Multirate Integration Architecture

Provably Resilient Systems by Design

Entalus
Computer Science Labs

www.entalus.com

November 2023

Publish-subscribe. Producers in a publish-subscribe communication architecture create topics (e.g. data such as temperature, location, pressure, but also events and commands) and publish messages accordingly, whereas subscribers express interest in one or more topics, and only receive messages that are of interest, without knowledge of which publishing nodes, if any, there are. This kind of loose coupling supports scalable and dynamic communication structures, and interoperable nodes may be developed to a local specification of input and output messages. This eases modular development, system integration, and maintenance. Popular instances of publish-subscribe architectures for embedded and cyber-physical systems include the Robotics Operating System (ROS 2.0) [1], the related Distributed Data Services (DDS) [10], and NASA's open-source cFS software bus [2].

On the downside, the loose coupling in publish-subscribe architectures incurs performance penalties, and there are virtually no guarantees on real-time and resilient behavior. This makes it extremely difficult, if not impossible, to convincingly argue the safety and resililience of embedded and cyber-physical systems with traditional publish-subscribe architectures.

RADL Model of Computation. Node integration in RADL [3, 4, 5] is based on the publish-subscribe paradigm, but it also implements a multi-rate model of computation, which arises naturally in distributed settings where computing units execute periodically

according to their local clocks and communicate among themselves via message passing.

More precisely, the RADL model of computation consists of nodes operating quasi-periodically at their own periods, and communication through bounded latency channels. Hereby, RADL assumes bounded drift for local clocks and bounded communication latency. Interactions between nodes are decoupled by having components that execute in isolation and communicate through nonblocking channels. In this way, when a fast component generates high-frequency input that is received and processed by a low-frequency component, then messages might be lost, but it is not possible to cause the slow component to react faster than its frequency.

The RADL model of computation is most closely related to the Loosely TimeTriggered Architecture (LTTA) [0], the main difference being that LTTA employs a shared memory bus instead of a publish-subscribe communication architecture. One may also think of RADL as being a multi-rate extension of ROS 2.0.

The RADL model of computation is also flexible enough to support the whole design space between completely asynchronous and time-triggered behavior. This allows to configure RADL according to application-specific needs, including operating domains and dynamically changing performance requirements.

RADL Integration Architecture. The RADL model of computation provides a number of system-level guarantees, which have been formally proven [6]:

- The latency for processing messages is bounded
- Possible enforcement that messages do not overtake each other § Consecutive message loss is bounded
- Bounded length of message queues for eliminating message loss

These kinds of architectural properties are crucial in efficiently arguing the resilience and safety of *cyber-physical systems*. In addition, the RADL model of computation is also provably resilient against certain cybersecurity threats such as denial-of-service attacks.

The RADL model of computation and communication therefore combines some main advantages of a publish-subscribe architectures with those of resilient real-time integration architectures such as time-triggered architectures. In fact, RADL configurations cover the complete design space in between asynchronous and synchronously communicating systems. RADL can therefore flexibly be configured for meeting specific application needs and requirements.

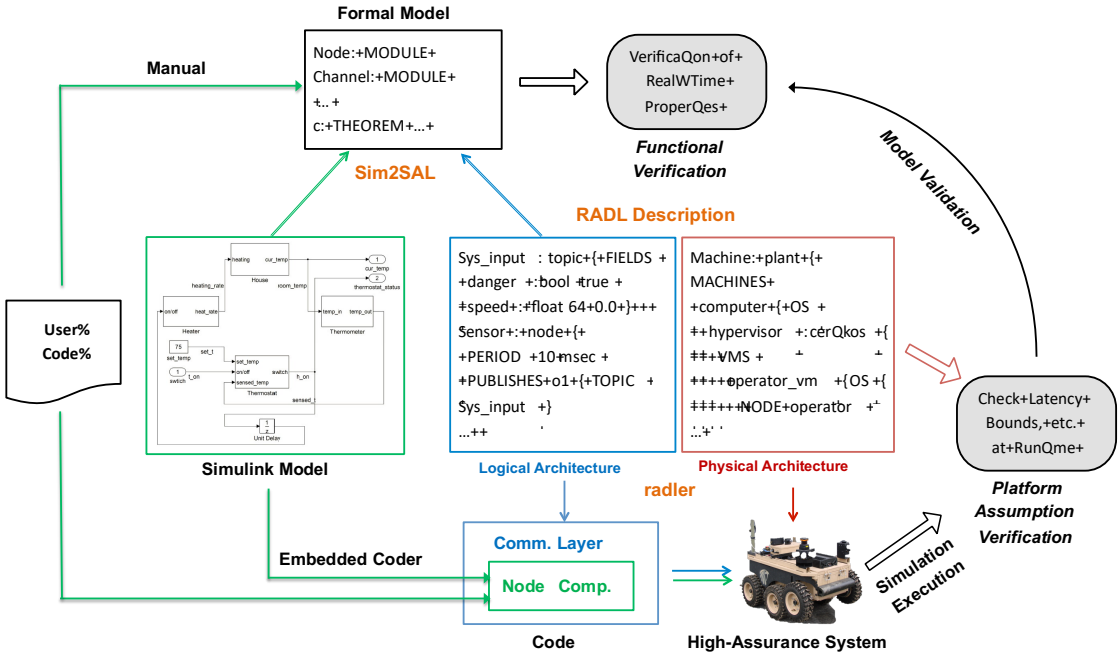

Figure 1: Overview of the RADL design and verification flow.

Design and Verification Flow. The RADL design and verification flow is depicted in Figure 1.

1. Capture of the logical architecture through the RADL architectural definition language.

2. Verification ofreal-time properties is based on calendar automata [7] and SMT-based infinite-state bounded model checking [8,9].

3. The RADLER build tool takes the validated logical architectural definition together with software components (in C, C++, or Simulink) as inputs, and generates executables based on a description of the underlying physical architecture.

4. The RADLER build tool also generates certain run-time health checks, which are conservative, light-weight, and locally computed at the level of individual nodes.

Engineering Support. Entalus supports clients in their development of resilient and secure products along the RADL design and verification flow by

1. Creating descriptions of logical and physical architectures
2. Verifying real-time and fault-tolerant system properties
3. Implementing and maintaining client-specific extensions to RADL

4. Architecting efficient assurance arguments for safe and resilient systems
5. Working out client-specific roadmaps for designing provably resilient multirate systems in clients' development environment and processes

Contact.

Harald Ruess
Principal Partner, Entalus LLC

email: harald.ruess@entalus.com
phone: +1 (941) 312-2144

References.

[0] Benveniste, Alberto. *Loosely Time-Triggered Architectures for Cyber-Physical Systems,* 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), 2010.

[1] https://github.com/ros2

[2] https://cfs.gsfc.nasa.gov/

[3] Li, Wenchao, Léonard, Gérard, and Natarajan Shankar. *Design and verification of multi-rate distributed systems.* 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), IEEE, 2015.

[4] Owre, Sam, and Natarajan Shankar. *Contract-Based Verification of Complex Time-Dependent Behaviors in Avionic Systems.* NASA Formal Methods. Vol. 9690. Springer, 2016.

[5] https://github.com/SRI-CSL/radler

[6] Larrieu, Robin, and Natarajan Shankar. *A framework for high-assurance quasi-synchronous systems.* 2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE). IEEE, 2014.

[7] Dutertre, Bruno, and Sorea, Maria. *Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata.* International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems. Springer, 2004.

[8] De Moura, Leonardo, Harald Ruess, and Maria Sorea. *Bounded Model Checking and Induction: From Refutation to Verification.* International Conference on Computer Aided Verification. Springer, 2003.

[9] De Moura, Leonardo, Harald Ruess, and Maria Sorea. *Lazy theorem proving for bounded model checking over infinite domains.* International Conference on Automated Deduction. Springer, 2002.

[10] https://www.dds-foundation.org/

[11] Kopetz, Helmut and Bauer. The time-triggered architecture. in Proceedings of the IEEE, vol. 91, no. 1, 2003.