Supporting Heterogeneity in Cyber-Physical Systems Architectures

Akshay Rajhans, *Member, IEEE*, Ajinkya Bhave, *Member, IEEE*, Ivan Ruchkin, *Student Member, IEEE*, Bruce H. Krogh, *Fellow, IEEE*, David Garlan, *Fellow, IEEE*, André Platzer, *Member, IEEE*, and Bradley Schmerl, *Member, IEEE*

***Abstract*—Cyber-physical systems (CPS) are heterogeneous, because they tightly couple computation, communication, and control along with physical dynamics, which are traditionally considered separately. Without a comprehensive modeling formalism, model-based development of CPS involves using a multitude of models in a variety of formalisms that capture various aspects of the system design, such as software design, networking design, physical models, and protocol design. Without a rigorous unifying framework, system integration and integration of the analysis results for various models remains *ad hoc*. In this paper, we propose a multi-view architecture framework that treats models as views of the underlying system structure and uses structural and semantic mappings to ensure consistency and enable system-level verification in a hierarchical and compositional manner. Throughout the paper, the theoretical concepts are illustrated using two examples: a quadrotor and an automotive intersection collision avoidance system.**

***Index Terms*—Control design, control engineering, formal verification, software architecture.**

I. Introduction

MODEL-BASED development refers to the use of computational and formal models in the system design process. The goal is to reduce costly testing and redesign: catching errors in models is significantly cheaper than finding them in the final system or even in prototype implementations. Model-based development of complex cyber-physical systems (CPS) involves creating models for a variety of different design perspectives, including computation and software design, communication and networking design, design of the physical dynamics, protocol design, and control design. Different modeling and analysis tools are well-suited for some of these aspects but not others. For control applications, control laws are typically derived and initially evaluated using control-oriented models, in which details of the implementation and the physical dynamics are simplified or neglected. These details are usually modeled and evaluated using other formalisms and tools. Heterogeneity of those models poses challenges to assessing the performance and correctness of the CPS as a whole.

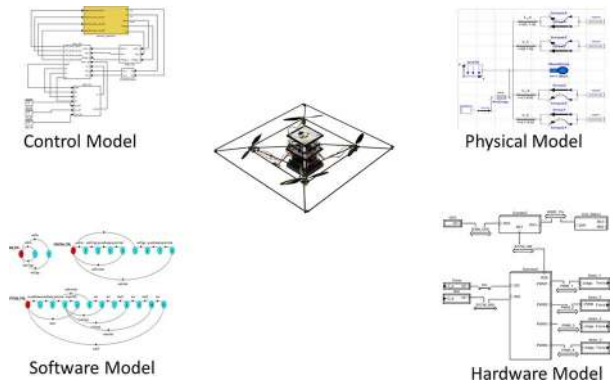


Fig. 1. Multi-domain models of the STARMAC quadrotor.

In our prior work (summarized in Section III), we introduced the concept of architectural views for CPS to support model-based development with heterogenous models, and demonstrated how mappings between the views can ensure the models are structurally consistent. Formal verification of safety-critical CPS using heterogeneous models requires richer semantic support; however, that goes beyond the semantics enforced by architectural structure. In this paper, we propose a framework to provide this support based on behavioral semantics to support heterogeneous multi-model development of CPS within a multi-view architectural framework.

To motivate the discussion, we consider two CPS examples. The first example is the Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC) [1], which is a quadrotor platform developed to test algorithms that enable autonomous operation of aerial vehicles. Fig. 1 depicts heterogeneous models of the quadrotor from four different design domains: a signal-flow model of the closed-loop feedback system for the vehicle, used for stability and performance analysis; an equation-based physical model to study the open-loop dynamic response of the vehicle to external forces and torques; a process algebra model of the on-board controller software, used to verify certain safety conditions; and a hardware model of the electronic units and their interconnection, used to study tradeoffs between specifications and system-level performance. Each of these models describes the same underlying vehicle, so the assumptions made in each model about the structure and properties of the quadrotor must be consistent with the complete system in some way. The notion of *structural consistency* makes it possible to trust analysis results obtained from

Manuscript received February 15, 2013; revised December 5, 2013; accepted December 13, 2013. Date of publication August 28, 2014; date of current version November 18, 2014. This work was supported in part by the National Science Foundation under Grants CNS 1035800-NSF and CCF-0926181. Recommended by Associate Editor C. J. Tomlin.

A. Rajhans, A. Bhave, and B. H. Krogh are with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: arajhans@alumni.cmu.edu; ajinkya@alumni.cmu.edu; krogh@ece.cmu.edu).

I. Ruchkin, D. Garlan, A. Platzer, and B. Schmerl are with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: iruchkin@cs.cmu.edu; garlan@cs.cmu.edu; aplatzer@cs.cmu.edu; schmerl@cs.cmu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2014.2351672

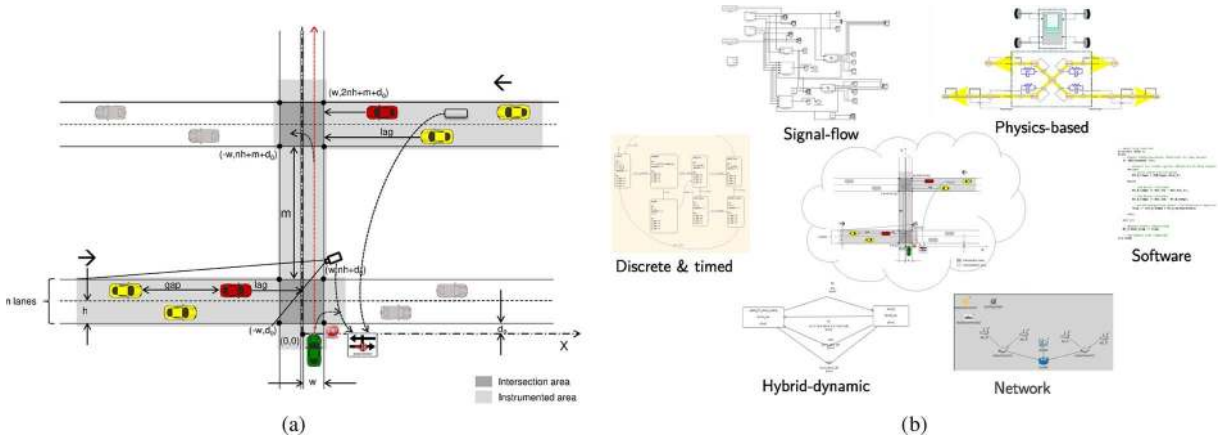


Fig. 2. Cooperative intersection collision avoidance system for stop-sign assist (CICAS-SSA) and a collection of models for it. (a) Illustration of CICAS-SSA. (b) A collection of models for CICAS-SSA.

each model separately and use the analysis results from each model as assumptions in other models. The structural richness of the STARMAC example helps illustrate the application of structural consistency.

The other example is a cooperative intersection collision avoidance system for stop-sign assist (CICAS-SSA), illustrated in Fig. 2(a), which aims to augment human judgment about safe gaps in oncoming traffic at stop-sign-controlled intersections [2]. Such systems involve: *sensing* for the heading of the oncoming vehicles; *communication* of these readings over networks to a decision system; and *computation* of safe gaps based on the *physical dynamics* of the vehicles and speed limits. In addition, there is also empirical information about the time required to alert and warn drivers in time for them to respond, driver behavior of acceptance and rejection of traffic gaps for different demographics, and the suitability of the systems for different intersection geometries [3]–[5]. There is no universal modeling framework that captures everything that is to be modeled for such a system, and even if there were, the analysis of such an all-inclusive model would not be tractable. As a result, one needs to decompose the underlying system into semantically different models, shown in Fig. 2(b), and analyze them separately. The semantic diversity of CICAS-SSA models helps demonstrate our approach to heterogeneous verification.

The need to use heterogeneous models and their analysis together towards correct system design presents a unique set of challenges. Lack of a system-level representation makes system integration an *ad hoc* and error prone activity. Formal verification of the overall safety-critical system without a comprehensive modeling formalism leads to the question: How can verification results from the different formalisms be combined to infer system-level properties? Complex systems are designed by composing subsystems and abstractions need to be made across heterogeneous modeling formalisms to make the analysis tractable. Each model represents some aspect of system design while occluding others by making simplifying assumptions that are often undocumented and have no supporting representation or are captured informally at best.

To address these issues, we present an architectural framework for the heterogeneous model-based development of CPS. We begin by reviewing related work in Section II. We then present our architectural framework in three parts: i) a summary of our work on the architecture-centric approach to consistent design of CPS in Section III; ii) the development of behavior semantics for addressing semantic heterogeneity and its use for multi-model hierarchical and compositional heterogeneous verification in Sections IV–VI; and iii) a unified framework that combines the multi-view structural analysis and multi-model semantic analysis, as presented in Section VII. Section VIII summarizes the contributions of the paper and outlines some directions for future work. The theoretical concepts are illustrated on the CICAS-SSA and STARMAC examples.

II. Related Work

A. Multi-Model Development

The field of computer automated multi-paradigm modeling is introduced in [6], and the current issues and promising approaches are outlined. System architecture virtual integration is an architecture-centric approach to the analysis of system models with respect to quality attributes, such as performance, safety, and reliability [7], but it lacks a uniform way to reason about heterogeneous model semantics. NAOMI is an experimental platform for enabling multiple heterogeneous models to work together [8], but it does not define consistency between models and the system, nor is there is a mechanism to define physical architectural elements. Ptolemy II enables simulation of heterogeneous models by integrating multiple "models of computation" hierarchically into a single simulation model in an actor-oriented formalism [9]. SysWeaver [10] is a model-based development tool that includes a flexible code generation scheme for distributed real-time systems, however it lacks support for a physical plant modeling view and a mechanism to define new views or relations between them.

Multiple models are also supported by model transformation/translation using meta-models [11]–[13], a suitable interchange format [14], [15] or the use of translation schemes [16]. The Vanderbilt model-based prototyping toolchain supports a subset of Simulink/Stateflow models with periodic execution, software architecture and hardware platform modeling, but has neither support for additional views (e.g., physical or verification models), nor a notion of consistency between additional system views. The approach in semantic anchoring [16] to transform between system models concentrates on the specification of the dynamic semantics of domain-specific modeling languages using a small set of behavior abstractions. The Metropolis design

framework [17] and its proposed extension Metro II [18] use platform-based design methodology for multi-model system design by orthogonalizing concerns such as communication-computation, function-architecture and behavior-performance.

The ModelicaML profile aims to integrate UML and Modelica for modeling and simulation of system requirements and design [19]. Similar work has been done for UML and Simulink [20]. There is also an ongoing effort to integrate Modelica with SysML (a UML 2.0 profile) for physical domain modeling [21]. However, in all these approaches, there is not an easy way to incorporate physical dynamics models into the overall framework. For example, SysML flow ports do not have a well-defined semantics to model flows of physical quantities (energy or torque). In addition, there are no consistent rules or guidelines on how to define relationships between multiple views in any of these frameworks.

These approaches focus on addressing specific sets of issues in supporting multi-model system development of CPS. Our approach is to develop a general unifying framework for supporting not only multi-model design, but also analysis and verification of CPS.

B. Software Architecture

Over the past decade *software architecture* has emerged as one of the primary techniques for disciplined engineering of large-scale software systems. A software architecture (SA) typically models a system as a graph of components and connectors, in which the components represent principal computational elements of a system's run-time structure and the connectors represent the pathways of communication between components [22], [23]. These elements are annotated with properties that characterize their abstract behaviors and allow one to reason and make tradeoffs at a systems level about qualities such as performance, reliability, security, and cost.

Given the importance of SA, there has been considerable research and development in: notations to support architectural specification, often called architecture description languages (ADLs): and tools to support their analysis and realization as code [24], [25]. Standardized notations, such as UML 2.0 [26], SysML [27], and AADL [28] provide a modeling vocabulary of components and connectors, as well as certain classes of properties. Additionally, a number of researchers have investigated the modeling of architectural behavior, for example as protocols characterized by process algebras or state machines [29], [30]. These notations are supported by tools that provide graphical editing and viewing, hierarchical development (in which components may be refined as more detailed architectures) [31], checking for component compatibility or substitutability [32], and evaluation of quality attributes such as performance, reliability, and security [25].

SAs also support reuse of design expertise and code infrastructure. In many cases an architecture of a system fits within a common family or design pattern, referred to as an *architectural style*, which is typically defined as a set of component and connector types, together with constraints that prescribe how elements can be composed [22]. Some ADLs allow one to define architectural styles, develop systems in that style, and provide tools for checking whether a system is compliant with a given style [33]. NASA has developed an architectural style for space systems, in which components represent sensors, actuators, state variables, and estimators, and then used this to model space rovers for Mars [34].

In principle, SA promises a natural solution path for providing a uniform modeling approach that supports the design and analysis of CPS. First, through high-level, hierarchical component-oriented models, it can provide representations that reduce complexity through abstraction and encapsulation. Second, through SA's uniform treatment of components, whether embodied by software or hardware, it can support integration of systems that combine physical and computational elements. Third, as we discuss later, it can form the basis for understanding the dependencies between various separable models that focus on partial analysis of the full system. Indeed, SA has been applied effectively to numerous embedded and control systems. ADLs such as Meta-H and AADL have been used to model avionics, automotive, and other control systems [35].

There are two fundamental shortcomings of current architecture modeling capabilities that limit their potential to fully address the engineering problems of large-scale, heterogeneous CPS: i) limited vocabulary to represent physical elements and their interactions; and ii) inadequate ways to support consistency relations between different (possibly heterogeneous) architecture views of the same system. Our work on cyber-physical system architectures that addresses these shortcomings is summarized in Section III.

C. Analysis and Verification

For model-based verification, heterogeneous abstractions have been used for specific pairs of formalisms, such as hybrid abstractions of nonlinear systems [36], [37], linear hybrid automata abstractions of linear hybrid systems [38], discrete abstractions of hybrid systems [39]–[41] and continuous abstractions of hybrid systems [42]. Our objective is to create a general framework for abstraction that applies to any set of heterogeneous formalisms.

Heterogeneous reactive systems can be compared and composed using the tagged-signal semantics [43], [44]. Julius creates a behavioral framework for modeling control as a behavior interconnection problem [45]. These approaches use system trajectories or behaviors as a mathematical framework for creating relations between the semantics of different modeling formalisms. In a similar spirit, we use mathematical relations and functions between behavior domains as the semantic mappings between heterogeneous models. In contrast to Julius's approach of incorporating behaviors in the definition of models, we see behaviors as the semantic interpretation of systems, which allows us to observe behaviors in different domains. This idea is similar to the one proposed in [46], where timed and time-abstract traces serve as different semantics for the same hybrid automaton. Contract-based design [47] and the use of vertical and horizontal contracts for abstraction and composition, similar to our hierarchical and compositional heterogeneous verification, has been presented in the context of CPS [48] and analog mixed-signal circuits [49].

For combining verification or analysis results across heterogeneous models, ontologies have been used as a knowledge-management approach. Lattice-based ontologies can be used to infer semantic relationships between elements of heterogeneous models [50]. Kumar *et al.* proposed an ontology-based approach for managing knowledge gained from heterogeneous

verification activities and for targeted knowledge acquisition [51]. Rather than treating verification activities as knowledge to be combined, we use logical combination of verification constructs to construct complex verification hierarchies. In a similar spirit, the temporal logic of actions proof system deploys a proof manager that breaks down a complex verification task logically into proof obligations that are proved using theorem provers and satisfiability-modulo-theories solvers [52], but this framework is primarily aimed towards software systems, whereas our framework supports more general (e.g., continuous, hybrid) dynamics and non-deductive analysis methods. Verification architectures have been developed to manage proofs by using high-level switching protocols to verify an overall system property, and require that each mode adhere to this high-level protocol [53]; however, the only supported formalisms are communicating sequential processes, object-Z and duration calculus.

Compositional reasoning is essential for multi-model analysis and verification. Some methods are defined in the context of a single formalism, such as such as assume-guarantee reasoning, with abstraction defined by language inclusion [54] and simulation relations [55], [56], or compositional methods based on deduction [57]. Another example is the behavior-interaction-priority framework for embedded software, which uses structured interaction invariants to support compositional analysis, but only for transition system models [58]. Our objective is to develop a general framework that elucidates the basic conditions for compositional abstraction between any pair of heterogeneous formalisms.

In summary, the existing approaches towards multi-model design and analysis of CPS focus on specific sets of problems, but fall short of providing a comprehensive solution. In the following sections we develop a framework that supports the unified structural representation of CPS and its constituent models, creates a mechanism for managing the models and ensuring consistency between them and the underlying system, and supports heterogeneous formal analysis of the system using the analysis of the individual models in a principled manner.

III. Cyber-Physical System Architectures

This section gives an overview of our architecture-centric approach to design and analysis of CPS with citations to our previous work in this area. We present an extensible architectural style for enabling architectural modeling of CPS and architectural view consistency for comparing the structure and semantics of the model associated with a particular view to the common system architecture.

A. An Architectural Style for CPS

The challenge in defining an architectural style for the physical domain of CPS is to strike a balance between specificity and generality. Architectural models should not have all the details required for a full simulation of the physical dynamics as they are often unnecessary at the architectural level. At the same time, the architectural components and connectors should correspond to intuitive notions of physical dynamics in the same way cyber components and connectors correspond to elements of computational systems. To achieve this balance, we introduce components and connectors based on a *behavioral view of open and interconnected physical systems*, as defined by Willems [59]. This provides a domain-independent perspective, including the ability to represent interactions between different physical domains and the possibility to specify system properties such as power flow and energy conservation laws. In the behavioral approach, laws that govern physical phenomena impose relations on a component's variables, while interconnection means that variables are shared between the connected components, i.e., component behaviors are coupled via their common variables.

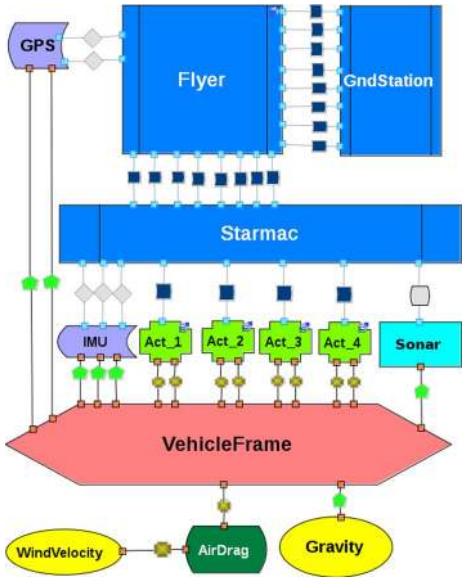


Fig. 3. Base Architecture of STARMAC in AcmeStudio.

The *base architecture* (BA) of a cyber-physical system is an instance of the CPS architectural style and provides the reference structure for all the models used for design and verification. It contains the set of system elements that are related to the analyses carried out in each model, as well as the elements that are common between the models. The BA should contain enough detail to describe the nature of the information exchanged and the physical quantities flowing between components, as well as component connectivity and coupling between physical variables represented by connectors.

Fig. 3 illustrates the use of the CPS style to model the BA of the quadrotor in our custom architecture design environment called AcmeStudio [33]. On the cyber side, each controller (attitude, position, and ground station) is mapped to a separate computation component that implements the control algorithm. The communication of setpoints from a higher-layer controller to a lower-layer controller is modeled as a *send-receive* connector. The periodic relaying of vehicle state from the lower control layer to the higher layer is modeled as a *publish-subscribe* connector. This illustrates the use of distinct connector types to represent different communication patterns between the same components. The vehicle frame is modeled as a rigid-body component, whose mass and moment of inertia are affected by the forces and moments acting at its ports, according to the dynamic equations of the quadrotor.

Each rotor and motor actuator is modeled as a single electromechanical transducer called *Act*, containing an electrical port and two mechanical ports, one each for the translational and rotational domains. The component models the conversion

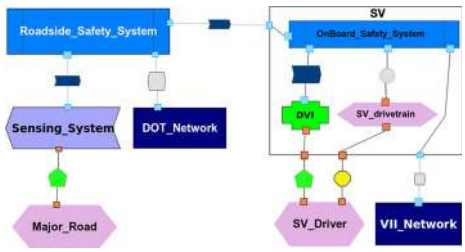


Fig. 4. Base architecture of CICAS-SSA.

of input motor voltage to an output thrust (force) and torque acting on the vehicle frame. As we refine the architecture, this composite component can have substructure, where the motor and rotor are separate components, with a torque connector between them. Each Act is connected to the vehicle frame by two physical coupling connectors, one for force balance and one for moment balance. This models the action and reaction phenomenon between each rotor assembly and the vehicle frame. The drag force is described as a dissipative component, whose magnitude depends on the wind velocity and the aircraft velocity, among other parameters. The complex empirical relationship of drag force to the velocities at its ports is annotated as a behavior property of the component. Gravitational force is modeled as a *flow source* component, since it exerts an independent force on the airframe. It is connected to the vehicle frame by a physical signal connector. Further details about the CPS architectural style and its application to the STARMAC are given in [60].

Fig. 4 shows an architectural model of the CICAS-SSA. It has a roadside safety system as a computation component that receives sensor readings from a highway sensing system, a physical-to-cyber transducer component that senses the physical coordinates of the vehicles on the major road, and a physical component that models the vehicles on the major road. The roadside safety system sends the readings to an encapsulated cyber-physical component of the subject vehicle (SV). The insert in Fig. 4 shows the architectural subcomponents that comprise the SV, namely an on-board safety system for computing, a computation component; the driver-vehicle interface (DVI), a cyber-to-physical transducer component; and the SV drive train, a physical component that can have further substructure not shown at this level. The SV driver is modeled as a physical component that sees the alerts or displays on the DVI and commands acceleration input to the SV drive train.

B. Consistency for Multi-View Architectures

An *architectural view* for a particular analysis domain is a mechanism to relate the architecture of the associated model to the system's common BA. In this context, well-defined mappings between a view and the BA are used to identify and manage semantically equivalent elements (and their connections) between the associated model and the underlying system. Hence, architectural views are an abstraction that represent the assumptions made in the model about the system's structure and connectivity. Views facilitate the separation of concerns during system design. Although views are usually constructed separately, the set of all system views must be related and consistent (in some sense) with the overall architecture, since each view contains a description of the same underlying system.

Since an architecture model represents the system as a graph of components and connectors, the problem of checking for the consistency between a view and the BA can be reduced to checking for the existence of a *graph morphism* between their associated component-connector graphs. We use undirected, typed graphs to model a system's architecture. Associating types with elements is necessary to distinguish between architectures that are topologically identical but represent semantically different systems. For example, one star-topology architecture might represent a mainframe computer being used by multiple thin clients, while another might represent the centralized control of several unmanned aerial vehicles by a single ground station. Mapping architectures into graphs allows us to leverage well-studied tools in graph theory that evaluate the topological similarity between two structures.

View consistency means that an architectural view satisfies the structural and semantic constraints imposed by components and connectors in the system's BA. A view is consistent with the BA if there exists an appropriate graph morphism between the *typed* graphs of the view and the BA. View consistency ensures that the elements in the associated model adhere to the connectivity constraints and physical laws present between elements in the BA. The use of typed graphs allows us to check for a limited notion of *semantic consistency* by capturing the properties of elements in the view and BA as labels of the nodes in the graphs. The graph matching algorithm automatically analyzes the constraints defined between the labels of corresponding nodes in the view and BA graphs as part of the view consistency check. Details of the application of our to the four STARMAC models and the view consistency tools implemented in AcmeStudio are given in [62].

C. Need for Richer Semantics

Multiview architectural modeling provides a rigorous framework for creating structurally consistent representations and managing heterogeneous model-based development of CPS, but safety-critical CPS also need formal verification of the heterogeneous models in order to guarantee correctness of the underlying systems. Hence, richer semantic support is needed in addition to the semantics enforced by the architectural styles and consistent structural deployment of functionality. For example, a sensing view of the CICAS-SSA that analyzes measurement errors [3] looking only at the sensing system component while ignoring others, a network simulation model that studies communication delay between a roadside unit and an intelligent vehicle [63] while ignoring the rest, a physical view that models the physics of the car to model how fast it can move, and an abstract verification view that models simple overapproximated dynamics of the SV and the oncoming vehicles and guarantees a safe intersection-entry strategy, can all have structurally consistent deployment of functionality. Correctness of these individual models along with structural consistency is not enough to conclude correctness of the overall system, however. A richer semantic support for dealing directly with the models themselves is necessary in addition to the structural mappings between their architectural abstractions.

The objective of this richer semantic support is to enable the associations between various heterogeneous models and their respective sets of behaviors in suitable behavior formalisms. The framework needs to be general enough to allow the use of

different modeling and specification formalisms, and different analysis methods and tools. There needs to be support for combining verification results from individual models in a meaningful way to reason about the correctness of the overall system. Complex systems are usually composed of smaller subsystems where different parts are often designed by different engineers possibly at different times, so the framework needs to support compositionality and distributed development in order to draw conclusions about the system correctness from that of the components in a distributed compositional manner. Additionally, semantic assumptions and simplifications are made while constructing models from particular perspectives that hide or abstract information from other perspectives of the system. The framework needs the ability to formally represent these assumptions and ensure system-wide semantic consistency of these assumptions. The remainder of the paper develops the semantic support for heterogeneous formal verification based on behavior semantics to address these issues.

IV. Behavior Semantics for Addressing Heterogeneity

A key challenge in using heterogeneous models to analyze the underlying system is addressing the *semantic heterogeneity*, i.e., relating the semantics of different models defined in different formalisms to the underlying system. Our goal is to create a formal framework that is independent of the specifics of any particular modeling formalism, yet works with every formalism. This section develops a framework based on behavior semantics and their mappings.

A. Modeling Formalisms and Semantic Domains

A *modeling formalism* $\mathcal{M}$ is a set of models of a particular type. Transition systems, hybrid automata, signal-flow models, acausal equation-based models, and network models are some of the modeling formalisms used in CPS. A *model* M is an element of some formalism $\mathcal{M}$. Let B denote a *behavior domain* defined using a given *behavior formalism* $\mathcal{B}$. The behavior formalism could be used to represent, for example, event traces, pairs of continuous input-output signals, or hybrid trajectories, and a particular behavior domain specifies a set of admissible behaviors defined using the given behavior formalism [64].

Let $[\![M]\!]^B$ denote the set of legal behaviors for a given model M with semantics defined in a given behavior domain B in a suitable formalism $\mathcal{B}$. With model semantics defined in different behavior domains possibly from different formalisms, we define *behavior relations* as follows to create mappings between those behavior domains.

Definition 1 (Behavior Relation): Given behavior domains B_1 and B_2 in possibly different behavior formalisms $\mathcal{B}_1$ and $\mathcal{B}_2$, a *behavior relation* is a set $R \subseteq B_1 \times B_2$ that associates pairs of behaviors from the two sets B_1 and B_2.

For a subset of behaviors $B_1' \subseteq B_1$, let $R(B_1')$ denote the set of behaviors in B_2 associated with behaviors in B_1', i.e., $R(B_1') = \{b_2 | \exists b_1 \in B_1' \text{s.t.} (b_1, b_2) \in R\}$. Similarly, for $B_2' \subseteq B_2$, let $R^{-1}(B_2')$ represent the set of behaviors in B_1 associated with behaviors in B_2', i.e., $R^{-1}(B_2') = \{b_1 | \exists b_2 \in B_2' \text{ s.t. } (b_1, b_2) \in R\}$.

This general yet mathematically precise definition supports a variety of formalisms and enables a mechanism to capture the associations and assumptions made while constructing the different models. For example, in case of CICAS-SSA, behavior relations could be used to define how the hybrid traces in an abstract verification model, the trajectories in a control-oriented signal-flow model and the evolution of forces and torques in a physical model relate with the underlying system behaviors and with each other. Note that these associations are often problem-specific and they are already assumed informally while creating these different models; and we facilitate writing them out precisely.

We consider special cases of behavior relations that are also functions, i.e., $R \subseteq B_1 \times B_2$ s.t. $(b_1, b_2) \in R$ and $(b_1, b_2') \in R$ only if $b_2 = b_2'$. We call these *behavior abstraction functions* and denote them as functions $\mathcal{A} : B_1 \to B_2$.

B. Heterogeneous Abstraction and Composition

For all but the most trivial cyber-physical systems, abstraction is essential for making analysis and verification tractable. When interpreted over the same behavior domain B, a model M_2 is an *abstraction* of a model M_1, written $M_1 \sqsubseteq^B M_2$, if $[\![M_1]\!]^B \subseteq [\![M_2]\!]^B$. Many different forms of abstraction have been considered in the literature. We focus on abstraction that corresponds to behavioral inclusion, for example, language or trace inclusion. This mathematical definition of a subset relation captures the notion of overapproximation, whose interpretation could depend on the particulars of the behavior formalism, e.g., in terms of language semantics, trace semantics, reachable sets, and so on.

Definition 2 (Heterogeneous Abstraction): Given behavior domains B_1, B_2 and a behavior relation $R \subseteq B_1 \times B_2$, a model M_2 is an *abstraction* of a model M_1 through R, written $M_1 \sqsubseteq^R M_2$, if

$$[\![M_1]\!]^{B_1} \subseteq R^{-1}\left([\![M_2]\!]^{B_2}\right).$$

This definition asserts that for every behavior in B_1 of model M_1, the behavior relation R associates at least one corresponding behavior in B_2 of model M_2. For example, given a detailed control-oriented Simulink model, an abstract verification model, and a behavior relation between their respective domains, this definition states that we have an abstraction only when for each Simulink trajectory, there is a corresponding abstract trace of the verification model.

When abstract and concrete models across different formalisms are composed of smaller models of interacting components, our objective is to associate the behaviors of component models in isolation with those of the compositions, so that the analysis for the component models can be used to reason about the compositions. We start with a simple special-case scenario in which the semantics of component models P and Q are defined in the same behavior domain. In this case, we define the *semantic composition* of two component models as follows.

Definition 3 (Semantic Composition): Given component models P and Q from the same modeling formalism $\mathcal{M}$ with semantics defined in behavior domain B, the composition $P \| Q$ is a model in $\mathcal{M}$ s.t.

$$[\![P \| Q]\!]^B = [\![P]\!]^B \cap [\![Q]\!]^B. \tag{1}$$

This definition of composition as the intersection of behavior sets is consistent with the literature for composition using many but not all behavior domains [43]–[45]. In particular, it requires

the semantics to allow variables to change arbitrarily if left unspecified in the models. For a given modeling formalism $\mathcal{M}$, syntactic techniques may exist for creating a composition, e.g., construction of product automata. We allow all such procedures to be used to define the composition, so long as (1) holds.

In a more general scenario, the component models P and Q can have different local behavior domains B^P and B^Q in behavior formalism $\mathcal{B}$. These local behavior domains have only the variables pertaining to a specific component while excluding others. In such a case, we need to lift the local semantics of the components to common global behavior domains before we can compose them. We begin by defining the relationship between behaviors in a global domain and a local domain.

Definition 4 (Behavior Localization): Given a behavior formalism $\mathcal{B}$ and two behavior domains $B, B' \in \mathcal{B}$, an onto function $\downarrow\colon B \to B'$ (i.e., every element of B' has at least one pre-image in B) is called a *(behavior) localization* of domain B to domain B'.

Given a localization $\downarrow$ of B to B', for $b \in B$, we will let $b\downarrow$ be a shorthand for $\downarrow(b)$. The set-valued extension of localization can be defined in the usual way. For $b' \in B'$ we will let $b'\uparrow$ denote the set-valued function $\uparrow\colon B' \to 2^B - \{\emptyset\}$ defined by $\uparrow(b') = \downarrow^{-1}(b') = \{b \in B | b\downarrow = b'\}$. We will call the function $\uparrow$ a *(behavior) globalization* of B' to B. Note that $b'\uparrow$ is always non-empty since the localization function $\downarrow$ is onto.

Behavior localization and globalization are generally inferred from relationships between models from given modeling formalisms and associated definitions of the relationships between model primitives and their semantic interpretations. Given the definitions of localization and globalization of behavior domains, we define model globalization as follows.

Definition 5 (Model Globalization): Given a global behavior domain B, a model P with its local behavior domain B', and a behavior localization function $\downarrow\colon B \to B'$, the *(model) globalization* of P is any model P^G s.t. $[\![P^G]\!]^B = [\![P]\!]^{B'}\uparrow$.

For a given modeling formalism $\mathcal{M}$, syntactic approaches for globalization may exist, e.g., addition of self loops for newly-added event labels for discrete transition systems, or addition of state variables with unconstrained dynamics for continuous dynamic systems. We allow the use of all such syntactic preprocessing procedures that lead to models with the correct set of behaviors $[\![P]\!]^{B'}\uparrow$ before composition.

The following definition generalizes the notion of semantic composition from Def. 3.

Definition 6 (Globalized Semantic Composition): Given a global behavior domain B, component models P and Q with their corresponding local behavior domains B^P and B^Q, and behavior localizations $\downarrow^P\colon B \to B^P$ and $\downarrow^Q\colon B \to B^Q$, the *globalized semantic composition* of P and Q in the global behavior domain B, denoted by $P\|^G Q$ is the semantic composition of models P^G and Q^G, which are the globalizations of P and Q, respectively, i.e., $P\|^G Q = P^G \| Q^G$.

While syntactic procedures can produce different model globalizations, they still yield semantically equivalent compositions in terms of sets of behaviors.

C. Example

Consider the task of establishing the collision freedom of CICAS-SSA for one oncoming vehicle, called the principal other vehicle (POV) and one subject vehicle (SV) waiting at the stop sign, as shown in Fig. 5. The collision freedom of each POV and each SV is an integral part in establishing the correctness of the overall system.[1] The POV appears in the sensing range, continues to approach the intersection, enters the intersection and eventually clears it. The SV can either go straight or turn right to merge into the POV's path. Once they crosses the intersection, we no longer treat the POV or the SV to be relevant as next vehicles take their place. Assuming the road coordinate along the POV path is X and those along the straight and right-turn paths of the SV are Y and Z, the potential conflict area in and adjacent to the intersection is shown with dimensions depicted by the bold line segments. Fig. 5 also shows typical intersection geometry parameters such as the width of a highway lane, the range of when a vehicle is first detected, and a conservative estimate of when it is too close (300m) assuming it travels at highway speeds between 20 m/s and 30 m/s.

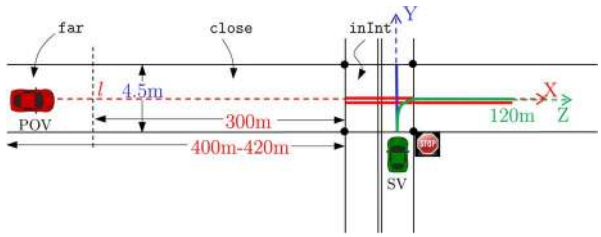


Fig. 5. Instance of the CICAS-SSA with one near-side oncoming lane with one POV.

Fig. 6 shows a hybrid-automaton model M_0 for the system in Fig. 5. The model shows a parallel composition of POV and SV component models. The continuous dynamics of the POV can be represented by a differential inclusion based on highway speed limits. The SV has a choice of entering the intersection or merging into the traffic only if it is safe to do so. It is forced to stay stopped if unsafe or can decide to stay stopped by choice. Once it enters the intersection or merges into the traffic, it continues to drive with some minimum bounds of acceleration along either y or z and eventually clears the conflict zones. Because we do not care about what happens after either car clears the intersection, we ignore the corresponding dynamics after clearing the intersection by not updating the dynamics for SV and by choosing appropriate invariant for POV.

The behavior formalism of choice for this model is hybrid trajectories, with local behavior domains for the component models being the set of all 1-D trajectories in the variable x and the set of all 5-D hybrid trajectories in the variables x, y, v_y, z, and v_z, which stand for POV position and SV position and velocities while going straight and merging right. Parallel composition of hybrid automata can be used to form a composition, after addition of unrestricted dynamics along y, v_y, z and v_z serves as the globalization for the POV component. In the next two sections, we present the hierarchical and compositional heterogeneous verification of this model.

In summary, this section develops a general semantic framework for associating behaviors across different domains. Behavior relations serve as mappings between domains at different levels of abstraction, and localization/globalization mappings serve as mappings between local and global semantics. The local semantics defines the meaning of the component models

[1]The overall system verification for CICAS-SSA is presented as a case study in [65].

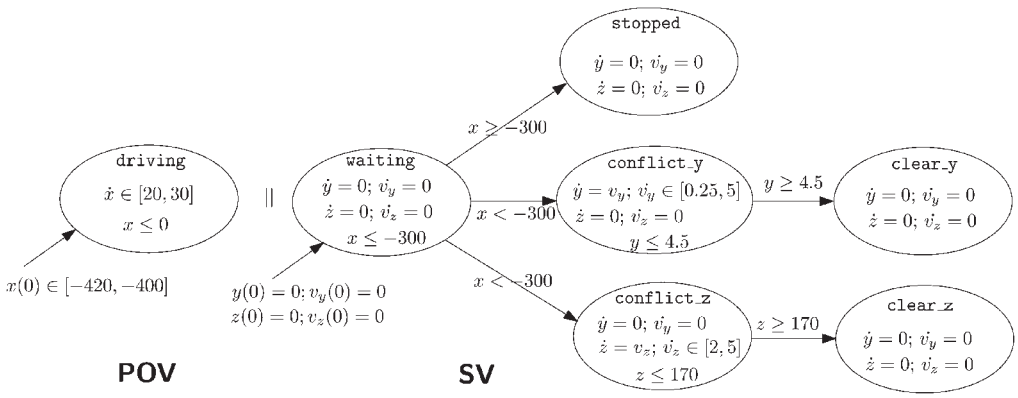


Fig. 6. Hhybrid automaton model M_0 for CICAS-SSA.

when considered in isolation, and the global semantics defines the meaning of the whole system models when the components are composed to a system. We exploit these semantic mappings to develop hierarchical heterogeneous verification across different levels of abstraction in Section V, and compositional heterogeneous verification across individual component models at a given level of abstraction in Section VI.

V. Heterogeneous Verification

A *specification* S in a *specification formalism* $\mathcal{S}$ is an indirect representation that captures what the system can or cannot do, typically without any implementation details. Specifications could be written in, for example, various temporal logics, Kripke structures, automata, sets of unsafe states to be avoided, or even in English language, so long as their semantic interpretation is clear in terms of a given behavior domain in the associated behavior formalism.

The *semantic interpretation* of S in a behavior domain B, denoted by $[\![S]\!]^B$, is defined as the set of all behaviors in B for which the specification is satisfied. When semantically interpreted over the same set of behaviors B, a specification S_2 is said to imply a specification S_1, written $S_2 \Rightarrow^B S_1$ if $[\![S_2]\!]^B \subseteq [\![S_1]\!]^B$. This simply asserts that any behavior that satisfies S_2 also satisfies S_1.

The following definition extends this notion to heterogeneous behavior spaces using behavior relations.

Definition 7 (Heterogeneous Implication): Given behavior domains B_1, B_2 and a behavior relation $R \subseteq B_1 \times B_2$, we say that specification S_2 *implies* specification S_1 *via* R, written $S_2 \Rightarrow^R S_1$, if

$$R^{-1}\left([\![S_2]\!]^{B_2}\right) \subseteq [\![S_1]\!]^{B_1}.$$

This definition requires that if a behavior $b_1 \in B_1$ is associated through R with a behavior in $b_2 \in B_2$ that satisfies S_2, then b_1 satisfies S_1.

In a given behavior domain B, a model M *entails* a specification S, written $M \models^B S$, if $[\![M]\!]^B \subseteq [\![S]\!]^B$. When true, this simply asserts that the set of behaviors of the model M do not violate the set of safe behaviors allowed by the specification S. To establish this type of entailment, formal approaches such as reachability analysis and theorem proving, or semiformal approaches like systematic state-space exploration, need to be used. The following proposition states general conditions under which one can perform heterogeneous verification.

Proposition 1 (Heterogeneous Verification): Given two behavior domains B_0 and B_1 in behavior formalisms $\mathcal{B}_0$ and $\mathcal{B}_1$, models M_0 and M_1 in modeling formalisms $\mathcal{M}_0$ and $\mathcal{M}_1$, specifications S_0 and S_1 in specification formalisms $\mathcal{S}_0$ and $\mathcal{S}_1$, and a behavior relation $R \subseteq B_0 \times B_1$, if $M_0 \sqsubseteq^R M_1$, $M_1 \models^{B_1} S_1$ and $S_1 \Rightarrow^R S_0$, then $M_0 \models^{B_0} S_0$.

Proof: From $M_0 \sqsubseteq^R M_1$, we have

$$\begin{aligned}[\![M_0]\!]^{B_1} &\subseteq R^{-1}\left([\![M_1]\!]^{B_1}\right)\\ (\text{From } M_1 \models^{B_1} S_1) &\subseteq R^{-1}\left([\![S_1]\!]^{B_1}\right)\\ (\text{From } S_1 \Rightarrow^R S_0) &\subseteq [\![S_0]\!]^{B_0}.\end{aligned}$$

Therefore, $M_0 \models^{B_0} S_0$. ■

There are two natural ways of using *multiple* models and specifications together.

1) Conjunctive Multi-Model Heterogeneous Verification: In this case, each model is a heterogeneous abstraction of the underlying system and we need to ensure that the specifications checked against each model together imply the specification of the underlying system. The following definition makes this notion formal.

Definition 8 (Conjunctive Heterogeneous Implication): Given system behavior domain B_0, behavior domains B_i and behavior relations $R_i \subseteq B_0 \times B_i$, specifications S_i for $i = 1, \ldots, n$ *conjunctively imply* the system specification S_0 if

$$\bigcap_i R_i^{-1}\left([\![S_i]\!]^{B_i}\right) \subseteq [\![S_0]\!]^{B_0}.$$

This definition allows the individual specifications S_i to not imply S_0, but their conjunction (intersection of the allowed behaviors) is required to be stronger than S_0.

Proposition 2 (Heterogeneous Conjunctive Analysis): For a system model M_0 with a behavior domain B_0 and specification S_0, given models M_i with the corresponding behavior domains B_i, specifications S_i and behavior relations $R_i \subseteq B_0 \times B_i$, if $M_0 \sqsubseteq^{R_i} M_i$, specifications S_i conjunctively imply S_0, and $M_i \models^{B_i} S_i$ for each $i = 1, \ldots, n$, then $M_0 \models^{B_0} S_0$.

Proof: From $M_0 \sqsubseteq^{R_i} M_i$ for each i, we have

$$\begin{aligned}[\![M_0]\!]^{B_0} &\subseteq \bigcap_i R_i^{-1}\left([\![M_i]\!]^{B_i}\right)\\ (\text{since } M_i \models^{B_i} S_i) &\subseteq \bigcap_i R_i^{-1}\left([\![S_i]\!]^{B_i}\right)\\ (\text{Conj. Het. Implication}) &\subseteq [\![S_0]\!]^{B_0}.\end{aligned}$$

Therefore, $M_0 \models^{B_0} S_0$. ■

2) Disjunctive Multi-Model Heterogeneous Verification: Now we consider the case where different models are built to represent different subsets of behaviors of a system. This is typically useful when there are different behaviors in different operating regimes best modeled by different models, where neither one fully represents the whole set of behaviors of the system, but their union does. This notion is made formal by the following definition.

Definition 9 (Model Coverage): For a system model M_0 with a behavior domain B_0, given a set of models M_i with corresponding behavior domains B_i and behavior relations $R_i \subseteq B_0 \times B_i$, models M_i, $i = 1, \ldots, n$ *cover* M_0 if there exists a partition $\{B_0^1, B_0^2, \ldots, B_0^n\}$ of $[\![M_0]\!]^{B_0}$ s.t. $\forall i = 1, \ldots, n$

$$B_0^i \subseteq R_i^{-1}\left([\![M_i]\!]^{B_i}\right).$$

This definition requires that every behavior of the underlying system M_0 be accounted for by at least one model.

Lemma 1: If models M_i cover M_0 through R_i, $i = 1, \ldots, n$, we have

$$[\![M_0]\!]^{B_0} = \bigcup_{i=1}^{n} R_i^{-1}\left([\![M_i]\!]^{B_i}\right).$$

Proof: From the definition of partition, we have

$$\begin{aligned}[\![M_0]\!]^{B_0} &= \bigcup_{i=1}^{n} B_0^i \\ (\text{Def. } 9) &\subseteq \bigcup_{i=1}^{n} R_i^{-1}\left([\![M_i]\!]^{B_i}\right).\end{aligned}$$

■

In the disjunctive case, no model is a proper abstraction of the underlying system, only all models together cover it. Hence, in order to make sure that a specification holds for the underlying system we need to verify that each of the disjunctive models satisfies that specification.

Proposition 3 (Heterogeneous Disjunctive Analysis): For system model M_0 with a behavior domain B_0 and specification S_0, given models M_i with the corresponding behavior domains B_i, specifications S_i and behavior relations $R_i \subseteq B_0 \times B_i$, if each specification S_i heterogeneously implies S_0, models M_i cover M_0, and $M_i \models^{B_i} S_i$ for each $i = 1, \ldots, n$, then $M_0 \models^{B_0} S_0$.

Proof: From Lemma 1, we have

$$\begin{aligned}[\![M_0]\!]^{B_0} &\subseteq \bigcup_{i} R_i^{-1}\left([\![M_i]\!]^{B_i}\right) \\ (\text{since } M_i \models^{B_i} S_i) &\subseteq \bigcup_{i} R_i^{-1}\left([\![S_i]\!]^{B_i}\right) \\ (\text{Het. Implication}) &\subseteq [\![S_0]\!]^{B_0}.\end{aligned}$$

Therefore, $M_0 \models^{B_0} S_0$. ■

Finally, we note that the conjunctive and disjunctive analysis constructs can be nested arbitrarily. For example, the jth conjunctive verification subtask $M_j \models^{B_j} S_j$ can be broken down disjunctively into its subtasks $M_{ji} \models^{B_{ji}} S_{ji}$ by creating new models that cover M_j and specifications that imply S_j. Thus, using the nesting of conjunctive and disjunctive constructs, any arbitrary propositional logical breakdown of a system verification task can be achieved. This is illustrated in an example in the following subsection.

A. Example

Consider the following safety verification problem for the CICAS model M_0 from Fig. 6. A safety violation (potential collision) occurs if by the time POV enters the intersection, SV is still in the conflict zone. The absence of this violation can be written as a temporal logic specification $S_0 : \Box\neg((x == 0 \wedge 0 < y < 4.5) \vee (x == 0 \wedge 0 < z < 170))$. The objective is to show that M_0 satisfies S_0.

1) Disjunctive Analysis: We note that the SV has to be safe irrespective of whether it is crossing or merging into the lane. We can model these behaviors individually and verify their safety independently in a disjunctive verification construct. Fig. 7 shows a model M_1 of the system where SV is only allowed to intersect the traffic. A similar model M_2 can be constructed for the merging case.

In place of the behavior domain B_0 of model M_0 as the class of all 5-D hybrid traces, the behavior domains B_1 and B_2 for models M_1 and M_2 are classes of all 3-D hybrid traces. The behavior relations between these domains and the original domain are:

- $R_1 : \{(b_0, b_1) | b_0 \downarrow_{z,v_z} == \bar{0}$ and $b_0 \downarrow_{x,y,v_y} == b_1\}$;
- $R_2 : \{(b_0, b_2) | b_0 \downarrow_{y,v_y} == \bar{0}$ and $b_0 \downarrow_{x,z,v_z} == b_2\}$;

where $\bar{0}$ represents a 2-D trace of zeros over all time and $\downarrow_{()}$ represents the projection on ().

We construct simpler specifications to be checked for the two models as:

- $S_1 : \Box\neg(x == 0 \wedge 0 < y < 4.5)$; and
- $S_2 : \Box\neg(x == 0 \wedge 0 < z < 170)$.

The heterogeneous implication $S_1 \Rightarrow^{R_1} S_0$ holds because $R_1^{-1}([\![S_1]\!]^{B_1})$ forces that y be conflict-free and z be 0, which implies that y is conflict-free and z is conflict-free. Similarly, we have $S_2 \Rightarrow^{R_2} S_0$. Further, we note that in every behavior of M_0, either $\{y, v_y\}$ or $\{z, v_z\}$ are zero and both the possibilities are covered by either model. Therefore, from Prop. 3, if $M_1 \models^{B_1} S_1$ and $M_2 \models^{B_2} S_2$, we can conclude $M_0 \models^{B_0} S_0$. Next, we show $M_1 \models^{B_1} S_1$ using conjunctive analysis. $M_2 \models^{B_2} S_2$ can be shown in a similar manner.

2) Conjunctive Analysis: Consider the subtask of showing $M_1 \models^{B_1} S_1$. We break down this task conjunctively by creating three models M_{1i} and constructing corresponding specifications S_{1i}, $i = 1, 2, 3$, as shown in Fig. 8. M_{11} models the behaviors of the POV, and is exactly the same as the POV automaton in M_1. M_{12} models the behavior of the SV only while it is in the conflict zone and has the same dynamics as that of the `conflict_y` location of M_1. M_{13} is a discrete model consisting of two elements. The component POV is a created by partitioning the component POV of M_1 into discrete states `far`, `close`, and `inInt` using predicates $x \leq -300$, $-300 \leq x \leq 0$, and $0 \leq x$. The second component SV is merely a discrete control graph of the hybrid automaton model for SV in M_1. The only synchronized pair of transitions is $(\texttt{far} \xrightarrow{\beta_1} \texttt{close})$ and $(\texttt{waiting} \xrightarrow{\beta_1} \texttt{stopped})$.

The behavior relations are:

- $R_{11} : \{(b_1, b_{11}) | b_{11} == b_1 \downarrow_x\}$;
- $R_{12} : \{(b_1, b_{12}) | b_{12} == s_1 \downarrow_{y,v_y}$ where s_1 is b_1 restricted to the discrete location (`driving`, `conflict_y`);
- $R_{13} : \{(b_1, b_{13}) | b_1$ is a hybrid trajectory that visits the discrete locations corresponding to ones in b_{13} in that order.

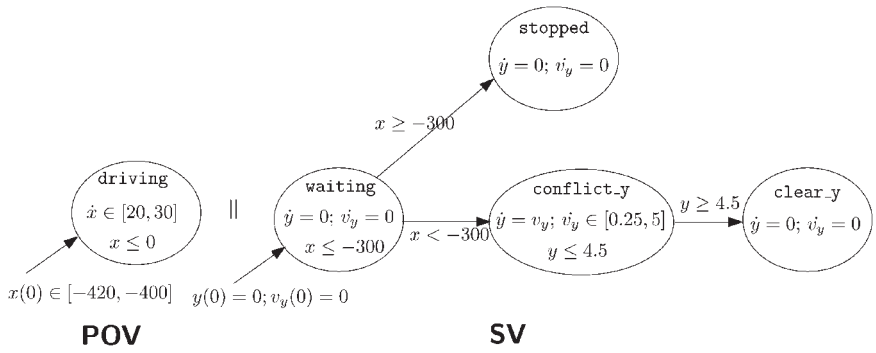


Fig. 7. Hybrid model M_1 for SV going only straight if safe.

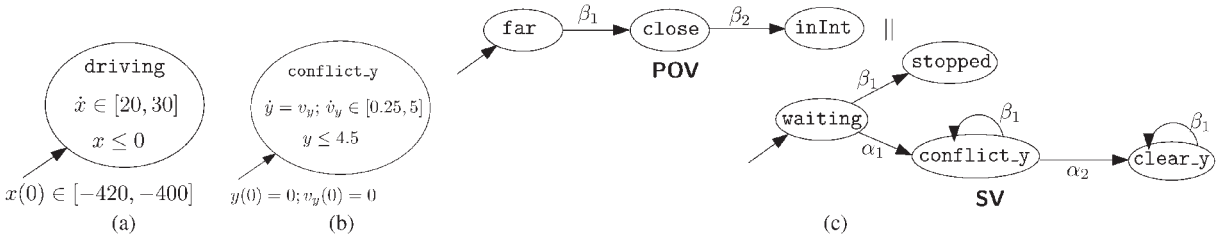


Fig. 8. Abstractions M_{1i} of M_1 representing the POV dynamics, the SV dynamics in the conflict zone and the discrete protocol. (a) Model M_{11}. (b) Model M_{12}. (c) Model M_{13}.

For these behavior relations, we first note that $M_1 \sqsubseteq^{R_{1i}} M_{1i}$ because neither of the models M_{1i} is more restrictive than M_1. The specifications for the three models are:

- $S_{11} : \Box(x == -300 \Rightarrow \Box_9 x < 0)$;
- $S_{12} : \Box(\diamond_8 \geq 4.5)$;
- $S_{13} : \Box((\texttt{POVclose} \wedge \neg\texttt{SVdriving}) \rightarrow \neg(\diamond\texttt{SVdriving}))$, where `POVclose` is satisfied in states $(\texttt{close}, \cdot)$ and $(\texttt{inInt}, \cdot)$; and `SVdriving` is satisfied in states $(\cdot, \texttt{conflict\_y})$.

The behaviors effectively allowed in B_1 by the specifications S_{1i} are as follows:

- $R_{11}^{-1}([\![S_{11}]\!])$: system behaviors where POV takes at least 9 seconds to get from $l = -300$ to the intersection.
- $R_{12}^{-1}([\![S_{12}]\!])$: system behaviors where SV clears the intersection within 8 seconds of starting to drive.
- $R_{13}^{-1}([\![S_{13}]\!])$: system behaviors where SV does not start driving after POV crosses l.

There can only be two cases:

1) The SV has already started driving before the POV crosses l and is in the intersection: in this case, from $R_{11}^{-1}([\![S_{11}]\!])$ and $R_{12}^{-1}([\![S_{12}]\!])$ together, it will clear the intersection in at most 8 seconds and the POV will not get to the intersection in at least 9 seconds; OR
2) The SV has not started driving when the POV crosses l: in this case, from $R_{13}^{-1}([\![S_{13}]\!])$, the SV cannot start driving anymore.

Therefore, from all the specifications put together, the two cars can't be in the intersection at the same time, which implies S_1, i.e., we have conjunctive heterogeneous implication.

$M_{11} \models^{B_{11}} S_{11}$ can be shown by algebraic computations: for the fastest velocity (30 m/s) it takes 10 s to travel 300 m. $M_{12} \models^{B_{12}} S_{12}$ can be shown by Newton's laws of motion: the longest time needed to cross 4.5 m with initial velocity 0 and minimum acceleration 0.25 m/s^2 is $\sqrt{(2 * 4.5)/0.25} = 6$ seconds. $M_{13} \models^{B_{13}} S_{13}$ can be shown by using Labeled Transition System Analyzer [30]. Under these conditions, using Prop. 2, we can infer that $M_1 \models^{B_1} S_1$.

In summary, in this section we developed the notion of hierarchical heterogeneous verification, which studies how verification results can be combined across different levels of abstraction. Conjunctive and disjunctive heterogeneous verification constructs can be nested arbitrarily to create arbitrary mixes of conjunctive and disjunctive analysis hierarchies. A detailed heterogeneous verification hierarchy for CICAS-SSA is presented in [65].

VI. Compositional Heterogeneous Verification

In this section, we study conditions that ensure that the composition of abstractions for individual components is an abstraction for the composition of the components. We use behavior abstraction functions as the semantic mappings since arbitrary behavior relations that are not functions are not sufficient for compositionality [66].

Fig. 9 illustrates the compositional heterogeneous abstraction problem considered in this paper. For each of the two levels of abstraction, $i = 0, 1$, we assume there is a modeling formalism $\mathcal{M}_i$ and a behavior formalism $\mathcal{B}_i$. Component models $P_i, Q_i \in \mathcal{M}_i$ have their semantics defined in terms of *local* behavior domains $B_i^P, B_i^Q \in \mathcal{B}_i$. These local domains include only the variables relevant to the given component. Heterogeneous abstraction between the two models of each component is established via behavior abstraction functions $\mathcal{A}^P$ and $\mathcal{A}^Q$ (Section IV) that are mappings between the respective local behavior domains. To compose the two models to form the system models $M_i \in \mathcal{M}_i$, the local semantics are lifted to global behavior domains $B_i \in \mathcal{B}_i$ to include variables from both components. We seek conditions under which heterogeneous abstraction between component models in their local behavior domains implies heterogeneous abstraction between the composite system models in the global behavior domains.

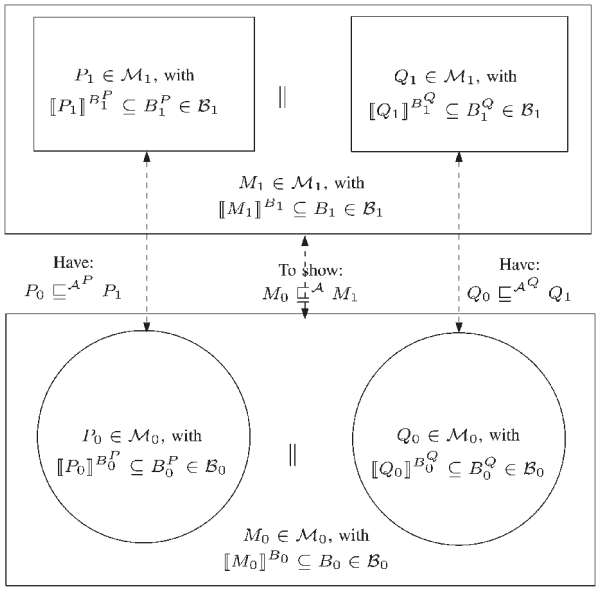


Fig. 9. Compositional heterogeneous abstraction analysis.

We note that for globalized semantic composition, models need to be globalized using behavior globalization as stated in Def. 5. Similarly, in order to relate the semantic mappings of the individual components together at a common level, we develop the following notion of globalization of behavior abstraction functions.

Definition 10 (Abstraction Globalization): Given two behavior formalisms $\mathcal{B}_0$ and $\mathcal{B}_1$, behavior domains $B_0, B_0' \in \mathcal{B}_0$ and $B_1, B_1' \in \mathcal{B}_1$ from each behavior formalism, localizations $\downarrow_i$ of B_i to B_i' for $i = 1, 2$, and a behavior abstraction function $\mathcal{A}'$ of B_0' to B_1', a behavior abstraction function $\mathcal{A}$ of B_0 to B_1 is said to be a *globalization* of $\mathcal{A}'$ if

$$\forall\, b_0 \in B_0 : \mathcal{A}'(b_0 \downarrow_0) = \mathcal{A}(b_0) \downarrow_1 . \tag{2}$$

We write $\mathcal{A} = \mathcal{A}' \Uparrow$ if $\mathcal{A}$ is a globalization of $\mathcal{A}'$. We call $\mathcal{A}'$ a *localization* of $\mathcal{A}$, written $\mathcal{A}' = \mathcal{A} \Downarrow$, iff $\mathcal{A} = \mathcal{A}' \Uparrow$. We have shown that globalization of a given local behavior abstraction function always exists, but is not necessarily unique; while the localization of a global behavior abstraction function may not always exist (because projections cause loss of information), but the localization is unique if it exists [66]. From the uniqueness of localization and non-uniqueness of globalization, we note that

$$(\mathcal{A}' \Uparrow) \Downarrow = \mathcal{A}'; \tag{3}$$

but $(\mathcal{A} \Downarrow) \Uparrow$ may not be equal to $\mathcal{A}$.

In the next two subsections we find conditions under which compositional heterogeneous abstraction w.r.t. Fig. 9 can be used.

A. Heterogeneous Abstraction in Global Behavior Domains

We start with a simple special-case scenario w.r.t. Fig. 9 in which $B_i^P = B_i^Q = B_i \in \mathcal{B}_i,\ i = 0, 1$. For this special case, only one behavior abstraction function $\mathcal{A}$ is sufficient, as we can set $\mathcal{A}^P = \mathcal{A}^Q = \mathcal{A}$. In this case, the following proposition gives conditions for compositional heterogeneous abstraction.

Proposition 4: For each abstraction level $i = 0, 1$, given component models P_i, Q_i with the semantics of each model interpreted over a behavior domain B_i, and a behavior abstraction function $\mathcal{A} : B_0 \to B_1$, if $P_0 \sqsubseteq^{\mathcal{A}} P_1$ and $Q_0 \sqsubseteq^{\mathcal{A}} Q_1$, then $P_0 \| Q_0 \sqsubseteq^{\mathcal{A}} P_1 \| Q_1$.

Proof: From $P_0 \sqsubseteq^{\mathcal{A}} P_1$ and $Q_0 \sqsubseteq^{\mathcal{A}} Q_1$, we have $[\![P_0]\!]^{B_0} \subseteq \mathcal{A}^{-1}([\![P_1]\!]^{B_1})$ and $[\![Q_0]\!]^{B_0} \subseteq \mathcal{A}^{-1}([\![Q_1]\!]^{B_1})$. Therefore, $[\![P_0 \| Q_0]\!]^{B_0} = [\![P_0]\!]^{B_0} \cap [\![Q_0]\!]^{B_0} \subseteq \mathcal{A}^{-1}([\![P_1]\!]^{B_1}) \cap \mathcal{A}^{-1}([\![Q_1]\!]^{B_1}) = \mathcal{A}^{-1}([\![P_1]\!]^{B_1} \cap [\![Q_1]\!]^{B_1}) = \mathcal{A}^{-1}([\![P_1 \| Q_1]\!]^{B_1})$. ■

This proposition states that with global semantics, composition of abstractions is the abstraction of the composition. Next we consider the general case where the local semantics of the two components are defined in terms of distinct behavior domains.

B. Heterogeneous Abstraction in Local Behavior Domains

The following lemma states that heterogeneous abstraction between model globalizations via a global abstraction function is equivalent to heterogeneous abstraction between original models via the localization of the global abstraction function.

Lemma 2: For abstraction levels $i = 0, 1$, given component models P_i with local behavior domains B_i', behavior localization functions $\downarrow_i$: $B_i \to B_i'$, let their corresponding globalized models be P_i^G with global behavior domains B_i. If $\mathcal{A} : B_0 \to B_1$ is a global behavior abstraction function and $\mathcal{A}' : B_0' \to B_1'$ is a localization of $\mathcal{A}$, then $P_0^G \sqsubseteq^{\mathcal{A}} P_1^G \Leftrightarrow P_0 \sqsubseteq^{\mathcal{A}'} P_1$.

Proof: From the definition of model globalization, we have

$$b_i \in \left[\!\left[P_i^G\right]\!\right]^{B_i} \Leftrightarrow b_i \downarrow_i \in [\![P_i]\!]^{B_i'} \tag{4}$$

and

$$b_i' \in [\![P_i]\!]^{B_i'} \Leftrightarrow b_i' \uparrow_i \subseteq \left[\!\left[P_i^G\right]\!\right]^{B_i} . \tag{5}$$

Case I: $P_0^G \sqsubseteq^{\mathcal{A}} P_1^G \Rightarrow P_0 \sqsubseteq^{\mathcal{A}'} P_1$: This can can be shown as follows. For any given $b_0 \in [\![P_0^G]\!]^{B_0}$, let $b_1 := \mathcal{A}(b_0)$. From $P_0^G \sqsubseteq^{\mathcal{A}} P_1^G$, we have $b_1 \in [\![P_1^G]\!]^{B_1}$. From (2), $\mathcal{A}'(b_0') = b_1 \downarrow_1$, where $b_0' := b_0 \downarrow_0$. Hence, from (4) we have that $\forall\, b_0' \in [\![P_0]\!]^{B_0'}$, $\mathcal{A}'(b_0') \in [\![P_1]\!]^{B_1'}$, which implies $[\![P_0]\!]^{B_0'} \subseteq \mathcal{A}'^{-1}([\![P_1]\!]^{B_1'})$, i.e., $P_0 \sqsubseteq^{\mathcal{A}'} P_1$.

Case II: $P_0^G \sqsubseteq^{\mathcal{A}} P_1^G \Leftarrow P_0 \sqsubseteq^{\mathcal{A}'} P_1$: This can be shown as follows. From $P_0 \sqsubseteq^{\mathcal{A}'} P_1$, we have $b_0' \in [\![P_0]\!]^{B_0'} \Rightarrow \mathcal{A}'(b_0') =: b_1' \in [\![P_1]\!]^{B_1'}$. From Def. 10 and (5), for any $b_0' \in [\![P_0]\!]^{B_0'}$, $b_0 \in b_0' \uparrow_0 \subseteq [\![P_0^G]\!]^{B_0} \Rightarrow \mathcal{A}(b_0) =: b_1 \in b_1' \uparrow_1 \subseteq [\![P_1^G]\!]^{B_1}$. Therefore, $[\![P_0^G]\!]^{B_0} \subseteq \mathcal{A}^{-1}([\![P_1^G]\!]^{B_1})$, i.e., $P_0^G \sqsubseteq^{\mathcal{A}} P_1^G$. ■

Lemma 2 implies the following reasoning indicated in Fig. 9. When the abstract and concrete models of a component are considered in isolation, it does not matter whether one does the heterogeneous abstraction analysis in the global domains or the local domains. We now use the result from Lemma 2 in a compositional setting when the component models are composed to form a system model.

For the following discussion, we let models M_i, with the global behavior domains B_i, be the globalized compositions $P_i \|^G Q_i$ of component models P_i and Q_i with their local behavior domains B_i^P and B_i^Q, for levels of abstraction $i = 0, 1$ as depicted in Fig. 9. We consider two scenarios in which the source of the abstraction is at the system level and component levels, respectively.

1) Centralized Development: First, we consider the case where we have an abstraction function $\mathcal{A} : B_0 \to B_1$ between the global behavior domains B_0 and B_1.

Proposition 5: For abstraction levels $i=0,1$, given component models P_i and Q_i with corresponding local behavior domains B_i^P and B_i^Q, let their globalized semantic compositions be $P_i \| ^G Q_i$ in global behavior domains B_i with behavior localizations $\downarrow_i^j : B_i \to B_i^j$, where $j = P, Q$, and a global behavior abstraction function $\mathcal{A} : B_0 \to B_1$. If localizations $\mathcal{A}\Downarrow^P$ and $\mathcal{A}\Downarrow^Q$ of $\mathcal{A}$ exist and $P_0 \sqsubseteq^{\mathcal{A}\Downarrow^P} P_1$ and $Q_0 \sqsubseteq^{\mathcal{A}\Downarrow^Q} Q_1$, then $M_0 \sqsubseteq^{\mathcal{A}} M_1$.

Proof: From $P_0 \sqsubseteq^{\mathcal{A}^P} P_1$ and $Q_0 \sqsubseteq^{\mathcal{A}^Q} Q_1$, we know from Lemma 2 that $P_0^G \sqsubseteq^{\mathcal{A}} P_1^G$ and $Q_0^G \sqsubseteq^{\mathcal{A}^Q} Q_1^G$, i.e., that $[\![P_0^G]\!]^{B_0} \subseteq \mathcal{A}^{-1}([\![P_1^G]\!]^{B_1})$ and $[\![Q_0^G]\!]^{B_0} \subseteq \mathcal{A}^{-1}([\![Q_1^G]\!]^{B_1})$. We have, $[\![P_0 \|^G Q_0]\!]^{B_0} = [\![P_0^G]\!]^{B_0} \cap [\![Q_0^G]\!]^{B_0} \subseteq \mathcal{A}^{-1}([\![P_1^G]\!]^{B_1}) \cap \mathcal{A}^{-1}([\![Q_1^G]\!]^{B_1}) = \mathcal{A}^{-1}([\![P_1^G]\!]^{B_1} \cap [\![Q_1^G]\!]^{B_1}) = \mathcal{A}^{-1}([\![P_1 \|^G Q_1]\!]^{B_1})$. ■

Prop. 5 states that we can establish $M_0 \sqsubseteq^{\mathcal{A}} M_1$ in the global behavior domains by establishing $P_0 \sqsubseteq^{\mathcal{A}\Downarrow^P} P_1$ and $Q_0 \sqsubseteq^{\mathcal{A}\Downarrow^Q} Q_1$ in the local behavior domains of the two components.

2) Decentralized Development: Now, we consider the case where the abstraction functions $\mathcal{A}^P : B_0^P \to B_1^P$ and $\mathcal{A}^Q : B_0^Q \to B_0^Q$ between the local behavior domains B_i^P and B_i^Q are given and heterogeneous abstractions of component models $P_0 \sqsubseteq^{\mathcal{A}^P} P_1$ and $Q_0 \sqsubseteq^{\mathcal{A}^Q} Q_1$ are established independently. This is the more common situation in practice, particularly for distributed development. In this case, the following proposition states that if the globalizations of abstraction functions $\mathcal{A}^P \Uparrow^P$ and $\mathcal{A}^Q \Uparrow^Q$ are defined consistently, the heterogeneous abstraction results for the components carry over to their compositions.

Proposition 6: For abstraction levels $i=0,1$, given component models P_i and Q_i with local behavior domains B_i^P and B_i^Q, let their compositions be $P_i \| ^G Q_i$ in global behavior domains B_i and local behavior abstraction functions be $\mathcal{A}^P$: $B_0^P \to B_1^P$ and $\mathcal{A}^Q$: $B_0^Q \to B_1^Q$ s.t. $P_0 \sqsubseteq^{\mathcal{A}^P} P_1$ and $Q_0 \sqsubseteq^{\mathcal{A}^Q} Q_1$. If $\mathcal{A}^P \Uparrow^P = \mathcal{A}^Q \Uparrow^Q =: \mathcal{A}$, i.e., then $P_0 \| ^G Q_0 \sqsubseteq^{\mathcal{A}} P_1 \| ^G Q_1$.

Proof: The result follows due to $(\mathcal{A}^P \Uparrow^P) \Downarrow^P = \mathcal{A}^P$ and $(\mathcal{A}^Q \Uparrow^Q) \Downarrow^Q = \mathcal{A}^Q$ from (3) and Prop. 5. ■

In order to make sure that the globalizations of the local abstraction functions from the two components agree, we either need the local behavior domains to be disjoint (no common variables), or the local abstraction functions to agree on the "intersection" of the two behavior domains, i.e., along the variables common to the two components [66].

C. Example

We consider the problem of establishing heterogeneous abstraction between models M_1 from Fig. 7 and M_{13} from Fig. 8(c) compositionally.

1) Heterogeneous Abstraction for POV: Consider the hybrid and discrete POV component models from Fig. 7 and Fig. 8(c) and call them P_0 and P_1 respectively. The local behavior domains are B_0^{POV}: 1-D hybrid traces, i.e., evolution of the hybrid state $h^{\mathrm{POV}} := (l^{\mathrm{POV}}, x)$ over time, with $l^{\mathrm{POV}} \in \mathcal{L}^{\mathrm{POV}} := \{\texttt{driving}\}$ and $x \in \mathbb{R}$; and $B_1^{\mathrm{POV}} := \Sigma^{POV^*}$ for set of event labels $\Sigma^{\mathrm{POV}} = \{\beta_1, \beta_2\}$. The model semantics are $[\![P_0]\!]^{B_0}$: the set of all hybrid traces with the discrete location `driving` and x that starts in the initial condition set $[-420, -400]$ and evolves along any arbitrary derivative in the range [20, 30], and $[\![P_1]\!]^{B_1}$: the singleton set $\{\beta_1\beta_2\}$.

A behavior abstraction function $\mathcal{A}^{\mathrm{POV}} : B_0^{\mathrm{POV}} \to B_1^{\mathrm{POV}}$ constructed by partitioning the continuous dimension x at boundaries $x = l$ and $x = 0$ is written mathematically as follows. Given $b_0^{\mathrm{POV}} = h^{\mathrm{POV}}(t) \in B_0^{\mathrm{POV}}$ and $b_1^{\mathrm{POV}} = \sigma_0\sigma_1 \cdots \in B_1^{\mathrm{POV}}$, $\mathcal{A}^{\mathrm{POV}}(b_0^{\mathrm{POV}}) = b_1^{\mathrm{POV}}$ iff $\exists$ times $t_i \in \mathbb{R}_+$ s.t. $\forall\, t' \in [0, t_0)$,$x(t') \in \mathsf{FROM}(\sigma_0)$; $\forall\, t' \in [t_{i-1}, t_i)$, $x(t') \in \mathsf{TO}(\sigma_{i-1}) \cap \mathsf{FROM}(\sigma_i)$ for $i = 1, \ldots, N$ for some $N \in \mathbb{N}$; and $\forall\, t' \geq t_N$, $x(t') \in \mathsf{TO}(\sigma_N)$, where $\mathsf{FROM}(\cdot)$ and $\mathsf{TO}(\cdot)$ are given in the following table. Otherwise, $\mathcal{A}^{\mathrm{POV}}(b_0^{\mathrm{POV}}) = \varepsilon$.

α	$\mathsf{FROM}(\alpha)$	$\mathsf{TO}(\alpha)$
β_1	$x \leq l$	$x \in [l, 0]$
β_2	$x \in [l, 0]$	$x \geq 0$.

Given that the boundary l is at -300, the range of velocities is positive, and the initial condition is in the range $[-420, -400]$, it is straightforward to show that $\forall b_0^{\mathrm{POV}} \in B_0^{\mathrm{POV}}$, $\mathcal{A}^{\mathrm{POV}}(b^{\mathrm{POV}}) = \beta_1\beta_2$. Therefore, $P_0 \sqsubseteq^{\mathcal{A}^{\mathrm{POV}}} P_1$. Note that if l is say -410, $\mathcal{A}^{\mathrm{POV}}(b^{\mathrm{POV}}) = \beta_2$ for some b^{POV} and $P_0 \not\sqsubseteq^{\mathcal{A}^{\mathrm{POV}}} P_1$.

2) Heterogeneous Abstraction for SV: Now consider the SV component of the hybrid and discrete models from Fig. 7 and Fig. 8(c) and call them Q_0 and Q_1. The local behavior domains are B_0^{SV}: the set of 3-D hybrid trajectories $h^{SV}(t)$, where $h^{SV} := (l^{SV}, x, y, v_y)$ are the hybrid states that take values in $\mathcal{L}^{SV} \times \mathcal{X}^{SV}$, for the discrete set of locations $\mathcal{L}^{SV} := \{\texttt{waiting}, \texttt{stopped}, \texttt{conflict\_y}, \texttt{clear\_y}\}$ and the continuous state space $\mathcal{X}^{SV} := \mathbb{R}^3$; and $B_1^{SV} := \Sigma^{SV^*}$ with $\Sigma^{SV} := \{\alpha_1, \alpha_2, \beta_1\}$, where α's signify SV entering and exiting the intersection.

A behavior abstraction function $\mathcal{A}^{SV} : B_0^{SV} \to B_1^{SV}$, constructed by only keeping the discrete part of the hybrid model and adding transition labels, is written formally as follows. Given $b_0^{SV} = h^{SV}(t)$, where $t \in \mathbb{R}_+$ and $h^{SV} = (l^{SV}, x, y, v_y)$, and $b_1^{\mathrm{POV}} = \sigma_0\sigma_1 \cdots$ with states $q_i^{SV} \in \mathcal{L}_i^{SV}$ s.t. $q_i^{SV} \xrightarrow{\sigma_i} q_{i+1}^{SV}$, $\mathcal{A}^{SV}(b_0^{SV}) = b_1^{SV}$ iff $\exists$ times $t_i \in \mathbb{R}_+$ s.t. $\forall t' \in [t_i, t_{i+1})$ with $t_0 = 0$, $l^{SV}(t') == q_i^{SV}$. Otherwise, $\mathcal{A}^{\mathrm{POV}}(b_0^{\mathrm{POV}}) = \varepsilon$.

Because Q_1 has the exact same discrete transition graph as that of Q_0, for every hybrid behavior $b_0^{SV} \in [\![Q_0]\!]^{B_0^{SV}}$, $\mathcal{A}^{SV}(b_0^{SV}) \in [\![Q_1]\!]^{B_1^{SV}}$, i.e., $Q_0 \sqsubseteq^{\mathcal{A}^{SV}} Q_1$.

3) Abstraction Between Compositions: The variables common to local behavior domains B_i^{POV} and B_i^{SV} are x and β_1. We have to make sure that the localizations $\mathcal{A}^{\mathrm{POV}} \Downarrow^{\cap}$ and $\mathcal{A}^{SV} \Downarrow^{\cap}$ of abstraction functions $\mathcal{A}^{\mathrm{POV}}$ and $\mathcal{A}^{SV}$ onto these common variables, i.e., the mappings from behaviors in x to behaviors in $\{\beta_1\}^*$ agree. $\mathcal{A}^{\mathrm{POV}} \Downarrow^{\cap}$ is essentially the same as $\mathcal{A}^{\mathrm{POV}}$, with the row for β_2 discarded. $\mathcal{A}^{SV}$ puts indirect restrictions on x due to the guard and invariant conditions of the hybrid transitions $(\texttt{waiting}, x) \to (\texttt{stopped}, x)$ that are mapped with the discrete transition $\texttt{waiting} \xrightarrow{\beta_1} \texttt{stopped}$. Such a hybrid transition occurs iff $x \leq l$ and $x \geq l$ hold before and after the transition, i.e., while crossing the boundary $x = l$ in the increasing direction, which agrees with $\mathcal{A}^{\mathrm{POV}} \Downarrow^{\cap}$. In the self-loop β_1 transitions, x does not appear and is therefore unrestricted, and in agreement with $\mathcal{A}^{\mathrm{POV}} \Downarrow^{\cap}$.

Therefore, using Prop. 6, we can conclude (without having to analyze models M_1 and M_{13} directly) that $M_1 \sqsubseteq^{\mathcal{A}} M_{13}$.

In summary, compositional heterogeneous abstraction makes it possible to use component models in isolation for establishing abstraction between the composite models. In hierarchical

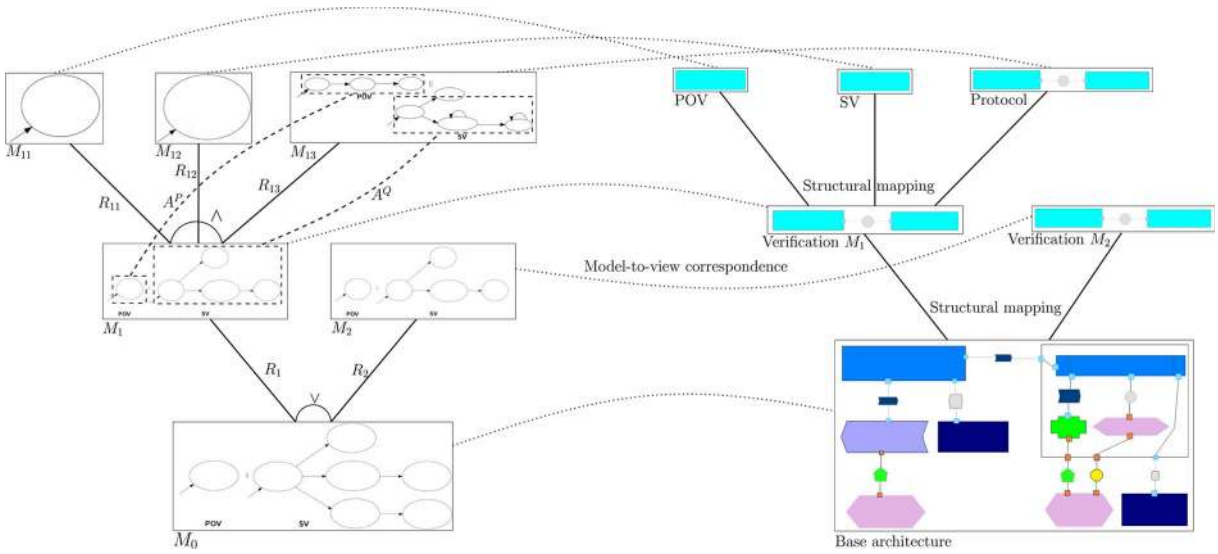


Fig. 10. CICAS-SSA: Semantic hierarchies of verification models (left) and structural hierarchies of architectural models (right).

heterogeneous verification, compositional abstraction can be used at each level whenever possible.

VII. Combined Structural and Semantic Analysis Framework

We have described structural analysis with architectural views in Section III and heterogeneous semantic analysis in Sections IV–VI, which, combined, form our multi-view architectural framework. We advocate the use of this framework as a CPS design approach that combines structural and semantic analyses for multi-model design and analysis of CPS, overcoming the limitations of the existing approaches. Each model has a corresponding structural representation in an architectural view. Depending on the particular formalisms involved in the heterogeneous analysis, models may explore and verify various aspects of a cyber-physical system design. The role of architectural views in this framework is to analyze the structure of models and store high-level information about models to enable inter-model analysis. A simple example of such analysis is structural consistency (cf., Section III).

The semantic and structural hierarchies for CICAS-SSA are shown in Fig. 10. On the left side, verification models described in this paper are linked with semantic mappings: component-wise abstraction and behavior relations. On the right side, architectural views are linked with structural mappings. As the figure shows, for this verification example we need to manage: various heterogeneous models and specifications, semantic mappings between various formalisms, abstractions or disjunctive coverage between models and conjunctive or individual implications between specifications given these mappings, component-wise semantic mappings, their local and global behavior domains, localization mappings for compositional development, etc.

Managing the multi-model design and analysis using our architectural framework provides the necessary rigor that *ad hoc* approaches lack. Drawing from our experience with the STARMAC and CICAS-SSA examples, we believe there are several benefits to multi-model development and analysis in the architectural framework as follows.

An extensible base for developing project-specific analyses of several models. A set of architectural views may contain the necessary meta-information about models to perform inter-model analysis. For example, such analysis could ensure that variables (e.g., a vehicle's coordinates) are propagated in similar ways through the system in all models.

A global system representation that individual modeling formalisms cannot provide. Architectural views serve as a unifying place for system-level information. This satisfies learning and documentation needs of a CPS project: as new engineers join the project, views and mappings between them help to communicate the high-level structure and relations between the system models [67]. Nontechnical stakeholders may find views useful to see the scope and main parts of a CPS.

Extensible architectural styles. Architectural styles support capturing the constraints of a domain and constraining the design process within them. A common way of constraining is to declare the spaces of valid and invalid architectural configurations. For example, the physical style can be extended into the electrical, mechanical, hydraulic, and other styles depending on the level of detail to be captured. Such styles facilitate creation and verification of corresponding systems.

Representation of model structures as architectural views. This enables structural consistency analysis, as described in Section III. Structural consistency guarantees that the models make compatible assumptions about the structure of the system. Such analysis may prevent or correct system design flaws [61].

Explicit representation of interdependent assumptions about models. Heterogeneous models often make simplifying assumptions about each other to keep the complexity of each model manageable for analysis. We have proposed explicitly representing these assumptions using constraints over parameters, and a notion of semantic consistency [68]. Such a consistency analysis can be supported by the multi-view framework as constraints over parameters and/or variables can be represented as properties of architectural elements. For example, a verification view of a system may assume a worst-case communication delay δ of 0.5 seconds, while a wireless communication view may calculate a maximum value of δ for communication delays.

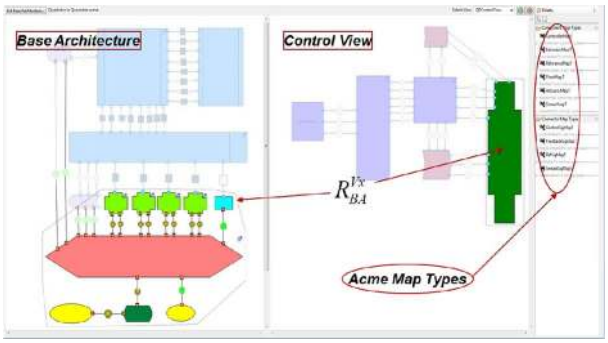


Fig. 11. AcmeStudio multi-view editor showing structural mappings between the control view of STARMAC and the BA. The latter is shown in Fig. 3 in detail.

The overall analysis is consistent if the assumption $\delta \leq 0.5$ actually holds.

Utilization of structural knowledge to simplify verification. The information about architectural topology may inform verification activities. For example, the presence of a connector means that the components share at least one variable. Conversely, absence of any connection means that the components lack directly shared variables. This information would let the verification engine avoid needless behavior consistency checks to satisfy the conditions of Prop. 6.

The practical need to support the multi-model architectural framework necessitates the creation of integrated design environments for CPS design and analysis. We used the AcmeStudio architectural design environment [33] to model views in the STARMAC and CICAS-SSA examples. One of the key benefits of AcmeStudio for heterogeneous design is a multi-view editor [61], illustrated in Fig. 11. Here several components in the base architecture are mapped to a single component in the control view of the STARMAC quadrotor. AcmeStudio allows specification of constraints over what element types can be mapped to each other. It also checks for appropriate correspondences between the mapped components and components connected to them in both views using graph morphism checking.

We are currently working on additions that allow stronger association between semantic models and architectural views: support for parametric assumptions, associating variables with components for verification views to enable more generic analyses, and linking architectural views to specific models like Simulink.

VIII. Conclusion and Future Research Directions

This paper presents an architecture-based framework with structural and semantic mappings to manage multi-model heterogeneous development of cyber-physical systems. We extend software architecture principles by adding architectural modeling vocabulary to include physical and cyber-physical interface elements and using architectural views to capture structure of various models and structural mappings to ensure consistency. Semantic mappings using behavior relations and abstraction functions enable the use of hierarchical and compositional heterogeneous verification. Finally, we combine the architectural views with behavior relations within a unified analytical framework to utilize the advantages of both during model-based development of CPS. For control system development, this framework creates a formal connection between the concerns addressed by control engineering models and tools, and the concerns addressed by the many other models and tools used to design and implement the complete system.

There are several future research directions that can build upon the work presented here. The behavior relations presented here are mathematical definitions at the level of mappings between entire behaviors. To make these more useful in practice, mappings between state spaces such as generalizations of simulation relations to heterogeneous domains may offer a constructive approach to creating relations between the resulting behaviors. Inter-formalism dependencies are currently captured using constraints over static parameters. We are working on extensions to dynamic constraints. Globalization/localization mappings currently exist for different levels of abstraction in behavior domains of the same formalism. Generalizations to heterogeneous component models would be interesting. We are investigating how the structural and semantic sides can help each other. The structural connectivity information could help simplify verifications or proofs. We are also developing tool support in AcmeStudio. Support for semantic consistency is being developed by exporting to external analysis tools such as the theorem prover KeYmaera [69].

References

[1] G. Hoffman, S. Waslander, and C. Tomlin, "Quadrotor helicopter trajectory tracking control," in *Proc. AIAA Guidance, Navigation, Control Conf.*, 2008.

[2] Minnesota Department of Transportation, CICAS-SSA: Concept of Operations.

[3] A. Gorjestani, A. Menon, P.-M. Cheng, C. Shankwitz, and M. Donath, "CICAS-SSA Report #2: The design of an optimal surveillance system," Minnesota Guidestar, 2008, Tech. Rep.

[4] A. Gorjestani, A. Menon, P.-M. Cheng, C. Shankwitz, and M. Donath, "CICAS-SSA Report #1: Alert and warning timing for CICAS-SSA," Minnesota Guidestar, 2008, Tech. Rep.

[5] A. Menon, A. Gorjestani, P.-M. Cheng, B. Newstrom, C. Shankwitz, and M. Donath, "CICAS-SSA Report #3: Macroscopic review of driver gap acceptance and rejection behavior in the US," Minnesota Guidestar, 2008, Tech. Rep.

[6] P. J. Mosterman and H. Vangheluwe, "Computer automated multiparadigm modeling in control system design," in *Proc. Int. Conf. Comput.-Aided Control Syst. Design*, 2000, vol. 12, pp. 65–70.

[7] L. W. P. Feiler and J. Hansson, "System architecture virtual integration: A case study," in *Proc. Embedded Real Time Software Syst. Conf.*, 2010.

[8] T. Denton, E. Jones, S. Srinivasan, K. Owens, and R. W. Buskens, "Naomi-an experimental platform for multi-modeling," in *Proc. 11th Int. Conf. Model Driven Eng. Lang. Syst.*, 2008, pp. 143–157, Springer-Verlag.

[9] S. S. Bhattacharyya, E. Cheong, and I. Davis, "PTOLEMY II heterogeneous concurrent modeling and design in Java," Univ. of California, Berkeley, CA, USA, 2003, Tech. Rep.

[10] D. de Niz, G. Bhatia, and R. Rajkumar, "Model-based development of embedded systems: The sysweaver approach," in *IEEE Real Time Technol. Applicat. Symp.*, 2006, pp. 231–242.

[11] J. Davis, "GME: The generic modeling environment," in *Companion of the 18th Annual ACM SIGPLAN Conf. Object-Oriented Programming, Systems Languages Applications, OOPSLA'03, ACM*, 2003.

[12] A. Ledeczi, J. Davis, S. Neema, and A. Agrawal, "Modeling methodology for integrated simulation of embedded systems," *ACM Trans. Model. Comput. Simul.*, vol. 13, pp. 82–103, Jan. 2003.

[13] H. L. M. Vangheluwe, "DEVS as a common denominator for multi-formalism hybrid systems modeling," in *Proc. IEEE Int. Symp. Comput.-Aided Control Syst. Design*, Anchorage, AK, USA, 2007.

[14] A. Pinto, L. P. Carloni, R. Passerone, and A. Sangiovanni-Vincentelli, "Interchange semantics for hybrid system models," in *Proc. 5th MATHMOD*, 2006.

[15] Z. Gu, S. Kodase, S. Wang, and K. Shin, "A model-based approach to system-level dependency and real-time analysis of embedded software," in *Proc. 9th IEEE Real-Time Embedded Technol. Applicat. Symp.*, May 2003, pp. 78–85.

[16] K. Chen, J. Sztipanovits, and S. Abdelwahed, "Toward a semantic anchoring infrastructure for domain-specific modeling languages," in *Proc. 5th ACM Int. Conf. Embedded Software*, Sep. 2005.

[17] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An integrated electronic system design environment," *Comput.*, vol. 36, pp. 45–52, Apr. 2003.

[18] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu, "A next-generation design framework for platform-based design," in *Proc. Conf. Using Hardware Design Verificat. Lang. (DVCon)*, 2007.

[19] W. Schamai, P. Fritzson, C. Paredis, and A. Pop., "Towards a unified system modeling and simulation with ModelicaML: Modeling of executable behavior using graphical notations," in *Proc. 7th Int. Modelica Conf.*, 2009.

[20] J. Shi, "Combined usage of UML and Simulink in the design of embedded systems: Investigating scenarios and structural and behavioral mapping," in *Proc. 4th Workshop Object-Oriented Modeling of Embedded Realtime Syst.*, 2007.

[21] T. Johnson, C. J. J. Paredis, and R. M. Burkhart, "Integrating models and simulations of continuous dynamics into SysML," in *Proc. 6th Int. Modelica Conf.*, 2008, pp. 135–145, Modelica Assoc.

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

[23] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2003.

[24] N. Medvidovic and R. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Trans. Software Eng.*, vol. 26, no. 1, pp. 70–93, Jan. 2000.

[25] D. Garlan and B. Schmerl, "Architecture-driven modelling and analysis," in *Proc. 11th Austral. Workshop Safety Related Programm. Syst.*, 2006, vol. 69.

[26] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*, 2nd ed. Boston, MA, USA: Addison Wesley, 2004.

[27] SysML.org, SysML. [Online]. Available: http://www.sysml.org/

[28] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis and design language (AADL): An introduction," Software Eng. Insti., Carnegie Mellon Univ., Tech. Rep. CMU/SEI-2006-TN-011, Feb. 2006.

[29] G. Abowd, R. Allen, and D. Garlan, "Formalizing style to understand descriptions of software architecture," *ACP Trans. Software Eng. Methodol.*, vol. 4, pp. 319–364, Oct. 1995.

[30] J. Magee and J. Kramer, *Concurrency: State Models and Java Programming*, 2nd ed. New York, NY, USA: Wiley, 2006.

[31] M. Moriconi and R. Reimenschneider, "Introduction to SADL 1.0: A language for specifying software architecture hierarchies," SRI Int., Tech. Rep. SRI-CSL-97-01, 1997.

[32] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Software Eng. Methodol.*, Jul. 1997.

[33] B. Schmerl and D. Garlan, "AcmeStudio: Supporting style-centered architecture development," in *Proc. 26th Int. Conf. Software Eng.*, 2004, pp. 704–705.

[34] D. Garlan, W. Reinholtz, B. Schmerl, N. Sherman, and T. Tseng, "Bridging the gap between systems design and space systems software," in *Proc. 29th Annu. IEEE/NASA Software Eng. Workshop*, 2005.

[35] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "Developing AADL models for control systems: A practitioner's guide," Software Eng. Inst., Carnegie Mellon Univ., Tech. Rep. CMU/SEI-2007-TR-014, 2007.

[36] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Algorithmic analysis of nonlinear hybrid systems," *IEEE Trans. Autom. Control*, vol. 43, no. 4, pp. 225–238, Apr. 1998.

[37] T. Dang, O. Maler, and R. Testylier, "Accurate hybridization of nonlinear systems," in *Proc. Int. Conf. Hybrid Syst.: Comput. Control*, 2010.

[38] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," *Int. J. Software Tools Technol. Transfer*, vol. 10, no. 3, pp. 258–273, 2008.

[39] R. Alur, T. A. Henzinger, G. Laffarriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 971–984, Jul. 2000.

[40] A. Chutinan and B. H. Krogh, "Verification of infinite-state dynamic systems using approximate quotient transition systems," *IEEE Trans. Autom. Control*, vol. 46, no. 9, pp. 1401–1410, Sep. 2001.

[41] R. Alur, T. Dang, and F. Ivancic, "Predicate abstraction for reachability analysis of hybrid systems," *ACM Trans. Embedded Comput. Syst.*, vol. 5, no. 1, pp. 152–199, 2006.

[42] M. Althoff, A. Rajhans, B. H. Krogh, S. Yaldiz, X. Li, and L. Pileggi, "Formal verification of phase-locked loops using reachability analysis and continuization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2011, pp. 659–666.

[43] E. A. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 12, pp. 1217–1229, Dec. 1998.

[44] A. Benveniste, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli, "Composing heterogeneous reactive systems," *ACM Trans. Embedded Comput. Syst.*, vol. 7, pp. 43:1–43:36, July 2008.

[45] A. A. Julius, "On interconnection and equivalence of continuous and discrete systems: A behavioral perspective," Ph.D. dissertation, Univ. of Twente, Enschede, The Netherlands, 2005.

[46] T. A. Henzinger, "The theory of hybrid automata," *Verificat. Digital Hybrid Syst.*, vol. 170, pp. 292–296, 2000.

[47] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *Formal Methods for Components and Objects*, vol. 5382, *Lecture Notes in Computer Science*, F. Boer, M. Bonsangue, S. Graf, and W.-P. Roever, Eds. Berlin/Heidelberg, Germany: Springer, 2008, pp. 200–225.

[48] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems," *Eur. J. Control*, vol. 18, no. 3, pp. 217–238, 2012.

[49] P. Nuzzo, A. Sangiovanni-Vincentelli, X. Sun, and A. Puggelli, "Methodology for the design of analog integrated interfaces using contracts," *IEEE Sensors J.*, vol. 12, no. 12, pp. 3329–3345, Dec. 2012.

[50] J. Leung, T. Mandl, E. Lee, E. Latronico, C. Shelton, S. Tripakis, and B. Lickly, "Scalable semantic annotation using lattice-based ontologies," in *Proc. 12th Int. Conf. Model Driven Eng. Lang. Syst.*, Oct. 2009, pp. 393–407.

[51] R. Kumar, B. H. Krogh, and P. Feiler, "An ontology-based approach to heterogeneous verification of embedded control systems," *Hybrid Syst.: Comput. Control*, vol. 3414, pp. 370–385, 2005, Springer Berlin/Heidelberg.

[52] K. Chaudhari, D. Doligez, L. Lamport, and S. Merz, "The TLA+ proof system: Building a heterogeneous verification platform," in *Proc. Int. Colloq. Theoret. Aspects Comput.*, Natal, Brazil, 2010, vol. 6256, p. 44.

[53] J. Faber, "Verification architectures for complex real-time systems," Ph.D. dissertation, Univ. of Oldenburg, Oldenburg, Germany, 2011.

[54] D. Kaynar and N. Lynch, "Decomposing verification of timed I/O automata," in *Proc. Joint Conf. Formal Modell. Anal. Timed Syst. Formal Techniques Real-Time Fault Tolerant Syst.*, 2004.

[55] G. Frehse, "Compositional verification of hybrid systems using simulation relations," Ph.D. dissertation, Radboud Univ. Nijmegen, Nijmegen, The Netherlands, 2005.

[56] T. A. Henzinger, S. Qadeer, S. K. Rajamani, and S. Tasiran, "An assume-guarantee rule for checking simulation," *ACM Trans. Program. Lang. Syst.*, vol. 24, pp. 51–64, Jan. 2002.

[57] A. Platzer, "Differential dynamic logic for hybrid systems.," *J. Autom. Reas.*, vol. 41, no. 2, pp. 143–189, 2008.

[58] S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis, "Compositional verification for component-based systems and application," *IET Software*, vol. 4, no. 3, pp. 181–193, 2010.

[59] J. Willems, "The behavioral approach to open and interconnected systems," *IEEE Control Syst. Mag.*, vol. 27, no. 6, pp. 46–99, Dec. 2007.

[60] A. Bhave, D. Garlan, B. Krogh, A. Rajhans, and B. Schmerl, "Augmenting software architectures with physical components," in *Proc. Embedded Real Time Software Syst. Conf.*, May 19–21, 2010.

[61] A. Bhave, "Multi-view consistency in architectural views for cyber-physical systems," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2011.

[62] A. Bhave, B. H. Krogh, D. Garlan, and B. Schmerl, "View consistency in architectures for cyber-physical systems," in *Proc. 2nd Int. Conf. Cyber-Phys. Syst.*, 2011.

[63] A. Gokhale, M. McDonald, S. Drager, and W. McKeever, "A cyber physical systems perspective on the real-time and reliable dissemination of information in intelligent transportation systems," *J. Netw. Protocols Algorithms*, vol. 2, no. 3, pp. 116–136, 2010.

[64] A. Rajhans and B. H. Krogh, "Heterogeneous verification of cyber-physical systems using behavior relations," in *Proc. 15th ACM Int. Conf. Hybrid Syst.: Comput. Control (HSCC)*, 2012.

[65] A. Rajhans, "Multi-model heterogeneous verification of cyber-physical systems," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2013.

[66] A. Rajhans and B. H. Krogh, "Compositional heterogeneous abstraction," in *Proc. 16th ACM Int. Conf. Hybrid Syst.: Comput. Control*, 2013.

[67] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, 2nd ed. Boston, MA, USA: Addison-Wesley, Oct. 2010.

[68] A. Rajhans *et al.*, "Using parameters in architectural views to support heterogeneous design and verification," in *Proc. 50th IEEE Conf. Decision Control*, Orlando, FL, USA, Dec. 2011.
[69] A. Platzer and J.-D. Quesel, "KeYmaera: A Hybrid Theorem Prover for Hybrid Systems.," in *IJCAR*, vol. 5195, *LNCS*. New York, NY, USA: Springer, 2008, pp. 171–178.

Akshay Rajhans (S'08–M'13) received the B.E. degree in electronics and telecommunication from the University of Pune, Pune, India, in 2003, the M.S.E. degree in electrical engineering from University of Pennsylvania, Philadelphia, PA, USA, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2007 and 2013, respectively.

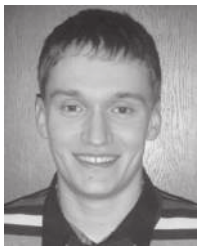
He has worked on the research and development of embedded control systems at Cummins India Limited and Bosch Research and Technology Center. He is currently a Senior Software Engineer in the Simulink Semantics group at The MathWorks, Inc., Natick, MA, USA. His research interests include model-based design and formal analysis of cyber-physical systems.

Dr. Rajhans is a recipient of the William J. McCalla Best Paper Award at the 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).

Ajinkya Bhave (S'06–M'12) received the B.E. degree in computer engineering from University of Mumbai, Mumbai, India, in 2001, the M.S. degree in robotics from the Robotics Institute, the M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2003 and 2011, respectively.

He was Special Faculty at Carnegie Mellon University in 2012 and currently works as Technical Leader at LMS Siemens, where he heads the Controls Group at the India Development Center. His research interests include embedded control systems, software architectures, and model-based systems engineering.

Dr. Bhave is a recipient of the NASA Group Achievement Award for the "Spaceward Bound 2006 Expedition: Chile," as a member of the autonomous rover team.



Ivan Ruchkin (S'12) received the Specialist degree in mathematics and computer science (with Honors) from Lomonosov Moscow State University, Moscow, Russia, in 2011. He is currently pursuing the Ph.D. degree in software engineering at Carnegie Mellon University, Pittsburgh, PA, USA.

He was a Software Engineer and a User Interface Designer in the Computer Systems Laboratory at Lomonosov Moscow State University and in SiTrans Ltd. and several open-source projects. His research interests include software architecture, formal methods, and cyber-physical systems.

Bruce H. Krogh (S'82–M'82–SM'92–F'98) received the B.S. degree in mathematics and physics from Wheaton College, Wheaton, IL, USA, and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, IL, USA, in 1975 and 1983, respectively.

In 1983, he joined Carnegie Mellon University, Pittsburgh, PA, USA, where he is currently Professor of electrical and computer engineering and Director of Carnegie Mellon University in Rwanda in Kigali, Rwanda. His current research interests include synthesis and verification of embedded control software, discrete event and hybrid dynamic systems, and distributed control strategies for the smart grid and other energy-related applications.

Dr. Krogh was an Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL and *Discrete Event Dynamic Systems: Theory and Applications*, and founding Editor-in-Chief of the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY.

David Garlan (S'82–M'87–SM'10–F'12) received the B.A. degree in mathematics from Amherst College, Amherst, MA, USA, in 1971 and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, USA, in 1987.

He is a Professor of Computer Science in the School of Computer Science at Carnegie Mellon University. His research interests include software architecture, self-adaptive systems, formal methods, and cyber-physical systems. He is a coauthor of two books on software architecture: *Software Architecture: Perspectives on an Emerging Discipline* (Prentice-Hall, Inc., 1996) and *Documenting Software Architecture: Views and Beyond* (Addison-Wesley Professional, 2011).

Dr. Garlan received a Stevens Award Citation for "fundamental contributions to the development and understanding of software architecture as a discipline in software engineering." (2005) and an Outstanding Research award from ACM SIGSOFT for "significant and lasting software engineering research contributions through the development and promotion of software architecture." (2011). He is a Fellow of the ACM.

André Platzer (S'06–M'09) received the M.S. degree in computer science from University of Karlsruhe (TH), and the Ph.D. degree in computer science from the University of Oldenburg, Oldenburg, Germany, in 2004 and 2008, respectively.

In 2008, he joined Carnegie Mellon University, Pittsburgh, PA, USA, as an Assistant Professor of computer science. His research interests include logical foundations of cyber-physical systems, logic in computer science, programming languages, formal methods, and automated theorem proving.

Dr. Platzer is recipient of an ACM Doctoral Dissertation Honorable Mention Award, an NSF CAREER award, and was named one of the Brilliant 10 Young Scientists by the Popular Science magazine 2009 and one of the AI's 10 to Watch 2010 by the IEEE INTELLIGENT SYSTEMS MAGAZINE.

Bradley Schmerl (M'94) received the B.S. degree in mathematics (with honors), majoring in computer science from the University of Adelaide, Adelaide, Australia, in 1991 and the Ph.D. degree in computer science from Flinders University, Bedford Park, Australia, in 1997.

In 1996, he as a Software Engineer with SKC, Ltd, Adelaide, Australia. From 1997 to 1998, he was a Lecturer at Flinders University, and from 1998 to 2000, he was an Assistant Professor at Clemson University, Clemson, SC, USA. He is currently a Senior Systems Scientist in the School of Computer Science at Carnegie Mellon University, Pittsburgh, PA, USA. His research interests include software engineering, software architecture, autonomic, and pervasive computing.