architectural views undergo continual change as engineers search for a design that satisfies the requirements. Often design exploration and refinement relies upon algorithms and tools, which read and change models. We call such tools *analyses* [22] [8]. Many analyses originate in different domains and make implicit interdependent assumptions about each other. For example, real-time scheduling assumes that there is sufficient electrical power for every processor at all times. At the same time, battery design process requires that computations do not consume more power than the battery can reasonably provide. Such analytic assumptions need to be explicitly considered and reconciled. Let us consider two analyses from the fields of sensor security and system reliability respectively:

- *Trustworthiness Analysis.* A_{trust} [34] modifies the system to ensure that in case of a malicious attack on sensors the system can still function within acceptable error margins. This is achieved by considering a particular attacker profile and determining the necessary number of sensors of each kind. A_{trust} operates over V_{trust}.
- *Failure Modes and Effects Analysis (FMEA).* A_{fmea} [35] determines failure modes and their probabilities. We consider a version of FMEA that redesigns the system so that it does not have critical failure modes (i.e., those where the system is unsafe) with likelihood more than some threshold α_{fail}. A_{fmea} operates over V_{fmea}.

In previous work [36], we considered A_{trust} and A_{fmea} to be integrated without dependency cycles. However, this is not a realistic solution: as more analyses are considered, cyclic dependencies are increasingly likely to occur, and cannot be avoided without significantly changing the analyses. Therefore, in this paper we consider A_{trust} and A_{fmea} to be separate but dependent on each other (as we elaborate later), thus introducing a cyclic dependency that needs to be addressed.

Formally, analysis A is a function that has system designs as its domain and codomain: $\mathsf{A} : \mathsf{M} \to \mathsf{M}$. Many analyses including A_{trust} and A_{fmea} operate only on their specific view V, in which case we can restrict an analysis to this view: $\mathsf{A} : \mathsf{V} \to \mathsf{V}$. In this case executing, or applying, an analysis A to a system model M requires two steps:

1) Obtaining $\mathsf{A}(\mathsf{M})$ and making it the new system under design.
2) Restoring consistency among views in M.

In our example each analysis modifies its own view, which means in step 2 the changes need to be propagated to the other view. This can be done using mapping R_V^V following existing approaches like change propagation [37] or model synchronization [38]. Although re-establishing view consistency is an important part of analysis the workflow, we do not concentrate on it in this paper.

An important assumption of our work is the idempotence property $\mathsf{A}(\mathsf{A}(\mathsf{V})) = \mathsf{A}(\mathsf{V})$. Both analyses that we consider in this paper are idempotent because they directly address a particular quality attribute, and do not modify the system if the attribute is already satisfied. We rely on idempotence in Sec. IV to resolve dependency cycles. Applicability of the discussed resolution techniques to non-idempotent analyses will be considered in the future work.

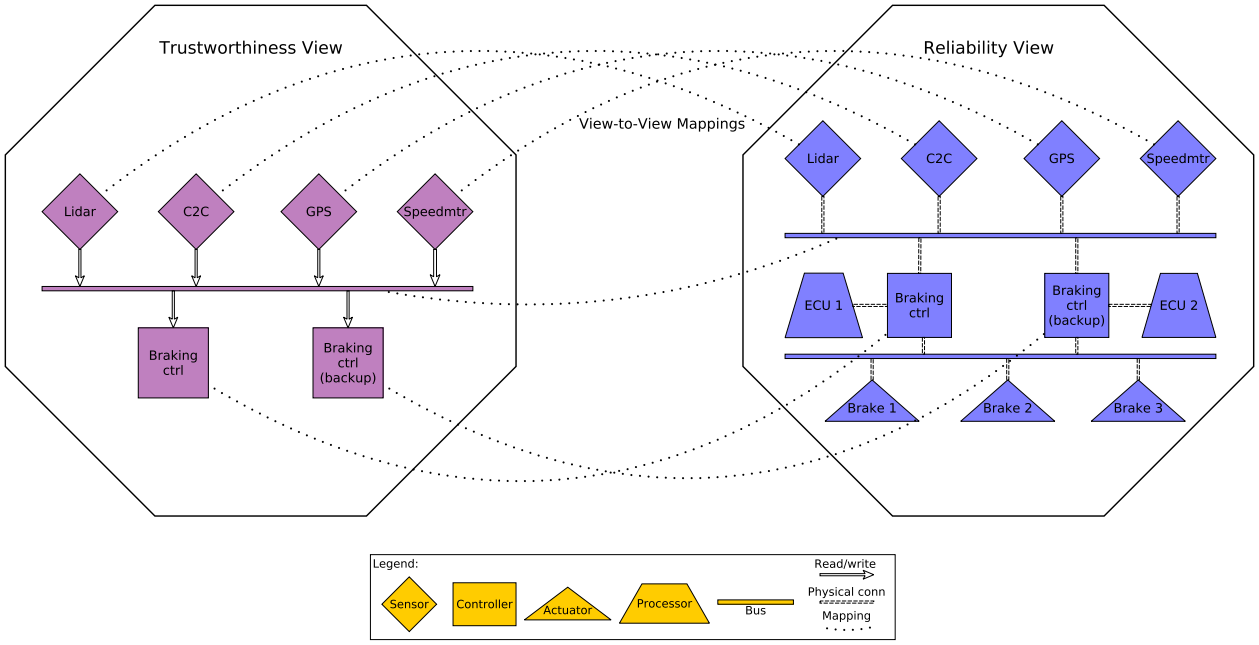

Fig. 2. A multi-view model M of the braking system: V_{trust}, V_{fmea}, and R^V_V.

Following [8], for each A we define analysis contracts as tuples of inputs I, outputs O, assumptions A, and guarantees G: $C_{\mathsf{A}} \equiv (I, O, A, G)$. For simplicity we will write $\mathsf{A}.I$ meaning $C_{\mathsf{A}}.I$. Therefore we have the following contracts[3]:

$$
\begin{aligned}
&\mathsf{C}_{trust}.I = \{\mathbb{S}, \mathsf{Place}, \dots\} \\
&\mathsf{C}_{trust}.O = \{\mathbb{S}, \mathsf{Trust}\} \\
&\mathsf{C}_{trust}.A = \dots \\
&\mathsf{C}_{trust}.G = \textit{“system is trustworthy”}\ ^{4} \\
&\mathsf{C}_{fmea}.I = \{\mathbb{S}, \mathsf{P}_{\mathsf{fail}}, \alpha_{fail}\} \\
&\mathsf{C}_{fmea}.O = \{\mathbb{S}, \dots\} \\
&\mathsf{C}_{fmea}.A = \dots \\
&\mathsf{C}_{fmea}.G = \textit{“system is reliable”}\ ^{5}
\end{aligned}
\tag{3}
$$

We limit our discussion in the rest of the paper to A_{trust} and A_{fmea}. However, in a typical engineering context there may be dozens of analyses from heterogeneous domains that have dependencies. For example, control analysis may determine whether a control algorithm satisfies control requirements such as rise time and percent overshoot [39]. Schedulability analyses such as binpacking and frequency scaling [22] determine the capacity behind the control algorithm to compute outputs in time, but at the same time depend upon the quantity of control computation and communication.

[3]Some inputs, outputs, and assumptions are omitted because they do not contribute to the discussion of dependency loops. Full contracts can be found in [36].

[4]System trustworthiness may have several different operationalizations [32]. For example, we could assume that the system is trustworthy when at least half of its sensors are trustworthy [31]. A particular operationalization of trustworthiness is outside of this paper's scope.

[5]Analogously, the interpretation of reliability may differ from system to system. We reason about reliability as a whole without binding ourselves to a particular definition.

According to Eqs. 3 both A_{trust} and A_{fmea} modify the set of system's sensors $\mathbb{S}$, meaning that these two analyses have a circular dependency on each other. This makes it impossible to find a valid sequence of their execution based on just their contracts. Therefore we need to study the nature of analytic dependency loops closer.

C. Dependency Loops

To characterize the dependency loops precisely, let us introduce several formal definitions for analyses, dependencies, and dependency loops between analyses. First, two analyses are dependent if inputs of one have commonalities with outputs of the other.

Definition 1: Analysis A_i *is dependent* on analysis A_j, denoted $d(\mathsf{A}_i, \mathsf{A}_j)$, if $\mathsf{A}_i.I \cap \mathsf{A}_j.O \neq \emptyset$.

Second, a dependency loop is a chain of analysis dependencies where the last element depends on the first one. The smallest dependency loop is a pair of mutually dependent analyses.

Definition 2: Analyses $\mathsf{A}_1 \dots \mathsf{A}_n$ form a *dependency loop*, denoted $\mathsf{Loop}(\mathsf{A}_1 \dots \mathsf{A}_n)$, if:

$$d(\mathsf{A}_1, \mathsf{A}_2) \wedge \dots \wedge d(\mathsf{A}_{n-1}, \mathsf{A}_n) \wedge d(\mathsf{A}_n, \mathsf{A}_1)$$

Eqs. 3 indicate that $\mathsf{Loop}(\mathsf{A}_{fmea}, \mathsf{A}_{trust})$. A dependency loop makes it impossible to use the graph-based ordering algorithm [8] to find a sound sequence of analyses because analysis contracts do not have sufficient specification to resolve the dependency. Therefore we explore other ways to resolve loops. Our ultimate goal is to "skip" the loop and find a design that would theoretically satisfy the loop. Such a design, when fed into each of the analyses, would not change. This situation resembles the fixed point concept from numeric analysis [40], hence we adapt definitions from that field.

Definition 3: A system model or view M is a *fixpoint* of an analysis set $\mathcal{AN}$, denoted $\mathsf{M} \in \mathsf{FP}(\mathcal{AN})$, if $\forall \mathsf{A} \in \mathcal{AN} \cdot \mathsf{A}(\mathsf{M}) = \mathsf{M}$.

From Def. 3 it follows that a fixpoint M satisfies the guarantees of all analyses in $\mathcal{AN}$. This is a necessary, but not sufficient, condition: a model may satisfy all guarantees of an analyses, but not be a fixpoint because the analysis may still modify the model (e.g., to optimize it further). We define models that satisfy all guarantees of an analysis as its *candidate fixpoints*. Also, a fixpoint may not satisfy some assumptions because analyses may exclude their fixpoints from the applicability set since no further changes are possible or needed.

Now consider a set of analyses $\mathcal{AN}$ and a system model M. Below are several mutually exclusive *cases* for fixpoints. These cases support two goals. First, they will help us qualitatively evaluate techniques for dependency loop resolution, which we present in the next section. Second, knowing the case of a particular loop narrows down the available techniques, thus streamlining the resolution of this loop.

- C1 *Strong convergence*: a fixpoint exists and is reachable by any sequence of analyses. This may happen when there are two analyses and their changes to the system do not practically overlap.
- C2 *Weak convergence*: a fixpoint exists and is reachable by some sequence of analyses. This is more likely to be the case when there are several analyses and they interact differently depending on their order of execution.
- C3 *Weak divergence*: a fixpoint exists but is not reachable by any sequence of analyses. E.g., there is a stable alternation between two designs with two analyses.
- C4 *Divergence*: a fixpoint does not exist, but at least one candidate fixpoint exists.
- C5 *Strong divergence*: no candidate fixpoints exist: no model satisfies a conjunction of guarantees of all analyses.

Now that the problem of analytic dependency loops is formally defined, we proceed to the methods of its resolution.

IV. Resolution of Dependency Loops

The goal of dependency loop resolution is, given a system M and a set of circularly dependent analyses $\mathsf{Loop}(\mathcal{AN})$, to find such analysis A' that would produce a fixpoint of $\mathcal{AN}$:

$$\mathsf{A}'(\mathsf{M}) \in \mathsf{FP}(\mathcal{AN}).$$

This problem has two sub-parts: *finding* a fixpoint and *verifying* that a given model is a fixpoint. For the former, we are not looking for a mathematically optimal solution or a specific fixpoint because system design is often done via *satisficing* [41] rather than optimizing. Many aspects of design are poorly quantifiable: supplier availability and negotiation, diverse qualities of the system, component compatibility, and so on. Therefore we prefer an acceptable suboptimal design to an exhaustive search of a design space that is often unbounded or too large. For the latter part however, we do require an accurate approach, otherwise analysis results may be unsound and potentially lead to design errors.

TABLE I. Convergence, example of C1 and C2.

Sensors	G_{trust}	G_{fmea}
A	✓	✗
B	✗	✗
AB	✓	✗
ABB	✗	✓
AABB	✓	✓

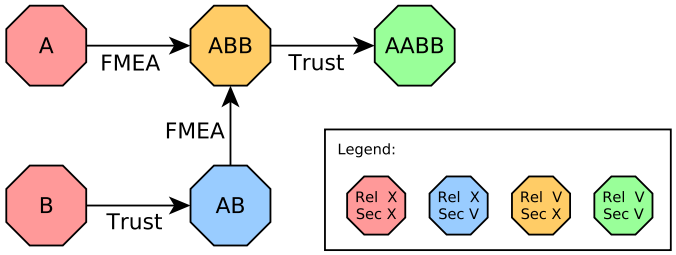

Fig. 3. Example workflow of analyses for convergence.

For further discussion consider the application of A_{fmea} and A_{trust} in several specific contexts. Assume that two types of sensors are given: A and B. A is trustworthy but unreliable, and B is reliable but untrustworthy – these characteristics are specified in the sensors' architectural types. The specific calculations of aggregates do not concern us at this moment, and we abstract reliability and trustworthiness as boolean properties.

First let us consider the convergence situation. Tab. I shows the evaluation of a sensor configuration for the convergence situation. Each line represents a configuration of the system in terms of sensors. AABB is the desired fixpoint configuration that is both trustworthy and reliable. As Fig. 3 indicates, the alternating analyses converge on the fixpoint.

Similarly, Tab. II and III represent divergence with and without a fixpoint respectively. Fig. 4 shows an alternation situation where ABB is not trustworthy and AAB is not reliable, and analyses keep alternating between designs without converging on an existing but unreachable fixpoint AABB.

TABLE II. Divergence, example of C3 and C4.

Sensors	G_{trust}	G_{fmea}
AB	✓	✗
ABB	✗	✓
AAB	✓	✗
AABB	✓	✓

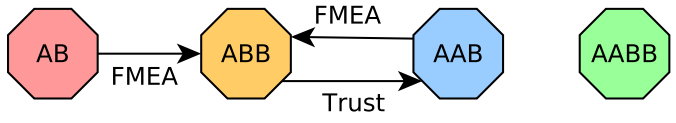

Fig. 4. Example workflow of analyses for divergence. See legend in Fig. 3.

To achieve practical dependency resolution we consider three methods: analysis iteration, constraint solving, and genetic search.

Analysis Iteration. This method iteratively searches for a fixpoint by applying analyses to the model in some sequence, similarly to a method of numeric computation of functional

TABLE III. STRONG DIVERGENCE, EXAMPLE OF C5.

Sensors	G_{trust}	G_{fmea}
A	✓	✗
B	✗	✗
AB	✓	✗
ABB	✗	✓
AAB	✓	✗
AABB	✓	✗
ABBB	✗	✓
AAAB	✓	✗

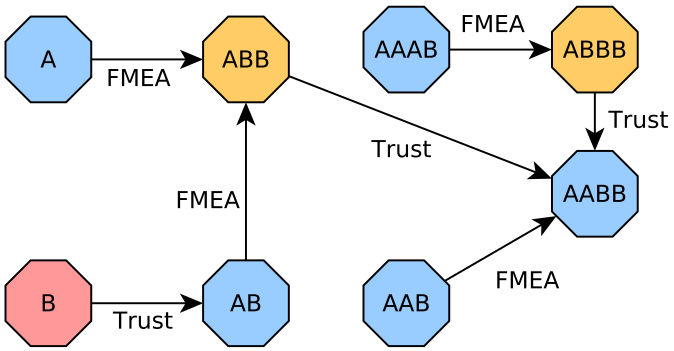

Fig. 5. Example workflow of analyses for strong divergence. See legend in Fig. 3.

fixpoints [40]. In our case, however, the order of analysis iteration is an open question. One option is to select random sequences of analyses, which would find a fixpoint in C1 and could find one in C2. A more sophisticated approach is to construct a contract-guided sequence: only analyses with satisfied assumptions are applied; from those, analyses with unsatisfied guarantees are given a priority. Selection may be random or lexicographical. Another to enhance analysis iteration is to define a partial order on each view, and apply analyses that move the views towards the goal. Analysis iteration can also be used as an accurate fixpoint verification method for C1, C2, C3 and candidate verification for C4 in accordance with Def. 3.

An advantage of analysis iteration is that it is simple and does not require extra specification. In particular, for Tab. I and Fig. 3 iteration would converge on the AABB model given a starting point of A or B. For larger models however iteration is computationally expensive[6], may not converge, and its success may depend heavily on the starting model: there are cases when iteration converges when started from one model but not from another. Therefore, we suggest two other approaches.

Constraint solving. This method searches for a fixpoint by constructing a constraint satisfaction problem and feeding it to a solver. We can use Satisfiability Modulo Theories (SMT) [42] as an example of a constraint solving approach. To set up a constraint problem, one needs to translate relevant architectural types from the model (denoted $SMT(\mathsf{M})$) and analysis guarantees into problem constraints using an existing theory (e.g., integers or reals). The set of sensors under search would become an underspecified part of the satisfaction problem, so that a solver can find its valuation that satisfies constraints. For instance, constraint solving would find AABB in Tab. II/Fig. 4, but analysis iteration would not find a path to it. Constraint solving would also demonstrate absence of a fixpoint in Tab. III/Fig. 5, although it would only explore within the given bounds.

Constraint solving can be successfully used to find a fixpoint in C1, C2, C3, find candidate fixpoints and demonstrate absence of fixpoints in C4, and demonstrate absence of candidate fixpoints in C5 – as long as a constraint problem can be constructed and a (candidate) fixpoint lies within the constraints. The possibility of constructing a constraint satisfaction problem depends on the particular solving framework. For instance, SMT does not yet have theories for calculating real numbers. Unfortunately, constraint solving cannot verify fixpoints because it does not directly execute analyses; nevertheless, it can verify candidate fixpoints. For instance, in the case of SMT if $SMT(\mathsf{M}) \wedge \neg G_1 \wedge \cdots \wedge \neg G_n$ is UNSAT then the M is a candidate fixpoint, at least within the checking bounds. We can overcome the checking bound limitation with the next cycle resolution approach.

Genetic search. This method executes for a system model M obtaining $\mathsf{A}_1(\mathsf{M}) \ldots \mathsf{A}_n(\mathsf{M})$ and deriving hybrids among the architectures, in a way similar to crossover in genetic algorithms [43]. For two analyses A_{fmea} and A_{trust} the set of candidates is $\mathsf{A}_{fmea}(\mathsf{M}) \oplus \mathsf{A}_{trust}(\mathsf{M}) \cup \forall i \subseteq \mathsf{A}_{fmea}(\mathsf{M}) \cap \mathsf{A}_{trust}(\mathsf{M})$, where $\oplus$ is an exclusive OR over sets. Genetic search may find fixpoints in C1, C2, C3, and C4. Genetic search may be particularly useful in cases where the fixpoint is outside the bounds of constraint solving but can be reached by a mutation. For instance, if AABB were outside of the constraint checking bounds in Tab. II/Fig. 4, genetic search would still have a chance to find this model.

A special case of genetic search for the case of two models crossover – *merging models* – can be useful in cases where constraint solving is not: crossover may find a candidate that does not satisfy some guarantees or constraints. Such candidates may provide insights to engineers that would lead to relaxing inappropriate constraints or finding important subspaces of the design space. Merging would rely on view-to-view mappings to achieve consistency in the produced architectural models. We expect merging to be practically limited to sets of components because merging connectors may lead to combinatorial explosion due to non-determinism of where connectors attach. Another drawback of genetic search is that it cannot perform fixpoint verification, and therefore it should be paired with another method like iteration. Thus, non-determinism of genetic search is both its strength and weakness.

All three methods and their expected applicability are summarized in Tab. IV. The constraint solving column assumes that a constraint problem can be formulated in one of the existing theories. This table indicates that no single method can capture all possible dependency cases, and their combination is necessary to provide a robust solution to this engineering problem. We have shown that even for two analyses, A_{trust} and A_{fmea}, the circular case may be different, which would lead to different approaches being fruitful.

This section explored solutions for the relatively simple example of two mutually dependent analyses, A_{trust} and

[6]For practical application of analysis iteration it is crucial that the consistency propagation algorithm is efficient since it is run after every iteration.

TABLE IV. SUMMARY OF APPLICABILITY FOR LOOP RESOLUTION METHODS.

Case	Analysis Iteration	Constraint Solving	Genetic Search
Find C1	✓	✓	✓
Verify C1	✓	✗	✗
Find C2	✗	✓	✓
Verify C2	✓	✗	✗
Find C3	✗	✓	✓
Verify C3	✓	✗	✗
Find C4	✗	✓	✓
Verify C4	✓	✓	✗
Detect C5	✗	✓	✗

A_{fmea}. In a more complex case many analyses depend on each other and make interrelated and often vague assumptions. One may use other ways to deal with this complexity. For example, one may think of re-writing several analyses as one monolithic multi-analysis with algorithms encapsulated. This approach has a benefit of being simple and more controllable (e.g., additional optimization can be applied during consolidation), however it is more fragile because the constituent analyses cannot be reused individually or combined in a different fashion. Instead, our contract-based approach emphasizes more general modular composition, formal verification, and scalability for larger numbers of analyses.

V. FUTURE WORK AND CONCLUSION

This paper explored the challenging problem of resolving analysis dependencies. As we showed, these dependencies often cannot be resolved using contract specification and needs extra description, such as architectural types and mappings. We sketched and exemplified three approaches to cycle resolution: iterative execution, constraint solving, and genetic algorithms. We expect these descriptions and approaches be applicable to other domains and analyses (e.g., cost-benefit analysis of architecture) in our future work.

An important future work direction is implementing and integrating dependency resolution algorithms into our architectural and analytic framework [24]. The tools would need access to architectural styles and analysis descriptions to perform the intended functions. A major step is a design of a general API so that dependency resolution can be extended with new techniques. An implementation of dependency resolution would also be helpful to demonstrate practical feasibility of our cycle resolution techniques. This implementation can be further enhanced in several ways. One is concurrent execution of different techniques and aggregation of their results. Another way to enhance loop resolution is to combine it with optimization and search for optimal fixpoints.

We envision our work to be more general and systematic than ad hoc analysis integration, so empirical validation is essential. We have previously formalized a number of scientific and engineering domains: real-time CPU scheduling, electrical and thermal analysis of batteries [8], reliability, sensor security, and secure control [36]. To demonstrate the generality and effectiveness of the described cycle resolution techniques we will revisit these domains to discover circular dependencies, which we previously designed away. Beyond that, we plan to look for realistic CPS projects to investigate the effect on analytic cycles on larger system designs.

An even deeper level of integration between the analytic and architectural approaches would involve using system invariants during analysis execution. Currently, satisfaction of system invariants is a concern orthogonal to analytic execution. One way to use invariants is to discharge analytic assumptions with them, instead of verifying the assumptions directly. Similarly, one can use analytic guarantees to discharge system invariants after running an analysis. We hope that this would lead to a significant reduction of verification time and effort. We expect that bringing analyses and architecture closer together would lead to a cohesive and versatile toolbox of domain integration tools that can be applied in various engineering contexts, such as aerospace, automotive, energy, and medical CPS.

ACKNOWLEDGMENTS

The authors would like to thank Ashwini Rao, Dionisio De Niz, and Sagar Chaki for their work on the initial vision of the system example and analyses in [36], and Nicholas Rouquette for a motivating discussion of circular dependencies in model-based engineering.

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

REFERENCES

[1] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *Proceedings of the 11th Symposium on Object Oriented Real-Time Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 363–369.

[2] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *2010 47th ACM/IEEE Design Automation Conference (DAC)*, 2010, pp. 731–736.

[3] S. Mitsch, K. Ghorbal, and A. Platzer, "On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles," in *Proc. of Robotics: Science and Systems*, 2013.

[4] I. Ruchkin, B. Schmerl, and D. Garlan, "Architectural Abstractions for Hybrid Programs," in *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering*, ser. CBSE '15. New York, NY, USA: ACM, 2015, pp. 65–74.

[5] D. Phan, J. Yang, D. Ratasich, R. Grosu, S. A. Smolka, and S. D. Stoller, "Collision Avoidance for Mobile Robots with Limited Sensing in Unknown Environments," in *Proc. of the 15th International Conference on Runtime Verification*, 2015.

[6] D. H. Stamatis and H. Schneider., *Failure Mode and Effect Analysis: FMEA from Theory to Execution*, 2nd ed. Milwaukee, Wisc: Amer Society for Quality, Jun. 2003.

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

[8] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "Contract-based Integration of Cyber-physical Analyses," in *Proceedings of the 14th International Conference on Embedded Software*, ser. EMSOFT '14. New York, NY, USA: ACM, 2014, pp. 23:1–23:10.

[9] M. Bartoletti, T. Cimoli, P. Di Giamberardino, and R. Zunino, "Vicious circles in contracts and in logic," *Science of Computer Programming*, vol. 109, pp. 61–95, Oct. 2015.

[10] A. Rajhans, A. Bhave, I. Ruchkin, B. Krogh, D. Garlan, A. Platzer, and B. Schmerl, "Supporting Heterogeneity in Cyber-Physical Systems Architectures," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3178–3193, Dec. 2014.

[11] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems*," *European Journal of Control*, vol. 18, no. 3, pp. 217–238, 2012.

[12] P. Derler, E. A. Lee, S. Tripakis, and M. Torngren, "Cyber-physical System Design Contracts," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ser. ICCPS '13. New York, NY, USA: ACM, 2013, pp. 109–118.

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

[14] S. Tripakis, C. Stergiou, C. Shaver, and E. A. Lee, "A Modular Formal Semantics for Ptolemy," *Mathematical Structures in Computer Science. Accepted for publication*, 2012.

[15] V. Subramonian and C. Gill, "Towards Integrated Model-Driven Verification and Empirical Validation of Reusable Software Frameworks for Automotive Systems," in *Model-Driven Development of Reliable Automotive Services*. Springer Berlin Heidelberg, 2008, pp. 118–132.

[16] A. Davare, D. Densmore, L. Guo, R. Passerone, A. L. Sangiovanni-Vincentelli, A. Simalatsar, and Q. Zhu, "metroII: A Design Environment for Cyber-physical Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1s, pp. 49:1–49:31, 2013.

[17] A. Bhave, "Multi-View Consistency in Architectures for Cyber-Physical Systems," Ph.D. dissertation, Carnegie Mellon University, Dec. 2011.

[18] G. Simko, D. Lindecker, T. Levendovszky, S. Neema, and J. Sztipanovits, "Specification of Cyber-Physical Components with Formal Semantics Integration and Composition," in *Model-Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, Jan. 2013, pp. 471–487.

[19] Sandeep Neema, Ted Bapty, and Janos Sztipanovits, "Multi-Model Language Suite for Cyber-Physical Systems," Institute for Software Integrated Systems, Vanderbilt University, Tech. Rep., 2013.

[20] J. Sztipanovits, G. Karsai, S. Neema, and T. Bapty, "The Model-Integrated Computing Tool Suite," in *Model-Based Engineering of Embedded Real-Time Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Jan. 2010, no. 6100, pp. 369–376.

[21] Y. Zhou and E. A. Lee, "A Causality Interface for Deadlock Analysis in Dataflow," in *Proceedings of the 6th ACM &Amp; IEEE International Conference on Embedded Software*, ser. EMSOFT '06. New York, NY, USA: ACM, 2006, pp. 44–52.

[22] 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*, ser. EMSOFT '11. New York, NY, USA: ACM, 2011, pp. 13–22.

[23] G. Bergmann, I. Roth, G. Varro, and D. Varro, "Change-driven model transformations," *Software & Systems Modeling*, vol. 11, no. 3, pp. 431–461, Mar. 2011.

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

[25] A. Qamar, "Model and Dependency Management in Mechatronic Design," Ph.D. dissertation, KTH Sweden, Stockholm, Sweden, 2013.

[26] A. Radjenovic and R. Paige, "The Role of Dependency Links in Ensuring Architectural View Consistency," in *Seventh Working IEEE/IFIP Conference on Software Architecture, 2008. WICSA 2008*, Feb. 2008, pp. 199 –208.

[27] Paul Gao, Russel Hensley, and Andreas Zielke, "A road map to the future for the auto industry," *McKinsey Quarterly*, Oct. 2014.

[28] A. Iliaifar, "LIDAR, lasers, and logic: Anatomy of an autonomous vehicle," 2013. [Online]. Available: http://www.digitaltrends.com/cars

[29] 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 *2010 IEEE Symposium on Security and Privacy (SP)*, May 2010, pp. 447–462.

[30] 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 *Proc. of the 20th USENIX Conference on Security*, Berkeley, CA, USA, 2011, pp. 6–22.

[31] H. Fawzi, P. Tabuada, and S. Diggavi, "Secure Estimation and Control for Cyber-Physical Systems Under Adversarial Attacks," *IEEE Transactions on Automatic Control*, vol. 59, no. 6, pp. 1454–1467, Jun. 2014.

[32] F. G. Marmol and G. M. Perez, "Towards pre-standardization of trust and reputation models for distributed and heterogeneous systems," *Computer Standards & Interfaces*, vol. 32, no. 4, pp. 185–196, Jun. 2010.

[33] A. Bhave, B. Krogh, D. Garlan, and B. Schmerl, "View Consistency in Architectures for Cyber-Physical Systems," in *2011 IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS)*, Apr. 2011, pp. 151 –160.

[34] 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*, vol. 79, no. 3, pp. 383–401, May 2013.

[35] 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*, 2011, pp. 361–366.

[36] I. Ruchkin, A. Rao, D. De Niz, S. Chaki, and D. Garlan, "Eliminating Inter-Domain Vulnerabilities in Cyber-Physical Systems: An Analysis Contracts Approach," in *Proc. of the First ACM Workshop on Cyber-Physical Systems Security & Privacy (CPS-SPC)*, Denver, Colorado, 2015.

[37] R. Eramo, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio, "A model-driven approach to automate the propagation of changes among Architecture Description Languages," *Software & Systems Modeling*, vol. 11, no. 1, pp. 29–53, Jul. 2012.

[38] Z. Diskin, "Algebraic Models for Bidirectional Model Synchronization," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Vlter, Eds. Springer Berlin Heidelberg, 2008, no. 5301, pp. 21–36.

[39] D. W. S. Clair, *Controller Tuning and Control Loop Performance*, 2nd ed. Newark: Straight-Line Control Co., Jan. 1990.

[40] D. Borwein and J. Borwein, "Fixed point iterations for real functions," *Journal of Mathematical Analysis and Applications*, vol. 157, no. 1, pp. 112–126, May 1991.

[41] H. Simon, "Rational choice and the structure of the environment," *Psychological Review*, vol. 63, no. 2, pp. 129–138, 1956.

[42] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories: From an Abstract DavisPutnamLogemannLoveland Procedure to DPLL(T)," *J. ACM*, vol. 53, no. 6, pp. 937–977, Nov. 2006.

[43] J. Holland, "Genetic Algorithms and the Optimal Allocation of Trials," *SIAM Journal on Computing*, vol. 2, no. 2, pp. 88–105, Jun. 1973.