Run-Time Monitoring Environments for Real-Time and Safety Critical Systems

Geoffrey Nelissen, Humberto Carvalho, David Pereira, Eduardo Tovar
CISTER/INESC TEC, ISEP
Polytechnic Institute of Porto
Porto, Portugal
Email: {grrpn, hjesc, dmrpe, emt}@isep.ipp.pt

***Abstract*—In this work, we present four different implementations of a run-time monitoring framework suited to real-time and safety critical systems. Two implementations are written in Ada and follow the Ravenscar profile, which make them particularly suited to the development of high integrity systems. The first version is available as a standalone library for Ada programs while the second has been integrated in the GNAT run-time environment and instruments the ORK+ micro-kernel. Information on the task scheduling events, directly originating from the kernel, can thus be used by the monitors to check if the system follows all its requirements. The third implementation is a standalone library written in C++ that can be used in any POSIX compliant run-time environment. It is therefore compatible with the vast majority of operating systems used in embedded systems. The last implementation is a loadable kernel module for Linux. It has for main advantage to be able to enforce complete space partitioning between the monitors and the monitored applications. It is therefore impossible for memory faults to propagate and corrupt the state of the monitors.**

I. Introduction

As a part of the development process of embedded systems, there is a need to verify that the functional and timing requirements defined in the system specifications will always be respected after the system deployment. This is even more important for safety critical systems, which must go through a thorough certification process. However, with the increasing complexity of embedded systems, it becomes always more complicated and sometimes impossible to statically verify offline that all the requirements will be respected at run-time. Specifically, with the advent of multicore processors, several new challenges arose: (i) the manufacturers sacrificed the determinism of their computing platforms to improve the average case performances, (ii) the number of applications running concurrently on the same processor and hence competing for the shared resources, is increased, (iii) the applications are becoming more complex and make use of intra-task parallelism to take advantage of the processing power offered by the several cores. Additionally, the integration of applications developed by different companies or development teams, the utilisation of legacy code and/or the lack of access to the source code of some of the executed functionalities, render the verification process even more complex.

Under such conditions, it becomes unrealistic to formally verify that all the system requirements will be respected under any possible execution scenario. The worst-case analyses that are usually performed before the system deployment are also based on a set of assumptions (e.g., minimum activation period, worst-case execution time, maximum release jitter) that may not always be respected at run-time. For all those reasons, run-time monitoring and run-time verification become an interesting alternative to the traditional offline verification. Run-time verification is based on the instrumentation of the target applications. Monitors are then added to the system to verify at run-time that the system requirements are respected during the execution. If a misbehaviour is detected, an alarm can be raised so as to trigger appropriate counter-measures (e.g., execution mode change, reset or deactivation of some of the functionalities).

Run-time monitoring and verification can be used during the system development phase to test and debug the applications. However, the monitors can also be left in the system after its deployment, in which case they play the role of a safety net, preventing the system to enter in an unexpected or dangerous state.

Safety related standards recommend the use of run-time monitoring and verification solutions in safety critical systems [1]–[3]. However, their use is not limited to safety critical applications. They can be very useful for the development of mission critical and business critical applications, or simply to improve the reliability of any embedded system.

II. Reference Architecture

In [4], a reference architecture for a safe and reliable run-time monitoring framework was proposed. This architecture is depicted on Figure 1. It is based on four main components: (i) event buffers in which events (i.e., a timestamp associated to a data) can be pushed by the instrumented application, (ii) event writers used by the monitored application to push events in the buffers, (iii) event readers that may be used by the monitors to access the events that are saved in the buffers, and (iv) monitors, implemented as periodic tasks, that read events and check that the application respect its specifications.

As shown on Figure 1, there can be only one writer per buffer. This avoids parallel accesses to the same buffer, which may have lead to unwanted blocking times. Thanks to this restriction, the writing operation in a buffer is wait free. There can however be more than one reader connected to the same

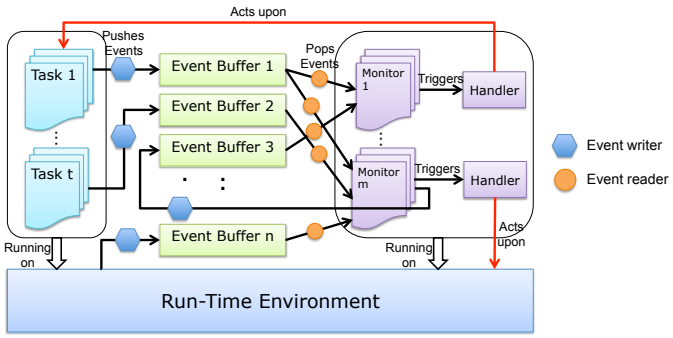

Fig. 1. Run-time monitoring reference architecture [4].

buffer, which allows several monitors to use the same events in parallel.

III. Implementations for Different Execution Environments

Four different implementations of the reference architecture proposed in [4] have been developed and should be presented during the demo session.

A. Ravenscar Compliant Ada Library

The first implementation is written in Ada, a programming language particularly suited to the development of critical applications. The library respects all the restrictions associated with the Ada Ravenscar profile [5]. The Ravenscar profile was defined to ensure timing predictability and hence ease the timing analysis of critical applications, by enforcing strict coding rules.

The developed library can be used in any application written in Ada (Ravenscar compliant or not). It provides all the needed facilities to instantiate monitors, buffers, buffer readers and writers discussed above.

B. Integration in the ORK+ Micro-Kernel

ORK+ is a Ravenscar compliant micro-kernel [6] implemented in Ada and integrated in the GNAT GPL 2011 compilation system developed by AdaCore. The kernel is packaged together with the compiler and the other libraries proposed by the GNAT runtime environment. ORK+ is currently one of the reference run-time environments in the ESA EagleEye reference mission [7] used for testing new technologies for future space applications.

The Ada library mentioned in the previous section was added to the GNAT runtime environment and has been used to instrument the ORK+ micro-kernel. This means that monitors can now have access to task scheduling related events extracted directly at the kernel level. Those events are saved in a set of predefined buffers that can be accessed by user-defined monitors.

C. POSIX Compliant C++ Library

The third implementation is written C++ and assumes a POSIX execution environment. It can thus be used in a vast majority of real-time operating systems available for embedded applications (e.g., Linux, RTEMS, ...). Similarly to the Ada library, the C++ version offers all the facilities required for the implementation of an efficient run-time verification framework compliant with the reference architecture described in Section II. Each monitor is encapsulated in a POSIX thread which is periodically executed.

D. Integration in Linux as a Kernel Module

In addition to the POSIX implementation, a loadable kernel module has been implemented for Linux. The major advantage of this Linux implementation is that it achieves total space partitioning. Indeed, the monitors can be instantiated in different processes than the monitored application. As each process runs within its own virtual sandbox, it is impossible for a monitor or a monitored application generating memory errors to corrupt monitors instantiated in other processes. Additionally, the buffers are living at the kernel level while the monitors and monitored applications are instantiated at the user level. Given that the event buffers are allocated in kernel memory, they cannot be corrupted, and will persist even if the monitored application crashes, thereby allowing the monitors to continue extracting critical information even when the system malfunctions. Finally, kernel land allows for finer-grained control over the hardware. The preemptions can thus be disabled during critical sections, guaranteeing wait-free read and write operations in the buffers.

IV. Demonstration

During the demo session, we will (i) show how applications running in the different execution environments described in the previous section can easily be instrumented, (ii) show how monitors can be implemented either for logging, for runtime verification purposes or for providing adaptive capabilities to the instrumented application, (iii) provide indications on the impact of the framework on the application performances.

Acknowledgments. This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0003/2012 - JU grant nr. 333053 (CONCERTO) and ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

References

[1] DO-178C, *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., 2011.
[2] ISO26262, *Road vehicles Functional safety*. ISO, 2011.
[3] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. ACM, 2015, pp. 139–148.
[4] G. Nelissen, D. Pereira, and L. M. Pinho, "A novel run-time monitoring architecture for safe and efficient inline monitoring," in *Reliable Software Technologies–Ada-Europe 2015*. Springer, 2015, pp. 66–82.
[5] A. Burns, B. Dobbing, and T. Vardanega, "Guide for the use of the ada ravenscar profile in high integrity systems," *ACM SIGAda Ada Letters*, vol. 24, no. 2, pp. 1–74, 2004.
[6] Universidad Politécnica de Madrid, "ORK+," 2014. [Online]. Available: http://www.dit.upm.es/ str/ork/index.html
[7] V. Bos, P. Mendham, P. K. Kauppinen, N. Holst, A. Crespo Lorente, M. Masmano, J. A. d. l. Puente Alfaro, and J. R. Zamorano Flores, "Time and space partitioning the eagleeye reference mission," 2013.