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

Huafeng Yu1, Prachi Joshi2, Jean-Pierre Talpin3, Sandeep Shukla2, Shin’ichi Shiraishi1,4

1 TOYOTA InfoTechnology Center USA, 2 Virginia Tech, 3 INRIA Rennes, 4 Potential author

ABSTRACT

Model-Based Engineering (MBE) is a promising approach to cope with the challenges of designing 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 an ongoing project to address these challenges with regard to automotive software control systems. A novel model-based formal integration framework is being developed in this project to enable architecture modeling, timing specification, formal semantics, design by contract and optimization in the system-level design. The main advantage of the proposed approach is 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, as well as 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 [13], particularly for the next generation autonomous vehicles. The complexity of the control software in itself is extremely challenging [11], due to the integration of newer functionalities, such as autonomous driving. Moreover, since these systems are considered safety-critical, they require higher assurance, requiring 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 done 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 [13].

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 early phases of design [30]. 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 constrains 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 [19] [16], 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 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.
In spite of providing isolated solutions to the previous issues, we propose a model integration framework for the automotive domain, considering both 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 [22] and formal contract [26], 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 \textit{Inside-out} and \textit{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 [29]. From a view-based design approach, behavioral, architectural, and timing views of the system [22] 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 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 \textit{inside-out} and \textit{outside-in}. Additionally, a concrete system integration is presented in Section 6. The conclusion is finally given in Section 7.

## 2. CHALLENGES FOR 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) [11]. The latter have the responsibility for the final integration. However, this integration is a big challenge due to isolated development, lack of integration specification and architecture specification, late phase for the integration, etc. However, very few studies have been carried out w.r.t. 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 [27] 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 [31] was not considered essential, thus rarely formalized in conventional automotive design processes. Consequently, it generally leads to a manual, error-prone, timing-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 [29], AUTOSAR [3] and EAST-ADL [14] 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 [16] [36].

**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 [11] [36]. 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 [37]. 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 [24], [23]. 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 [32]; Service-oriented Architecture [28]; and
heterogeneous models integration and simulation [15]. In addition, AUTOSAR[3] aims at component and platform-level integration for automotive systems, and System Architecture Virtual Integration (SAVI) program [16] 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 [12]. Specific model-based tool chains were developed as solutions, such as [2], 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 sections, we summarize an ongoing project, aiming at addressing previous issues in the context of model-based integration for automotive software control systems. This project is mainly inspired from [36], [16], [13], [32], [12].

### 3. AN INTEGRATION FRAMEWORK

The framework established in current project follows the concepts of a previous research project dedicated to the system-level co-modeling, analysis, verification and simulation in the avionic domain [37] [36], where AADL and MATLAB/Simulink were used to model a simplified Airbus A350 doors management system. A formal model of computation (MoC), called Polychrony [22], was adopted as a supporting semantic model to enable a 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, current work 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 considered by us as key solutions to the challenges mentioned in Section 1.

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 centric 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](image)

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 (more details in Section 6).

### 4. ARCHITECTURE AND TIMING SPECIFICATION

Models of architectures, of architecture variability, and time modeling 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 [29], whose semantic unambiguity, together with its forthcoming synchronous behavior annex [7, 6], based on previous experiments with concepts borrowed from synchronous programming [22], 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. 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.

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.

Moreover, time in system design is usually abstracted 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 to ground modeling and design practices.

Design of CPS has proved to often benefit from concepts of multiformal and logical time(s) for their natural description, which is why, in the spirit of SynCharts [1], and further exploiting the most recent advances of its successor SC-Charts (Sequentially Constructive Charts [34]), 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 [35]), clock calculi (logical synchronization analysis [8, 22]), real-time, abstract, affine scheduling (from multi-rate software to real-time hardware [10, 9]), and discrete/continuous interface using zero-crossing (non-standard interpretation [4]).

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

It not only reveals the timing structure of software as perceived, or related, to its environment (the thread and program scheduler), but is 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 subroutines into synchronous automata in static-single assignment form, compatible with the above timed semantic framework.

In the end, this development aims at offering capability to infer/analyze, verify (Fiascro [5]), schedule (Cheddar [25], Syndex [21]), simulate (Polychrony [18, 33]) 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 first clarified before the introduction of our methodology.

The automotive application modeled in tools like Simulink are then translated manually into software code that is cross compiled into the respective processor binaries. Unfortunately, in such a process, the physical attributes of the platform such as latencies, power consumptions, processor speeds, bus contents, arbitration delays, etc., are not directly modeled with the application, and hence the process of application design remains detached from the platform characterization. Thus, only after the application design, and its functional testing, can one discover any incompatibility with the platform characteristics that would require redesigning the application. This increases the number of application design - test on platform - fix application cycles, which is time consuming, and often yields suboptimal application results.

The second problem emerges due to the fact that the various subsystems supplied 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 brittle fixes, resulting in brittle system design, and possibly a system that is not reusable in the future generation of the same automotive series, because the interfaces of subsystems are too brittle to allow design changes.

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 which we call Inside-out and Outside-In (IO) methodology.

Our methodology is based on the notion of formal contracts. 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 we do not want to decompose a contract 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.

At the moment, we do this decomposition manually, while we work on the algorithmic decomposition process. The goal of the decomposition process is to provide the 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. If each supplier provides the subsystem design that exactly meets the contract, then the integration process will be smooth, and the subcontracts were created through decomposition of the 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 here, but the method is about how to design the inside of the component.

Image

Figure 2: An example of decomposition of component and corresponding contract [17].

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.

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. RELIABLE SYSTEM INTEGRATION

Another important aspect in the system development is system integration. One possible solution based on model based design and design by contract has been proposed in our work [20]. 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 to 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.

Image

Figure 3: Flow-chart for the proposed mapping technique.

7. CONCLUSION

In this paper, we have concisely presented several integration challenges in the context of model-based engineering for automotive software control systems. These challenges include ambiguous integration specification, heterogeneity in modeling and semantics, timing specification, integration frameworks, etc. To cope with these challenges, an integration framework is proposed, based on formal timing specification, as a forthcoming synchronous timing annex to the architecture analysis and design language (AADL); a dual approach, namely inside-out and outside-in, for contract decomposition and composition; and a co-modeling, integration and optimization approach. We consider a better adoption of the framework via standardization of the timing specification, as well as tool-independent trustworthy composition via formal contracts.
REFERENCES


