CISTER

Research Center in

Real-Time & Embedded

Computing Systems

Conference Paper

Run-time Monitoring Architecture for Real-Time Systems

Geoffrey Nelissen

David Pereira

Luís Miguel Pinho

CISTER-TR-151207

Run-time Monitoring Architecture for Real-Time Systems

Geoffrey Nelissen, David Pereira, Luís Miguel Pinho

CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

Verification and testing are two of the most costly and time consuming steps during thedevelopment of safety critical systems. The advent of complex and sometimes partiallyunpredictable computing architectures such as multicore commercial-of-the-shelf platforms,together with the composable development approach adopted in multiple industrial domains suchas avionics and automotive, rendered the exhaustive testing of all situations that could potentiallybe encountered by the system once deployed on the field nearly impossible. Run-time verification(RV) is a promising solution to help accelerate the development of safety critical applicationswhilst maintaining the high degree of reliability required by such systems. RV adds monitors inthe application, which check at runtime if the system is behaving according to predefinedspecifications. In case of deviations from the specifications during the runtime, safeguardingmeasures can be triggered in order to keep the system and its environment in a safe state, as wellas potentially attempting to recover from the fault that caused the misbehaviour. In thiscommunication, we present a novel run-time monitoring architecture suited to safety critical applications.

Run-time Monitoring Architecture for Real-Time Systems

Geoffrey Nelissen, David Pereira, Luís Miguel Pinho
CISTER/INESC-TEC, Polytechnic Institute of Porto, Portugal
email: <grrpn, dmpre, lmp>@isep.ipp.pt

Abstract. *Verification and testing are two of the most costly and time consuming steps during the development of safety critical systems. The advent of complex and sometimes partially unpredictable computing architectures such as multicore commercial-of-the-shelf platforms, together with the composable development approach adopted in multiple industrial domains such as avionics and automotive, rendered the exhaustive testing of all situations that could potentially be encountered by the system once deployed on the field nearly impossible. Run-time verification (RV) is a promising solution to help accelerate the development of safety critical applications whilst maintaining the high degree of reliability required by such systems. RV adds monitors in the application, which check at runtime if the system is behaving according to predefined specifications. In case of deviations from the specifications during the runtime, safeguarding measures can be triggered in order to keep the system and its environment in a safe state, as well as potentially attempting to recover from the fault that caused the misbehaviour. In this communication, we present a novel run-time monitoring architecture suited to safety critical applications.*

Run-time verification (RV) [1,2] entails adding pieces of code called monitors to a running application. The monitors scrutinise the system behaviour and check if it respects associated specifications. The monitors can detect anomalies during the execution of the application. That information may be logged and back propagated to the system designer. It is therefore an effient solution to detect bugs and other deficiencies when a complete static verification of the system is not possible. By keeping the monitors running in the system after its deployment, run-time verification can also be used as a tool to increase the safety and reliability of systems during their operation, triggering counter-measures when anomalies are detected and hence acting as a safety net around the monitored application.

With the advent of more and more complex computing platforms (e.g., multicore processors, many-core accelerators, networks on chip, distributed systems interconnected with various communication networks) and the adoption of new computing paradigms to exploit the power of those architectures, verifying whether a system respects its functional (e.g., order of execution and validity of results) and extra-functional (e.g., timing constraints) specifications became a big challenge. Static verification (i.e., the formal proof that the system is correct) has proven limited either because of the state space explosion problem as in the case of approaches based on model checking, or simply due to theoretical limitations related to the expressivity and decidability of approaches based on deductive verification. Furthermore, ensuring the correctness of extra-functional properties before the system deployment is usually subject to a high degree of pessimism, essentially because the data that must be manipulated (e.g., execution time, inter-arrival time or response time) are almost always available only at run-time, and depend on specificities of the underlying hardware (e.g., communication protocols on shared buses, replacement policies in caches, operation ordering in execution pipelines), interactions with the external physical environment and interference caused by concurrent applications.

Testing and simulations are often presented as solutions to improve the confidence in the system. However, one can never ensure an exhaustive testing of all the possible situations that may be encountered after deployment. Therefore, successfully passing all the tests does not provide any guarantee that the system is bug-free or that it will always respect all its requirements. Furthermore, today's practices in product development rely extensively on distributed development. Multiple partners and subcontractors develop different functionalities that are later integrated together to form the final product. As a result, most of the components are black-boxes and none of the partners knows exactly how they all have actually been implemented. Therefore, ensuring an exhaustive testing and fully trusted verification of the system before its deployment becomes nearly impossible. Keeping monitors running together with the applications in order to detect potential misbehaviours and trigger safe-guarding measures should therefore be considered as a promising solution to accelerate the product development cycle whilst maintaining the high safety requirements associated to such systems.

Most of the state-of-the-art on run-time verification focuses on the design of formal languages [3, 4, 5, 6, 7] for the specification of properties that must be verified at run-time. Those languages are then used to generate monitors in a correct-by-construction manner. However, run-time verification cannot work without appropriate mechanisms to actually monitor the system and extract meaningful information that can then be used by the monitors to assert the respect of the specifications. This basic infrastructure over which any run-time verification framework is built has received much less attention from the research community. Run-time monitoring is however a corner stone of an efficient and safe run-time verification framework as it plays the interface between the monitors and the monitored applications. Previous works have developed run-time monitoring solutions as a part of full run-time verification frameworks [6,7,8]. Most of them though, put the accent on the specification language and the monitor generation process, and not the implementation and run-time monitoring aspect. In this communication we specifically focus on the run-time monitoring architecture and present a new, efficient and safe solution to integrate monitors with application code. The design of the presented run-time monitoring framework is perfectly suited to safety critical systems such as avionics, space, railway or automotive applications, as well as any other embedded system.

The presented solution has been designed in order to fulfil the requirements of safety critical systems. The architecture has been kept simple so as to ease its implementation in various run-time environments. A prototype, written in Ada, of this new reference architecture is presented and experimental results show that the overhead caused by the instrumentation of the code is constant w.r.t. the worst-case execution time of the tasks constituting the monitored application.

References

[1] Delgado, N., Gates, A.Q., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans. Softw. Eng. 30(12), 859{872 (Dec 2004)

[2] Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming 78(5), 293 - 303 (2009), the 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS07)

[3] Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for ltl and tltl. ACM Trans. Softw. Eng. Methodol. 20(4), 14:1-14:64 (Sep 2011)

[4] Konur, S.: A survey on temporal logics for specifying and verifying real-time systems. Front. Comput. Sci. 7(3), 370-403 (Jun 2013)

[5] Sen, K.: Generating optimal monitors for extended regular expressions. In: In Proc. of the 3rd Workshop on Runtime Verification (RV03), volume 89 of ENTCS. pp. 162-181. Elsevier Science (2003)

[6] Pedro, A, Pereira, D, Pinho, L, Pinto, J, "A Compositional Monitoring Framework for Hard Real-Time Systems", NASA Formal Methods Symposium 2014 (NFM14), Springer International Publishing. 29, Apr, 2014, LNCS 8430, pp 16-30. Houston, TX, U.S.A..

[7] Pedro, A, Pereira, D, Pinho, L, Pinto, J, "Monitoring for a decidable fragment of MTLD", 15th International Conference on Runtime Verification, September 22 – September 25, 2015 Vienna, Austria.

[8] Pedro, A, Pereira, D, Pinho, L, Pinto, J, "Towards a Runtime Verification Framework for the Ada Programming Language", Lecture Notes in Computer Science, Reliable Software Technologies – Ada-Europe 2014 (LNCS), Springer International Publishing. 23 to 27, Aug, 2014, 8454, pp 58-73.