International Journal on Software Tools for Technology Transfer (2023) 25:427–429
https://doi.org/10.1007/s10009-023-00721-2

GENERAL

Special Issue: RV 2020

# Introduction to the Special Issue on Runtime Verification

**Jyotirmoy Deshmukh**[1] · **Dejan Ničković**[2]

Accepted: 10 October 2023 / Published online: 2 November 2023


**Abstract**
Runtime verification (RV) refers to methods for formal reasoning about all aspects of the dynamic execution of systems, including hardware, software, and cyber-physical systems. RV includes techniques to assess and enforce correctness of a system against systemic bugs or extrinsic uncertainties. These methods are typically considered lightweight as they may not involve exhaustive verification or proofs, but they provide a higher level of rigor and versatility compared to conventional testing methods. This article introduces the extended versions of selected papers from the peer-reviewed proceedings of the 20th International Conference on Runtime Verification (RV 2020). RV 2020 was supposed to be held in Los Angeles, California, USA in July 2020, but was instead held virtually due to the global Covid-19 pandemic.

**Keywords** Runtime verification · Program instrumentation · Empirical abstractions · Spatial-temporal specifications

## 1 Introduction

Runtime verification (RV) [2] is a lightweight verification approach that includes methods for monitoring, analyzing, and steering the runtime behavior of software and hardware systems. RV provides an important set of tools to reason about system correctness, robustness, and reliability. Originating from the formal methods community, RV provides an additional level of rigor and effectiveness compared to traditional testing. By its focus on the individual system execution traces, RV complements formal verification with a more practical, pragmatic, and scalable view. RV can be during various system design phases for testing, verification, and debugging purposes, and after deployment for ensuring reliability, safety, and security, for providing fault containment and recovery, as well as online system repair.

The main applications of RV are:

- specification- and algorithm-based system monitoring
- enforcement of safety and security policies during the operation of the system,
- extracting quantitative and statistical information from traces,
- fault localization and explanation,
- mining specifications from traces, and
- trace visualization.

There are a plethora of technologies based on RV that support the above activities, including:

- generation of monitors from specifications,
- system and program instrumentation and trace logging,
- combination of static and dynamic analysis, and
- runtime enforcement.

## 2 RV conference

RV started as a workshop in 2001 with the aim to bring researchers and practitioners from academia and industry together to discuss the problems of monitoring and steering system executions using rigorous analysis techniques. Originally, there were two major questions surrounding the field of RV. The first question was whether the use of formal methods at runtime was a viable alternative or complement to the more traditional verification approaches, such as the model checking or the theorem proving. The second question was whether the use of formal methods to analyze system executions adds value compared to classical monitoring techniques used, for instance, in performance monitoring, distributed debugging, or supervisory control and data acquisition. With the development of the RV field, many other fundamental

✉ D. Ničković
Dejan.Nickovic@ait.ac.at

J. Deshmukh
jyotirmoy.deshmukh@usc.edu

[1] Department of Computer Science, University of Southern California, 941 Bloom Walk, Los Angeles, 90089, CA, USA

[2] Center for Digital Safety and Security, Austrian Institute of Technology, Giefinggasse 4, Vienna, 1210, Austria

and practical questions arose. To reflect this growing interest of the research and industrial communities in the analysis of systems at their runtime, RV became a conference in 2010.

RV 2020 was organized on October 6–9, 2020. The conference was originally planned to take place in Los Angeles, USA, but due to the CoViD-19 pandemic, we settled instead for a virtual event. The conference called for contributions on the following non-exhaustive topics of interest:

- specification languages for monitoring,
- monitor construction techniques,
- program instrumentation,
- logging, recording, and replay,
- combination of static and dynamic analysis,
- specification mining and machine learning over runtime traces,
- monitoring techniques for concurrent and distributed systems,
- runtime checking of privacy and security policies,
- metrics and statistical information gathering,
- program/system execution visualization,
- fault localization, containment, recovery, and repair, and
- dynamic type checking.

The proceedings of RV 2020 have been published in Springer's Lecture Notes in Computer Sceince series (cf. Deshmukh and Ničković [1]).

## 3 Selected papers

In the remainder of this article, we briefly present the contributions of the four papers included in this special issue and put them in a broader context of runtime verification research and trends.

The paper "Program Analysis Using Empirical Abstraction" by Ho et al. [3] uses program instrumentation and program execution analysis, two essential RV methods, and combines them with clustering and abstract interpretation to build a framework for program analysis using empirical abstractions. This unconventional paper explores and discovers new synergies between dynamic and static analysis. The authors provide both the theoretical foundations for the proposed methods and their implementation in a tool. The originality of the approach and the quality of the results were rewarded with the RV 2020 Best Paper Award.

The paper "Efficient and Expressive Bytecode-Level Instrumentation for Java Programs" by Soueidi et al., [6] addresses one of the fundamental aspects of runtime verification, the program instrumentation. More specifically, the paper presents the tool BISM for efficient instrumentation of Java programs at the bytecode level. The authors introduce an expressive instrumentation language that is aware of the program's control flow and allows modular instrumentation. BISM accommodates both novice and advanced users by enabling instrumentation with or without proficiency in bytecode. Finally, BISM admits both static instrumentation at build-time and dynamic instrumentation at load-time.

The paper "Concurrent Runtime Verification of Data Rich Events" by Shafiei et al., [5] addresses the problem of actor-based runtime verification. For that purpose, the paper introduces MESA (Message-based System Analysis), an open-source runtime verification tool for concurrent monitors that uses the actor model. MESA can be instantiated with different monitoring systems. In addition, it supports data slicing for individual monitors. The paper provides an extensive empirical study of MESA, in which the tool is used to monitor flights in the U.S. airspace from live data streams. The aim is to ensure that each flight follows its planned route. In particular, the paper demonstrates that the concurrent monitoring using data slicing can significantly increase monitoring efficiency.

Finally, the paper "MoonLight: A Lightweight Tool for Monitoring Spatio-Temporal Properties" by Nenzi et al. [4] addresses the problem of monitoring spatio-temporal specifications. The properties of mobile and spatially distributed interacting entities that evolve in time naturally arise in many application domains, including biology, Internet-of-Things, and smart cities. This paper presents MoonLight, a Java tool for runtime verification of specifications expressed in Spatio-Temporal Reach and Escape Logic, a formalism in which the spatial aspects are expressed using weighted graphs that describe topological configurations in which entities are arranged.

**Funding** Open access funding provided by AIT Austrian Institute of Technology GmbH.



## References

1. Deshmukh, J., Nickovic, D. (eds.): Runtime Verification – 20th International Conference, RV 2020 Los Angeles, CA, USA, October 6–9, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12399. Springer, Berlin (2020)
2. Falcone, Y., Havelund, K., Reger, G.: A tutorial on runtime verification. In: Engineering Dependable Software Systems, pp. 141–175 (2013)

3. Ho, V.M., Alvin, C., Lawson, J.D., Mukhopadhyay, S., Peterson, B.: Program analysis using empirical abstraction. Int. J. Softw. Tools Technol. Transf. (2023). (In this issue.)
4. Nenzi, L., Bartocci, E., Bortolussi, L., Silvetti, S., Moonlight, M.L.: A lightweight tool for monitoring spatio-temporal properties. Int. J. Softw. Tools Technol. Transf. (2023). (In this issue.)
5. Shafiei, N., Havelund, K., Mehlitz, P.: Concurrent runtime verification of data rich events. Int. J. Softw. Tools Technol. Transf. (2023). (In this issue.)
6. Soueidi, C., Monnier, M., Falcone, Y.: Efficient and expressive bytecode-level instrumentation for Java programs. Int. J. Softw. Tools Technol. Transf. (2023). (In this issue.)