Towards Customizable CPS: Composability, Efficiency and Predictability

Wang Yi(✉)

Uppsala University, Uppsala, Sweden
yi@it.uu.se

Abstract. Today, many industrial products are defined by software, and therefore *customizable* by installing new applications on demand - their functionalities are implemented by software and can be modified and extended by software updates. This trend towards customizable products is extending into all domains of IT, including Cyber-Physical Systems (CPS) such as cars, robotics, and medical devices. However, these systems are often highly safety-critical. The current state-of-practice allows hardly any modifications once safety-critical systems are put in operation. This is due to the lack of techniques to preserve crucial safety conditions for the modified system, which severely restricts the benefits of software.

This work aims at new paradigms and technologies for the design and safe software updates of CPS at operation-time – subject to stringent timing constraints, dynamic workloads, and limited resources on complex computing platforms. Essentially there are three key challenges: *Composability*, *Resource-Efficiency* and *Predictability* to enable modular, incremental and safe software updates over system life-time in use. We present research directions to address these challenges: (1) Open architectures and implementation schemes for building composable systems, (2) Fundamental issues in real-time scheduling aiming at a theory of multi-resource (inc. multiprocessor) scheduling, and (3) New-generation techniques and tools for fully separated verification of timing and functional properties of real-time systems with significantly improved efficiency and scalability. The tools shall support not only verification, but also code generation tailored for both co-simulation (interfaced) with existing design tools such as Open Modelica (for modeling and simulation of physical components), and deployment on given computing platforms.

1 Background

Our life is becoming increasingly dependent on software. Many industrial products are defined by software, thus *customizable* as smart phones: their functionalities, features and economical values are realized by software and can be changed on demand over their life-time through software update. Indeed these products often serve as an open platform through software to access numerous services provided by remote servers in the cloud thanks to the emerging technologies of Internet-of-Things (IoT), cloud storage, cloud computing, data centers etc. The

© Springer International Publishing AG 2017
Z. Duan and L. Ong (Eds.): ICFEM 2017, LNCS 10610, pp. 3–15, 2017.
https://doi.org/10.1007/978-3-319-68690-5_1

trend towards customizable products is extending into all application domains of IT including Cyber-Physical Systems (CPS) such as cars, robotics and medical devices. Today software in our cars may be updated in service workshops; Tesla even allows customers to upgrade remotely the software system of their electric vehicles. Even avionics, traditionally a very conservative area, is moving from functionally separated solutions on uniprocessors to integrated systems on multi-core platforms with the capability to re-configure during operations [10]. However, CPS are often highly safety-critical, thus utmost care must be taken to ensure crucial safety conditions.

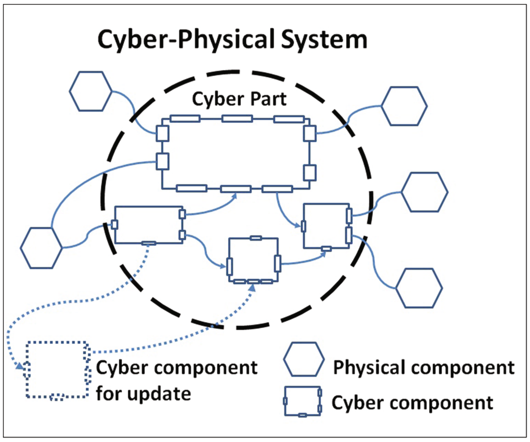

Fig. 1. Towards an open architecture for updatable CPS

Current design methodologies for CPS offer limited support for software updates on systems in operation. Although updates are possible in areas where certification is not mandatory, it is often restricted to updates either offered by professional service providers or software upgrading prepared (through intensive verification and test in the lab [23]) by the manufacturers e.g., Tesla. In general, the current state-of-practice allows hardly any modifications once safety-critical systems are put in operation due to the lack of technology to preserve the safety conditions of the modified systems. A classic example is civil avionics [28]: once a passenger aircraft built by Boeing is certified for operation, it should fly for life-time (estimated 50 years) without modifications to its electronic system and for maintenance the company must purchase the original electronic control units and store them for 50 years. It is remarkable that in the era of IoT when everything is connected and everything is changing over time, we are flying in a machine running outdated software made decades ago. This largely restricts the benefits of software.

2 Why Update CPS in Operation?

In less safety-critical areas, software updates are widely adopted by users to increase system safety by software patches or extend system functionalities for better utilization of the computational resources by installing new applications on demand. Smart phones and notebooks are examples. Apart from the lack of technologies for safety preservation, there seems to be no reason why software for CPS in general should not be updated.

CPS may be small embedded devices or large-scale networked embedded systems often viewed as systems of (sub-)systems with underlining communication infrastructure; a (sub-)system may have the overall architecture as shown in Fig. 1, consisting of a collection of cyber components (software components) interacting with physical components in real time through sensors and actuators. At operation-time, the user or customer may want to update the software system by installing a new application (a software component) purchased from a software provider. The update should be done by herself, not a professional service provider or the manufacturer who has full access to the software system. For example, consider smart transportation. When travelling in North Europe in the winter with a future self-driving car, we may want to install *ourselves* applications for ice- and elk-detection for safe driving. Another example is medical device e.g., pacemaker. In a possible future scenario, for a patient carrying such a device, a new heart problem may manifest over time (e.g., due to aging of the patient). To treat the upcoming problem, a doctor may propose to install a new application instead of replacing the device.

3 The Challenges

Clearly, the examples illustrate that software updates may lead to more reliable and cost-efficient solutions. However, for both examples, we must make sure to preserve the following two basic conditions before the intended updates are realized:

Condition 1 the new application will not interfere with the existing system: they should not block each other due to synchronization and their input and output values should be compatible, satisfying required relationships – the functional correctness must be preserved.

Condition 2 the computing platforms have enough computational resources to run the new application without being overloaded or violating any timing constraints – non-functional correctness must be preserved.

These two safety conditions illustrate the following key challenges for the design and update of CPS:

Composability (The design challenge) to build systems that are updatable at operation-time, allowing for modular updates that should neither require re-designing the original system, nor interfere with the functionalities of the original system (Condition 1).

Efficiency (The run-time challenge)to optimize resource utilization for preserving Condition 2 such that incremental updates may be applied over the system's life-time.

Predictability (The verification challenge) to enable safe updates through verification of the two conditions (Condition 1 and 2) on demand before the intended updates are committed.

The three challenges are often contradicting. For example, to optimize resource utilization, a global solution may be preferred. However, globally dynamic resource sharing may result in unmanageable non-determinism leading to poor predictability. Similarly designing systems for predictability using monolithic-threading or time-triggered approaches may not be an advantage for achieving the composability because these approaches require all computation jobs and resource accesses must be scheduled at design-time, which leaves little possibility for updates after deployment. In fact, current design methodologies of embedded systems allow for systems that are either predictable or composable, and often resource in-efficient in many cases due to resource over-dimensioning. For instance, synchronous systems [22] designed for predictable and deterministic behavior are often hard to modify and difficult for integration of new functionalities without re-designing the whole system; whereas concurrent systems [6] with multi-threading can be extended easily by new threads for new functionalities, thus are better for composability, but poor for predictability as they are hard to verify due to non-determinism.

For the design of updatable systems, naturally we take a component-based approach, which allows for modular changes. Component-based software development has attracted a large amount of research; in the past decades, various component models have been developed e.g. [9] for a classification of software models in the context of software engineering. In the domain of embedded systems, considerable efforts have been investigated within the ARTIST initiatives on model-based design (see e.g. [5]). An interesting line of work is the theories of interfaces e.g. [12] for timed systems, [8,41] for resource modeling and scheduling and more recently, contract-based systems design [3,13]. However, all previous attempts address only issues on the design of systems. Our focus will be on updates after deployment. Conceptually the existing techniques are useful, but not applicable. For updates, we must address composability issues at run-time. First, we must build systems that are updatable. Second, we must make sure that the updates are safe before they are realized.

4 Objectives

We distinguish design-time, operation-time and run-time. Operation-time means when systems are in operation after deployment, which can be off-line or on-line but not necessarily run-time. Any requirement at operation-time is more demanding than design-time but less than run-time. Therefore for operation-time updates, we assume that the overall architecture of a system and also its components (or sub-systems) are all designed, verified and deployed at design-time and abstract models tailored for operation-time verification of the two conditions are available.

The overall objective of this work is three-fold. First we aim at new implementation schemes for building updatable systems. Second, we develop scheduling techniques to optimize resource utilization at run-time and thus enable incremental updates over system's life-time. Third, we develop verification techniques and tools to validate the safety of updates on demand. Now we outline our ideas to reach the goals.

Composability shall be achieved by (1) multi-threading and (2) non-blocking communication that preserves synchronous semantics for data exchange among components. The objective of this work is to develop open system architectures as illustrated in Fig. 1 offering open interfaces and new implementation schemes to build systems allowing for integration of new software components by simply creating new threads. The threads will be coordinated by a centralized run-time system to ensure that the synchronous semantics of data exchange among threads (by reading and writing requests) is preserved [7] and the timing constraints on computation jobs released by threads are satisfied. For updating such multi-threading systems with new software components under the described requirements, we need to solve optimization problems similar to retiming of synchronous circuits, a classic problem in circuits design [29].

Resource-efficiency will be addressed by static partitioning and run-time scheduling. The objective of this work is two-fold. First, we study fundamental issues in real-time scheduling, addressing the optimality and complexity of scheduling algorithms [15] in particular questions related to dynamic workloads with complex release patterns of computation jobs and parallel computing platforms such as multi- and many-cores with massive parallel and heterogeneous processing units. The goal is to develop a parallel version of the real-time calculus [40] aiming at a unified theory for characterization of parallel and heterogeneous resource demands and resource supplies, as well as optimal mapping between them, as a scientific foundation for multiprocessor scheduling, which is a hard open problem in the field of real-time systems. Second, a practical approach will be taken to achieve near-optimal solutions for applications under assumptions in systems building such as non-blocking data exchange.

Safety-conditions will be ensured by verification on demand before the intended updates are committed. The objective of this work is to develop a new generation of verification techniques and tools for CPS in particular a new version of UPPAAL with significant improvements on efficiency and scalability by fully separating the analysis of timing and functional correctness. Functional properties will be specified and verified in a contract-based framework supported with SMT-based verification techniques. Timing and non-functional properties will be specified on computation jobs and verified using scalable techniques developed for scheduling analysis [34,35]. For uniprocessor platforms, the existing techniques and tools e.g. [1] scale well with industrial size problems. The future focus will be on multicore platforms.

5 Work Directions

To address the challenges, we propose the following work directions.

5.1 Towards Open Architectures for Updates

We consider CPS that may be large-scale networked embedded systems of (sub-)systems. A (sub-)system with its own computing platform may have a set of software components (cyber part as shown in Fig. 1) deployed based on a data-flow-like diagram with basic blocks representing its components (which may have hierarchical structures) and (links representing the input and output relation among the components via interfaces. A sub-system may contain local network links for which extra blocks should be created, modelling the delays for data exchange if the delays are not ignorable. The diagram may also contain cycles; however a cycle should contain a delay block to avoid infeasible behaviors. For abstraction, each physical-component is assumed to have a set of data buffers (e.g., implemented by a driver) as its interface for data exchange with software components.

To enable updates at operation-time after deployment, we must build systems that enjoy the following properties (see e.g. [2]): (1) integrating a new component should not require re-designing the whole system, and (2) a newly integrated component should not interfere with the existing components. Apart from resource sharing that shall be addressed separately, there are essentially two sources for potential interferences:

- The outputs of a component are not needed by the others, violating the functional correctness and
- The Components may block each other due to synchronization mechanisms for keeping data coherence.

In the following we propose solutions to disable these potential interferences.

Components, Interfaces and Contracts. We do not restrict how a component is implemented inside but it must offer a well-defined interface containing a set of input and output data buffers open for updates allowing for integration of new components. The functional correctness of a component is specified by a contract on its interface consisting of a pre-condition on input buffers and a post-condition on output buffers. The contract is a local invariant satisfied by the computational behavior of the component, which should be verified at design-time.

Furthermore, a workload model specifying the timing constraints and resource requirement of each component should be available (created at design-time), which may be considered as part of the contract. The workload model (or task model) of a component specifies the release patterns of three types of requests: reading, writing and computing (jobs). At run-time, the computing jobs will be scheduled and executed according to the timing constraints. The reading and writing requests will be non-blocking and coordinated to preserve the synchronous semantics.

Non-blocking Data Exchange that Preserves Synchronous Semantics. To implement the components and the original system, any synchronization schemes may be adopted to keep data coherence. However for integration of new components at operation-time, we have to adopt non-blocking data exchange. For non-blocking writing on an input buffer, only one-writer is allowed; but an output buffer may allow arbitrary number of readers. Data items written should be considered as non-consumable. The rationale is that new integrated components may only read data from an open interface of the existing system for computing their own output. The computed values may be used for realizing new functionalities or write back to the existing system to improve the existing functionalities on input buffers that previously have default values before the integration.

Reading is enabled (non-blocking) at any time; it is only copying (but not consuming) the data; writing will over-write the previously written data; thus only the latest data (i.e., the most fresh) values are available in the input buffers if the buffer capacity is not enough and the readers are slower than the writer.

To keep data coherence, we will develop new synchronization protocols to preserve the synchronous semantics of data exchange [7], ensuring two conditions:

- Globally all readers should receive the same data if the reading requests are issued after the same writing request and
- Locally for each component, the writing of an output value should correspond to the input value by the preceding reading request.

Essentially the arrival order of reading and writing requests should be enforced by the centralized run-time system; whereas the computation jobs may be scheduled in any orders satisfying the timing constraints provided that the local order of reading (input), computing (jobs) and writing (output) is preserved for each component. For the simple case when computation jobs take no time, the the DBP protocol (Dynamic Buffering Protocols [7]) can be used to preserve the synchronous semantics. Here we have a challenging case where computation jobs will have non-zero computation times (specified by WCETs) and timing constraints such as deadlines. The computation jobs may be released according to any patterns e.g., specified using graph-based real-time task models e.g., [34]. Our goal is to develop scheduling algorithms and data buffering protocols to preserve both the timing constraints and also the synchronous semantics. The hard technical challenge is to design algorithms and protocols which can be re-configured at operation-time to handle software updates. This requires to solve non-trivial optimization problems similar to retiming of synchronous circuits [29]. We aim at techniques for near-optimal solutions. This work shall be driven and evaluated by case studies including a large-scale industrial application to build a solar-powered electric vehicle [31].

5.2 Towards Precise Workload Modelling and Optimal Scheduling

Product customization often refers to incremental modifications. For CPS, it is about (1) integrating a new software component for extensions with new functionalities or (2) updating an existing software component for improvements on the systems functionalities. Both cases may increase resource requirement incrementally and so incease the system workload. Eventually it will hit the limit of resource utilization (the ideal case is 100%) when the system is infeasible or when a timing constraint for a computation job is violated. Thus run-time resource management and scheduling is crucial for the customization of systems in operation if not for the original design where the ad hoc solution in practice is often by over-dimensioning the system resources with redundancy, which is not an option for customization. If system resources are not utilized in an optimal manner, the possibility for customization will not last long. There are two technical challenges. First, the workload (or resource requirement) of each software component (and the whole system) should be modelled and characterized as precisely as possible to reduce the pessimism of scheduling analysis, thus potentially allow for more applications to run concurrently on the platform. Second, the workload should be mapped and scheduled on the platform to achieve optimal resource utilization.

For a survey on real-time workload models, see e.g. [25, 36]. There is a full hierarchy of workloads models available of different expressive powers and degrees of analysis difficulty as shown in Fig. 2. In the context of real-time systems, often simplistic models such as periodic or sporadic models (e.g. the classic task model L&L due to Liu and Layland [30]) are adopted to over-approximate the workload generated by physical- and software-components, which in many applications may lead to pessimistic analysis and resource over-dimensioning. To faithfully describe the resource requirements and timing constraints of software components, as the basis for workload characterization, we will use the di-graph real-time task model [34] and further extend it within manageable complexity for automated analysis with features including parallel OpenMP like structures [37,38], synchronization [32] as well as mixed criticality workloads [14,18]. To capture the dynamic workload triggered by physical components, a new line of work has been proposed in [4,33] to compute faithful abstract models from hybrid models for precise schedulability analysis. The work will be further developed to compute the

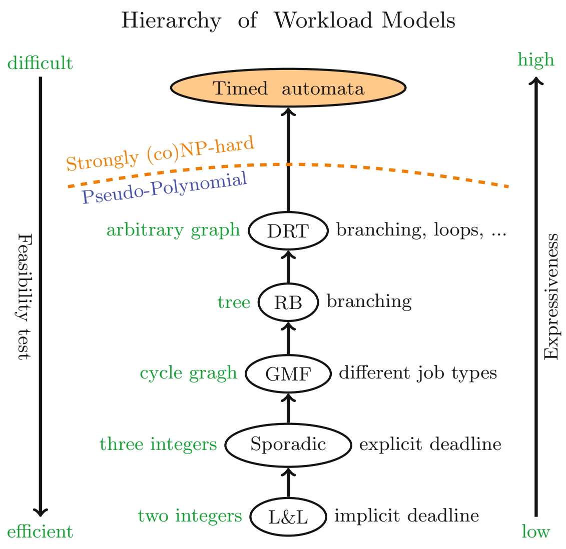

Fig. 2. Expressiveness vs. analysis efficiency

di-graph models from general hybrid automata based on the theory of optimal control and abstraction techniques [24].

Second, efficient scheduling algorithms and methods must be develop to optimize resource utilization at run-time. For uniprocessor platforms, the theoretical foundation of real-time scheduling has been established in the past decades with various scheduling and analysis techniques available. Several fundamental problems are solved recently [16,17,19]. However to implement the synchronous semantics of data exchange in a multi-threading setting, run-time scheduling must consider dependent tasks imposed by the input and output relations defined by a data-flow like diagram as shown in Fig. 1. It is a non-trivial technical challenge to design optimal scheduling algorithms for graph-based real-time models under precedence constraints even for uniprocessor platforms. To enforce the synchronous semantics, we foresee that memory consumption must be handled efficiently to make multiple local copies of the same data dynamically to serve the reading requests. Thus memory requirements must be considered in run-time scheduling, which brings another dimension of complexity in uniprocessor scheduling. On the theory side, there are still open issues in uniprocessor scheduling, including the complexity of uniprocessor scheduling of sporadic tasks with arbitrary deadlines and optimality of mixed-criticality scheduling [15].

For multicore platforms, the research community has produced a large number of insightful theoretical results [11], with the hope to extend the well-established theory for uniprocessor systems developed in the last three decades to the multiprocessor setting, e.g. our work on extending the classic result of Liu and Layland on rate-monotonic scheduling to multiprocessor setting [20]. We aim at obtaining such results also for heterogeneous platforms. In particular, we will study the application mapping problem on heterogeneous multiprocessor platforms that may have processor cores e.g. GPU, CPU with different processing speeds, and non-trivial interaction with I/O devices, as well as memory requirements, which is often the case in embedded applications in the context of CPS. The ultimate goal is to develop a parallel version of the real-time calculus [21,26,39,40], as a scientific foundation for multi-resource (including multiprocessor cores) scheduling, which is one of the challenging open problems in the field of real-time systems.

However, we will also take a practical approach to develop real-time applications for updatable systems on platforms with massive parallel processing units such as multi- and many-core, for which near-optimal solutions may be possible under the assumption in systems building such as non-blocking data-exchange.

5.3 Towards Fully Separated Verification of Timing and Functional Properties

To validate the safety of operation-time updates, we must developed powerful and scalable techniques for automated verification of the safety conditions as outlined earlier. Since the invention of model checking, the area of verification has advanced significantly with tremendous success in industrial applications.

Complex systems with millions of states and configurations may be verified today in seconds.

Model checking technology has been adopted to verify real-time systems where UPPAAL [27] is one of the leading tools. However, it is well-recognized that the technique suffers from the scalability problem, which is even more critical for real-time systems where the tool must handle not only functional properties but also timing constraints. We will take a different approach and fully separate the verification of functional and non-functional properties to improve the scalability of the tool. This is a lesson learnt from the development of UPPAAL. Mixing up functional and timing behaviors in modelling harms significantly the efficiency of the tool, which is the critical barrier for its scalability. In the implementation of UPPAAL, for verifying functional correctness e.g. deadlock-freeness or mutual exclusion properties, it demands a large amount of memory for keeping track of the timing constraints. Unfortunately UPPAAL had to treat these aspects in one unified framework which is not adequate for verification with manageable complexity. Major safety-critical properties should be guaranteed independently of timing. For example, a system should be deadlock-free independently of how fast a component is executed. There is room for great improvements.

The term *predictability* refers often to *easy-to-verify*. It concerns two parts. First, the system must be built *verifiable*. The model selected for design and verification should be as expressive as possible to express interesting system features; however it should not be too expressive with unnecessarily expressive power which may harm the analysis efficiency. The models for verification should be carefully selected for efficient analysis and fast termination. For many applications, timed automata are often too powerful. For example, to model real-time task release patterns, only lower bounds on clocks are needed to express the minimal release distances of computation jobs, mimicking the delay statements in real-time programming languages e.g. Ada. However for reasoning about timing constraints on computations, upper bounds on clocks are needed to express deadlines. Separating lower bounds on task releases and upper bounds on computation jobs leads to an adequate model for real-time systems [34] for which feasibility analysis can be verified efficiently in pseudo polynomial time [17]. This model will be the basis for our work on verification of timing and non-functional properties.

For non-functional correctness, as part of its interface to the computing platform, each component will have a workload model specifying its timing constraint and resource requirement within the tractable hierarchy as shown in Fig. 2 (see [36] for details) for efficient operation-time checks on demand. However, the demanding challenge is in the scheduling and analysis on complex platforms such as multi-cores. A partition-based approach is promising [20], which may reduce the analysis problems to the uniprocessor setting. In connection with the work on developing the theory of multiprocessor scheduling outlined earlier, different strategies will be evaluated for multiprocessor schedulability analysis.

To reason about functional correctness, a theory of contract-based interfaces will be developed based on first-order logic where a component interface will be specified using pre- and post-condition on input and output data as a local invariant of the component. Given contracts for each component, two essential properties have to be verified: (1) each component satisfies its own contract; and (2) the contracts of components combined in a system are compatible, i.e., each component produces the outputs needed by the other components. The first aspect (1) is essentially a problem of software verification, and will be addressed using techniques from abstract interpretation, software model checking, and SMT-based verification. A central concern in (1) will be the handling of real-valued quantities, which are today in software represented mainly as fixed-point or floating-point data, and supported by only few of the existing analysis tools (and usually with limited scalability). A technical challenge is to advance the state of the art in floating-point verification and develop improved SMT techniques for this theory. In (2), logical relationship between multiple contracts must be checked, a problem that is today primarily addressed with the help of SMT solving. Again, the main concern will be to scale up SMT methods to handle the relevant data-types and the extent of contracts needed for real-world systems.

Acknowledgement. For discussions on ideas presented in this document, the author wish to thank Jakaria Abdullah, David Black-Schaffer, Gaoyang Dai, Pontus Ekberg, Peter Fritzon, Nan Guan, Bengt Jonsson, Morteza Mohagheghy, Christer Nordström, Philipp Ruemmer, Joseph Sifakis, Martin Stigge, Janos Sztipanovits and Aleksandar Zeljic.

References

1. Abdullah, J., Dai, G., Guan, N., Mohaqeqi, M., Yi, W.: Towards a tool: times-pro for modeling, analysis, simulation and implementation of cyber-physical systems. In: Aceto, L., et al. (eds.) Larsen Festschrift. LNCS, vol. 10460, pp. 23–639. Springer, Heidelberg (2017). doi:10.1007/978-3-319-63121-9_31
2. Attie, P., Baranov, E., Bliudze, S., Jaber, M., Sifakis, J.: A general framework for architecture composability. Formal Aspects Comput. **28**(2), 207–231 (2016)
3. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J., Reinkemeier, P., Vincentelli, A.S., Damm, W., Henzinger, T., Larsen, K.G.: Contracts for systems design: theory. INRIA report, France (2015)
4. Biondi, A., Buttazzo, G., Simoncelli, S.: Feasibility analysis of engine control tasks under edf scheduling. In: Proceedings of ECRTS15, pp. 139–148. IEEE (2015)
5. Bouyssounouse, B., Sifakis, J.: Embedded Systems Design: The ARTIST Roadmap for Research and Development, vol. 3436. Springer, Heidelberg (2005)
6. Burns, A., Wellings, A.: Concurrent and Real-Time Programming in Ada. Cambridge University Press, New York (2007)
7. Caspi, P., Scaife, N., Sofronis, C., Tripakis, S.: Semantics-preserving multitask implementation of synchronous programs. ACM Trans. Embed. Comput. Syst. **7**(2), 15:1–15:40 (2008)
8. Chakabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.I.A.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003 (2003)

9. Crnkovic, I., Sentilles, S., Vulgarakis, A., Chaudron, M.R.V.: A classification framework for software component models. IEEE Trans. Softw. Eng. **37**(5), 593–615 (2011)
10. Certainty (Deliverable D1.2): Certification of real time applications designed for mixed criticality (2014). www.certainty-project.eu/
11. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. **43**(4), 35:1–35:44 (2011)
12. de Alfaro, L., Henzinger, T.A., Stoelinga, M.I.A.: Timed interfaces. In: EMSOFT 2002, pp. 108–122 (2002)
13. Derler, P., Lee, E.A., Tripakis, S., Törngren, M.: Cyber-physical system design contracts. In: Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems, ICCPS 2013, pp. 109–118. ACM (2013)
14. Ekberg, P., Yi, W.: Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. Real-Time Syst. **50**(1), 48–86 (2014)
15. Ekberg, P., Yi, W.: A note on some open problems in mixed-criticality scheduling. In: Proceedings of the 6th International Real-Time Scheduling Open Problems Seminar (RTSOPS) (2015)
16. Ekberg, P., Yi, W.: Uniprocessor feasibility of sporadic tasks remains conp-complete under bounded utilization. In: Proceedings of RTSS15, pp. 87–95 (2015)
17. Ekberg, P., Yi, W.: Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly conp-complete. In: ECRTS 2015, pp. 281–286 (2015)
18. Ekberg, P., Yi, W.: Schedulability analysis of a graph-based task model for mixed-criticality systems. Real-Time Syst. **52**(1), 1–37 (2016)
19. Ekberg, P., Yi, W.: Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard. In: Proceedings of RTSS17, Paris (2017)
20. Guan, N., Stigge, M., Yi, W., Yu, G.: Fixed-priority multiprocessor scheduling with liu and layland's utilization bound. In: Proceedings of RTAS 2010, Stockholm, pp. 165–174 (2010)
21. Guan, N., Yi, W.: Finitary real-time calculus: efficient performance analysis of distributed embedded systems. In: RTSS 2013, pp. 330–339, December 2013
22. Halbwachs, N.: Synchronous Programming of Reactive Systems. The Springer International Series in Engineering and Computer Science. Springer, New York (2013)
23. Holthusen, S., Quinton, S., Schaefer, I., Schlatow, J., Wegner, M.: Using multi-viewpoint contracts for negotiation of embedded software updates. In: Proceedings 1st Workshop on Pre- and Post-Deployment Verification Techniques, Iceland, pp. 31–45, June 2016
24. Krčál, P., Mokrushin, L., Thiagarajan, P.S., Yi, W.: Timed vs. time-triggered automata. In: Proceedings of CONCUR 2004, London, pp. 340–354 (2004)
25. Krcál, P., Yi, W.: Decidable and undecidable problems in schedulability analysis using timed automata. In: Proceedings of TACAS 2004, pp. 236–250 (2004)
26. Lampka, K., Bondorf, S., Schmitt, J., Guan, N., Yi, W.: Generalized finitary real-time calculus. In: Proceedings of IEEE INFOCOM 2017, Atlanta, GA, USA (2017)
27. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. STTT **1**(1), 134–152 (1997)
28. Lee, E.A.: Time for high-confidence cyber-physical systems. In: ICES workshop on Embedded and Cyber-physical Systems - Model-Based Design for Analysis and Synthesis, 6 February 2012, Stockholm, Sweden (2014)
29. Leiserson, C.E., Saxe, J.B.: Optimizing synchronous systems. In: FOCS 1981, the 22nd Annual Symposium on Foundations of Computer Science, pp. 23–36. IEEE (1981)

30. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM **20**(1), 46–61 (1973)
31. Lv, M., Guan, N., Ma, Y., Ji, D., Knippel, E., Liu, X., Yi, W.: Speed planning for solar-powered electric vehicles. In: Proceedings of the Seventh International Conference on Future Energy Systems, Waterloo, ON, Canada, 21–24 June 2016, pp. 6:1–6:10 (2016)
32. Mohaqeqi, M., Abdullah, J., Guan, N., Yi, W.: Schedulability analysis of synchronous digraph real-time tasks. In: Proceedings of ECRTS 2016, France, pp. 176–186 (2016)
33. Mohaqeqi, M., Abdullah, S.M.J., Ekberg, P., Yi, W.: Refinement of workload models for engine controllers by state space partitioning. In: Proceedings of ECRTS 2017, Croatia, pp. 11:1–11:22 (2017)
34. Stigge, M., Ekberg, P., Guan, N., Yi, W.: The digraph real-time task model. In: Proceedings of RTAS 2011, Chicago, IL, USA (2011)
35. Stigge, M., Yi, W.: Combinatorial abstraction refinement for feasibility analysis. In: Proceedings of RTSS 2013 (2013)
36. Stigge, M., Yi, W.: Graph-based models for real-time workload: a survey. Real-Time Syst. **51**(5), 602–636 (2015)
37. Sun, J., Guan, N., Wang, Y., He, Q., Yi, W.: Scheduling and analysis of real-time openmp task systems with tied tasks. In: Proceedings of RTSS 2017, Paris (2017)
38. Sun, J., Guan, N., Wang, Y., Deng, Q., Zeng, P., Yi, W.: Feasibility of fork-join real-time task graph models: hardness and algorithms. ACM Trans. Embed. Comput. Syst. **15**(1), 14:1–14:28 (2016)
39. Tang, Y., Guan, N., Liu, W., Phan, L., Yi, W.: Revisiting gpc and and connector in real-time calculus. In: Proceedings of RTSS 2017, Paris (2017)
40. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: ISCAS 2000, vol. 4, pp. 101–104 (2000)
41. Thiele, L., Wandeler, E., Stoimenov, N.: Real-time interfaces for composing real-time systems. In: Proceedings of the 6th ACM & Amp; IEEE International Conference on Embedded Software, EMSOFT 2006, pp. 34–43. ACM (2006)