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, this paper proposes languages and algorithms to specify and verify important integration properties, such as correct analysis execution and rich consistency of architectural views. According to the results to date, this approach shows promise in detection and prevention of implicit errors, which would be difficult to fix otherwise.**

I. Introduction

Modern software systems are growing more distributed, autonomous, and embedded in physical world [1]. Such systems are increasingly important because they offer socioeconomic benefits beyond classic embedded systems. For example, fully automated self-driving cars promise dramatic reductions in the accident rate [2]. Such systems are often called *Cyber-Physical Systems* (CPS)[1]because they are combine software with complex physical dynamics.

Safety-critical 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 from various scientific and engineering fields, such as artificial intelligence, control theory, and mechanical design. This diversity of methods leads to complex and heterogeneous models that are hard to combine for one system's design. For example, at least six distinct models of computation may need to coexist in a single system model [3].

Failure to combine multidisciplinary modeling methods properly may lead to miscommunication and inconsistencies, which turn into design errors and ultimately system failures [4]. In this paper, such issues are generally called the *Problem of Modeling Methods Integration (MMI)* – absence of integration leading to erroneous designs. Although partial solutions to the MMI problem exist, CPS community has not yet developed general, effective, and practical ways to integrate CPS modeling and design methods [5]. As a result, safety-critical CPS are prone to implicit errors that take a substantial amount of time, effort, and funds to discover and fix. For example, the GM ignition switch recall was caused by an unexpected interaction between the mechanical and electrical designs of the ignition switch, leading to failures, loss of lives, and expensive recalls [6].

One of the approaches to the MMI problem focuses on choosing an appropriate component abstraction (a*"view"*) for each CPS formalism, using annotated graphs as an underlying notation [7]. This approach takes advantage of flexibility of graph annotations to support custom 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 *algorithmic changes to models* (we call these algorithms *analyses*). Finally, consistency properties have *limited expressiveness* confined solely to the architectural level, incapable of expressing richer properties.

This paper introduces the *two-level integration approach* that overcomes the above limitations. One level of abstraction is the *architectural view level* that represents the aspects of each model that are relevant for integration. The other level is the *analysis level* that focuses on algorithms that change models and infer information from them. Combining these two levels leads to a holistic treatment of CPS modeling integration issues. In particular, this research makes the following major 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, even for models that are not structured based on components, e.g., hybrid programs [8]. Preliminary work (Sec. V) shows that automation of model-view relations is feasible and provides auxiliary modeling benefits, such as component-based analysis and reuse of models.
- **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. Preliminary

[1]Such systems are also sometimes referred to as autonomous robotics and mechatronics.

work shows that this framework prevents modeling errors that occur due to incorrect ordering of analysis execution.

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. This language incorporates model-specific terms beyond the architectural level, and can used in Contributions **A** and **B**. Preliminary work demonstrates that this language can detect integration errors that would not have been detectable purely at the level of architectural views.

All three elements of the approach will be implemented in popular architecture modeling languages and environments, and validated on realistic academic and industrial case studies to show effectiveness and generality of the approach.

Next section gives more background on CPS modeling methods and existing work that addresses the MMI problem. Sec. III contains more detail on the MMI problem, and Sec. IV presents the two-level approach to MMI. The last section discusses research completed to date and concludes the paper.

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

A. Modeling Methods for Cyber-Physical Systems

Modeling methods for CPS differ depending on a scientific field where they originate. Since CPS engineering revolves around the boundary between discrete digital and continuous physical worlds, one of the most important characteristics of modeling methods is their treatment of potentially continuous phenomena, such as time and space. On the one hand of this spectrum are classic software engineering models like statecharts and process algebras [9]. These models are easier to compose and verify using such techniques as model checking. However, their treatment of continuous quantities is very limited and not satisfactory for CPS [10] A special place among discrete formalisms is taken by *architectural models* such as UML, SysML, and AADL [11]. These models include flexible elements such as profiles, types, and annexes to allow specialization and extension, making architectural models particularly suitable to be a foundation of CPS integration.

On the other hand of the spectrum are classic control models written in differential or difference equations [8] and their engineering counterparts like Simulink [12]. Although these models are well-suited for traditional control settings like physical process control, it is increasingly difficult to apply such models to complex intelligent systems. For instance, it is challenging to analyze behavioral planning, which in important to CPS, in signal-flow control models.

The field of hybrid systems aims to combine discrete and continuous system dynamics. A common model is a hybrid automaton [13] that combines continuous differential equations with discrete state jumps. 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. Another limitation is that hybrid models lack typical modularity mechanisms [14], and hence are difficult to combine with common models in software and systems engineering.

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 hopefully serve as a lingua franca of all CPS modeling methods. Such solutions often lead to very complex descriptions and quick explosion of state space [15], thus becoming inapplicable to large systems. The second way is to preserve the diversity and heterogeneity of models through model integration. The rest of this section reviews several such frameworks.

Software and systems engineering have a significant heritage in compositional methods, some of which have been adapted to CPS. One strand of research composes components using contracts [16]: each component has an interface with a formal contract. This approach works well for distributed development of systems, but is often inappropriate for cross-cutting qualities like safety and security. Another way to compose system parts is by unifying components through their behavior relations [17]. This is practical when behaviors are known and easily specified, which, however, is not always the case for complex systems. The work in this paper uses well-known contract-based reasoning at a novel level of analyses.

Ptolemy II [18] is a platform for rich simulation of diverse models of computation. It can be used to co-simulate heterogeneous models like state machines, Petri nets, and timed automata. Although this is a promising integration method for hands-on engineering, it does not provide strong theoretical guarantees (like those behind formal verification). Another obstacle is that not every CPS model has a well-defined computation model: some models are more declarative than imperative, for example logical and functional models. The two-level integration approach incorporates modeling methods and verification regardless of how definable models of computation are.

Another approach to the MMI problem is based on logical integration through metamodels and formal semantics, such as in OpenMETA and CyPhyML [19]. This method provides strong theoretical guarantees and practical engineering support when a metamodel is known. However, there is little guidance for models that do not have explicit metamodels, or whose metamodels are too computationally complex and only partial integration is possible. Another limitation is that metamodel integration does not directly support verification of changes to

models. Our research, on the other hand, does not depend on explicit metamodels and provides structured change support that can augment metamodel integration.

Prior work on architecture-based CPS model consistency defined a vocabulary of CPS terms, and used it to check structural consistency [20]. It however does not address the question of how model-view relations are created and maintained, leaving it to error-prone and expensive manual effort. Prior work on analysis contracts offers limited, bounded verification with Alloy and focuses on openness of runtime models [21]. In contrast, our work on analysis contracts advances that state of the art through an extensible contracts language with theoretically defined semantics and verification mechanisms.

III. Problem: Modeling Method Integration

As mentioned in Sec. I, the *Modeling Method Integration (MMI) problem* is caused by a critical lack of integration between CPS modeling methods that leads to design errors and ensuing substantial operational losses. Consider for example a self-driving car that autonomously navigates through an urban environment, supported by intelligent infrastructure, such as smart traffic signals [22]. 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 invalidate each other, it is likely that difficult-to-find discrepancies would be introduced into the design.

Some parts of the MMI problem have been successfully addressed in related research. For example, architectural integration abstractions have been used to represent model structure and behavior, check consistency, and compose independent components [23]. Model integration properties can be expressed as parametric constraints over sets of views [24].

However, several important integration issues have not been resolved. One of them is the informality of relations between heterogeneous models and their integration-level representations (such as views). These relations may be easier to establish and maintain for component-based models, such as Simulink and Verilog. However, some CPS models do not have native support for components. For example, hybrid programs are difficult to componentize [8] because they formally are sequences of potentially non-deterministic discrete jumps and continuous evolutions. Traditionally, such fuzzy model-view relations are handled by an engineer's judgment and insight, which is an effort-intensive and error-prone way.

Another aspect of the MMI problem is that system designs undergo constant change. It is increasingly common to use automated tools and algorithms to analyze and augment models. We call such tools and algorithms *analyses*. 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 [25] are analyses that derive a more optimal architecture of a real-time system. If analyses change models locally, it is impractical to maintain consistency in a traditional way: for every local change, many global properties may need to be re-verified before another local change can be made. 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 not only in terms of architectural elements (components and connectors), but also in domain- and model-specific terms (e.g., current battery cell charge [25]) that are not defined in architectural models. Often such terms are too semantically low-level, and fully defining them in architectural views would be impractical because one would have to "import" the full detailed semantics of a model, thus defeating the purpose of integration abstractions.

To clarify the expressiveness issue, consider an assumption of the frequency scaling analysis – behavioral deadline-monotonicity of threads on each processor [25]. That is, if a thread preempts another, it should have a smaller 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 (implicitly modeled) current moment of time. Then 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) [26] requiring that predicate P holds at all times; and $\mathsf{Dline}(t)$ is the offline deadline of thread t. We need to systematically and scalably interpret and verify statements like the above, and the existing frameworks fall short of doing so.

A. Challenges

The MMI problem can be encountered in a variety of engineering settings outside of CPS. For example, traditional software engineering uses several notations and reasoning styles, and sometimes their interoperability is required [27]. It is however in the CPS context that this problem becomes *particularly challenging* due to several reasons:

- No single discipline in CPS owns a full solution. Heterogeneous methods need to, as least partially, reconcile conflicting paradigms to achieve practical integration.
- Correct integration often depends upon satisfaction of implicit assumptions. Many engineers do not challenge the assumptions of their discipline, and aren't aware of the assumptions in other disciplines. Discovering such assumptions and their violations is therefore a difficult task with unpredictable outcomes.

- Usually models and source code of safety-critical CPS cannot be found on popular collaborative websites such as GitHub. Conducting an evaluative case study of integration methods requires overcoming organizational and legal barriers to access protected systems.

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 two-level integration approach is shown in Fig. 1. Consider two heterogeneous models to be integrated. The models are not completely independent, and there exists some relationship between them. This relationship is however too complex to express or verify directly. Instead, 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).

To support systematic change of models, analyses are considered first-class entities. Analyses read and change views, which propagate the changes to models. Analyses often have their applicability limited by assumptions. If an analysis' assumptions are not satisfied, this analysis may produce an incorrect result should not be executed. Some analyses have guarantees – statements that should hold after an analysis is executed. If this is not the case, the changes made by this analysis should be rolled back. Since some analyses modify the same set of views, input-output dependencies arise and have to be resolved.

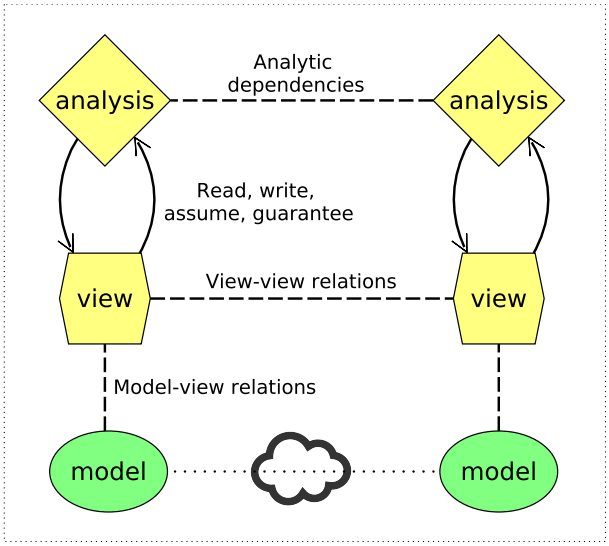

Fig. 1. Architectural and analytic approach to CPS model integration.

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 ADLs 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 [28] or synchronization [29].

Model-view relations, on the other hand, require a more special link between views and potentially less structured models. A view needs to abstract out some semantics and structure of a model. We use view-model mappings and transformations to automatically support model-view relations. These mappings and transformations have to be customized to the particular formalism in order to be effective. The approach takes advantage of the flexibility of architectural styles to support customization and tailor transformation algorithms. For instance, the hybrid program view [14] describes which component "owns" how each hybrid program variable.

Another function of the view level is establishing consistency between models through their views. That is done by specifying and verifying consistency rules, which take form of constraints over multiple views and can be verified with constraint solving [30]. Properties that contain model-specific terms like CanPrmpt (see Eq. 1) require more sophisticated verification methods, such as model checking or theorem proving.

B. Analysis Level

The analysis level automates sound execution of model-based analyses, which depends on correct ordering and satisfaction of assumptions and guarantees. The language of *analysis contracts* 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 go in order of their dependencies. For example, if analysis A_1 depends on analysis A_2[2], 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 for this method is when there are cyclical dependencies, which requires more sophisticated methods of dependency resolution. Assumption and guarantee verification is done in the same way as consistency property verification on the view level.

The two-level integration approach demonstrated 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.

[2]That is, one of A_2 outputs is one of A_1 inputs.

- Important but 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 is promising to make CPS modeling method integration more effective and less effortful. The next section presents preliminary evidence that supports this claim.

V. Results to Date

This section describes two significant 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 (d$\mathcal{L}$) [8], is particularly difficult to integrate with other modeling methods, in part due to HP expressiveness and lack of support for modularity. Each hybrid program contains fragments of various concerns, from the physical environment of the system to the type of sensing and actuation. These fragments are highly intertwined with each other across many statements of the program. 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 of hybrid programs [14].

To incorporate hybrid programs into the approach, we defined an architectural HP view, which defines how architectural elements are transformed into hybrid programs. That enabled high-level design and reasoning about HPs and at the same time eliminated manual effort of model-view consistency maintenance. An architectural HP view, which contains actors HPA, composers CPR, and connectors HPC, can be transformed into a single HP via transformation functions of CPR and HPC. Given a view, it is possible to reuse its parts and express its properties in d$\mathcal{L}$, thus the level of abstraction is elevated to components and systems from individual statements. I have also defined an analysis to check whether a view has a proper compositional structure, e.g., whether an actor violates the laws of causality by manipulating variables of another actor outside existing connectors.

This work on architectural abstractions for hybrid programs towards Contribution **A**, implemented as a plugin to AcmeStudio [31], demonstrated feasibility of automated maintenance of model-view relationship. It opens future research directions in construction of integration tools and investigation of theoretical soundness for HP view analyses.

B. Analysis Contracts Framework

This work investigated theoretical and practical aspects of using *analysis contracts* for integration (towards Contribution **B**) [25]. Theoretical goals were designing the syntax and semantics for contracts and devising algorithms that ensure sound execution of analyses. Practical goals included formalization of existing domains beyond the original one (thread scheduling) and creation of an extensible framework for analysis execution and contract verification.

Towards the theoretical goals, analysis contracts as quadruples C (see Sec. IV above for details). Their semantics is described over *verification domains* – collections of sets and functions that describe the essential elements of a technical domain. Towards the practical goal, the ACTIVE tool [32] was designed and implemented[3] to support correct execution of analyses in an AADL environment OSATE2 [33].

This research showed that analysis contracts are suitable for detection and prevention of integration errors in several domains: threads scheduling, battery scheduling [25], sensor trustworthiness, reliability, and control [34]. This work demonstrated the improvements in effectiveness and cost-efficiency in CPS modeling method integration.

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 case studies of model integration in realistic academic and industrial cyber-physical systems, such as NASA Europa Orbiter [4] and Carnegie Mellon Andy Rover [5]. Candidates for case studies would be systems with diverse modeling and analytic artifacts that can be used to validate effectiveness and generality of the approach.

Acknowledgements

I thank my advisor David Garlan for his guidance and support. I thank Dionisio De Niz, Sagar Chaki, Bradley Schmerl, Ashwini Rao, and other collaborators for contributing to this research. I am also grateful to anonymous reviewers for their helpful comments and suggestions.

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.

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

[4] nasa.gov/europa

[5] lunar.cs.cmu.edu

REFERENCES

[1] 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.

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

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

[4] 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," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 29–44, Jan. 2012.

[5] M. Wolf and E. Feron, "What Don'T We Know About CPS Architectures?" in *Proceedings of the 52Nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 80:1–80:4.

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

[7] P. Fradet, D. Metayer, and M. Perin, "Consistency Checking for Multiple View Software Architectures," *Software Engineering ESEC/FSE 99*, vol. 1687, pp. 410–428, 1999.

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

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

[10] E. A. Lee, "CPS Foundations," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 737–742.

[11] 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 Professional, Oct. 2012.

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

[13] R. Alur, T. A. Henzinger, and H. Wong-toi, "Symbolic Analysis of Hybrid Systems," 1997.

[14] 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.

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

[16] 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.

[17] A. Rajhans and B. H. Krogh, "Compositional Heterogeneous Abstraction," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '13. New York, NY, USA: ACM, 2013, pp. 253–262.

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

[19] 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 (FPS 2014)*. Grenoble, France: Springer, Apr. 2014.

[20] 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.

[21] 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.

[22] D. Millward, "Smart traffic lights to stop speeders," *The Telegraph*, May 2011. [Online]. Available: http://www.telegraph.co.uk/motoring/news/8521769/Smart-traffic-lights-to-stop-speeders.html

[23] 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.

[24] A. Rajhans, A. Bhave, S. Loos, B. Krogh, A. Platzer, and D. Garlan, *Using Parameters in Architectural Views to Support Heterogeneous Design and Verification*, 2011, submitted for publication.

[25] 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.

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

[27] D. D. Ruscio, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio, "Model-Driven Techniques to Enhance Architectural Languages Interoperability," in *Fundamental Approaches to Software Engineering*. Springer Berlin Heidelberg, 2012, no. 7212, pp. 26–42.

[28] M. Kessentini, H. Sahraoui, and M. Boukadoum, "Model Transformation as an Optimization Problem," in *Model Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, 2008, no. 5301, pp. 159–173.

[29] Z. Diskin, "Algebraic Models for Bidirectional Model Synchronization," in *Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, no. 5301, pp. 21–36.

[30] P. Kaufmann, M. Kronegger, A. Pfandler, M. Seidl, and M. Widl, "A SAT-Based Debugging Tool for State Machines and Sequence Diagrams," in *Software Language Engineering*. Springer International Publishing, Sep. 2014, no. 8706, pp. 21–40.

[31] B. Schmerl and D. Garlan, "AcmeStudio: Supporting Style-Centered Architecture Development," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 704–705.

[32] 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.

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

[34] 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.