Towards a Safe Compositional Real-Time Scheduling Theory for Cyber-Physical Systems *

Linh Thi Xuan Phan
University of Pennsylvania

Abstract

Modern cyber-physical systems are becoming increasingly complex and distributed. These trends are making it more and more difficult to ensure the timing guarantees of these systems: traditional approaches were developed for much simpler systems and are difficult to scale. As system sizes are growing further, designing future cyber-physical systems is going to be even more challenging.

Compositional design and compositional analysis have emerged as an effective means to address this challenge. Several interface models and interface computation methods have been developed, which can be used to analyze complex systems in an efficient manner. The existing theories provide a foundation for ensuring the timing guarantees of cyber-physical systems; however, they also have several important limitations. This position paper discusses open challenges in this domain, and it highlights several research directions towards a safe and resource-efficient compositional theory for cyber-physical systems.

1. Introduction

Cyber-physical systems (CPS) are becoming increasingly complex and distributed at large scale. One way to scale timing analysis to such complex systems is to develop them in a compositional manner [39]: real-time workloads (such as tasks) are encapsulated in components, which expose their resource needs through resource-aware interfaces. These components can be composed under a scheduling algorithm or input/output interconnections to form larger components, and their interfaces can be computed efficiently via component abstraction and interface composition. Timing constraints of a component can then be guaranteed by ensuring that the platform satisfies the component's interface.

Several interface models and interface computation techniques have been developed (see e.g., [39, 9, 16, 27, 34, 38, 21, 14, 42, 30, 6, 10, 41, 20]), which enable efficient integration and isolation of independently-developed cyber-physical components. However, these existing theories face several important limitations: they ignore the platform overhead and the correlation between scheduling and communication, which could lead to timing violations in practice; they assume that nodes' clocks are closely synchronized, which is difficult and expensive to accomplish in a complex networked setting; they consider exclusively timeliness and ignore the semantics of timed interactions between components, or vice versa; and they focus only on the cyber aspects, while making implicit assumptions about the physical aspects that may not always be realizable.

Some of these assumptions are not unique to the current compositional theories; they come from the underlying schedulability and timing analysis. However, in large-scale distributed CPS, they are much more likely to be violated and can no longer be ignored. In this paper, we discuss these limitations in more detail, and we briefly sketch how the current theories could be extended to overcome them.[1]

2. Platform overhead matters

The current compositional theories assume a somewhat idealized platform in which all overhead is negligible. In practice, the platform overhead – such as release delay, preemption overhead, cache effects, context switches, and interrupt delay – can substantially interfere with the execution of tasks. Without considering such overhead, the computed interfaces can underestimate the components' resource needs; hence, the components can violate their timing constraints even if their interfaces are satisfied [35].

At first glance, it may seem that this issue can be solved by inflating the worst-case execution time (WCET) of each task by the overhead it experiences. However, this approach can be unsafe: including the overhead as part of the tasks' WCETs implies that the sources of overhead are assumed to be scheduled together with tasks, but this does not hold for certain types of overhead, such as interrupts and task release events. Further, it is difficult to compute a safe bound on the overhead experienced by a task, since such overhead accumulates with the number of tasks in the entire system[2], but the task-level details of one component is hidden from another component in a compositional setting.

Accounting for the platform overhead on multicore processors is even more challenging because of the complex interactions between various platform resources. In a compositional setting, the overhead a component incurs, e.g., due to cache interference, is harder to quantify: it depends not only on the direct interference between the component's own tasks but also on the indirect interference between these tasks and the interfaces of other components,

* This research was supported in part by the ARO grant W911NF-11-1-0403, ONR grant N00014-13-1-0802, and NSF grants CNS-1117185, ECCS-1135630 and CNS-1329984.

The 4th Analytic Virtual Integration of Cyber-Physical Systems Workshop
December 3, 2013, Vancouver, Canada.
 Proceedings are available at:
http://dx.doi.org/10.3384/ecp13090
AVICPS website:
http://www.analyticintegration.org/

[1] Interface theories for ensuring functional correctness have been studied extensively (see e.g., [37, 32, 36]); in this paper, we focus primarily on interface theories from the scheduling perspective.

[2] This is because a task within a component may be delayed by the interrupt processing or task release events in other components.

and the latter in turn depends on the actual values of the interfaces and their implementations; consequentially, the interface computation becomes a cyclic process, which is often expensive and may not always converge.

In our recent work, we have proposed a new notion of overhead-aware interfaces, as well as methods for computing the interfaces that take into account platform overhead [35, 43]. While these initial solutions are promising, much work remains to be done to achieve a compositional theory that is both safe and resource-efficient in practice.

3. Data-dependent components

Existing compositional theories typically assume independent execution of tasks; in practice, however, CPS often operate on data flows with end-to-end timing constraints. Therefore, new interface models and interface analysis methods that can provide end-to-end timing guarantees for data-dependent and distributed components are necessary. Although there are a number of relevant formalisms that provide partial solutions to this problem, such as assume-guarantee interfaces [21, 14, 42], data flow graphs with LET semantics [30], and interfaces for network resource [38], none of them considers end-to-end timing constraints and compositional scheduling concurrently.

To meet the above needs, the component model and composition semantics need to be extended to capture not only resource sharing but also data communication semantics and input/output connections. Ideally, the component model should be self-sufficient for the analysis, i.e., it should contain all the information necessary for computing the interfaces, such as local timing constraints (e.g., local deadlines) and activation patterns (e.g., periods or arrival patterns) of tasks. However, such information about a component is difficult to obtain, since it depends on the local constraints assigned to other data-dependent components and, consequently, on those components' interfaces. In addition, deriving a composition semantics that encapsulates the intricate correlation between scheduling and communication is also non-trivial.

To illustrate the above cyclic relationship, consider an end-to-end data flow with end-to-end deadline D that is processed sequentially by two tasks, T_1 and T_2, which are located in components C_1 and C_2, respectively. Then, the deadline of T_2 is inverse proportional to the deadline of T_1. Further, the arrival pattern of T_2's input data – which is needed to compute C_2's interface – depends on the arrival pattern of T_1's output data, which in turn depends on T_1's deadline as well as the resource supply of C_1's interface.

One approach to tackle the above challenge is to adapt deadline decomposition methods, coupled with synchronization protocols, such as those outlined in [28]. This direction offers a self-sufficient component model, and it enables an efficient interface computation by directly applying existing results. However, it can also result in non-optimal interfaces, due to the cyclic relationship between the component model and the interfaces (discussed above). Therefore, enhancements of the deadline decomposition methods and synchronization protocols are required to improve the analysis accuracy.

Another interesting direction is to explore *parametric interfaces*, where an interface can be represented as a function of variables that denote unknown factors, such as local timing constraints, and the interface computation can be performed symbolically. The concrete values of the interfaces can then be realized at the top-level component based on the end-to-end constraints. Since the size of the composed interface grows with more composition steps, it would be useful to refine the interface, e.g., using a safe approximation of the interface, at each composition step to improve the analysis efficiency.

4. Clock synchronization

The current real-time scheduling theory and interface analysis methods assume a common notion of time. However, achieving strong synchrony in a distributed system is a known hard problem: due to clock drift and network propagation delays, the local clocks of the different nodes are always slightly different, and even frequent resynchronization (which would have a high overhead) would not be sufficient to achieve perfect synchrony.

Without perfect synchrony, the notion of a deadline becomes ambiguous. Since CPS interact directly with the physical environment, deadlines are usually given in terms of the physical time; however, when a control or data flow passes through multiple nodes, each node's local clock can deviate from the physical time, which can result in jobs being scheduled too early or too late; also, nodes can disagree whether a deadline has been missed or met.

At first glance, it may seem that timing variance due to clock drift is too small to matter in practice; however, even small discrepancies can cause scheduling anomalies and thus 'snowball' into large anomalies. For instance, suppose nodes A and B are expected to send messages to node C at a certain time to trigger tasks T_A and T_B on C, and suppose further that node A's message is meant to be sent first, so that T_A is released before T_B. If B's clock is slightly faster than A's, then B may send its message too soon, causing the messages to be reordered and T_B to be released and scheduled before T_A; as a result, T_A may miss its deadline, even though the schedulability analysis (which assumes perfect synchrony) may have predicted that the deadline would be met. Worse, since C's schedule is now different, the timing of C's own messages is also affected, which may lead to further changes on other nodes. Thus, while the original discrepancy on B is small, the resulting effect on the system as a whole can be much larger.

One approach towards solving this problem is to extend the system model with a bound on the clock drift and the length of the synchronization intervals, so that it becomes possible to reason about possible deviations from the reference time, and to make scheduling decisions accordingly. To enable this approach, component interfaces would have to be extended to expose information about drift and synchronization, and the information would have to be carried over when interfaces are composed and analyzed. To keep the complexity of the analysis manageable, it may be useful to 1) abstract some of the details in the interface, and/or 2) specify requirements for other subsystems that the com-

ponent interacts with, such as an upper bound on the acceptable clock drift. One challenge is to determine good abstractions; another is to determine which requirements are useful and can be satisfied.

5. The gap between real-time scheduling theory and high-level formal models

Cyber-physical systems are traditionally modeled using two different paradigms: high-level models of computation and real-time task/resource models. These models capture different timing properties: The former focus on the high-level formal specifications of timed interactions, communications, and synchronization among a collection of independent processes or subsystems; examples include timed automata [4], I/O automata [29], and real-time process algebra [26]. The latter capture implementation-level task timing information (e.g., execution time, deadline, priority) and details on physical resources and resource sharing (e.g., processing speed, network bandwidth, memory, scheduling policy). Although both categories are intertwined, they are typically considered in isolation. On the one hand, verification techniques for timed concurrent models verify temporal properties based solely on the high-level model, without considering the platform aspects, such as communication delay and scheduling overhead (e.g., synchronization is assumed to be instantaneous). On the other hand, the task and resource models used in real-time scheduling theory are based on the execution platforms and the source-code of the software; unlike the high-level models, they do not take the semantics of communication into account. Thus, even if a higher-level property (e.g., a safety property) is proven in the higher-level model, it does not necessarily hold in the implemented system.

Several efforts have been made to bridge this gap by adding platform aspects to the high-level models. For instance, existing work on the implementability of timed automata incorporates the platform information in the timed automata model by explicitly modeling the execution platform [3] or by modifying the timed automata semantics to reflect the implementation platform, such as the sampling-based [23], almost ASAP [15, 11], time-triggered [22], and probabilistic and topological [8] semantics. Real-time scheduling has also been combined with timed automata in [2, 7] and with process algebra in [26, 31]. In addition, a number of automata- and actor-oriented scheduling interfaces have also been developed [6, 10, 41, 20].

However, the above approaches are not only expensive in terms of analysis complexity but also assume a very simple model of resources – for instance, the processor is assumed to have unit speed and be fully available. As we move towards multicore and distributed systems, these assumptions no longer hold: the processor is not always fully available due to various types of platform overhead (such as I/O, interrupts, communication), and these types of overhead can even vary between different nodes. We believe that it would be useful to develop new models and analysis techniques that combine both aspects.

One direction is to establish an intermediate 'glue layer' that connects the two classes of models, e.g., similar to the hierarchical heterogeneity approach for composing high-level models of computation in Ptolemy [17], or the functional mock-up interface for co-simulation and model exchange [1, 12]. The glue layer could precisely capture the assumptions (e.g., synchronization semantics) that the higher-level model makes about the platform, and it could be used to mechanically verify that a given platform satisfies these assumptions. The assumptions must be realistic (i.e., realizable on common platforms) but should abstract low-level details of the platform as much as possible. Since the assumptions may be relevant to other subsystems that the component communicates with, they should be taken into account by the interface analysis.

6. Analyzing state-based systems

While real-time scheduling theory provides a clean system abstraction and enables efficient analysis, it currently cannot be applied to scheduling state-based systems. For instance, a system might schedule tasks depending on the current state of their input buffers, or it might handle data differently based on its current buffer state (e.g., append new data items to an input buffer if there is room, and discard them otherwise). State-based scheduling is also inherent in adaptive systems, which need to respond to the external environment. For example, a video encoder component might send encoded video frames to a receiver component depending on how quickly the receiver can decode the video to avoid buffer overflows and buffer underflows. Although it is sometimes possible to derive a stateless approximation of the system, such approximation often leads to overly pessimistic analysis results.

A common approach to analyzing such systems is to use state-based models, such as timed automata [4] or event count automata [13], and to perform the analysis using formal verification. Here, if very small time steps are used, the resulting state space can be very large, and thus, the analysis can become expensive; hence, it is desirable to use large time steps wherever possible. However, large time steps can decrease accuracy, i.e., the analysis may fail for systems that are actually schedulable, or may succeed only with additional resources. It seems promising to apply abstraction refinement in these cases. Furthermore, some systems may contain a mixture of state-based and stateless components; to efficiently support this case, it would be useful to have a way to interconnect components of both types. It seems interesting to explore hybrid techniques, along the lines of [33] and [24].

7. Beyond resource-aware interfaces

The current compositional theories focus only on the cyber layer, such as timing and resource aspects, while making implicit assumptions about the physical system and the environment. As a result, failures may occur in the system, e.g., when these hidden assumptions are violated. To guarantee the safety and trustworthy of the system, it is critical to extend these theories beyond cyber concerns.

We believe that it would be essential to develop a notion of *safety-aware interfaces* to enable the compositional analysis of safety properties for CPS. For this, the com-

ponent and interface models would need to be extended to capture not only the cyber aspects but also the physical aspects and the cyber-physical interactions of components, including e.g., the dynamics of the physical system, the control algorithm, safety-criticality levels of tasks, and safety goals of components under different environment conditions. One approach is to integrate control-theoretic multi-mode systems [25, 5] with a mixed-criticality extension of real-time multi-mode interfaces [34].

A challenge towards a safety-aware compositional theory is how to detect unsafe interactions between components, as well as to enforce their absences, during the interface composition. Undesirable interactions between components on the same cyber-physical platform can arise from multiple dimensions, such as via shared data and variables, via shared actuators and sensors, via computational and communication resource sharing, and via the physical environment. For instance, an unsafe interaction between an adaptive cruise control component and a collision avoidance component in an automotive system might arise via the physical environment when the former requests an increase in speed while the latter simultaneously requests a sharp turn, which could cause the vehicle to roll over.

New approaches for analyzing the coupling between control, safety and resource aspects are especially needed to detect undesirable interactions such as above. It seems useful here to adapt results on hazard analysis and safety assessment from the safety engineering domain (e.g., [18]) for the modeling and interface analysis. Further, as it may not be feasible to identify all interactions statically, it would be interesting to incorporate the idea of run-time monitoring and recovery [40] to enable automatic refinements of the interfaces and component interactions during run time.

Besides safety, there is a growing need for a *security-aware* compositional theory for CPS. This is highly challenging, as it may not be possible to 'decompose' the impact of an attack down to the component level, and it is also difficult to predict the attack behavior (e.g., of a malicious adversary). Existing work on compositional security [19] might provide useful insights to address this challenge.

8. Conclusion

As cyber-physical systems continue to grow in complexity, we believe that compositional approaches will remain to be effective for future CPS design and analysis. We have discussed in this paper several important existing limitations and research challenges in this research area. The list is not exhaustive, but through it we hope to inspire future research towards a safe, secure and resource-efficient compositional theory for CPS that is fully realizable in practice.

References

[1] Functional Mockup Interface. `http://www.fmi-standard.org`.

[2] Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *TCS*, 354(2):272–300, 2006.

[3] K. Altisen and S. Tripakis. Implementation of timed automata: An issue of semantics or modeling? In *FORMATS*, 2005.

[4] R. Alur and D. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.

[5] R. Alur, V. Forejt, S. Moarref, and A. Trivedi. Safe schedulability of bounded-rate multi-mode systems. In *HSCC*, 2013.

[6] R. Alur and G. Weiss. Rtcomposer: a framework for real-time components with scheduling interfaces. In *EMSOFT*, 2008.

[7] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: a tool for schedulability analysis and code generation of real-time systems. In *FORMATS*, 2004.

[8] C. Baier, N. Bertrand, P. Bouyer, T. Brihaye, and M. Größer. Probabilistic and topological semantics for timed automata. In *FSTTCS*, 2007.

[9] S. Baruah and N. Fisher. Component-based design in multiprocessor real-time systems. In *ICESS*, 2009.

[10] P. Bhaduri and I. Stierand. A proposal for real-time interfaces in speeds. In *DATE*, 2010.

[11] P. Bouyer, K. G. Larsen, N. Markey, O. Sankur, and C. Thrane. Timed automata can always be made implementable. In *CONCUR*, 2011.

[12] D. Broman, C. Brooks, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter. Determinate composition of fmus for co-simulation. In *EMSOFT*, 2013.

[13] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *RTSS*, 2005.

[14] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *EMSOFT*, 2001.

[15] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost asap semantics: From timed models to timed implementations. In *HSCC*, 2004.

[16] A. Easwaran, I. Shin, and I. Lee. Optimal virtual cluster-based multiprocessor scheduling. *RTS*, 43(1):25–59, 2009.

[17] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity-the ptolemy approach. *Proc. IEEE*, 91(1):127–144, 2003.

[18] C. Ericson et al. *Hazard analysis techniques for system safety*. Wiley-Interscience, 2005.

[19] D. Garg, J. Franklin, D. Kaynar, and A. Datta. Compositional system security with interface-confined adversaries. *ENTCS*, 265:49–71, 2010.

[20] M. Geilen, S. Tripakis, and M. Wiggers. The earlier the better: a theory of timed actor interfaces. In *HSCC*, 2011.

[21] T. A. Henzinger and S. Matic. An interface algebra for real-time components. In *RTAS*, 2006.

[22] P. Krčál, L. Mokrushin, P. Thiagarajan, and W. Yi. Timed vs. time-triggered automata. In *CONCUR*, 2004.

[23] P. Krčál and R. Pelánek. On sampled semantics of timed systems. In *FSTTCS*, 2005.

[24] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In *EMSOFT*, 2009.

[25] J. Le Ny and G. J. Pappas. Sequential composition of robust controller specifications. In *ICRA*, 2012.

[26] I. Lee, J.-Y. Choi, H. H. Kwak, A. Philippou, and O. Sokolsky. A family of resource-bound real-time process algebras. In *FORTE*, 2001.

[27] H. Leontyev and J. H. Anderson. A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *RTS*, 43(1):60–92, Sep. 2009.

[28] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, 1st edition, 2000.

[29] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Info. and Comp.*, 185(1):105–157, 2003.

[30] S. Matic and T. A. Henzinger. Trading end-to-end latency for composability. In *RTSS*, 2005.

[31] M. Mousavi, M. Reniers, T. Basten, and M. Chaudron. Pars: a process algebra with resources and schedulers. In *FORMATS*, 2004.

[32] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of program analysis*. Springer, 1999.

[33] L. T. X. Phan, S. Chakraborty, P. S. Thiagarajan, and L. Thiele. Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In *RTSS*, 2007.

[34] L. T. X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *ECRTS*, 2010.

[35] L. T. X. Phan, M. Xu, J. Lee, I. Lee, and O. Sokolsky. Overhead-Aware Compositional Analysis of Real-Time Systems. In *RTAS*, 2013.

[36] B. C. Pierce. *Types and programming languages*. The MIT Press, 2002.

[37] J. A. Rowson and A. Sangiovanni-Vincentelli. Interface-based design. In *DAC*, 1997.

[38] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida. Multi-level hierarchical scheduling in ethernet switches. In *EMSOFT*, 2011.

[39] I. Shin and I. Lee. Compositional Real-Time Scheduling Framework. In *RTSS*, 2004.

[40] O. Sokolsky, U. Sammapun, I. Lee, and J. Kim. Run-time checking of dynamic properties. In *RV*, 2005.

[41] I. Stierand, P. Reinkemeier, T. Gezgin, and P. Bhaduri. Real-time scheduling interfaces and contracts for the design of distributed embedded systems. In *SIES*, 2013.

[42] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *RTAS*, 2006.

[43] M. Xu, L. T. X. Phan, I. Lee, O. Sokolsky, S. Xi, C. Lu, and C. D. Gill. Cache-Aware Compositional Analysis of Real-Time Multicore Virtualization Platforms. In *RTSS*, 2013.