The Challenge of Interoperability: Model-Based Integration for Automotive Control Software

Huafeng Yu[1], Prachi Joshi[2], Jean-Pierre Talpin[3],
Sandeep Shukla[2], Shinichi Shiraishi[1]
[1]TOYOTA InfoTechnology Center, [2]Virginia Tech, [3]INRIA Rennes
hyu@us.toyota-itc.com, {prachi, shukla}@vt.edu, talpin@inria.fr

Invited

ABSTRACT

Model-Based Engineering (MBE) is a promising approach to cope with the challenges of designing the next-generation automotive systems. The increasing complexity of automotive electronics, the platform, distributed real-time embedded software, and the need for continuous evolution from one generation to the next has necessitated highly productive design approaches. However, heterogeneity, interoperability, and the lack of formal semantic underpinning in modeling, integration, validation and optimization make design automation a big challenge, which becomes a hindrance to the wider application of MBE in the industry. This paper briefly presents the interoperability challenges in the context of MBE and summarizes our current contribution to address these challenges with regard to automotive software control systems. A novel model-based formal integration framework is being developed to enable architecture modeling, timing specification, formal semantics, design by contract and optimization in the system-level design. The main advantages of the proposed approach include its pervasive use of formal methods, architecture analysis and design language (AADL) and associated tools, a novel timing annex for AADL with an expressive timing relationship language, a formal contract language to express component-level requirements and validation of component integration, and the resulting high assurance system delivery.

Keywords

Model-based design, integration framework, interoperability, architecture modeling, design by contract, timing

1. INTRODUCTION

Current automotive software systems are increasingly complex, heterogeneous, and decentralized, while at the same time, they are required to be safer, more reliable, optimized in resource usage, and adaptive [1], particularly for the next generation autonomous vehicles. The complexity of the control software in itself is extremely challenging [2], due to the integration of newer functionalities, such as autonomous driving. Moreover, since these systems are considered safety-critical, they require higher assurance, which in turn mandates a rigorous design process involving modeling, integration, implementation, and high assurance verification. It is well known that the cost of validation and resulting redesign increases exponentially if performed at the late-phase implementation and integration. Therefore, system design is expected to be validated as early as possible, and design validation in an early phase has become one of the key approaches to reduce time to market constraints and the overall verification & validation cost [1].

High-level modeling languages, like SysML to capture the overall system specification, and MATLAB/Simulink for the specification of control algorithms are widely used to model and validate at the early phases of design [3]. However, a plethora of high-level models and heterogeneous models of computation makes model integration a severe challenge for Model-Based Engineering (MBE). This issue hampers the wider adoption of model-based design approaches, particularly in the early-phase verification and validation at the system level. To avoid aforementioned issues, a promising approach is the trustworthy *virtual integration* [4, 5], the validation of the virtually integrated model, and design refinement on the virtual prototype. However, at the level of virtual models, capturing timing relations between component executions, real-time constraints, architectural constraints, and challenges of composability of actually implemented components, resource usage by the real components, etc., can be very challenging. Therefore, the virtual model must have a representation of such implementation-specific constraints, and platform-specific yet essential properties represented in them. Furthermore, since the component models are captured with heterogeneous models of computation, integration of such components at the virtual model level might not work at the implementation level unless platform-specific constraints are not captured in such models - beyond just their models of computation. For example, the platforms that these components would eventually run on, the properties of their communicating substrate (buses with different properties and timing) could highly influence their composability and correctness of composition. The real-time requirements, resource usage, etc., for the end-to-end system might also be influenced by the properties of the platform. Thus, virtual integration at just the functional model level is not a good approach to achieve high assurance design implementation.

DAC'15 , June 07 - 11, 2015, San Francisco, CA, U.S.A.
Copyright 2015 ACM 978-1-4503-3520-1/15/06...$15.00
http://dx.doi.org/10.1145/2744769.2747945

Instead of providing isolated solutions to the aforementioned issues, we propose a model integration framework for the automotive domain, considering correct by construction, software architecture, view-based modeling and integration approach, and system optimization based on characteristics related to specific platforms. We adopt formal specification in an early stage of design at the modeling level. The formal specification is based on multi-clock timing [6] and formal contract [7], extracted from informal automotive requirements, to enable a correct composition of system models. Multi-clock timing specification are based on the modeling of synchrony and time as software and hardware events, and are related to synchronization in an architecture specification. Compared to real time, synchronous logical time, applied on both software and architecture, provides an algebraic framework in which both event-driven and time-triggered execution policies can be specified. The formal contracts, used for describing the functional and non-functional specifications of the components, take into account the architecture and platform models as well as their related properties. A dual design methodology, called *Inside-out* and *Outside-in*, is proposed to formally address the decomposition and composition of contracts such that both systems (from car manufacturers) and subsystems (from suppliers) satisfy all contracts.

In the framework, software architecture is explicitly specified with a standardized architecture description language, AADL [8]. From a view-based design approach, behavioral, architectural, and timing views of the system [6] are all considered, along with the compositional view of system models. Multiple formalism and languages are adopted for each view, for example, Simulink for behavioral view, AADL for architectural view, contract for compositional view and synchronous languages for timing view. Formal evaluation of the system is first performed on each view in a separated way, and then in an integrated way in our proposed integration framework.

Outline. One of the main challenges in model-based engineering, i.e., model integration, is characterized in Section 2. A synopsis of our ongoing project as well as the proposed framework, to address the model integration issue, is depicted in Section 3. Section 4 presents our contribution related to architecture modeling and timing specification. Section 5 describes the contract-based design for automotive: a dual approach of *inside-out* and *outside-in*. Additionally, a concrete system integration is presented in Section 6. The conclusion is finally given in Section 7.

2. CHALLENGES IN INTEGRATION

In the automotive domain, the development of electronics and related control software is generally achieved in an isolated and parallel manner by suppliers, according to the requirements provided by car manufacturers (OEM) [2]. The latter have the responsibility for the final integration. However, this integration is a big challenge due to isolated development, lack of integration and architecture specification, late phase for the integration, etc.. Very few studies have been carried out with regard to rigorous model-based virtual integration for automotive systems. In this section, the challenges of automotive model-based system integration will be briefly presented first.

Formal specification. Automotive specification is mainly based on informal requirements. As a consequence, it frequently leads to ambiguity and misunderstanding between OEMs and their suppliers. However, a complete formal specification is also impossible due to the limitations in language expression, time, cost, performance, etc. Therefore, an appropriate formal specification, created in an incremental, composable, and reusable manner is indispensable for the building of reliable model-based integration and formal verification. This specification is expected to be independent from languages, tools, and platforms for wider adaptation and good reusability in industry.

Modeling. In system design, various general or domain-specific modeling languages are used [9] because of different needs, including different application domains, performance, expertise, cost, etc.. For instance, UML is used as a general modeling language, and SysML is adopted in systems engineering related applications. MATLAB/Simulink and Modelica are used as domain-specific modeling languages in a wide span of domains, while SCADE is mostly used for safety-critical systems and requires a strong background in rigorous design. Each language and its associated toolset provide good support for their own development process from modeling to implementation. However a virtual integration using multiple languages and models turns out to be complicated, ambiguous and unpredictable.

Architecture modeling. Software architecture [10] was not considered essential, thus rarely formalized in conventional automotive design processes. Consequently, it generally leads to a manual, error-prone, time-consuming architecture exploration and validation. To avoid this problem, formalization, formal reasoning, and early-phase exploration are required, along with explicit quality attributes associated with particular architectural entities. Currently, architectural aspects of the system are not well expressed by general modeling languages. Architecture description languages, such as AADL[8], AUTOSAR [11] and EAST-ADL [12] were therefore proposed for embedded systems, especially avionic and automotive systems. A system-level design, considering both architecture and behavior, is becoming a promising solution to promote the virtual integration solution for embedded control systems [5, 13].

Timing specification. Semantics interoperability is one of the main issues in the composition of models, due to semantic dissimilarity between models and their inherent formalism, particularly for timing specification. The timing issue is among the most significant concerns in automotive system design [2, 13]. In general, the timing issue becomes more explicit when architecture is considered and the system is integrated, due to the gap between software and architecture design. To cope with timing-related semantics interoperability, one of the feasible solutions is to have a common formal model as the intermediate semantic model, and translate all other models into this common model. The intermediate model provides the formal semantics, based on which, the expected properties of the original models and their integration are checked. An example can be found in [14]. However, this approach requires a semantics preservation in the model translation, which is not practical in most cases. Another solution is related to unified formalism [15, 16]. However, this approach is more theoretical and not yet well applied in industry.

Integration frameworks. Research on model-based system integration has been discussed with regard to cyber-physical systems [17]; Service-oriented Architecture [18]; and heterogeneous models integration and simulation [19]. In addition, AUTOSAR[11] aims at component and platform-level integration for automotive systems, and System Architecture Virtual Integration (SAVI) program [5] targets avionic system integration. However due to multiple challenges in the model integration, integration frameworks, as opposed to specific integration solution, are becoming increasingly important. These frameworks are expected to consider formal specification and analysis, multi-view, and orthogonal attributes of the system, as well as *correct by construction* and *separation of concerns*, to reduce design complexity and validation time.

Tool support. Tool support of MBE is one of the key concerns from an industry adoption viewpoint [20]. Specific model-based tool chains were developed as solutions, such as [21], for safety-relevant automotive embedded systems. However, the lack of serious consideration of formal aspects and integration semantics in these tool chains limits their support for a reliable integration.

In the following, we summarize our current contribution, aiming at addressing previous issues in the context of model-based integration for automotive software control systems. This work is mainly inspired from [13, 22, 5, 1, 17, 20].

3. AN INTEGRATION FRAMEWORK

The proposed framework follows the concept of a previous research project, dedicated to the system-level co-modeling, analysis, verification and simulation in the avionic domain [14, 13], where AADL and MATLAB/Simulink have been used to model a simplified Airbus A350 doors management system. A formal model of computation (MoC), called Polychrony [6], was adopted as a supporting semantic model to enable an unambiguous and trustworthy integration. The later formal analysis, verification, and scheduling were mainly performed on the basis of the same MoC.

Based on the previous exploration, our current research mainly copes with a formal virtual integration framework for next-generation design of automotive control software. The framework promotes *correct by construction* in the early design phase, rather than a posteriori *Verification & Validation.* The main research topics involved in this framework include: formal timing specification, architecture modeling, design by contract, semantics interoperability and system optimization, as well as specification and modeling for different views and properties of the system, such as behavior, architecture, composition, and timing. Other views, such as cost and performance, can also be considered in the next step. All these techniques are adopted in the framework as key solutions to the challenges presented in Section 2.

Formal timing specification and design by contract play a core role in the trustworthy model integration in our approach. High-level, formalized, multi-clock timing specification, considered as a central topic, is to be defined, observed and analyzed, based on software architecture. Considering abstraction in the system design, we advocate the modeling of synchrony and time as software and hardware events, which are related to synchronization in an architecture. Compared to real time, synchronous logical time, applied on both software and architecture, provides an algebraic framework in which both event-driven and time-triggered execution policies can be specified. The formal contracts, used for describing the functional and non-functional specifications of the components, consider the architecture and platform models as well as their associated properties. A dual design methodology, called *Inside-out* and *Outside-in*, is proposed, where the first part addresses decomposition of a contract into sub-contracts, such that the latter can independently be given to automotive suppliers, instead of natural language specifications. The second part deals with a reliable integration of sub-systems to obtain the required system satisfying all contracts. Timing specification and formal contract will be detailed in Section 4 and Section 5 respectively.

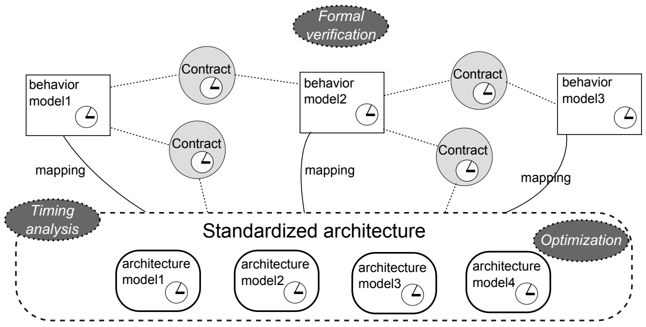

Figure 1: An overview of the integration framework

Figure 1 briefly illustrates the integration framework. High-level automotive requirements are initially analyzed, from which *formalizable* requirements are extracted, according to the technical formalizability and verifiability. These requirements are then used to create formal contracts for properties of timing, safety, performance, cost, etc. In addition to these aspects, multiple modeling languages are applied for different views of the system in the framework, for instance, AADL for architecture modeling and Simulink for behavior modeling. Both timing specification and contracts are defined on these models, to enable a reliable integration. In the final step, behavior models are mapped onto the architecture model, considering pre-defined optimization criteria (see more details in Section 6 and [23]).

4. ARCHITECTURE AND TIMING SPECIFICATION

Models of architectures, models of architecture variability, models of time and quanta, play intertwined and crucial roles in our model integration framework. Architecture modeling has to be thought from the perspective of its direct usability for formal analysis, verification, and simulation. This is an essential reason for the choice of the SAE standard AADL [8], whose semantic unambiguity, together with its forthcoming synchronous behavior annex [24, 25], can serve as a domain-specific (the system architect's domain), yet robust foundation, to support formal reasoning using contracts, and implement the suited meet-in-the-middle design methodology.

Tracing assumptions about time and inferring timing guarantees in an architecture model is essential, in order to maintain the global consistency of its safety-critical functions. This is the essence of synchronous programming, yet reduced to modeling a logical or algebraic abstraction of time suitable for software or hardware (verification and compilation/synthesis).

However, a cyber-physical (or reactive, or embedded) system is the integration of heterogeneous components originating from several design viewpoints: reactive software, some of which is embedded in hardware, interfaced with the physical environment through mechanical parts. One simple example is one of our previous experiments: co-simulating the Airbus A-350 doors management system using models in Matlab/Simulink and the AADL[14].

Time takes different forms when observed from each of these viewpoints: it is discrete and event-based in software, discrete and time-triggered in hardware, continuous in mechanics or physics. Moreover, high-level modeling and programming formalisms used to represent software, hardware, and physics significantly alter this perception of time. In the model of the environment, the continuous evolution of time is represented by differential equations, whose computerized resolution is inherently discrete. In hardware models, the system clock is an abstraction of the electrical behavior of the circuit. It is usually further approximated by coarser-grain abstractions: register transfer level (RTL), transaction-level modeling (TLM) or system-level modeling.

Consequently, time in system design is usually abstracted in a way to serve the purpose of one of many design problems: simulation, profiling, performance analysis, scheduling analysis, parallelization, distribution, simulation, or virtual prototyping. For example, in non-real-time commodity software, timing abstraction, such as number of instructions and algorithmic complexity, is sufficient: software will run the same on different machines, except slower or faster. Alternatively, in cyber-physical extensions, multiple recurring instances of meaningful events may create as many dedicated logical clocks, on which ground modeling practices.

Design of CPS has proved to often benefit from concepts of multiform and logical time(s) for their natural description, which is why, in the spirit of SynCharts [26], and further exploiting the most recent advances of its successor SCCharts (Sequentially Constructive Charts [27]), our aim is to serve the AADL behavior annex with the capability to describe multi-form and multi-rate time through timing relations: logical and algebraic relations (intra-domain relations), abstractions, and concretization, i.e., Galois connections (inter-domain relations).

Instances are constructivity analysis (logical causality analysis [28]), clock calculi (logical synchronization analysis [29, 6]), real-time, abstract, affine scheduling (from multi-rate software to real-time hardware [30, 31]), and discrete/continuous interface using zero-crossing (non-standard interpretation [32]).

In the AADL synchronous behavior annex alone, logical time can simply be installed through (observably time-consuming) delayed state transitions, similar to SyncCharts, as abstraction of the above to non-observable, local computations. This simple addition enables to establish inferable and traceable links between software program points and hardware events, and built relations among them (a clock calculus).

Such a model of computations and communications not only reveals the timing structure of software as perceived by, or related to, its environment (the scheduler), but is it also applicable to all other AADL concepts: ports, threads, buses, memory, and CPU protocols can be associated with behavioral and timed abstractions.

Moreover, the synchronous behavior annex is backward compatible with (most) legacy AADL automata by employing code transformation techniques similar to those used in imperative programming language (pause injection), by transforming untimed, reactive, transition systems and subprograms into synchronous automata in static-single assignment form, compatible with above timed semantic framework.

In the end, this development aims at offering capability to infer/analyze, verify (Fiacre [33]), schedule (Cheddar [34], Syndex [35]), simulate (Polychrony [36, 37]) and link/trace the architecture model with its supportive contract-based design system.

5. CONTRACT BASED DESIGN

Our *contract-based* design methodology aims at formalizing some current practices in automotive industry and keeping with the engineering conventions of OEMs, while at the same time bringing further rigor through formal methods, so that better correctness guarantees can be provided, and virtual prototyping based models can be subject to formal analysis and predictions.

Two problems are clarified first. Models, like Simulink, of an automotive application are translated into software code that is cross-compiled into the respective processor binaries. However, the physical attributes of the platform, such as latencies, processor speeds, bus contentions, are independent from the application. Consequently, this design process remains detached from the platform characterization. Only after the application design, and its functional testing, can one discover any incompatibility with the platform characteristics. This increases the number of cycles of design - test on platform - fix, which is time consuming, and often yields suboptimal application results.

The second problem emerges due to the fact that the various subsystems provided by suppliers may individually meet certain functional and/or performance criteria. However, due to incompatibilities of the subsystems at their interfaces, the integration process requires either going back to the suppliers, or making vulnerable fixes, resulting in brittle system design that ruins variability. For example, a system is not reusable because subsystem interfaces are inflexible to allow design changes and evolution.

A Dual approach of Inside-out and outside-in

In order to solve these two problems, namely - the need to formally relate the platform model to application model, and the need to formally guarantee the interoperability of subsystems contracted out to different suppliers, we propose a dual design methodology, called *Inside-out* and *Outside-In* (IO^R) methodology. This methodology is based on formal contracts, defined on the interfaces of components (or sub-systems), which capture and constrain the functional and nonfunctional behavior of the signals/variables visible at the interface. The formalization of contracts considers compatibility as well as closure properties of contract composition. Moreover, composition operations carry algebraic properties, such as associativity and commutativity. Given the formal contract language, and its algebraic properties, our methodology works in two ways:

Inside-out: for the first problem mentioned above, we develop and devise algorithmic methods for decomposing a contract into sub-component contracts. This process is quite complex, because a contract is not supposed to decomposed into arbitrary subcontracts, but rather into subcontracts that correspond to components to be running on specific processors/controllers on the platform. Thus this contract decomposition process can only be automated, provided a formal platform model can be taken as input to the system, with its various physical and functional attributes formally represented.

An algorithmic decomposition process is currently under development. The goal of the this process is to provide formal contracts of the subsystems to the suppliers, so that instead of natural language narrative description of the subsystems that they are asked to design, they can design to the formal contracts. Formal verification methods are applied by them to ensure that their design meets the requirements captured in the contracts. Each supplier is encouraged to provide the subsystem design that exactly meets the contract, which leads to a smooth integration process, as the subcontracts were created through the decomposition of system-level properties.

Outside-in: the second part of the IO^R methodology concerns the supplier's perspective. When a formal contract is provided to the supplier along with the platform model (as most OEMs use specific platforms for their automotive), the question is how to optimally design the subsystem to satisfy the contract including its non-functional requirements, such as latency and real-time characteristics. This part we call outside-in, because the contract describes the subsystem from outside and the method dealing with the inside design of the component.

Decomposition of contracts

One of the important issues in the transaction between the OEM and the suppliers is the communication of unambiguous requirements to the suppliers. Let us take an example to explain this issue and discuss a possible solution to it using design by contract.

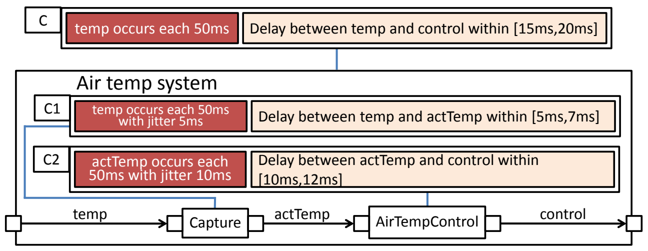
C
temp occurs each 50ms
Delay between temp and control within [15ms,20ms]
Air temp system
C1
temp occurs each 50ms with jitter 5ms
Delay between temp and actTemp within [5ms,7ms]
C2
actTemp occurs each 50ms with jitter 10ms
Delay between actTemp and control within [10ms,12ms]
temp
Capture
actTemp
AirTempControl
control

Figure 2: An example of decomposition of component and corresponding contract [38].

In Figure 2, contract C is a high-level contract for the component $AirTempSystem$. This contract is decomposed into two sub-contracts $C1$ and $C2$ such that C refines the composition of $C1$ and $C2$. This is an example of decomposition of a high-level contract into sub-contracts which satisfy the main contract C when they are integrated. For the development of this component, one possible solution for the OEM is to provide the contracts $C1$ and $C2$ to two suppliers, who in turn provide the corresponding components that satisfy $C1$ and $C2$ respectively. The OEM, can then validate and verify these components easily and integrate them in order to obtain the desired system.

6. INTEGRATION AND OPTIMIZATION

A concrete model-based approach for system co-modeling, integration, and optimization is briefly presented in this section, based on our previous research achievements [23]. In this work, we first design the architecture and behavior models of the system in AADL and Simulink respectively. The architecture model defines the hardware components and the communication channels in the system, and the behavior model defines the software running on the hardware (processors/controllers) and their interaction using the bus(es). The behavior model is then mapped onto the architecture model using an optimization technique, which minimizes the end-to-end communication latency in executing a distributed function on the specified platform architecture. The flow-chart describing the procedure for mapping is given in Figure 3. The behavior model is transformed into data-flow graph formalism. A pseudo static schedule for the data-flow graph (DFG) is derived. The proposed mapping technique assumes a fixed architecture model and creates partitions on the functional nodes, in order to group them. Each group of these nodes is then assigned to a processor such that the end-to-end delay for the execution of each function is optimized. This technique is illustrated with a case study of an adaptive cruise control, for which the mapping is performed manually so far.

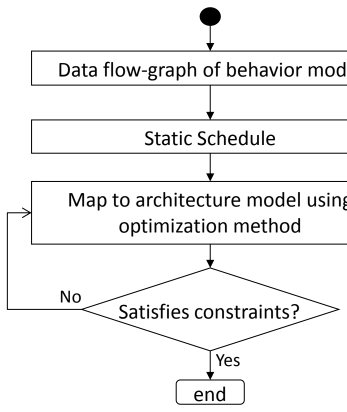
Data flow-graph of behavior model
Static Schedule
Map to architecture model using optimization method
Satisfies constraints?
No
Yes
end

Figure 3: Flow-chart for the mapping technique.

7. CONCLUSION

In this paper, we concisely presented several of the challenges ahead of our goal to engineer a model-driven approach dedicated to virtual integration applied on automotive software control systems. These challenges include ambiguity incurred from the heterogeneity of models and semantics to integrate, the variety of timing and quantitative specifications, of existing modeling frameworks, etc. To cope with these challenges, a formal model-based integration framework is proposed, based on formal timing specifications, using the forthcoming synchronous timing annex to the architecture analysis and design language (AADL); along with a dual, meet-in-the middle, top-down/bottom-up, inside-out and outside-in, methodology for contract refinement and composition; implemented using a co-modeling, integration and optimization approach at the system level. Our proposal aims at a better adoption of the framework by standardizing key timing and quantitative specification aspects, as well as a tool-independent and trustworthy composition framework using formal contracts.

8. REFERENCES

[1] DARPA, "Adaptive Vehicle Make (AVM) Program." `http://www.darpa.mil/Our\_Work/TTO/Programs`.

[2] M. Broy, "Challenges in Automotive Software Engineering," in *International Conference on Software Engineering (ICSE)*, 2006.

[3] D. Schmidt, "Model-Driven Engineering," *IEEE Computer*, vol. 39, no. 02, pp. 25–31, 2006.

[4] P. Giusto, A. Ferrari, L. Lavagno, J.-Y. Brunel, E. Fourgeau, and A. Sangiovanni-Vincentelli, "Automotive Virtual Integration Platforms: Why's, What's, and How's," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD'02)*, 2002.

[5] P.-H. Feiler, J. Hansson, D. de Niz, and L. Wrage, "System Architecture Virtual Integration: An Industrial Case Study," tech. rep., Software Engineering Institute, 2009. CMU/SEI-2009-TR-017.

[6] P. Le Guernic, J.-P. Talpin, and J.-C. L. Lann, "Polychrony for System Design," *Journal for Circuits, Systems and Computers*, vol. 12, pp. 261–304, 2002.

[7] B. Meyer, "Applying "Design by Contract"," *IEEE Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[8] SAE Aerospace (Society of Automotive Engineers), "Aerospace Standard AS5506A: Architecture Analysis and Design Language (AADL) ," *SAE AS5506A*, 2009.

[9] K. Müller-Glaser, G. Frick, E. Sax, and M. Kühl, "Multiparadigm Modeling in Embedded Systems Design," *IEEE Transactions on Control Systems Technology*, vol. 12, no. 2, pp. 279–292, 2004.

[10] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall Englewood Cliffs, 1996.

[11] AUTOSAR (AUTomotive Open System ARchitecture). `http://www.autosar.org/`.

[12] EAST-ADL. `http://www.east-adl.info`.

[13] H. Yu, Y. Ma, T. Gautier, L. Besnard, J.-P. Talpin, and P. Le Guernic, "Polychronous Modeling, Analysis, Verification and Simulation for Timed Software Architectures," *Journal of Systems Architecture (JSA)*, vol. 59, no. 10, pp. 1157–1170, 2013.

[14] H. Yu, Y. Ma, Y. Glouche, J.-P. Talpin, L. Besnard, T. Gautier, P. Le Guernic, A. Toom, and O. Laurent, "System-level Co-simulation of Integrated Avionics Using Polychrony," in *ACM Symposium on Applied Computing (SAC)*, 2011.

[15] D. Mathaikutty, H. Patel, S. Shukla, and A. Jantsch, "Modelling Environment for Heterogeneous Systems based on MoCs," in *Forum on specification and Design Languages (FDL)*, pp. 291–303, 2005.

[16] E. A. Lee and A. Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, pp. 1217–1229, 2006.

[17] J. Sztipanovits, X. D. 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, 2012.

[18] A. Rossignol, "The Reference Technology Platform," in *CESAR - Cost-efficient Methods and Processes for Safety-relevant Embedded Systems*, Springer, 2013.

[19] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity - the Ptolemy Approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[20] M. Broy, M. Feilkas, M. Herrmannsdoerfer, S. Merenda, and D. Ratiu, "Seamless Model-Based Development: From Isolated Tools to Integrated Model Engineering Environments," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 526–545, 2010.

[21] E. Armengaud, M. Zoier, *et al.*, "Model-based Toolchain for the Efficient Development of Safety-Relevant Automotive Embedded Systems," in *SAE World Congress & Exhibition*, 2011.

[22] Y. Ma, H. Yu, T. Gautier, P. Le Guernic, J.-P. Talpin, L. Besnard, and M. Heitz, "Toward Polychronous Analysis and Validation for Timed Software Architectures in AADL," in *Design, Automation, and Test in Europe (DATE)*, pp. 1173–1178, 2013.

[23] P. Joshi, S. K. Shukla, J.-P. Talpin, and H. Yu, "Mapping Functional Behavior onto Architectural Model in a Model Driven Embedded System Design," in *ACM Symposium on Applied Computing (SAC)*, 2015. To appear.

[24] L. Besnard, A. Bouakaz, T. Gautier, P. L. Guernic, Y. Ma, J.-P. Talpin, and H. Yu, "Timed Behavioural Modelling and Affine Scheduling of Embedded Software Architectures in the AADL using Polychrony," *Science of Computer Programming*, 2014. doi:10.1016/j.scico.2014.05.014.

[25] L. Besnard, E. Borde, P. Dissaux, T. Gautier, P. Le Guernic, and J.-P. Talpin, "Logically Timed Specifications in the AADL : a Synchronous Model of Computation and Communication (Recommendations to the SAE Committee on AADL)," Technical Report RT-0446, INRIA, 2014.

[26] C. André, "Representation and Analysis of Reactive Behaviors: A Synchronous Approach," in *Computational Engineering in Systems Applications*, 1996.

[27] R. von Hanxleden, B. Duderstadt, *et al.*, "SCCharts: Sequentially Constructive Statecharts for Safety-critical Applications," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 372–383, 2014.

[28] R. von Hanxleden, M. Mendler, *et al.*, "Sequentially constructive concurrency A conservative extension of the synchronous model of computation," in *Design, Automation Test in Europe (DATE)*, pp. 581–586, March 2013.

[29] L. Besnard, T. Gautier, P. Le Guernic, and J.-P. Talpin, "Compilation of Polychronous Data Flow Equations," in *Synthesis of Embedded Software* (S. K. Shukla and J.-P. Talpin, eds.), pp. 1–40, Springer, 2010.

[30] A. Bouakaz and J.-P. Talpin, "Design of Safety-Critical Java Level 1 Applications Using Affine Abstract Clocks," in *International Workshop on Software and Compilers for Embedded Systems*, pp. 58–67, 2013.

[31] A. Bouakaz and T. Gautier, "An Abstraction-Refinement Framework for Priority-Driven Scheduling of Static Dataflow Graphs," in *ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2014.

[32] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet, "Non-standard Semantics of Hybrid Systems Modelers," *J. Comput. Syst. Sci.*, vol. 78, no. 3, pp. 877–910, 2012.

[33] B. Berthomieu, S. Dal Zilio, and T. Fronc, "Model-Checking Real-Time Properties of an Aircraft Landing Gear System Using FIACRE," in *ABZ 2014: The Landing Gear Case Study*, vol. 433, pp. 110–125, Springer, 2014.

[34] J. McCormick, F. Singhoff, and J. Hugues, *Building Parallel, Embedded, and Real-Time Applications with ADA*. Cambridge University Press, Apr. 2011.

[35] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine, "The SynDEx Software Environment for Real-Time Distributed Systems, Design and Implementation," in *European Control Conference (ECC)*, 1991.

[36] A. Gamatié, T. Gautier, P. Le Guernic, and J.-P. Talpin, "Polychronous Design of Embedded Real-Time Applications," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 2, 2007.

[37] J.-P. Talpin, J. Brandt, M. Gemünde, K. Schneider, and S. Shukla, "Constructive Polychronous Systems," *Science of Computer Programming*, vol. 96, no. 3, pp. 377–394, 2014.

[38] M. Förster, "Dependable Reuse & Guarded Integration of Automotive Software Components," tech. rep., FAT-Series, 2013.