Scalable Assurance of Cyber-Physical Systems

Dionisio de Niz
dionisio@sei.cmu.edu

Cyber-Physical Systems (CPS) are software-reliant systems that interact with the physical world. As such, their kinetic effects frequently have safety-critical consequences. The scientific community has recognized this challenge and created techniques to provide mathematical proof techniques for three main aspects (among many others) of their kinetic effect, namely (i) that the software calculates the correct action on the physical process (e.g., full braking) (ii) at the right time (e.g., 100ms after sensing braking threshold) ensuring the (iii) correct physical effect (e.g., dynamical system reach desired state – stopped – before hitting the wall). Unfortunately, the application of these techniques become impractical due to scalability issues. In this paper we discuss three key scalability issues: (i) multi-criticality, (ii) artifact size, and (ii) cognitive design overload. These issues are clearly not orthogonal and we will discuss their interactions when we describe them.

Multi-Criticality

The practitioner community has recognized for a long time that not all aspects of a CPS are safety-critical. While the scientific community has been aware of this, the need to carve a reasonably isolated scope has force the community to sometimes ignore this characteristic focusing on strong guarantees but weak scalability. More recently the academic community has understood the need to combine assurance of different parts of the system at different levels of rigor. These efforts have been conducted from independent scientific disciplines such as mixed-criticality scheduling [1] for timing. These approaches allow the application of different degrees of rigor to different layers of functionality. For timing, for instance, it uses a larger over-approximation of the worst-case execution time (WCET) of tasks to reduce the likelihood of an overrun developing verification tests that only guarantees meeting the deadline of a subset of tasks (the higher-critical tasks) with the strictest WCET and guarantee the deadlines of more tasks with lower strictness of WCET (next layer of lower-criticality tasks). This requires new scheduling mechanisms that monitors CPU consumption of the tasks and stops lower-criticality tasks if an overload is detected to ensure that higher-criticality tasks are guaranteed to terminate by their deadlines.

More recently, new efforts to combine logical and timing guarantees within the same platform have given birth to the real-time mixed-trust computation framework [2]. Moreover, in this framework not only different levels of verification rigor are supported (supporting both verified and unverified components) but also the protection of verified component from misbehavior of unverified ones is provided. In this case, the approach is to insert small pieces of verified components to monitor the output of the system for safety properties and replace these outputs for safe ones if they are deemed unsafe. This way we can (i) guarantee a system-wide safety property by only (ii) verifying small components. These components are known as *enforcers*. This framework is implemented with a VM where the unverified components run. This VM runs on top of a verified hypervisor (HV) [3] with a real-time scheduler that ensures that if a unverified task does not finish on time, a corresponding verified hypertask issues a safe action. At the same time the hypervisor protects a verified component that checks if the unverified component output is safe. The combined unverified (guest) task running in a VM and the verified hypertask running in the HV is know as a mixed-trust task and is scheduled with a combine VM and HV scheduler known as mixed-trust scheduler.

Artifact Size

The artifact size has been a problem for software-reliant system verification for a long time. It is well known that formal techniques like model checking cannot scale to large program sizes even with recent scalability advances. However, new tools (e.g., Frama-C) have allows us to apply these techniques to real programs in popular programming languages (e.g., C) for selective properties and components.

The enforcer concept presented before allows us to ignore a large number of components in the system and focus only on small pieces of code that can make strong verification possible. However, as discussed before, it is necessary to understand and protect against misbehavior of unverified components.

The concept of an enforcer has also been explored to verify dynamical system properties with the use of Lyapunov functions [5]. Lyapunov functions enables the evaluation of the tolerance of a system to a misbehavior of an unverified component. In particular, a Lyapunov function defines a positive invariant set that allows us to verify that if the misbehavior is eliminated (i.e., enforced) before the system leaves this set, the system can go back to its designated setpoint.

The combination of the logical [4], temporal [6], and dynamical [5] enforcer techniques in a properly protected framework [2] allows us to scale this powerful verification techniques to sizes of practical systems.

Cognitive Design Overload

Perhaps a subtler challenge related to the size of a system is the cognitive design overload. More specifically, the design of large CPS starts with humans defining what the system should do at an abstract level. This first level includes the functional decomposition of the system and high-level properties that we want the system to exhibit. The first level typically involves the development of the specification that will be use for acquisition. After this first level of design, the system and its part are further refined requiring to keep track how the additional specified details interact with all the other parts of the system and how they relate to the desired system-level properties. Clearly, the human cognition quickly reaches its limit when faced with this task. Model-Based Engineering (MBE) techniques have been developed for this purpose. In this case, a model of the system is built at a high level and continuously refined as the system is developed. The purpose of this model is to use automated analysis to verify the desired properties of the system. The benefits of this continuous analysis yield benefits right in the early models when analyses can discover errors that can be very costly if discovered in later stages such as system integration or fielding [11].

A number of modeling languages like SysML [7] and AADL [8] have been created for this purpose. AADL in particular, was created primarily for CPS systems with a strong core semantics to capture the interactions of software and hardware and its consequences to quality attributes like timing, reliability, and security. From the start AADL was designed to be extensible allowing, not only the incorporation of new analyses as natural plugins into its toolset, but also as extensions to the language itself in what is known as language annexes. However, this level of openness and the capacity to extend the expressiveness and analytic capability of AADL brings a new challenge to the cognitive design overload, namely analyses assumptions. Specifically, analysis make assumptions about the model that must be verified, otherwise the results can be rendered invalid. The verification of these assumptions grows in complexity as analyses grow in sophistication. Moreover, as multiple analyses are applied on the model they can make conflicting assumptions. In addition, as analyses are used at different refinement levels, the analysis results at one level can be invalidated by refinements to the model. To address this problem a number of research efforts have created analyses contracts [9,10] that enables the definition of contracts in the form of assumption/guarantee pairs. The assumptions, define conditions that the model should exhibit before the analysis can be applied. The guarantees, on the other hand, define the conditions that a successful analysis guarantees on the model. With these contracts it is possible to automate the assumption verification and analyses interactions of a model reducing the cognitive load of a modeler. Moreover, analyses contracts are key to apply analyses coming from different scientific disciplines whose details can only be fully understood by a discipline specialist.

The scalable assurance of CPS is by no means a solved issue. However, in this paper we presented a few of the key challenges that it entails.

Acknowledgements

References

1. Dionisio de Niz, Karthik Lakshmanan, and Raj Rajkumar."On the Scheduling of Mixed-Criticality Real-Time Tasksets." RTSS 09.

2. Dionisio de Niz, Bjorn Andersson, Mark Klein, John Lehoczky, Amit Vasudevan, Hyoseung Kim, and Gabriel A. Moreno. Mixed-Trust Computing for Real-Time Systems. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2019.
3. Design, implementation and verification of an extensible and modular hypervisor framework. A Vasudevan, S Chaki, L Jia, J McCune, J Newsome, A Datta. 2013 IEEE Symposium on Security and Privacy.
4. Bjorn Andersson, Sagar Chaki, and Dionisio de Niz. "Combining Symbolic Runtime Enforcers for Cyber-Physical Systems." International Conference in Runtime Verification. 2017.
5. R. Romagnoli, B. H. Krogh, B. Sinopoli (2019). Design of Software Rejuvenation for CPS Security Using Invariant Sets. In 2019 American Control Conference (ACC).
6. Sagar Chaki and Dionisio de Niz. "Formal Verification of a Timing Enforcer Implementation." EMSOFT 2017.
7. https://www.sysml.org.
8. SAE Architecture Analysis & Design Language (AADL) AS5506C.
9. Ivan Ruchkin, Dionisio de Niz, Sagar Chaki, and David Garlan.”"Contract-Based Integration of Cyber-Physical Analyses." EMSOFT 2014.
10. Ivan Ruchkin, Dionisio de Niz, Sagar Chaki, and David Garlan.”"ACTIVE: A Tool for Integrating Analysis Contracts." AVICPS 2014.
11. Peter H. Feiler, Jorgen Hansson. Dionisio de Niz, and Lutz Wrage. "System Architecture Virtual Integration: An Industrial Case Study." TECHNICAL REPORT CMU/SEI-2009-TR-017 ESC-TR-2009-017.