MODELS: G: Architectural and Analytic Integration of Cyber-Physical System Models

Ivan Ruchkin
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA 15213
iruchkin@cs.cmu.edu

***Abstract*—Modeling methods for Cyber-Physical Systems (CPS) originate in various engineering fields, and are difficult to use together due to their heterogeneity. Inconsistencies between mutually oblivious models and analyses often lead to implicit design errors, which may cause catastrophic failures of critical CPS. Such consistency issues are not fully solved by the state-of-the-art integration methods, which lack generality, formal guarantees, and effectiveness. To overcome these limitations and achieve better integration, this paper outlines a two-level integration approach based on architectural views and analysis contracts. In particular, we propose languages and algorithms to specify and verify important integration properties, such as correct analysis execution and rich consistency of architectural views. Based on the results to date, this approach shows promise in detection and prevention of implicit errors, which would be difficult to discover and fix otherwise.**

I. Introduction

Modern software systems are becoming more distributed, autonomous, and embedded in the physical world. Such systems are increasingly important because they offer socioeconomic benefits beyond classic embedded systems. For example, self-driving cars promise dramatic reductions in accident rates [1]. Such systems are often called *Cyber-Physical Systems* (CPS)[1] because they combine software with complex physical dynamics. CPS are difficult, but important, to engineer correctly. To tackle complex analog and digital processes, CPS design and quality assurance rely on model-driven engineering methods from various scientific fields, such as artificial intelligence, control theory, and mechanical design. These diverse methods and models may be hard to combine for one system's design [2].

Failure to combine multidisciplinary modeling methods properly may lead to miscommunication and inconsistencies, which turn into design errors and ultimately system failures [3]. Although partial solutions to such issues exist, the CPS community has not yet developed general, effective, and practical ways to integrate CPS modeling methods [4]. We term these issues the *Problem of Modeling Methods Integration (MMI)* – integration problems between heterogeneous CPS modeling methods that introduce errors into designs. One example of the MMI problem is a recent GM ignition switch recall that was caused by an unexpected interaction between the mechanical and electrical designs of the ignition switch, leading to loss of lives and expensive recalls [5].

One of the approaches to the MMI problem focuses on creating and relating appropriate component-based abstractions for models used in each CPS modeling method. These abstractions, called *views*, are based on annotated graphs of components and connectors [6]. This approach, known as *architectural*, takes advantage of flexibility of graph annotations to support diverse models and a variety of consistency verification methods. Despite its advantages, the architectural approach currently has several significant limitations. First, model-view relations are *informal* and require substantial manual effort to be created and updated throughout the engineering process. Another limitation is that consistency is fragile due to frequent *changes to models* (often made by model transformations and analytic tools, and we call these changes *analyses*). Finally, consistency properties have *limited expressiveness* confined solely to the architectural level, incapable of expressing richer properties.

This paper introduces a *two-level integration approach* that overcomes the above limitations. One level of abstraction is the *architectural view level* that represents the aspects of each CPS model that are relevant for integration. The other level is the *analysis level* that manages algorithms and procedures that change models and infer information from them. Combining these two levels leads to a holistic treatment of CPS model integration. Specifically, this research makes the following contributions to the field of cyber-physical systems:

- **A** *Formalization of, and automated support for, CPS model-view relations* – a formalism and algorithms for automated creation and updating of model-view relations.
- **B** *A framework for automated analysis-driven change in CPS models* – a language for specifying analysis dependencies with formal contracts, and an algorithm for sound ordering and execution of analyses.
- **C** *A method for domain-specific specification and verification of CPS model consistency and analytic soundness* – a language to express model-specific CPS properties, and an algorithm to verify such properties.

All three contributions are being implemented in popular architecture modeling languages and environments, and vali-

[1] They are also referred to as autonomous robotics and mechatronics.

dated on realistic academic and industrial case studies to show effectiveness and generality of the approach.

The next section discusses existing work related to the MMI problem. Sec. III details the MMI problem, and Sec. IV presents the two-level approach to MMI. The last section discusses research completed to date and future directions.

II. Background and Related Work

A *modeling method* is a cohesive set of formalisms, algorithms, and processes to represent, design, and analyze a system towards satisfaction of certain properties. CPS engineering combines various modeling methods to address systemic properties like safety, stability, schedulability, security, and others.

The related work can be split in two categories: individual CPS modeling methods that can be supported by an integration approach, and CPS model integration approaches that are seen as alternative solutions to the MMI problem.

A. Modeling Methods for Cyber-Physical Systems

Since CPS engineering revolves around the boundary between discrete digital and continuous physical worlds, one important characteristic of modeling methods is their treatment of continuous phenomena, such as time and space. On the one hand of this spectrum are classic software engineering models like statecharts and process algebras [7]. These models are easier to compose and verify using techniques like model checking. However, their treatment of continuous quantities is limited and not satisfactory for CPS [8]. A special place among discrete formalisms is taken by *architectural models*, such as UML and AADL [9], that have flexible elements (profiles, types, and annexes) for specialization and extension, making these models a promising foundation of CPS integration.

On the other hand of the spectrum are classic control models written in differential or difference equations [10] and their engineering counterparts like Simulink [11]. These models are well-suited for traditional control settings (e.g., physical process control), but it is increasingly difficult to apply such models to complex intelligent systems. For instance, it is challenging to analyze behavioral planning in signal-flow control models. To compensate for such drawbacks, hybrid systems (e.g., hybrid automata [12]) aim to combine discrete and continuous dynamics in one model. Although hybrid systems have enjoyed success in symbolic and numeric analysis, hybrid models are notoriously complex and have limited scalability, and thus have to be applied selectively in practice.

B. Integration Approaches

Currently there are two major ways of addressing the MMI problem. One is to create a single language or formal system with universal semantics that would serve as a lingua franca of all CPS modeling methods. Such solutions typically lead to complex descriptions and quick explosion of state space [13], thus becoming inapplicable to large systems. The second way, which we review below, is to preserve the diversity and heterogeneity of models by relating some of their aspects.

Some integration approaches focus on structural elements of models that they integrate. For example, models can be composed as components with formal interfaces and contracts [14]. Component contracts work well for composition of system parts, but are often inappropriate for cross-cutting qualities like safety and security. Another idea is logical integration through metamodels and formal semantics, as in OpenMETA and CyPhyML [15]. These frameworks provide strong theoretical guarantees when a metamodel is known, but there is little guidance for models that do not have explicit metamodels, or when models or metamodels undergo frequent change.

Other related approaches integrate behavioral aspects of models. Unfortunately, it is hard to create a comprehensive integration framework based only on behavior. One example is Ptolemy II [16] – a rich simulation platform that can co-execute heterogeneous computation models, but sacrifices completeness guarantees. In contrast to Ptolemy, the behavior relations approach [17] uses mappings between behavior traces of heterogeneous models. This approach is practical when behaviors are well-known and easily specified in relatively limited models, but does not scale to realistic systems.

Our work unites the research strands of architectural CPS consistency and analysis contracts into a single approach. Prior work on architecture-based CPS model consistency defined a vocabulary of CPS terms, and used them to check structural consistency [6]. However, it did not address creation and maintenance of model-view relations, leaving it to error-prone and expensive manual effort. Prior work on analysis contracts offers limited, bounded verification with Alloy [18]. In contrast, our work provides an extensible contracts language with formally defined semantics and verification mechanisms.

III. Problem: Modeling Method Integration

The *Modeling Method Integration (MMI) problem* is caused by integration errors between CPS modeling methods, leading to substantial operational losses. Consider for example a self-driving car that autonomously navigates through an urban environment, supported by intelligent infrastructure like smart traffic signals. All aspects of the car design – electrical, mechanical, thermal, power, control, and communication – need to be properly aligned and, ideally, verified before manufacturing in order to ensure safety and acceptable performance of the system. If modeling methods that address these aspects contradict each other, it is likely that difficult-to-find discrepancies would be introduced into the design.

Although some parts of the MMI problem have been addressed, several important issues have not been resolved. One of them is the informality of relations between heterogeneous models and their integration representations, such as views. These relations are easier to establish and maintain for component models, such as Simulink and Verilog. However, some CPS models do not have native support for components. For example, hybrid programs [10] are hard to componentize because they formally are sequences of non-deterministic discrete jumps and continuous evolutions. Usually, such informal

model-view relations are handled by an engineer's judgment and insight, which is effort-intensive and error-prone.

Another aspect of the MMI problem is that system designs undergo constant change. It is increasingly common to use automated tools and algorithms to transform, analyze, and augment models. These *analyses* are based on theories and methods from diverse engineering and scientific domains. For example, in the domain of real-time processor scheduling, thread-to-processor allocation via bin-packing and processor frequency scaling [19] are analyses that derive an optimized architecture of a real-time system. When many analyses change models, it is impractical to re-establish consistency manually: for every change, many properties may need to be re-verified. Besides, analyses often make implicit assumptions about the system and its environment, and it's important to verify these assumptions – otherwise analytic results may be incorrect.

Finally, some model consistency properties and analytic assumptions need to be expressed in domain- and model-specific behavioral terms (e.g., current battery cell charge [19]). Usually such terms are too semantically low-level to be fully defined in architectural views. The reason for that is complex model semantics, bringing which into a view would defeat the original purpose of the abstraction. As a result, verification of complex integration properties has to be done manually or with ad-hoc automation.

To clarify the expressiveness issue, consider an assumption of a frequency scaling analysis – behavioral deadline-monotonicity of threads on each processor [19]. That is, if a thread preempts another, it should have an earlier deadline. To express this statement logically, in addition to the architectural concepts (threads, processors, bindings, and deadlines), we refer to the non-architectural concept of thread preemption. In this case, preemption is modelled as a logical predicate $\mathsf{CanPrmpt}$ over a pair of threads that holds iff the first thread preempts the second at the current moment of time. The assumption can be written as:

$$\begin{aligned} &\forall t_1, t_2 \in \mathbb{T} \cdot t_1 \neq t_2 \wedge \mathsf{CPUBind}(t_1) = \mathsf{CPUBind}(t_2) : \\ &\mathsf{G}\ (\mathsf{CanPrmpt}(t_1, t_2) \implies \mathsf{Dline}(t_1) < \mathsf{Dline}(t_2)), \end{aligned} \tag{1}$$

where $\mathbb{T}$ is the set of all threads in the system; $\mathsf{CPUBind}(t)$ is the processor that thread t executes on; $\mathsf{G}\ (P)$ is a global modality in linear temporal logic (LTL) [20] requiring that predicate P holds at all times; and $\mathsf{Dline}(t)$ is the offline deadline of thread t. This statement has two parts: a first-order statement, and an LTL predicate. We need to systematically and scalably interpret and verify statements like these, and existing approaches do not support combining logics and abstraction levels in one language.

A. Challenges

The MMI problem also appears in a variety of engineering settings outside of CPS. For example, traditional software engineering uses several notations and reasoning styles (e.g., various static and dynamic analyses), and sometimes their interoperability is required [21]. In the CPS context this problem becomes *particularly challenging* for two reasons:

1) No single discipline in CPS owns a full solution. Heterogeneous methods and models need to reconcile conflicting paradigms to achieve practical integration.
2) Correct integration often depends upon satisfaction of implicit assumptions. Many engineers aren't aware of assumptions in other disciplines. For instance, a programmer may rely on functionally correct code, without considering bit flips from radiation inside processing units. Discovering such assumptions and their violations is therefore a difficult task with unpredictable outcomes.

Nevertheless, it is possible to improve CPS modeling method integration. The next section presents a two-level approach to the MMI problem.

IV. Approach: Architectural and Analytic Integration

The overall scheme of the proposed integration approach (Fig. 1) shows how we add two levels of abstraction on top of existing CPS models. Consider a pair of heterogeneous models to be integrated. These models are not completely independent, and there exists some relationship between them. This relationship is, however, too complex to express or verify directly. Instead, as a first step, we create architectural view abstractions with integration-relevant information for each model. The views need to be general enough to accommodate different formalisms (e.g., hybrid programs or Simulink) and CPS application domains (e.g., avionics or transportation).

In the second step, we add support for systematic change of models and views by making analyses first-class entities. Analyses read and change views, which propagate these changes to models. Analytic applicability is often limited by assumptions. If assumptions of an analysis are not satisfied, this analysis may produce an incorrect result and should not be executed. Some analyses have guarantees – statements that hold after an analysis is executed. If guarantees of an analysis do not hold, the changes from this analysis should be rolled back.

Thus, the two-level approach uses architectural and analytic constructs to integrate CPS models. In the rest of the section, we will discuss each level in more detail.

A. View Level

The view level is used to mediate complex interaction between analyses and models. An *view* is an architectural model with components, connectors, and properties that are defined in a particular *architectural style* – a custom vocabulary of architectural elements. Using views for integration requires creating and maintaining two kinds of relations: *view-view* and *model-view*. The former is more straightforward because views are specified in architecture description languages that have generally homogeneous structure of components and connectors. Therefore this relationship can be maintained using a number of well-established techniques such as model transformation and synchronization [22].

Model-view relations, on the other hand, require a more sophisticated link between views and potentially less structured models. A view needs to abstract out some semantics

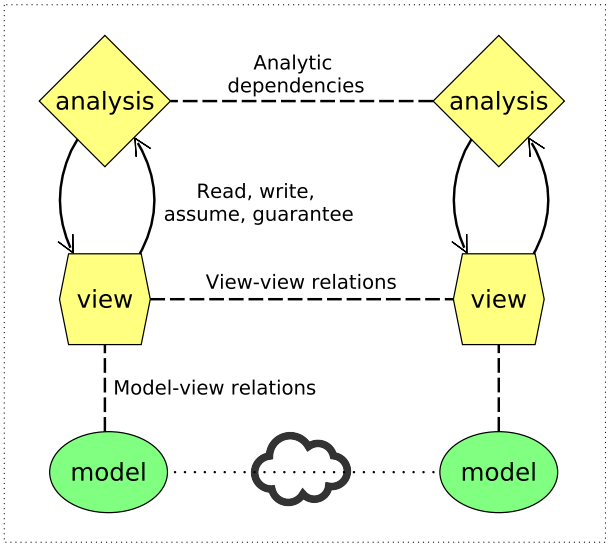

Fig. 1. Architectural and analytic approach to CPS model integration.

and structure of a model. We use view-model mappings and transformations to automatically support model-view relations. To be effective, these mappings and transformations have to be customized to each formalism. We advantage of the flexibility of *architectural styles* (vocabularies of component and connector types that describe domain-specific architectural properties) to support customization and tailor transformation algorithms. For instance, the hybrid program view [23] describes which component "owns" each variable in a hybrid program.

Another function of the view level is establishing consistency between models through their views. That is done by specifying and verifying consistency rules, which are specified as constraints over multiple views and can be verified with constraint solving [19]. Properties that contain model-specific terms like CanPrmpt (see Eq. 1) require more sophisticated verification methods, such as model checking or theorem proving. Our approach provides a general scheme to incorporate such verification into multi-view consistency checking.

B. Analysis Level

The analysis level automates sound execution of model-based analyses, which depends on correct ordering and satisfaction of assumptions and guarantees. We provide a language of *analysis contracts* that facilitates soundness checking. Every analysis is accompanied by its contract C that specifies inputs I, outputs O, assumptions A, and guarantees G of the analysis, in short $C \equiv (I, O, A, G)$.

Correct analysis ordering is one where all analyses are in order of their dependencies. For example, if analysis A_1 depends on analysis A_2 (i.e., one of A_2 outputs is one of A_1 inputs), then A_2 should be executed before A_1. For instance, a CPU scheduling analysis, which determines the voltage required by CPUs, should be followed by a battery design analysis, which uses the voltage as a requirement. A sound sequence of analyses is built by creating an analysis dependency graph and selecting any topological ordering that ends with the desired analysis. The only exception is when there are cyclical dependencies, which requires more sophisticated methods of dependency resolution [24]. Assume/guarantee verification is done in the same way as consistency property verification on the view level.

The integration approach illustrated in Fig. 1 has the following advantages:

- By combining architectural and analytic perspectives, it incorporates a large body of prior work and offers diverse opportunities to integrate modeling methods.
- Important yet subtle integration properties that span multiple domains and formalisms can now be expressed and verified at an appropriate abstraction level, thanks to Contribution **C**.
- The bottom-up philosophy behind the approach enables adding new modeling methods without up-front planning, in contrast with existing top-down approaches that are dependent on specific formalisms.

The approach also has several limitations:

- It may be difficult to provide integration for incomplete and informal models, which often require unique formalizations and algorithms.
- The scalability of formal verification techniques depends on carefully designed abstractions and high-quality tools, which may not always be available.
- It may be impractical to invest into an up-front formal integration of views and analyses in CPS projects that do not use many heterogeneous modeling methods.

To summarize, the two-level approach promises to make CPS MMI more effective and less costly. The next section presents preliminary work that supports this claim.

V. Results to Date

This section describes two major results: architectural views for hybrid programs and the analysis contracts framework.

A. Architectural Views for Hybrid Programs

The hybrid program modeling method, based on hybrid programs (HP) and differential dynamic logic (dℒ) [10], is particularly difficult to integrate with other modeling methods, in part due to HP expressiveness and lack of support for components. Each hybrid program contains highly intertwined fragments of various concerns, from the physical environment to the type of sensing and actuation. Specification of interacting components, e.g., a robot and an obstacle, is also dispersed through the body of a HP. Leading to poor HP modularity, these issues inspired work on creating an architectural abstraction for hybrid programs [23].

To incorporate hybrid programs into our approach, we defined an architectural HP view, which defines how architectural elements are transformed into hybrid programs. This view enabled high-level design and reasoning about HPs and eliminated manual effort of maintaining model-view consistency. An architectural HP view, which contains actors HPA, composers CPR, and connectors HPC, can be transformed

into a single HP via transformation functions customized in the view. Given a view, it is possible to reuse its parts and express its properties in $\mathsf{d}\mathcal{L}$, thus the level of abstraction is elevated from individual statements to components and systems. We have also defined an analysis to check whether a view has a proper structure, e.g., whether an actor violates causality by manipulating variables of another actor outside of connectors.

Implemented as a plugin to AcmeStudio [25] (towards Contribution **A**), architectural abstractions for hybrid programs showed feasibility of automated maintenance of model-view relationship. This work opens future research directions for theoretical study of HP view analyses, and, more broadly, frameworks for automated relating of models and views.

B. Analysis Contracts Framework

Our work investigated theoretical and practical aspects of using *analysis contracts* for integration (towards Contribution **B**) [19]. In the theoretical part, analysis contracts were formalized as quadruples C (see Sec. IV above for details). The semantics of these contracts was described over *verification domains* – collections of sets and functions that capture the essential elements of a technical domain (such as battery design). In terms of practical aspects, the ACTIVE tool [26] was designed and implemented[2] to support correct execution of analyses in OSATE2 [27] – an AADL modeling environment.

This research showed that analysis contracts are suitable for detection and prevention of integration errors in several domains: threads scheduling, battery scheduling [19], sensor trustworthiness, reliability, and control [28]. Through this work we showed improvements in effectiveness and cost-efficiency of CPS modeling method integration and, more generally, feasibility of unified reasoning about model-model consistency and model transformation/analysis in the CPS domain.

To summarize, this paper outlined the two-level integration approach for CPS modeling methods. The next steps in this research are finalizing the design of the integration property language (Contribution **C**), and conducting validation case studies of model integration in realistic academic and industrial cyber-physical systems, such as NASA Europa Orbiter[3] and Carnegie Mellon CoBot[4]. Our long-term research vision extends beyond integrating models and analyses, to incorporating heterogeneous datasets and humans into a unified CPS integration framework [29].

ACKNOWLEDGEMENTS

This work is funded and supported by the National Science Foundation under Grant CNS-0834701, by the National Security Agency, and 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.

[2] Available at github.com/bisc/active

[3] nasa.gov/europa

[4] cs.cmu.edu/~coral/projects/cobot

REFERENCES

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

[2] P. Derler, E. A. Lee, and A. L. Sangiovanni-Vincentelli, "Addressing Modeling Challenges in Cyber-Physical Systems," University of California, Berkeley, Tech. Rep. UCB/EECS-2011-17, Mar. 2011.

[3] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang, "Toward a Science of Cyber-Physical System Integration," *Proc. of the IEEE*, 2012.

[4] M. Wolf and E. Feron, "What Don't We Know About CPS Architectures?" in *Proc. of DAC*, 2015, pp. 80:1–80:4.

[5] A. Valukas, "Report to Board of Directors of General Motors Company Regarding Ignition Switch Recalls," Jenner & Block, Tech. Rep., 2014.

[6] 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.

[7] J. Magee and J. Kramer, *Concurrency: State Models & Java Programs*. Wiley, Apr. 1999.

[8] E. A. Lee, "CPS Foundations," in *Proceedings of the 47th Design Automation Conference*. New York, NY, USA: ACM, 2010.

[9] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*, 1st ed. Upper Saddle River, NJ: Addison-Wesley, 2012.

[10] A. Platzer, "Differential Dynamic Logic for Hybrid Systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, Aug. 2008.

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

[12] R. Alur, T. A. Henzinger, and H. Wong-toi, "Symbolic Analysis of Hybrid Systems," in *Proc. of the IEEE CDC*, 1997.

[13] R. Mateescu, "Model Checking for Software Architectures," in *Software Architecture*. Springer Berlin Heidelberg, 2004, no. 3047, pp. 219–224.

[14] 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.

[15] 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*, Grenoble, France, 2014.

[16] C. Ptolemaeus, *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, Sep. 2013.

[17] A. Rajhans and B. H. Krogh, "Compositional Heterogeneous Abstraction," in *Proc. of HSCC*. ACM, 2013, pp. 253–262.

[18] M.-Y. Nam, D. de Niz, L. Wrage, and L. Sha, "Resource allocation contracts for open analytic runtime models," in *Proc. of EMSOFT*, 2011.

[19] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "Contract-based Integration of Cyber-physical Analyses," in *Proceedings of EMSOFT*. New York, NY, USA: ACM, 2014, pp. 23:1–23:10.

[20] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science, 1977*, Oct. 1977, pp. 46–57.

[21] D. D. Ruscio, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio, "Model-Driven Techniques to Enhance Architectural Languages Interoperability," in *FASE*, 2012.

[22] H. Giese and R. Wagner, "Incremental Model Synchronization with Triple Graph Grammars," in *Proc. of MoDELS*, 2006.

[23] I. Ruchkin, B. Schmerl, and D. Garlan, "Architectural Abstractions for Hybrid Programs," in *Proc. of CBSE*, 2015.

[24] I. Ruchkin, B. R. Schmerl, and D. Garlan, "Analytic Dependency Loops in Architectural Models of Cyber-Physical Systems," in *ACES-MB at MODELS 2015*, Ottawa, Canada, 2015.

[25] B. Schmerl and D. Garlan, "AcmeStudio: Supporting Style-Centered Architecture Development," in *Proc. of ICSE*, 2004.

[26] 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.

[27] P. Feiler, L. Wrage, J. Delange, and J. Siebel, "OSATE2," 2015, github.com/osate. [Online]. Available: https://github.com/osate

[28] 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 CPS-SPC*, Denver, CO, 2015.

[29] I. Ruchkin, "Integration Beyond Components and Models: Research Challenges and Directions," in *Proc. of the Third Workshop on Architecture Centric Virtual Integration (ACVI)*, Venice, Italy, 2016.