Programming adaptive real-time systems

Clément Ballabriga
Univ. Lille, France
clement.ballabriga@univ-lille1.fr

Julien Forget
Univ. Lille, France
julien.forget@univ-lille1.fr

Giuseppe Lipari
Univ. Lille, France
giuseppe.lipari@univ-lille1.fr

***Abstract*—In this panel proposal, we would like to discuss the programming of adaptive real-time systems. Here, the term *adapative* refers to a system capable of modifying its behaviour at run-time, so as to cope with variations (execution time variation, environment variation, ...) that are difficult, or even impossible, to predict. We would like to discuss such systems, not only from a timing analysis point-of-view, but also from a functional point-of-view, that is to say, how are such systems actually programmed?**

I. Motivation

There are mainly three sources of variation that may require to adapt the behaviour of a real-time system at run-time. First, task execution times often exhibit large variability; scheduling analysis has to be performed based on the worst-case execution-time, which is usually significantly more than the actual average execution time. Second, the complexity of modern hardware platforms makes it difficult to predict their behaviour. Third, the system physical environment is in many respects unpredictable, even though the engineer expertise may enable to make assumptions on the range of values that can be expected for the system inputs.

Real-time systems are usually programmed according to a rigid model: the system is implemented with a fixed set of tasks, with fixed timing parameters. The result is guaranteed safe, but usually this involves making conservative (pessimistic) hypotheses about software, hardware, and system environment characteristics that cannot be determined statically. Allowing system adaptation at run-time enables to significantly reduce these over-estimations.

II. Definitions

A real-time system is usually modelled as a set of *tasks* $\mathcal{T}$, where each task $\tau_i \in \mathcal{T}$ has a set of real-time attributes (T_i, C_i, D_i). T_i is the task period, or minimal inter-arrival time, C_i is its worst-case execution time and $D_i \leq T_i$ is its relative deadline [1]. An *adaptive real-time system* is a real-time system that can modify its task set $\mathcal{T}$ at run-time as follows:

- Some tasks are added to or removed from $\mathcal{T}$;
- The periods of some tasks in $\mathcal{T}$ are modified;
- The deadlines of some tasks in $\mathcal{T}$ are modified.

Note that we do not consider execution time variations as adaptations, since they do not correspond to modifications *decided* by the system itself, but rather witnessed by it.

III. Functional semantics of real-time adaptation

From a software point-of-view, handling variation through real-time adaptation is a complex problem [2]. First, monitoring mechanisms must be provided, e.g. to monitor execution time variations, either by the programming language or by the operating system. Second, an adaptation procedure that will be executed in case of variation must be provided. There are many possible ways to implement this. In the simplest case, we may afford to let the task continue as planned, if the variation has no negative impact on the system. However, assessing when to adapt the system at run-time can be a complex process. In other cases we may have to stop the current task execution. If the task is stopped, we may have to undo its side-effects or start a recovery task. The adaptation procedure can also require a more general reconfiguration of the system, creating new tasks or altering existing ones. Finally, we must plan for means to transition back to the system initial behaviour. This complex process is mostly ignored by existing approaches in the real-time research community, since system models generally abstract from the system functional behaviour.

IV. Focus of the panel

We propose a panel discussion that will focus on the software point-of-view of adaptive real-time systems programming, instead of the timing analysis point-of-view. We believe that this topic is currently not very well covered by the literature, and yet is of prime importance to the community. First, as mentioned in the previous section, because it is a non-trivial problem and second, because assumptions made at the real-time model level may be impracticable at the functional level. For example, the popular Mixed Criticality (MC) scheduling approach [3] assumes that low criticality tasks can be *dropped* when a high criticality task exceeds its initial execution time budget. However, a task that is dropped before completion must not produce side-effects, otherwise the functional correctness of the system may be compromised.

The choice of the design and programming tools is a central question here. Adaptive real-time systems can be programmed with existing low-level languages (like C for instance), however it might be difficult to design and analyze large programs with complex adaptive behaviours and still guarantee correctness. In this panel we would like to discuss higher level languages (synchronous languages, Simulink, etc), their benefits and limitations for modeling adaptive real-time systems.

V. Related Works

Several methods have been proposed to deal with large variability between worst-case and average case execution times. First, more precise task execution time models that take variability into account have been proposed. For instance, the probabilistic approach [4], [5] represents execution times using probability distribution functions. The parametric WCET approach [6], [7] represents execution times as formulas that depend on various parameters.

Mixed Criticality (MC) Scheduling [8] has been proposed as a way to deal with execution time variability, focusing on the scheduling problem. The main idea behind this technique is to change the *criticality mode* of the system upon detection of an extreme value of the execution time, and then execute only the high criticality tasks. The MC Scheduling Problem has received much attention from the research community in the last decade [3].

Scheduling analysis of systems with *mode changes* has also been studied in the past [9]. In such a multi-mode system, each mode is characterized by a different task set. This model shares similarities with the MC model, though here mode changes are driven by environment variations, rather than execution time variations in the MC model.

In the context of real-time scheduling, sensitivity analysis [10], [11] aims at determining acceptable deviations from the specifications of a problem. It consists in studying the consequences, in terms of deadline-misses, of deviations from the specified task real-time characteristics (WCET, deadline, period). This is an important topic since it can help determine under which conditions run-time error-prevention measures must be considered.

Since timing-faults may not only impact the task that caused the fault but also other tasks, real-time programmers usually rely on low-level asynchronous mechanisms to handle them. This includes classic POSIX event handling mechanisms, which can be used to plan a recovery procedure, should a WCET overrun occur. More complex asynchronous transfer of control mechanisms are available in Java [12], Ada [13] or Real-time Euclid [14], which enable the programmer to specify the behaviour of a function in case it gets interrupted due to an overrun occurring in a different function.

Synchronous languages [15] have successfully been applied to the implementation of real-time systems for many years now and language constructs such as mode automata [16] enable clear and clean specification of mode chances. However, these languages mostly abstract from real-time, focus on mono-periodic systems, and so provide little means to specify real-time characteristics and even less means to modify these characteristics dynamically. Some notable exceptions are the introduction of *Futures* in Lustre [17], which enables to use computations with unbounded execution durations in a synchronous program, and the language Prelude, which focuses on multi-periodic synchronous systems [18].

References

[1] J. Y.-T. Leung and M. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information Processing Letters*, vol. 11, no. 3, pp. 115 – 118, 1980.

[2] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*, 4th ed. USA: Addison-Wesley Educational Publishers Inc, 2009.

[3] A. Burns and R. Davis, "Mixed criticality systems: A review," Department of Computer Science, University of York, Tech. Rep., 2013.

[4] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla, "Measurement-Based Probabilistic Timing Analysis for Multi-path Programs," in *23nd Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 2012.

[5] A. Melani, E. Noulard, and L. Santinelli, "Learning from probabilities: Dependences within real-time systems," in *8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2013.

[6] S. Bygde and B. Lisper, "Towards an Automatic Parametric WCET Analysis," in *8th International Workshop on Worst-Case Execution Time Analysis (WCET'08)*, ser. OpenAccess Series in Informatics (OASIcs), R. Kirner, Ed., vol. 8. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2008. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2008/1659

[7] A. Colin and G. Bernat, "Scope-tree: A program representation for symbolic worst-case execution time analysis," in *14th Euromicro Conference on Real-Time Systems (ECRTS)*. Washington, DC, USA: IEEE Computer Society, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=787256.787341

[8] S. Vestal, "Preemptive Scheduling of Multi-Criticality Systems with Varying Degrees of Execution Time Assurance," in *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*, December 2007.

[9] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[10] L. George and P. Courbin, *Reconfiguration of Uniprocessor Sporadic Real-Time Systems: The Sensitivity Approach*. IGI Global, 2011.

[11] E. Bini, M. Di Natale, and G. Buttazzo, "Sensitivity analysis for fixed-priority real-time systems," *Real-Time Systems*, vol. 39, no. 1-3, pp. 5–30, 2008.

[12] A. Wellings, *Concurrent and Real-Time Programming in Java*. Wiley, 2004.

[13] A. Burns and A. Wellings, *Concurrent and Real-Time Programming in Ada*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[14] E. Kligerman and A. D. Stoyenko, "Real-time euclid: A language for reliable real-time systems," *IEEE Trans. Software Eng.*, vol. 12, no. 9, pp. 941–949, 1986.

[15] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, R. De Simone *et al.*, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[16] J.-L. Colaço, B. Pagano, and M. Pouzet, "A conservative extension of synchronous data-flow with state machines," in *Proceedings of the 5th ACM international conference on Embedded software*. ACM, 2005, pp. 173–182.

[17] A. Cohen, L. Gérard, and M. Pouzet, "Programming parallelism with futures in lustre," in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ser. EMSOFT '12. New York, NY, USA: ACM, 2012, pp. 197–206.

[18] J. Forget, F. Boniol, D. Lesens, and C. Pagetti, "A Real-Time Architecture Design Language for Multi-Rate Embedded Control Systems," in *25th ACM Symposium On Applied Computing*, Sierre, Switzerland, Mar. 2010.