Towards Integration of Modeling Methods for Cyber-Physical Systems

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

***Abstract*—Safety-critical Cyber-Physical Systems (CPS) are growing increasingly more distributed, autonomous, and embedded in our society. CPS engineering relies on modeling methods from different fields. Such methods are difficult to combine due to their complexity and heterogeneity. Inconsistencies between models and analyses can lead to implicit design errors, which lead to critical CPS failures. Existing approaches to CPS model integration fall short in terms of their flexibility, effectiveness, and formal guarantees. To overcome these limitations and achieve better integration, I propose an integration approach based on architectural views and analysis contracts. To enable my approach I develop a model-view consistency support framework, an analysis contracts framework, and a verification method for multi-model integration properties. I claim that my approach is feasible, more effective, and more cost-efficient than the existing ones. I plan to validate my claims on realistic industrial academic case studies of CPS modeling.**

I. Problem: Modeling Methods Integration

Modern software systems are growing increasingly more distributed, autonomous, and embedded in physical world. Such systems are important in science and technology because they offer socioeconomic benefits beyond classic embedded systems. For instance, self-driving cars promise dramatic reductions in the accident rate [1]. I will call systems with these characteristics *Cyber-Physical Systems* (CPS) because they are software-controlled and interact with complex physical world, although other names such as autonomous robotics and mechatronics are often used to describe such systems as well.

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 engineering fields, such as artificial intelligence, control theory, and mechatronics. This diversity of methods leads to complex and heterogeneous engineering processes that are hard to combine for one system's design. For example, at least six distinct models of computation may need to co-exist in a single system model [2].

Ad hoc integration between diverse modeling methods may lead to miscommunication and inconsistencies, which turn into design errors and ultimately system failures [3]. I will refer to such critical lack of integration as the *Problem of Modeling Methods Integration (MMI)*. 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 [4]. 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, in the General Motors ignition switch recall case it took years to discover an unexpected interaction between the mechanical and electrical designs of the ignition switch that lead to failures, loss of lives, and expensive recalls [5].

Some aspects of the integration problem have been successfully addressed in related research (see next section for details). However, several important integration issues have not yet been adequately addressed. One of them is the *informality of relations* between models and their integration-level representations (such as views). This relationship may be straightforward to establish and maintain for component-based models such as Simulink[1] and Verilog[2]. However, some CPS models do not have syntactic support for component, or their components are significantly different from the traditional object-oriented modularization. For example, it is difficult to componentize hybrid programs [6] which formally are sequences of non-deterministic discrete jumps and continuous evolutions. One way to deal with the absence of model structure is to rely on the engineer's judgment and insight to maintain the relationship to a view. However, this is effort-intensive and error-prone.

Another aspect of the problem is that system designs undergo constant change. It is increasingly common to use automated tools and algorithms to analyze models and derive their updated versions. I call such tools and algorithms *analyses*. Analyses are based on theories from specific engineering and scientific domains. For example, in the domain of processor scheduling one finds thread-to-processor allocation via bin-packing and processor frequency scaling [7] to derive an optimal architecture of a real-time system. Some analyses change models: frequency scaling adjusts the frequency property of processor components. For such analyses, it is impractical to re-establish consistency after every change: for every change many global properties may need to be re-verified before another change is executed. Besides, analyses often make implicit assumptions about the system or its environment, and it is important to verify these assumptions.

[1] mathworks.com/products/simulink
[2] verilog.com

Finally, some multi-model consistency properties and analytic assumptions need to be expressed not only in terms of architectural elements (like components and connectors), but also in domain-specific terms that are not defined in the architecture. 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 semantics of the model, thus defeating the purpose of integration abstractions. As the next section describes, current integration approaches lack a way to *express model-specific terms* without fully bringing the model semantics to the architectural level.

II. Related Work

CPS engineering combines various modeling methods to address systemic properties like safety, stability, schedulability, efficiency, security, and others. A *modeling method* is a cohesive set of formalisms, algorithms, and processes to represent, design, and analyze a system towards satisfaction of certain properties. Much recent work on CPS modeling has focused on formalisms and models. The related work can be split into two categories: individual CPS modeling methods that I build upon and try to incorporate into my 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 the scientific field from which 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. At one end of this spectrum are classic software engineering models like statecharts and process algebras [8]. These have support for composability and automated verification. However, their treatment of continuous phenomena is often too limited for CPS.

At the other end of the spectrum are models that include continuities, like differential and difference equations [6] and engineering tools like Simulink [9]. Although these models are well-suited for traditional control settings, it is increasingly difficult to apply such models to complex autonomous systems. For instance, it is challenging to analyze behavioral planning in signal-flow control models. The field of hybrid systems aims to reconcile discrete and continuous system dynamics. A common model is a hybrid automaton [10] that combines continuous evolutions along differential equations with discrete state jumps. Although this field has enjoyed success in symbolic and numeric computation for analysis of hybrid models, these models are notoriously complex, have limited scalability, and lack typical modularity mechanisms [11], which makes them difficult to combine with common software and systems engineering methods.

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 complex descriptions and an state space explosion, thus not scaling properly for large systems. The second way is to preserve the diversity and heterogeneity of models through model integration. I will review several such frameworks in the remainder of this section.

Software and systems engineering have a long heritage in compositional methods, some of which are being adapted to CPS. One strand of research uses component contracts for composition [12]: each component has an interface with a formal contract. This approach works well for distributed development of systems, but is often not appropriate for cross-cutting qualities like safety and security, since these qualities would need to be propagated to almost every component interface, leading to scalability issues. Another way to compose system parts is by unifying components through their behavior relations [13]. This is practical when behaviors are known and can be easily specified, which, however, is not always the case for complex systems. My work takes the ideas of contract-based reasoning to a novel level of model-based analyses.

Ptolemy II [14] is an environment for simulation of diverse models of computation like state machines, timed automata, and differential equations. Unfortunately, simulation does not provide strong theoretical guarantees like verification would, and not every CPS model has an explicit computation model. OpenMETA [15] is a platform based on formal logical semantic integration through metamodels. Despite its strong theoretical guarantees there is little guidance for models that do not have explicit metamodels. Another limitation is that metamodel integration does not directly support verification of changes to models. My research overcomes the limitations of these platforms.

A promising set of architectural approaches to the MMI problem focuses on choosing appropriate views for each CPS formalism using annotated graphs as an underlying formalism [16]. Flexibility of graph annotations enables customization for each model and a variety of possible consistency verification methods [17]. However, the architectural approach currently has several limitations. First, model-view relations are informal and require substantial manual effort to create and update throughout the engineering process. Another limitation is that consistency is fragile due to frequent algorithmic changes to models. Finally, consistency properties have limited expressiveness confined solely to the architectural level, incapable of expressing richer properties.

III. Proposed Solution: Multi-Level Architectural and Analytic Approach

My research aims to improve the state-of-the-art in CPS modeling method integration by employing a *multi-level approach* to the MMI problem. One abstraction is architectural views that represent model aspects that are relevant for integration. The other level is the analysis level that considers algorithms that change models and infer information from

them. Combining these two levels leads to a holistic and effective treatment of CPS modeling integration issues.

The overall scheme of my 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 (the cloud). However, this relationship is often too complex to express or verify directly. Instead, I create architectural view abstractions with integration-relevant information for each model. The views need to be general enough to accommodate different formalisms and CPS application domains.

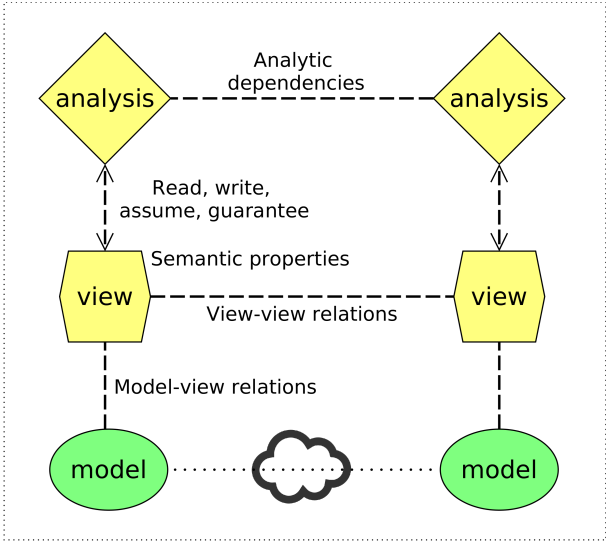

Fig. 1: The multi-level architectural and analytic approach.

To support systematic change of models I introduce analyses as part of the conceptual framework. Analyses read and change views, which propagate the changes to models. Analyses often make assumptions that must be satisfied for the analysis to be correct. For instance, frequency scaling is only applicable if the system is deadline-monotonic [7]. If an analysis' assumptions are not satisfied, this analysis may produce an incorrect result, and therefore should not be executed. Since some analyses modify the same set of views, input-output dependencies arise and have to be properly resolved.

A. View Level

The view level is used to mediate complex interaction between analyses and models. A key to this mediation is 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 [18] or synchronization [19].

Model-view relations, on the other hand, require a more special link between architectural descriptions and potentially less structured models. I employ partial transformations and annotations to overcome this problem. Mechanisms to establish and update model-view relations have to be customized to the particular formalism in order to be effective. I take advantage of the flexibility of *architectural styles* – custom vocabularies of architectural elements – to support customization and tailor transformation algorithms.

Another function of the view level is establishing consistency between models through their views. This is done using consistency rules, which take form of constraints over multiple views and can be verified with constraint solving, for example SMT [7]. Properties that contain model-specific terms (e.g., the current charge of a battery cell) 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 sound ordering and satisfaction of assumptions and guarantees. To facilitate soundness checking I designed the language of *analysis contracts*. Every analysis is accompanied by its contract C that specifies inputs I, outputs O, assumptions A, and guarantees G of the analysis, in short $C \equiv (I, O, A, G)$.

Sound analysis ordering is one where all analyses go in order of their dependencies. For example, if analysis A_1 depends on analysis A_2, then A_2 should be executed before A_1. 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.

To summarize, my approach promises more effective and less expensive CPS modeling method integration. The next section presents preliminary evidence to supports that claim.

IV. Preliminary Work

Here I describe two significant results: architectural views for hybrid programs and the analysis contracts framework.

A. Architectural View for Hybrid Programs

The hybrid program (HP) modeling and proving method, based on hybrid programs and differential dynamic logic ($\mathsf{d}\mathcal{L}$) [6], is particularly difficult to integrate with other modeling methods, in part due to HP expressiveness and lack of language support for modularity. Each hybrid program contains fragments of various concerns that are highly intertwined with each other, leading to poor modularity and possibility of compositional errors [11].

To incorporate hybrid programs into my approach, I defined how architectural elements can be 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. A foundational abstraction for HP is an architectural view that contains actors HPA, composers CPR, and connectors HPC. I defined an algorithm to transform a view 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 $\mathsf{d}\mathcal{L}$, 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, implemented as a plugin to AcmeStudio [20], demonstrated feasibility and auxiliary benefits of automated support for model-view relationships.

B. Analysis Contracts Framework

This work investigated theoretical and practical aspects of using *analysis contracts* for integration [7]. Theoretical goals were designing a syntax and semantics for contracts and creating algorithms that ensure sound execution of analyses. Practical goals included application for existing domains beyond the original thread scheduling and creation of an extensible framework for analysis execution and contract verification.

To reach the theoretical goal I defined the syntax of analysis contracts and described their semantics over *verification domains* – collections of sets and functions that describe the essential elements of a technical domain. Towards the practical goal I designed and implemented the ACTIVE tool [21] [3] that supports execution of analyses in the OSATE2 architectural environment [4] for AADL.

This research showed that analysis contracts are suitable for detection and prevention of integration errors in several domains: threads scheduling, battery scheduling [7], sensor trustworthiness, reliability, and control [22]. This work demonstrated the improvements in effectiveness and cost-efficiency in my approach to CPS modeling method integration.

V. Expected contributions

If successful, the proposed research will make the following contributions to the *theory* of model-driven engineering and cyber-physical systems:

- A formal description of the model-view consistency mechanism and algorithms for its continuous update.
- A language for analysis contract specification and algorithms to support sound execution of analyses. These algorithms include resolution of analytic data dependencies and analysis contract verification.
- A language for expressing model-specific consistency properties and analytic assumptions.

If successful, the proposed research will make the following contributions to the *practice* of model-driven engineering and cyber-physical systems:

- Implementation of the model-view formalism for Acme in the AcmeStudio architectural environment.
- Implementation of the analysis contracts approach for AADL in the OSATE2 architectural environment.
- Implementation of the property language for AADL in the OSATE2 architectural environment.
- A case study of integrating modeling methods in a realistic industrial or academic CPS project.

Depending on the available time and resources, a number of optional contributions can be made:

- A library of reusable CPS analyses, their contracts, and view consistency rules;
- An instantiation of the model-view mechanism for another representative CPS formalism like mechanical 3D CAD models or Simulink;
- Theoretical analysis of the property specification method in terms of its expressiveness and soundness.

VI. Plan for Thesis Evaluation and Validation

Below I make research claims about feasibility, correctness, effectiveness, and generality of my approach to CPS modeling method integration.

Feasibility: it is be possible to implement my approach in a tool environment that integrates CPS models. I.e., the tool should be capable of integrating representative CPS models and integration abstractions by specifying and verifying integration consistency. I plan to validate feasibility by implementing the approach in a software tool.

Correctness: verification procedures for the analysis contracts and integration properties are sound. That requires demonstrating that the algorithms theoretically achieve their goals of detecting and preventing inconsistencies in models.

Effectiveness: my approach semi-automatically detects and prevents modeling method integration errors that would otherwise be missed. I plan to validate effectiveness of the approach by applying it to collections of models for several systems and showing that it can detect errors that would otherwise have not been made explicit.

Generality: my approach applies to a broad range of CPS modeling methods. I will validate this claim by demonstrating the applicability of my approach to several representative CPS modeling methods.

Since a significant part of this thesis research is applied, it is critically important to evaluate these claims on practical cyber-physical systems and projects. Therefore, I plan to combine several validation methods. First, I check feasibility of my constructs and approach by implementing prototypes of suggested tools. I selected AcmeStudio and OSATE2 because the former already served as a platform for multi-view CPS model consistency research [17], and the latter supports multiple architectural CPS analyses based on AADL. Second, I plan to evaluate effectiveness and generality of my research on realistic industrial or academic projects (examples are below) by taking them as case studies and applying my approach to integrate modeling methods. In addition to the first two approaches, I will use theoretical validation to investigate formal guarantees of my research in a form of theorems.

Finding appropriate CPS case studies can be challenging. To simplify the search I establish the following criteria for the desired case study projects:

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

[4] wiki.sei.cmu.edu/aadl/index.php/Osate_2

- *Heterogeneity:* the project has at least two heterogeneous models or informal representations that are not integrated.
- *Applicability:* the system should have or have a possibility of having discrepancies between models or representations that would lead to critical design errors.
- *Realism:* a project is intended for practical use in industry or academia.
- *Scale:* the project should involve at least three engineers.
- *Timing:* the scope of the system can be adjusted so that validation does not take longer than one person-year.

Currently the candidate systems for conducting an integration case study are: the NASA Europa spacecraft, [5] the Andy Lunar Rover,[6] the SMACCMPilot quadrotor, [7] the STARMAC quadrotor, [8] and the Toyota powertrain [23].

VII. Current Status

Most of the fundamental research towards my dissertation has been completed, but several theoretical and practical aspects remain. The tasks are summarized in Tab. I, with the required work estimated to be completed in 11–16 months. The next steps in my research focus on finalizing the design of the multi-model property language and conducting a case study of model integration.

Task	Completion	Months left
Contract framework design	100%	0
Contract framework implementation	90%	0.5
Model-view mechanism design	90%	0.5
Model-view mechanism implementation	75%	1
Multi-model property language design	50%	2-3
Multi-model property verification implementation	50%	1
Case study search	50%	1-2
Case study execution	0%	2-4
Thesis writing	15%	2-3
Thesis defense	0%	1
(Optional) Theoretical evaluation of model-view mechanism	0%	2
(Optional) Another instantiation of model-view mechanism	0%	1-2
(Optional) Contract & property library	25%	1-2
Total without optional		11-16
Total with optional		15-22

TABLE I: Thesis tasks.

Acknowledgements

I thank my advisor David Garlan for his guidance and support, and my collaborators Dionisio De Niz, Sagar Chaki, Bradley Schmerl, Ashwini Rao, and others for contributing to this research. This work is 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.

[5] nasa.gov/europa
[6] lunar.cs.cmu.edu
[7] smaccmpilot.org
[8] hybrid.eecs.berkeley.edu/starmac

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

[4] 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.

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

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

[7] 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.

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

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

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

[11] 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.

[12] 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.

[13] 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.

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

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

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

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

[18] J. d. Lara and H. Vangheluwe, "AToM3: A Tool for Multi-formalism and Meta-modelling," in *Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE '02. London, UK, UK: Springer-Verlag, 2002, pp. 174–188.

[19] Istvn Rth, Andrs krs, and Dniel Varr, "Synchronization of abstract and concrete syntax in domain-specific modeling languages," *Software and Systems Modeling*, vol. 9, no. 4, pp. 453–471, 2010.

[20] 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.

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

[22] I. Ruchkin, A. Rao, D. De Niz, S. Chaki, and D. Garlan, "Eliminating Inter-Domain Vulnerabilities in Cyber-Physical Systems: An Analysis Contracts Approach," 2015, submitted for publication.

[23] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain Control Verification Benchmark," in *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '14. New York, NY, USA: ACM, 2014, pp. 253–262.