Integration Beyond Components and Models: Research Challenges and Directions

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

***Abstract*—Recent research in embedded and cyber-physical systems has developed theories and tools for integration of heterogeneous components and models. These efforts, although important, are insufficient for high-quality and error-free systems integration since inconsistencies between system elements may stem from factors not directly represented in models (e.g., analysis tools and expert disagreements). Therefore, we need to broaden our perspective on integration, and devise approaches in three novel directions of integration: modeling methods, data sets, and humans. This paper summarizes the latest advances, and discusses those directions and associated challenges in integration for cyber-physical systems.**

I. Introduction

Integration means bringing together elements of a system to make them operate cohesively, and it is an essential concern in software and systems engineering. It Integration differs from composition in that integration often refers to bringing together heterogeneous parts that were not necessarily intended to work together, or have significant challenges in doing so [1]. Research on early model-based integration has been focused on uniting components and models to create a system, and connecting systems to create larger systems.

As software becomes more pervasive and embedded in physical world, cyber-physical systems (CPS) place substantially higher demands on integration [2]. First, *complexity and scale* are leaping forward: what used to be a self-contained open-loop temperature control is now expected to interact and learn from a multitude of other devices in a smart home, or even a network of smart homes. Second, interdisciplinary engineering methods increase *heterogeneity* of system elements, making integrating these elements harder. Finally, *autonomy* requires systems to be aware of their own heterogeneity and complexity, and this awareness needs to be preserved or even constructed during integration.

How can existing integration approaches be augmented to deal with these challenges? To respond to complexity and heterogeneity of CPS, we argue for extending the scope of concerns addressed by integration. Doing so will enable us to better address issues that originate outside of components and models. For example, human interaction plays an increasingly prominent role due to CPS pervasiveness, and component integration is largely oblivious of humans.

This paper focuses on a vision for broadening the scope of integration in CPS. We first summarize recent advances in integrating components and models. After, we examine three ways in which integration can be extended: *modeling methods*, *data*, and *humans*. We will discuss research challenges in each of these directions.

II. Advances in Integration

In this paper we consider integration that is carried out early in an engineering lifecycle, also known as *virtual integration* [3]. Such integration is performed on digital blueprints of a system, and is often used in model-driven engineering.

Over the last decade, two directions of CPS integration research have been significantly advanced: component integration and model integration. We will review each direction individually in the remainder of this section.

A. Component Integration

Component integration[1] is among the most common types of integration. It is applied to systems separated into a number of components to reassemble these systems from them. This approach permits flexible sourcing of components: manual implementation, generation from specifications, or using black-box off-the-shelf components [4].

One necessary element of component integration is the *interface of components*, which usually takes the form of ports. Interface specification determines relative expressiveness, computational complexity, and flexibility of using that interface. Architecture Analysis and Design Language (AADL) [4], for example, fixes several forms of component interfaces based on data and events with simple semantics. It is possible to extend that semantics with annexes (i.e., language extensions) [4] with arbitrarily complex extra specifications and analyses. Such annexes are an example of flexibility of interface descriptions that enables customization to specific CPS domains and models.

Another fundamental idea in component integration is *assume-guarantee reasoning*. Each component makes assumptions about its inputs and environment, and provides guarantees about its outputs. Assume-guarantee reasoning depends on finding intermediate assumptions between components that would conveniently isolate and simplify reasoning (e.g., see the AGREE tool [5]). Usually the existence of such assumptions depends on system design (some system decompositions

[1]Integration of homogeneous components is often called *composition*.

are easier to reason about than others) and manual search for assumptions, which is time-consuming in complex models. This can be overcome by learning assumptions from models using machine learning and abstraction refinement [6].

One major downside of component integration is the (intended) difficulty of bypassing interfaces when more information is needed than available in these interfaces. For example, it is challenging to synchronize timing of components (which use different clocks and execution paradigms) in a model where interfaces contain only functional information, such as algebraic relations between inputs and outputs. Another shortcoming of component integration is that pervasive and cross-cutting qualities are difficult to integrate on component basis. For example, imagine we want to model energy usage. One option is to augment each component with a behavioral energy specification, but then their composition may lead to a statespace explosion. Another option is to ignore component boundaries in energy specifications, but then we risk losing the benefits of having components in the first place. Therefore, we would benefit from integration techniques beyond components to overcome the above limitations.

B. Model Integration

Model integration[2] looks to bring together whole models – not merely components. The crucial difference from component integration is that the models may overlap in their *referents* – system parts that they model. For instance, a system controller (referent) may be represented with a mathematical function in a control model and a periodic algorithm in a discrete event model. Thus, in models there may not be, and often is not, a clear separation of components, making model integration a more complex problem.

A general recipe for integrating models relies on application of two techniques: *abstraction* and *relation*. Abstraction factors out crucial commonalities of models into a simplified space (in a metamodel language, a universal format for all models traces, etc.). Once that space is constructed, some elements of models are related through it. That relation is checked for consistency properties [7], or used for model synthesis or transformation to achieve consistency [8].

Orthogonally to abstraction and relation, existing research on CPS model integration can be put into two categories of approaches: *structural* and *behavioral*. Structural approaches rely on metamodeling, viewpoints [9], megamodeling [10], and multi-view modeling [11] to create a higher-level abstraction of model structure to relate models. The relationship between a model and its structural abstraction is a major factor in what consistency qualities can be checked [7]. Behavioral approaches focus on behaviors that models permits; if an appropriate behavioral form is found, it can be used to relate and connect behaviors in heterogeneous models. Examples of behavioral approaches include heterogeneous simulation [12] and verification with behavior relations [13].

[2]Integration of homogeneous models is often called model composition; e.g., parallel composition of automata.

Model integration, although generally successful within its scope, has one significant drawback: it is usually fragile with respect to model changes. Once models are integrated, any change to them makes engineers either re-integrate models again (expending substantial effort), or potentially abandon the integrated state (risking design errors) – both of which are suboptimal. This issue leads us to examine ways of broadening the scope of integration in the next section.

III. Beyond Components and Models

Our motivation to extend integration beyond components and models is two-fold. First, such extension would help us handle integration problems at appropriate levels of abstraction; e.g., resolving a dependency between transformations once is more effective than fixing every model inconsistency that arises from it. Second, by integrating other engineering artifacts (such as data) with modeling, we expect to improve system designs during early modeling (e.g., achieve desired sensitivity to noise in data).

Early attempts to extend the scope of integration can be traced to integrating models on multiple levels. For example, in the OpenMETA toolchain [14], models are integrated at component level, tool level, and execution platform level. In another example, a methodology for power design [15] builds upon multiple types of specification, maps them to behavior and reliability models, and performs different model-based operations: component synthesis, control synthesis, and simulation. Notice that news levels and abstractions handle specific concerns and add synergistic value to model and component integration. We would like to follow the same pattern for extending integration.

In the remainder of this paper we explore three directions that appear most promising for future research on CPS integration: *modeling methods*, *data*, and *humans*.

A. Modeling Method Integration

Modeling method integration broadens the scope from models to *modeling methods* – cohesive approaches to modeling. A modeling method encompasses, in addition to a formalism and a set of models created using it, referents in the system that the method is applied to (e.g., concurrent threads for process-algebraic modeling) and ways of transforming and analyzing these models [16]. These ways are often embodied in tools and semi-automated procedures, such as real-time analyses [17]. Modeling method integration builds directly upon model integration, and presents several novel challenges.

One challenge is combining modeling methods that have the same or significantly *overlapping referents*. Should these modeling methods address different properties of their shared referent? Or should they use different approaches to get at the same property of the referent? Or is there perhaps a more complex synergy between them? An example of such synergy is found in [15]: discrete modeling, given its more restrictive semantics, synthesizes new controllers, while hybrid modeling is used to optimize and verify, due to its richer semantics.

Modeling method integration intends to merge heterogeneous design processes, which can run into conflicts otherwise [17]. Most modeling methods rely upon certain *assumptions* about the system and its environment; these assumptions need to hold throughout the engineering lifecycle so that the modeling method produces correct results. Hence, situations when the assumptions are violated need to be detected and/or avoided. Another challenging aspect is that modeling methods often *depend* (sometimes circularly) on information from each other to proceed. Scheduling analysis, for instance, constrains possible hardware optimizations, and vice versa. Therefore, sequencing the steps of modeling methods is crucial to their conflict-free integration. One promising approach to managing assumptions and dependencies in modeling methods is augmenting analyses and transformations with formal contracts about assumptions, guarantees, and data dependencies [16].

Finally, the notion of dependency itself needs to be broadened for modeling methods in cyber-physical systems. Dependencies arise between inputs and outputs of components, between related requirements, between system properties in different models [18], between procedures for analysis and transformation, and between other artifacts and model elements. These dependencies may have effects on each other: adding a component interface dependency may add or remove dependency between analytic operations executed on that component model. Therefore, frameworks for integrating dependency management need to be developed, accompanied with automated tools to discover and resolve dependencies.

B. Data Integration

Data integration refers to treating datasets as explicit engineering artifacts, and data analysis as a systematic lifecycle activity. Often in model-driven engineering, performance and testing data is considered a "second-class citizen" compared to models. For instance, a dataset obtained from a hardware-in-the-loop simulation can be used to evaluate and tune models. However, it is relatively rare for datasets per se to be systematically analyzed (e.g., for anomalies), compared among each other, and fed universally into system design.

One data integration challenge arises in model-driven engineering that heavily relies on (semi-)automated proofs of system properties (e.g., [19]). Systems engineering tools often fail to enable *efficient proving processes*. For instance, incomplete proofs are artifacts-in-progress, so it is important to support common patterns of proving (e.g., by recording tactics in a special language [20]) and relating incomplete proofs to other models (e.g., to find counterexamples). A more ambitious goal is to support distributed proof engineering, where several people work on different parts of the same proof simultaneously. One reason that distributed proving is hard is that traditional information hiding interferes with using necessary premises (e.g., if they are encapsulated in a component and not present in its interface). Another reason is that a model is often tuned to simplify a particular proof, and changing that model leads to invalidation of other proofs. Providing tools to detect and alleviate such conflicts would lead to better integration of proofs and models.

Another data integration challenge is *merging data* that comes from heterogeneous sources: simulations, sensor datasets, human feedback, past failures and incidents, etc. It is common for data to be incomplete due to different collection frequencies, fidelities, and various barriers to data collection. This incompleteness needs to be reconciled to draw valid conclusions from data [21]. For instance, missing data from taxi passengers who do not use their phones creates a gap between modeled and real taxi traffic [21].

The last challenge to mention in this section is using data to *inform system design* more broadly. Typically, data from a system is used to build more efficient controllers by providing more sophisticated algorithms. For instance, a robot navigating in a university hallway during a break between classes, would use more conservative parameters for collision avoidance, expecting multiple people coming from different directions. Going forward, can we extend the use of data to affect other elements of system design, such as placement of probes and actuators, system architecture patterns to apply, and priorities of validation and verification activities? Doing so could drastically improve the efficiency of engineering processes by automating design space exploration even further and widening the envelope of low-risk system evolution.

C. Integration with Humans

Humans are involved in creation and operation of cyber-physical systems in a variety of ways. For example, a person may be a user of a smart home system, an obstacle to avoid for an autonomous car, or an engineer who models a spacecraft. Let us separate all potential roles of humans into two categories: (i) external agents, such as users and operators, who interact with a system at its runtime, and (ii) engineers, such as designers and developers, who shape the system at its design time. We will consider each category separately.

When treating humans as users and operators, it is tempting, and often reasonable, to objectify these humans in a set of impersonal and fixed requirements, such as functionality, timing, and reliability. However, as cyber-physical systems are increasingly embedded in our society, these simple "interfaces" between a human and a system are inadequate to capture a complex reality. More parameters need to be considered for accurate modeling, like attention span and knowledge of humans [22], calling for *better human models*.

As we refine models of humans, two integration challenges arise. First, most human models are context-sensitive, and hence fragile when the context switches. For example, important variables for air traffic controllers may be different than those for server farm operators. Hence, integration should *merge human models* across related contexts, producing more reliable and reusable domain-specific models. The second challenge is to develop human models that can be practically *combined with system models*. The issue is that complex human models, when used alongside system models for analysis,

are likely to lead to statespace explosion for reachability analysis, model checking, and similar methods [23].

As engineers, humans think of systems from their subjective viewpoints. In a multidisciplinary environment, such subjectivity can be rooted in education and training in some discipline, thus leading to biased preferences of tools and techniques. Also, many other subtle questions may lead to disagreements among engineers: what aspects and qualities are important in the system? What concepts and abstractions are best used to describe the system? How should faults be identified and resolved? These questions open up a vast field of inquiry into *human factors in modeling and integration* – a field which so far has been explored sparsely[3]. Understanding mismatches between perspectives of engineers may shed light on deeper reasons behind integration issues, and suggest novel solutions.

One research direction of interest is relating differences in human training to consistency issues between models. For instance, it is plausible that a controller implementation does not match its model because software developers are not fully aware of the theoretical assumptions (and implications of their violation) of a controller, as described by control theory. Is it possible to find so called "boundary concepts" [24] that span several disciplines to establish reliable communication, or can we discover "boundary holes" – situations where interdisciplinary modeling repeatedly fails? If so, we can create tool support for model-based integration by building upon common successes and avoiding common pitfalls.

IV. Conclusion

This paper revisited the latest achievements in integration of CPS components and models, and identified new directions for integration research: modeling methods, data, and humans. These directions are promising to overcome existing limitations in integration, ultimately improving quality of cyber-physical systems and reducing costs and effort of integration.

Acknowledgment

Ideas expressed in this paper are based on collaboration with Dionisio de Niz, Sagar Chaki, Bradley Schmerl, David Garlan, and other researchers at Carnegie Mellon University.

This material is based on research sponsored by AFRL and DARPA under agreement number FA8750-16-2-0042, the U.S. Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, and through the Office of the Assistant Secretary of Defense for Research and Engineering (ASD(R&E)) under Contract HQ0034-13-D-0004. The Software Engineering Institute is a federally funded research and development center. The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Software Engineering Institute, Carnegie Mellon, the AFRL, DARPA, the United States Department of Defense, ASD(R&E), or the U.S. Government.

[3]See hufamo.compute.dtu.dk.

References

[1] A. Isazadeh, "Software Engineering: Integration," *Applied and Computational Mathematics*, vol. 3, no. 1, pp. 56–66, 2004.

[2] 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.

[3] P. Feiler and J. Hansson, "System Architecture Virtual Integration: An Industrial Case Study," p. 48, 2009.

[4] 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, 2012.

[5] M. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. Heimdahl, and S. Rayadurgam, "Your What Is My How: Iteration and Hierarchy in System Design," *IEEE Software*, vol. 30, no. 2, pp. 54–60, Mar. 2013.

[6] L. Feng, T. Han, M. Kwiatkowska, and D. Parker, "Learning-Based Compositional Verification for Synchronous Probabilistic Systems," in *Proc. of ATVA*. Springer Berlin Heidelberg, 2011, pp. 511–521.

[7] 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.

[8] M. E. Kramer, M. Langhammer, D. Messinger, S. Seifermann, and E. Burger, "Change-Driven Consistency for Component Code, Architectural Models, and Contracts," in *Proc. of the 18th CBSE*, ser. CBSE '15. New York, NY, USA: ACM, 2015.

[9] M. Torngren, A. Qamar, M. Biehl, F. Loiret, and J. El-khoury, "Integrating viewpoints in the development of mechatronic products," *Mechatronics*, 2013.

[10] R. Hilliard, I. Malavolta, H. Muccini, and P. Pelliccione, "On the Composition and Reuse of Viewpoints across Architecture Frameworks," in *Proc. of 2012 WICSA and ECSA*, Aug. 2012, pp. 131–140.

[11] J. Reineke and S. Tripakis, "Basic Problems in Multi-View Modeling," in *Proc. of TACAS'14*. Springer Berlin Heidelberg, Jan. 2014.

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

[13] A. Rajhans and B. H. Krogh, "Heterogeneous verification of cyber-physical systems using behavior relations," in *Proceedings of the 15th ACM HSCC*. New York, NY, USA: ACM, 2012, pp. 35–44.

[14] 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.

[15] A. L. S.-V. Pierluigi Nuzzo, "Methodology and Tools for Next Generation Cyber-Physical Systems: The iCyPhy Approach," *INCOSE International Symposium*, vol. 25, no. 1, pp. 235–249, 2015.

[16] I. Ruchkin, "Towards Integration of Modeling Methods for Cyber-Physical Systems," in *Proceedings of the Doctoral Symposium at MODELS 2015*. Ottawa, Canada: CEUR-WS, 2015.

[17] 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.

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

[19] A. Platzer, *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*, 2010th ed. Heidelberg: Springer, Sep. 2010.

[20] N. Fulton, S. Mitsch, J.-D. Quesel, and A. Platzer, "KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems," in *Proc. of CADE-25*, vol. 9195, Berlin, Germany, 2015.

[21] D. Zhang, J. Zhao, F. Zhang, and T. He, "UrbanCPS: a cyber-physical system based on multi-source big infrastructure data for heterogeneous model integration," in *Proc. of ICCPS'15*, 2015.

[22] S. Rosenthal, M. Veloso, and A. K. Dey, "Learning Accuracy and Availability of Humans Who Help Mobile Robots," in *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, San Francisco, California, 2011, pp. 1501–1506.

[23] S. Munir, J. A. Stankovic, C.-J. M. Liang, and S. Lin, "Cyber Physical System Challenges for Human-in-the-Loop Control," 2013.

[24] P. P. Mollinga, "The rational organisation of dissent: Boundary concepts, boundary objects and boundary settings in the interdisciplinary study of natural resources management," ZEF Paper Series, Tech. Rep., 2008.