Poster: Reusable software components to prototype and evaluate mixed-criticality scheduling policies

Alan Le Boudec*†, Hai Nam Tran†, Stéphane Rubini†, Alexandre Skrzyniarz*, Frank Singhoff†
* Thales DMS, Brest, France
† Lab-STICC, UMR CNRS 6285, University of Brest, France

I. Introduction

Nowadays, many functionalities are integrated in embedded real-time systems, leading to an increase in the number of their software and hardware components. In this context, the interest in mixed-criticality scheduling is growing [1].

A key feature of mixed-criticality systems (MCS), introduced by Vestal [2], is that task execution budgets depend on criticality levels. Under normal conditions, both low (LO) and high (HI) criticality tasks use optimistic budgets. In overloaded situations, the system switches to a critical mode where HI tasks rely on pessimistic budgets, and LO tasks are degraded or even stopped to ensure HI constraints. MCS scheduling must handle both task and system complex behaviors. Furthermore, each task may have various parameters depending on the system execution mode or their criticality. MCS scheduling policies handle much more task and system parameters compared to usual scheduling policies, which makes complex their implementation.

[Problem statement] Scheduling algorithms of MCS are typically evaluated by scheduling simulations. Such scheduling simulations are complex to implement. First because they need to handle more parameters than usual scheduling policies. Second, because many of those parameters have to be randomly generated. Unfortunately, there is no reusable tool that helps the implementation of MCS scheduling policies for evaluation purposes.

[Contributions] In this article, we propose reusable software components to ease the implementation of MCS scheduling simulators and guidelines to use such software.

II. Reusable software components

This section presents the reusable software components we have implemented inside the Cheddar tool [3] to help the community to evaluate their MCS scheduling policies by scheduling simulations. Cheddar is a real-time scheduling simulator written in Ada which contains various components required for scheduling simulations.

Figure 1 shows the 5 Cheddar components we have extended or adapted for MCS scheduling simulations. Component A generates random data of architecture models of the real-time systems to simulate. Components B and C are responsible for respectively writing or reading these models. Components D and E compute the scheduling simulations on the duration required by the analysed model.

To use these software components, we follow the flow of the Figure 1 from component A to E.

First, component A automatically generates the architectural models. These models contain both a description of the hardware platform and the task parameters. The component A proposes several means to generate task parameters. For example, the task execution time budgets are computed as the product of the period and the processor utilization rate. The period may be fixed or randomly generated. The processor utilization rate may be also obtained randomly from distribution laws, i.e. with a uniform law or UUnifast. The architecture models are then saved in an XML file by component B.

The following step involves reading the XML files. Component C read and parses these files. Then, the component D computes the feasibility interval [4] as well as simulation data. For example, the concept of computing budget involves a random execution time during the simulation, which must be computed according to a probability distribution law.

Finally, the scheduling analysis is performed using the generated simulation data and a given scheduling algorithm. The component E computes and produces scheduling analysis results. The metrics computed by component E are collected and saved in an XML file.

At completion, Cheddar provides metrics that are specific to MCS, such as the time spent in LO and HI modes, the number of overruns, and the number of interrupted tasks. It also computes classical metrics, including the number of missed deadlines and the worst-case response times.

III. Evaluation

The software components described above have been used to implement and simulate a MCS scheduling algorithm based on the concept of quality for Thales [5]. To implement this algorithm, we extended an object class of the source code of Cheddar to express the scheduling algorithm run by the simulator. We also adapted a Cheddar program to randomly generate the simulation parameters such as the feasibility interval and the task parameters (e.g. period, computing budgets, etc). Implementing this new MCS scheduling policy required about 350 lines of code.

By using the proposed software components, we benefit from various subprograms already existing in Cheddar: subprograms to generate random data needed for the simulation with Normal or Box-Muller distribution laws, subprograms to write and read architecture models, subprograms to drive

2325-1301/25/$31.00 ©2025 IEEE
DOI 10.1109/RTCSA66114.2025.00038

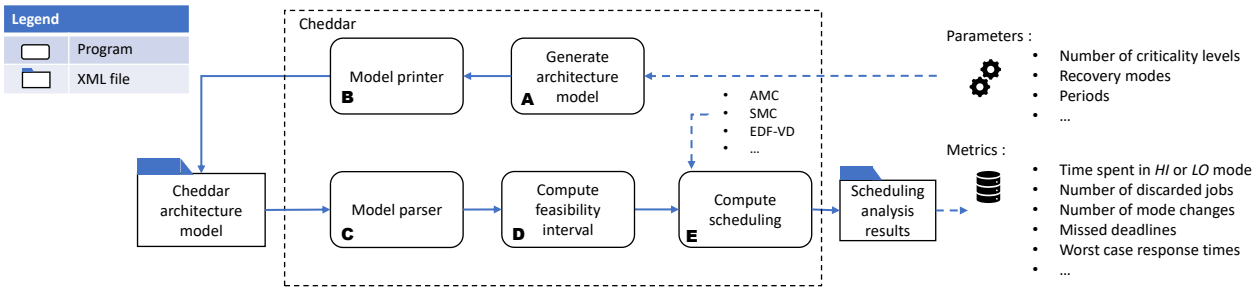

Fig. 1: Software components within Cheddar [3]

the scheduling simulations, and subprograms to automatically compute the metrics expected by the users from simulation results. These subprograms provided by Cheddar represent about 128 000 lines of code.

As a result, the effort to implement the MCS scheduling policy introduced in [5] represents only 0.3% of the overall simulator source code, showing a significant reusability of the simulator source code.

IV. Related work

Since the seminal Vestal publication [2], numerous authors have proposed MCS algorithms, including simulators [6]. Unfortunately, most of the time, these simulators are not designed to be re-used.

A simulator for MCS scheduling policies have to generate various random simulation data. Task parameters may be generated using real-life data [6], arbitrary values [7], or according to uniform distributions [8]. CPU utilizations are commonly derived using UUnifast [9], though simpler uniform laws are also used. A scheduling simulator has also to manage task execution time during simulation. Both uniform [6] and triangular distributions [10] have been proposed for such purpose. In our work, we propose to use the Box-Muller method, allowing the distribution to be adjusted according to mean and standard deviation.

Another important parameter to run scheduling simulations is the feasibility interval. Asyaban [11] proposes an estimate based on an interval corresponding to four times the length of the hyperperiod. Others simply propose a value supposed to be high enough, for example 10^{11} time units [6]. Our implementation benefits from Cheddar that provides various feasibility interval implementations. However, MCS implies that some tasks may not be executed, making it difficult to determine a feasibility interval that enables the full behaviour of the system to be covered.

To the best of our knowledge, no work offers reusable software components, integrated into a larger tool such as Cheddar.

V. Conclusion

Research in MCS has led to the exploration of various scheduling algorithms [1]. Most of the time, these algorithms are evaluated using scheduling simulations. However, to the best of our knowledge, authors usually implement ad-hoc simulators and no reusable tool is available for the community.

In this article, we propose software components to implement and evaluate MCS scheduling algorithms inside the Cheddar tool. We detail guidelines to implement MCS scheduling policies, the metrics computed by the simulator, and how to generate simulator parameters.

The proposed simulator components have been used to prototype a MCS algorithm based on the quality for Thales [5]. The effort to implement such algorithm represents only 0.3% of the Cheddar simulator source code[1], showing a significant reusability.

References

[1] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–37, 2017.

[2] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE international real-time systems symposium (RTSS 2007)*. IEEE, 2007, pp. 239–243.

[3] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *Proceedings of the 2004 annual ACM SIGAda international conference on Ada*, 2004, pp. 1–8.

[4] J. Goossens and R. Devillers, "Feasibility intervals for the deadline driven scheduler with arbitrary deadlines," in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306)*. IEEE, 1999, pp. 54–61.

[5] A. L. Boudec, H. N. Tran, S. Rubini, A. Skrzyniarz, and F. Singhoff, "q-amc: Integrating quality management in mixed criticality scheduling," in *Accepted in 30th International Conference on Emerging Technologies and Factory Automation, Porto, Portugal*. IEEE, September 2025.

[6] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 2015, pp. 259–268.

[7] H.-M. Huang, C. Gill, and C. Lu, "Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*. IEEE, 2012, pp. 23–32.

[8] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 147–152.

[9] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-time systems*, vol. 30, no. 1, pp. 129–154, 2005.

[10] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, 2012, pp. 155–165.

[11] S. Asyaban and M. Kargahi, "Feasibility interval for fixed-priority scheduling of mixed-criticality periodic tasks with offsets," *IEEE Embedded Systems Letters*, vol. 11, no. 1, pp. 17–20, 2018.

[1] Components available at http://beru.univ-brest.fr/cheddar