Building Correct Cyber-Physical Systems: Why we need a Multiview Contract Theory*

Susanne Graf[1], Sophie Quinton[2], Alain Girault[2] and Gregor Gössler[2]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, Grenoble, France
[2] Univ. Grenoble Alpes, INRIA, CNRS, Grenoble INP, LIG, Grenoble, France

Abstract. The design and verification of critical cyber-physical systems is based on a number of models (and corresponding analysis techniques and tools) representing different viewpoints such as function, timing, security and many more. Overall correctness is guaranteed by mostly informal, and therefore basic, arguments about the relationship between these viewpoint-specific models. We believe that a more flexible contract-based approach could lead to easier integration, to relaxed assumptions, and consequently to more cost efficient systems while preserving the current modelling approach and its tools.

1 Introduction

Building correct Cyber-Physical Systems (CPS) is a challenge in critical application domains such as avionics, automotive, etc. It is getting ever more difficult because CPSs are of increasing complexity: indeed, CPS are nowadays composed of a large number of components and subsystems of heterogeneous nature and of different criticality levels. In addition, non-functional aspects, or *viewpoints* – such as timing, memory footprint, energy, dependability, temperature, and more recently also security – are as important as functionality.

There exist many analyses and tools for verifying CPS, but their underlying model is always specific to a *single* viewpoint, and there is currently limited support to relate viewpoints semantically. In practice, the assumptions that a viewpoint-specific analysis makes on the other viewpoints remain mostly implicit, and whenever explicit they are handled mostly manually. In this paper, we argue that the current design process over-constrains the set of possible system designs and that there is a need for methods and tools to formally relate viewpoint-specific models and corresponding analysis results.

More specifically, we claim that *contract-based* design can be relevant to address the challenges raised by the use of multiple viewpoints. The term "design by contract" has been introduced in [30].

The rest of this paper is organized as follows. Section 2 provides a short overview of some viewpoint-specific models and techniques. Section 3 describes existing efforts

* This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

toward some level of integration of viewpoints. In Section 4 we motivate the need for a flexible contract-based approach to formally relate viewpoint-specific models and analysis results. Finally, we present in Section 5 initial remarks and possible research directions toward such a framework.

2 Multiple models for multiple viewpoints

In practice, CPS designers make use of several models focusing on specific aspects of a system, called *viewpoints* – typical examples are function, timing, safety, reliability, security, energy, etc. Often, different viewpoints correspond to distinct disciplines, possibly very different levels of granularity, and are supported by their own domain-specific software tools. We now briefly discuss the most relevant ones for CPS.

2.1 Function

The functionality of a system consists of a set of control functions, some of them intended to be executed cyclically with a cycle time that may be specific to each function, and others sporadically, typically for the treatment of alarms. The functionality is itself split between a continuous part (automatic control laws) and a discrete part (finite state machines), hence two viewpoints. Usual requirements for the former part include observability and stability, but also robustness to perturbations, delays, noise, and so on. For the latter part, engineers are concerned with safety and reachability properties. For all of these, a large body of results has been developed.

Function design is nowadays done in a component-based manner, sometimes using directly the C language or using domain specific languages proposed by design environment such as MATLAB Simulink (well suited for ODE based models), Scade [13] (well suited for safety critical systems), or Modelica [33] (well suited for DAE based systems). Most of these design environments include analysis tools to ensure that the control algorithms, provided by the control engineer, are correctly implemented (based on the semantics of the programming language). Analysis tools have also been developed to take into account both the continuous and discrete viewpoints, with tools such as PHAVer [19] or d/dt [4].

2.2 Timing

Timing requirements on CPSs are typically expressed in terms of deadlines on the *Worst-Case Response Time* (WCRT) of the system. The WCRT is the time required to compute and send the outputs of the system to its actuators starting from the values on the inputs obtained from its sensors. Depending on how the system's functionalities are implemented, these deadlines will be expressed in a different ways: e.g., end-to-end latency for a task based implementation, or worst-case reaction time for a periodic loop implementation.

The schedulability of a system, that is, the guarantee that no deadline can be missed, is proven in two steps: (i) computing the Worst-Case Execution Time (WCET) of the basic components of the system (e.g., its tasks, its C functions, ...), and (ii) performing

the schedulability analysis strictly speaking. The goal of the WCET analysis is to upper bound the maximum time it may take for a program to complete on a given hardware platform, assuming there is no interference from other programs that may concurrently execute. WCET analysis is based on low-level code (obtained by compilation from C) and on an abstraction of the hardware platform of the system under analysis, including memory access policies, caches, pipelines, and so on. Commercial tools like aiT [1,42] from Absint perform such an analysis.

Schedulability analysis integrates the results of the WCET analysis with an analysis of how different programs (or tasks) may interfere due to the fact that they share computation and communication resources (e.g., on a multi-core processor). Schedulability analysis is based on a model of the software represented as a set of tasks scheduled according to some scheduling policy. Tools performing schedulability analysis include SymTA/S from Symtavision [25] and RT-Druid from Evidence [2].

2.3 Dependability

Dependability is a crucial notion for CPS systems. It is defined as the ability for the system to "deliver a service that can justifiably be trusted" [6]. This generic notion encompasses many concepts, including availability, reliability and safety. Among those, the one on which we focus in this section is *reliability*, which is defined as the probability that the systems works correctly during a given time interval. Being a probability, it varies in the interval $[0, 1]$. For instance, fly-by-wire civil flight control systems must exhibit a reliability greater than $1 - 10^{-9} = 0.999999999$ over 10 hours (the "nine nines rule") [37].

When addressing the dependability viewpoint, engineers must provide the *fault model*, which identifies how the components of the CPS being designed can fail: this concerns both the hardware (processors, communication media, memory banks, sensors, actuators, and so on) and the software (tasks, OS, and middleware). For instance, the hardware failures can be transient or permanent. The fault model depends not only on the physical environment of the CPS (the temperature range, vibrations, radiations, and so on), but also on the chosen manufacturing technology (which CMOS size, which packaging), and on the operating mode (which voltage, frequency, and so on).

Then, the user specifies a minimal reliability r that the CPS under design must comply to. Improving the reliability requires some form of *redundancy*, which can be spatial or temporal when dealing with hardware failures [21]. Engineers thus use dedicated analysis tools to derive how much redundancy must be added to the system, and where it must be added, to achieve this bound r. Examples of such tools include fault-trees, reliability block diagrams, and so on.

3 Efforts toward integration of viewpoints

In addition to viewpoint-specific techniques, an increasing number of methods and tools provide some support for handling multiple viewpoints.

3.1 Tool integration

There exist integration mechanisms between design tools such as MATLAB Simulink, Scade, TargetLink[1], schedulability analysis tools such as SymTA/S or RT-Druid and WCET analysis tools, in particular aiT. This means that the functional model can be annotated with task information allowing for

- the extraction of a scheduling model so as to guarantee that there is always a well defined mapping relating functions and tasks;
- the extraction of low level code for WCET analysis;
- the injection of the computed WCETs into the scheduling model to perform the schedulability analysis.

Such tool support is obviously very useful for guaranteeing the consistency between viewpoint-specific models. Yet, the exchange of information between viewpoints takes place mostly at a syntactical level. For example, the assumption made in the functional model on schedulability is implicit.

3.2 Theoretical results relating automatic control and other viewpoints

Several approaches have been proposed to formally link automatic control objectives with discrete computation and real-time scheduling. Consider for instance the stability objective mentioned in Sec. 2.1. For a given control law, this issue can be addressed purely from the continuous viewpoint (e.g., by defining a suitable Lyapunov function and proving its convergence), but doing so ignores the discrete changes occurring in the system, which may cause the system to switch from one control law to another. Such switches between several control laws makes the stability problem very difficult to solve. Taking into account both viewpoints is therefore necessary, and attempts at this have been made in a contract-based manner.

With the goal of reasoning about how discretized signals evolve over time, *change and delay contracts* have been proposed in [32], while [28] introduces a theory of stochastic contract over Stochastic Signal Temporal Logic.

Co-design of control and real-time scheduling has been studied by many authors, see e.g. [27,18,20]. A set of timing contracts between control and software engineers is proposed in [15]. Stability of embedded control systems under timing contracts, synthesis of timing contracts ensuring stability, and synthesis of scheduling policies ensuring satisfaction of timing contracts are studied in [3]. In contrast, [14] proposes a component library for bottom-up construction of hybrid controllers ensuring safety and stability properties.

Of particular interest for multiview contracts is the *symbolic control* [41] approach operating on a finite abstraction of the infinite state space. Control aspects may interfere with other aspects in particular through the system state, the computing power spent to compute the control actions, and through delays and jitter. Consider for instance the delays: they can occur at three places of a controlled closed-loop system: at the inputs

[1] TargetLink is a production code generation tool from dSPACE

(due to sensors dynamics), in the state (due to modeling assumptions), and at the outputs (due to actuators dynamics). These delays have been addressed by the automatic control community and given rise to the "delay systems" research area. Taking into account delay systems within a multiview contract approach and studying the robustness to delays raises several interesting challenges: from the systems and control viewpoint, the control engineer could study the stability, observability, and controllability of their system without considering the delays, and then from the timing viewpoint they could study the robustness of their control law with respect to the delays.

3.3 Other approaches for the integration of multiple viewpoints

As seen in Sec. 2.1, the continuous and the discrete viewpoints belong both to the functionality of the system. To exemplify the benefits of multiview contracts, it is essential to address also non-functional viewpoints. Consider for instance the timing and the reliability viewpoints. As explained in Sec 2.3, improving the system's reliability requires some form of redundancy. For instance, a given task (or C function) can be replicated to reach the desired reliability, and potentially, each task can be replicated a different number of times. But this has an obvious negative impact on the timing of the system, because the system's WCRT will increase due to these task replications. So both viewpoints must be addressed jointly, as in [22].

Furthermore, consider now in addition the energy viewpoint. Decreasing the energy consumption of the system is classically achieved thanks to *Dynamic Voltage and Frequency Scaling* (DVFS) by choosing a lower (frequency, voltage) operating point for some tasks of the system. Potentially, each task can use a different (frequency, voltage) operating point. Again, this incurs an obvious negative impact on the timing viewpoint, because lowering the frequency increases the WCET of the tasks. But, perhaps less known is the negative impact on the system's reliability, because lowering the voltage makes the system sensitive to noise and lower energy particles, which are likely to create a critical charge leading to a transient failure [44]. Here again, these intricate dependencies between the viewpoints call for integrated methods and tools, as in [39] for the timing, energy, and temperature viewpoints, or in [5] for the timing, energy, and reliability viewpoints.

A large number of results exist as well on the connection between real-time and fault tolerance [7,12], and more recently on the integration of real-time and security [17,26].

Still, these multiview approaches consider two, at most three viewpoints and are not compatible with the existing workflow discussed previously.

In practice, the overview on all models and corresponding analysis activities is mostly in the hands of a human. As a consequence, the reasoning must be kept simple and thus the constraints imposed on each individual model are very restrictive, as we discuss in more detail in the next section.

4 Problem statement

We have explained in the previous sections that the verification of cyber-physical systems is mostly performed on viewpoint-specific models. If all these models were com-

pletely independent, this would be sufficient [29] but this is of course not the case, as discussed in Sec. 3.3. We see two main issues with the current situation:

1. There is no theoretical framework that can encompass all viewpoints.
2. As a result, the interface between viewpoints must be simple enough to be handled manually, possibly while remaining implicit.

Let us illustrate the above mentioned shortcomings on an example. Functional analysis is based on some ideal (possibly mathematical) semantics of a programming language. In practice, the actual platform on which the code will be running may be compromised by various kinds of failures occurring at runtime, which can be due to insufficient resources (memory, computation time, etc.) or due to physical faults of the hardware platform. In particular, a major verification effort is spent to guarantee the absence of such runtime errors due to timing (thanks to schedulability analysis), as well as to guarantee a very low probability of failures due to the hardware components (thanks to dependability analysis).

Note that in the function model a property requiring *absence of runtime errors* cannot even be expressed. This property is an *assumption* to guarantee the *validity of the idealized mathematical semantics* used to make functional analysis feasible. It must be be guaranteed by the platform, and at analysis level by the viewpoints dealing with those errors explicitly.

In the current methodology, this assumption is not explicitly formulated, and this means also that it cannot be relaxed.

Indeed, it has has been proven to be unnecessarily restrictive for a large class of CPSs. A system may still satisfy its functional requirements under a *weaker* assumption: for example, a component implementing some continuous control law may still be perfectly safe (i.e., stable in the sense of automatic control) even if a deadline is missed – that is, an increased control delay is observed – from time to time [20,16].

This example underlines the need for a comprehensive tool support backed by a strong formal theory to handle explicitly the dependencies between any two viewpoints. The contract framework we are aiming at must permit to guarantee system properties based on analysis results obtained on viewpoint-specific models. In this context, contracts are attached to viewpoints which may be of very different nature, but all model the same system. Note that existing contract frameworks do not solve our problem.

In the Design-by-Contract approach introduced in [30] for the programming language Eiffel, and in all similar frameworks, proving contract satisfaction boils down to pre/post condition reasoning, which does clearly not fit our needs. Closer to our needs are general frameworks proposed for components composed under some parallel composition operators.

The meta-theory of contracts proposed in [11] extends many existing contract frameworks. It proposes a set of interesting concepts based on work done in the SPEEDS project [34] and a very powerful theory at semantic level. Unfortunately, it assumes a unifying formalism and all concepts are represented as an algebra on sets of runs. We want to reason at a higher level.

Rely/Guarantee[2] reasoning frameworks [31,36] consider, like we, contracts $(\mathcal{A}, \mathcal{G})$ where $\mathcal{A}$ is an assumption on the environment under which the component is able to guarantee $\mathcal{G}$, and they propose proof rule based reasoning frameworks. In our case, contracts are attached with viewpoints instead of components. In the already mentioned project, we have also developed a general contract framework, with proof rules for avoiding the composition of heterogeneous models by composing verification results instead [23] but it is too abstract to directly usable.

We aim at building *domain specific reasoning frameworks* adapted to a multi-model and multi-tool based methodology: basic facts should be derived on individual viewpoint models using their specialized tools, the system designer should be able to prove integration correctness using a set of domain specific contracts and proof rules, where the deep semantic level proofs requiring reasoning on the underlying *global system model* are only used to prove the correctness of the framework, or to extend the framework when needed.

5 Discussion

In this section, we emphasize what we believe are key issues that must be taken into account by a multiview contract theory. Our aim is to provide the system engineer who is currently in charge with system integration in a multi-model based approach, with additional tool support for guaranteeing their consistency.

5.1 On abstraction and proof rules

Recall that the proof system that we want to develop is meant to ensure system properties from viewpoint specific analysis results. For now, let us consider some of the problems we may face with preserving properties from viewpoints to the system.

In the simplest case, a viewpoint model M_{vp} is an *abstraction* of the global model M_G, that is, $M_{vp} = \alpha(M_G)$ for a function α preserving a class of properties Φ. In this case, using results on property preserving abstractions (e.g. [29]), for any $\varphi \in \Phi$ we get immediately the proof rule[3]

if (1) M_{vp} satisfies φ then (2) M_G satisfies φ

In practice, the mapping α does quite often not define such a "property preserving abstraction". Only when some property $\mathcal{A}$ holds, α is a Φ-preserving abstraction. We say that M_{vp} is a *conditional abstraction*, and the assumption $\mathcal{A}$ is the condition that guarantees that it is a Φ-preserving abstraction. As an example take the already mentioned condition of *absence of failures and timing errors*, or more generally that no

[2] more commonly called Assume/Guarantee reasoning, but we adopt here the terminology of [11]

[3] where it may be necessary to "translate" the viewpoint property to a system property, but this requires technical arguments which beyond the purpose of this paper.

other viewpoint can "break" the function model, which is indeed required to guarantee that the function model M_{Fun} represents a correct abstraction of the system. This gives us immediately the proof rule

if (1) M_G satisfies $\mathcal{A}$ and (2) M_{vp} satisfies φ then (3) M_G satisfies φ

Because of the restriction that verification should be restricted to individual viewpoint models (or small groups of them that can be handled jointly by the same tool), (1) cannot be checked directly, but $\mathcal{A}$ has to be "projected" on individual viewpoints, and therefore (1) can be replaced by verification condition of the form

(1') M^1_{vp} satisfies $\mathcal{A}_1$, ..., and M^k_{vp} satisfies $\mathcal{A}_k$

On our running example, this means: in all viewpoints one has to identify events that could "break" M_{Fun}, and prove that such events will never occur. This demonstrates that projecting $\mathcal{A}$ on individual viewpoints may be reasonably simple.

We now have given a hint on how to formalize the current proof methodology for strong "cannot break" assumptions. But our aim is to be able to propose more relaxed assumptions. For example, in [24] we have proposed "interface automata" to represent more general "no break" conditions.

But we want to go beyond conditional abstraction. What can we propose, if for example, the condition "absence of deadline misses" is not satisfied, that is, the function model M_{Fun} is *not* an abstraction, at least not for the standard definition[4]? Current practice cannot handle this case in a satisfactory manner. Even if one knows that occasional deadline misses do not harm, it makes it mandatory to achieve schedulability (if needed by adding more resources) because there is no possibility to modify the "contract".

Could one replace this "contract"? Could one come up with a set of proof rules that would allow us to conclude from (1) M_{sched} satisfies *schedulable 9 times out of 10*, (2) M_{Fun} satisfies *always outputs a correct control action or does nothing* – that is, we replace the guarantee by a weaker one, and (3) possibly some more proofs, that M_{Fun} satisfies *outputs a correct control action 9 times out of 10*? This would then allow us to come up with a set of contracts to be satisfied by the set of viewpoint models.

Finally, could one systematize this approach for more complex "conditions" involving several viewpoints? possibly in a very viewpoint specific manner, using results mentioned in Sec 3.

5.2 Viewpoint composition

In order to *prove the correctness* of the reasoning framework to be defined, one obviously needs to reason on the global semantic model $\mathcal{M}_\mathcal{G}$, hopefully without ever building it. There are many proposals of unifying semantic frameworks proposed with the aim to provide a uniform representation of systems consisting of heterogeneous viewpoints or composed from parts based on different *models of computation*. We can here

[4] Note that sometimes it may be sufficient to relax the notion of abstraction to obtain a conditional abstraction

discuss only a few of them. One may in particular distinguish work on unified behaviour models whose purpose is the expression of behaviours stemming from heterogeneous viewpoints. We would like to mention in particular stochastic hybrid automata (SHA) [35] or at a lower semantic level Tag machines [10]. Another line of interesting work is on heterogeneous composition, in particular Metropolis [8], Ptolemy [43] or BIP [9]. There is also some work with a similar motivation as ours, where unifying models explicitly address viewpoint integration. We would like to mention [38] which defines a framework for a discrete setting and discusses problems of inter-viewpoint validation, and [40] which discusses a framework for service oriented systems. It proposes to restrict inter-viewpoint verification to verification of their consistency.

To summarize, in order to define a global model representing all relevant viewpoints, we need:

1. a common semantic model, rich enough to represent the behaviour of any viewpoint model
2. define the actual mappings from viewpoint models to the common semantic model
3. a notion of viewpoint composition

Let us discuss some of the needs and difficulties.

1. Behaviour semantics: the needs depend on the considered viewpoints. In those we are aware of, runs can be naturally represented as sequences of events representing *discrete state changes*, where between events, the discrete state remains stable, and the continuous state evolves according some laws. Some viewpoints may constrain the frequency of the occurrence of events by probabilities, occurrence patterns or other distributions. For the viewpoints mentioned in Section 2, a formalism such as the already mentioned SHA may be an option.

2. Semantic mapping: A difficulty for defining such a mapping stems from the fact that in different viewpoints, events may have a different granularity. To obtain a set of behaviour models that can be composed, each viewpoint model has to be refined sufficiently to be able to interact with other viewpoints on all relevant events.

Consider a well-known example that illustrates the granularity problem: in a function model according to a synchronous approach, in the simplest case, events are "ticks" representing the cycle period in which states and outputs are updated "instantaneously". This semantic model is useful as it simplifies verification of temporal properties, but using it makes the implicit assumption that the computation can always be completed within the cycle period. And the combined semantic model of both, the function and the corresponding task model, requires to refine "tick events" into event sequences such that all events of the kind *start task* and *end task* of the task model can be identified with some event in the function model.

This combined model may for example be used to easily prove the correctness of the contract saying that "as long as no deadline misses occur, the task model cannot disturb/break the function model".

6 Conclusion

This paper has been motivated by actual difficulties that system designers of large safety-critical cyber-physical systems have to face: how to keep consistent a system de-

sign without overconstraining it, in a context where multiple viewpoints are addressed separately using specialized tools. There is presently no framework that would allow a system engineer to manage the interplay between all viewpoints and the overall consistency in a flexible way.

There is a large body of theoretical work addressing interdependency and contracts for specific pairs or small groups of viewpoint; few of them are used in actual design processes. Among the contract frameworks that have been proposed for the purpose of achieving consistent integration, most consists in general theory.

The framework we have in mind would provide viewpoint specific contract patterns guaranteeing inter-viewpoint consistency in a flexible manner. We tried to motivate that this is a meaningful approach on hand of examples.

But most of the work remains to be done. On the application side, we need a more complete picture of existing inter-viewpoint models. The theory that will allow us to do the correctness proofs is also needed, but the theory should be done depending on the needs on the application side.

References

1. aiT. `https://www.absint.com/ait/`.
2. RT-Druid. `http://www.evidence.eu.com/products/rt-druid.html`.
3. M. Al Khatib, A. Girard, and T. Dang. Scheduling of embedded controllers under timing contracts. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, New York, NY, USA, 2017. ACM.
4. E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification, CAV'02*, volume 2404 of *LNCS*, Copenhagen, Denmark, 2002. Springer-Verlag.
5. I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems. *Int. J. Software Tools for Technology Transfer*, 15(3), 2013.
6. A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secure Comput.*, 1(1), 2004.
7. P. Axer and R. Ernst. Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*. ACM, 2013.
8. F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. *Computer*, 36(4), 2003.
9. A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T. Nguyen, and J. Sifakis. Rigorous component-based system design using the BIP framework. *IEEE Software*, 28(3), 2011.
10. A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli. Composing heterogeneous reactive systems. *ACM Trans. Embedded Comput. Syst.*, 7(4), 2008.
11. A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J. Raclet, P. Reinkemeier, A. L. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen. Contracts for system design. *Foundations and Trends in Electronic Design Automation*, 12(2-3), 2018.
12. A. Bhat, S. Samii, and R. R. Rajkumar. Recovery Time Considerations in Real-Time Systems Employing Software Fault Tolerance. In S. Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

13. D. Brière, D. Ribot, D. Pilaud, and J.-L. Camus. Methods and specifications tools for Airbus on-board systems. In *Avionics Conference and Exhibition*, London, UK, 1994. ERA Technology.
14. W. Damm, H. Dierks, J. Oehlerking, and A. Pnueli. Towards component based design of hybrid systems: Safety and stability. In Z. Manna and D. Peled, editors, *Essays in Memory of Amir Pnueli*, volume 6200 of *LNCS*. Springer, 2010.
15. P. Derler, E. Lee, S. Tripakis, and M. Törngren. Cyber-physical system design contracts. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ICCPS '13, New York, NY, USA, 2013. ACM.
16. R. Ernst, R. Henia, and S. Quinton. Beyond the deadline: New interfaces between control and scheduling for the design and analysis of critical embedded systems. Tutorial at ESWeek, 2017.
17. J. Fellmuth, T. Göthel, and S. Glesner. Instruction Caches in Static WCET Analysis of Artificially Diversified Software. In S. Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
18. D. Fontanelli, L. Greco, and L. Palopoli. Soft real-time scheduling for embedded control systems. *Automatica*, 49(8), 2013.
19. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *International Workshop on Hybrid Systems: Computation and Control, HSCC'05*, volume 3414 of *LNCS*, Zurich, Switzerland, 2005. Springer.
20. G. Frehse, A. Hamann, S. Quinton, and M. Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*. IEEE Computer Society, 2014.
21. F. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1), 1999.
22. A. Girault and H. Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Trans. Dependable Secure Comput.*, 6(4), 2009.
23. S. Graf, R. Passerone, and S. Quinton. Contract-based reasoning for component systems with rich interactions. In A. L. Sangiovanni-Vincentelli, H. Zeng, M. D. Natale, and P. Marwedel, editors, *Embedded Systems Development, From Functional Models to Implementations*. Springer Verlag, 2014.
24. S. Graf and B. Steffen. Compositional minimization of finite state systems. In E. M. Clarke and R. P. Kurshan, editors, *Computer-Aided Verification, Proceedings of a DIMACS Workshop 1990, New Brunswick, New Jersey, USA, June 18-21, 1990*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1990.
25. R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysisthe symta/s approach. *IEE Proceedings - Computers and Digital Techniques*, 152, 2005.
26. K. Krüger, M. Völp, and G. Fohler. Vulnerability Analysis and Mitigation of Directed Timing Inference Based Attacks on Time-Triggered Systems. In S. Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
27. P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele. A hybrid approach to cyber-physical systems verification. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12. ACM, 2012.
28. J. Li, P. Nuzzo, A. Sangiovanni-Vincentelli, Y. Xi, and D. Li. Stochastic contracts for cyber-physical system design under probabilistic requirements. In *Proceedings of the 15th ACM-*

IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE '17, New York, NY, USA, 2017. ACM.
29. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1), 1995.
30. B. Meyer. Applying "design by contract". *IEEE Computer*, 25(10), 1992.
31. J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng.*, 7(4), 1981.
32. A. Müller, S. Mitsch, W. Retschitzegger, W. Schwinger, and A. Platzer. Change and delay contracts for hybrid system component verification. In M. Huisman and J. Rubin, editors, *Fundamental Approaches to Software Engineering*. Springer, 2017.
33. M. Otter, S. Mattsson, and H. Elmqvist. Multidomain modeling with Modelica. In *Handbook of Dynamic System Modeling*. Chapman and Hall/CRC, 2007.
34. R. Passerone, I. B. Hafaiedh, S. Graf, A. Benveniste, D. Cancila, A. Cuccuru, S. Gerard, F. Terrier, W. Damm, A. Ferrari, L. Mangeruca, B. Josko, T. Peikenkamp, and A. L. Sangiovanni-Vincentelli. Metamodels in europe: Languages, tools, and applications. *IEEE Design & Test of Computers*, 26(3), 2009.
35. G. A. G. A. Perez Castaneda, J.-F. Aubry, and N. Brinzei. Stochastic hybrid automata model for dynamic reliability assessment. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 225(1), 2011.
36. A. Pnueli and K. Apt. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems ...*, volume 13 of *NATO ASI Series*. Springer, 1985.
37. D. Powell. Failure mode assumption and assumption coverage. In *International Symposium on Fault-Tolerant Computing, FTCS-22*, Boston (MA), USA, 1992. IEEE. Research report LAAS 91462.
38. J. Reineke and S. Tripakis. Basic problems in multi-view modeling. In E. Ábrahám and K. Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
39. H. Sheikh and I. Ahmad. Sixteen heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-cores. *ACM Trans. on Parallel Computing*, 3(2), 2016.
40. B. Steffen. Unifying models. In R. Reischuk and M. Morvan, editors, *STACS 97, 14th Annual Symposium on Theoretical Aspects of Computer Science, Lübeck, Germany, February 27 - March 1, 1997, Proceedings*, volume 1200 of *Lecture Notes in Computer Science*. Springer, 1997.
41. P. Tabuada. *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009.
42. H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise WCET prediction by separated cache and path analyses. *Real-Time Systems*, 18(2/3), 2000.
43. Y. Zhao, Y. Xiong, E. A. Lee, X. Liu, and L. C. Zhong. The design and application of structured types in ptolemy II. *Int. J. Intell. Syst.*, 25(2), 2010.
44. D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *International Conference on Computer Aided Design, ICCAD'04*, San Jose (CA), USA, 2004.