ORCA RT-Bench: A Reference Architecture for Real-Time Scheduling Simulators

Anderson R. P. Domingues[1,2], João Benno[1], Alexandre M. Amory[3], Fernando Gehm Moraes[1]
[1]*School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil*
[2]*Institute of Informatics, Federal University of Rio Grande do Sul – UFRGS – Porto Alegre, Brazil*
[3]*Scuola Universitaria Supeiore Pisa at Sant'anna – Pisa, Italy*
anderson.domingues@inf.ufrgs.br, joao.benno@edu.pucrs.br, alexandre.amory@santannapisa.it, fernando.moraes@pucrs.br

2021 XI Brazilian Symposium on Computing Systems Engineering (SBESC) | 978-1-6654-4311-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/SBESC53686.2021.9628369

***Abstract*—Real-time analysis is an ever-increasing branch of computer science whose study supports the development of peripheral areas such as embedded systems and robotics. As more and more approaches for real-time analysis emerge, it becomes challenging to choose among the variety of available tools. Such tools include both academic and industrial tools, mostly implementing just a small set of scheduling algorithms. This situation is aggravated when practitioners and academics must recondition such tools to meet specific purposes, e.g., implement new algorithms, as most tools present poor documentation or no support for their extension. In this work, we present a reference architecture for real-time scheduling simulators. The goal of our work is to accelerate the development of in-house scheduling simulators and teaching tools. By providing a set of models and a core implementation, we establish a framework from which both engineers and teachers can quickly implement and test scheduling algorithms without requiring entire operating system kernels or outdated tools.**

***Index Terms*—real-time systems, discrete-event simulation, software architecture**

I. Introduction and Motivation

Real-time systems are an ever-increasing branch of computer science. As such, it has delivered to the community a massive set of models, techniques, and tools to aid in the construction and analysis of the so-called real-time systems – systems in which deterministic behavior and predictability are core to their operation. One way of guaranteeing the timing behavior of tasks in a real-time system is through *scheduling*, in which an algorithm decides which task would occupy a determined resource such that all tasks fairly share that resource's time and none of them fail. Specifically for hard real-time applications, missing any deadline may jeopardize that application's operation. In some cases, a failing system may result in catastrophic outcomes such as financial loss, environmental damage, and risk to human lives.

A substantial amount of scheduling algorithms exist. As far as we know, there is no review or list to enumerate all algorithms. However, any quick search on the literature would bring up algorithms for dealing with uni-processed and multi-processed systems, soft- and hard-real time applications, static and dynamic algorithms, and multi-objective algorithms. Practitioners and academics may rely on simulators to support the development and study of scheduling algorithms. We call a scheduling simulator any tool capable of generating the schedule for a specific algorithm without requiring implementing a whole system kernel or operating system. The goal of a scheduler simulator is to reduce the complexity of experimenting with real-time operating systems, accelerating the study of scheduling algorithms.

We searched the literature for tools and frameworks that could be used to simulate scheduling algorithms. Although we found many tools, most of the tools are outdated or are not supported anymore. Most of the tools implement a restricted set of algorithms, providing no support for further extension. For instance, none of the tools provide documentation to support the implementation of new algorithms. More specific issues make some of the tools unusable in some scenarios, e.g., the lack of support for simulation traces, statistics, interruption emulation, and outdated technology/tool-chain.

A. Goals and Scope

This work aims to support academics and practitioners in developing, analyzing, and validating scheduling algorithms by providing a reference architecture (RA) [1] for scheduling simulators. Our RA is both a stand-alone simulator and framework whose components can be reprogrammed to form new simulators. Due to the variety of scheduling algorithms, we limit the scope of our RA to the simulation of algorithms targeting uni-processed systems. Additional goals of this work include:

- Accelerate the analysis of existing real-time systems by providing features to emulate specific characteristics of that systems. For instance, our RA permits accounting for scheduling time (the time taken by the scheduler to select the next task to run), interruptions, and system calls. We present the features of our simulator in Section IV.
- Provide a core implementation of a simulator for the canonical scheduling algorithms to help academic projects in graduate and undergraduate courses. We present the implemented algorithms in Section IV-E.
- Present a method for performance assessment of discrete event-based simulators, which could be used as part of other schedulers such as online schedulers [2, 3]. We present our performance model in Section IV-D.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior – Brasil (CAPES) – Finance Code 001, and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) grant 309605/2020-2.

We organize the rest of this paper as follows. In Section II, we highlight our findings when searching the literature for scheduling simulators. We present our reference architecture in Section III, briefly discussing each of its building blocks. Section IV presents ORCA-RT Bench, the tool that we implemented to validate our architecture, where we demonstrate some of the options to implement the components of our architecture. In Section V, we construct a performance model for our tool, taking into account the models in our reference architecture. Finally, we present our last considerations and future work in Section VI.

II. Related Work

We searched the literature for tools, frameworks, and reports on scheduling simulation. For instance, we found 12 simulators only by searching the term "CPU scheduling simulator" in GitHub. We gathered information on each project from their websites, reference papers, and "readme" files from repositories. We included in our research only open-source tools with documentation/paper/website written in English and directed to the simulation of uni-processed systems. Due to space limitations, we highlight only some of these tools below.

Yaashuwanth and Ramesh [4] propose an academic tool for teaching real-time scheduling, implementing a couple of scheduling algorithms, e.g., EDF (earliest deadline first), LLF (least laxity first), RM (rate monotonic), and DM (deadline monotonic). In contrast to our environment, their tool relies exclusively on a GUI to interact with the user. Our environment is intended to provide programmers and students with an insight into the internal structures of a real-time scheduler. Thus, we provide an uncoupled, stand-alone back-end program.

Casile et al. [5] propose a tool for distributed real-time systems. Similar to our work, they provide trace files as output, a GUI arrangement for displaying simulation steps, and an uncoupled backend. However, they do not present any analysis on simulation performance, nor do they report the information necessary to characterize applications. We provide a model for assessing simulation performance, allowing our tool to be deployed as a component of a larger system.

The RTSim tool [6] is a simulator based on *metasim*, an discrete-event simulation library. They provide a GUI tool for visualizing simulation traces, called RTTracer. This environment is very similar to ours, as our simulator has its built-in discrete-event module, which is a modification of another engine used in a previous work [7]. For the GUI, our tool generates output for ORB_KProfiller and KProfiler tools, both similar to RTTracer. RTSim website has not been updated since 2011, and the last public version of their tool was released in 2007. We could not find any documentation supporting the extension of the tool.

Manacero et al. [8, 9] proposed RTsim (lower case S), another academic simulator. Their website reports that the simulator supports RM scheduling for single processors and other multiprocessor systems algorithms. However, we could not evaluate the simulator as the download link is broken.

III. A Reference Architecture for Scheduling Simulators

The proposed reference architecture relies on five building blocks, which are sufficient to represent the implementation of any scheduling simulator: (*i*) event model, (*ii*) system model, (*iii*) simulation model, (*iv*) scheduling algorithm, and (*v*) performance model. The event model can be further split into task model and interruption model, the latter being optional. The performance model is optional as well. We dedicate the remainder of this section to explain each of the building blocks, depicted in Figure 1.

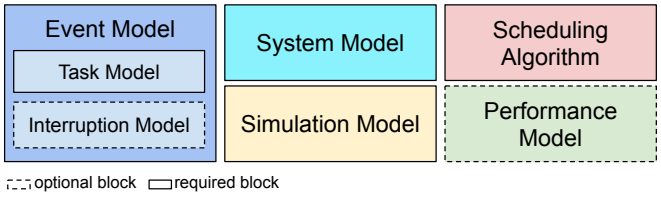

Fig. 1. Building blocks of the proposed reference architecture, comprising optional and mandatory blocks.

A. Event Model

Our proposed RA uses event-driven simulation to mimic the behavior of real-time schedulers. By doing so, we must describe the behavior of a real scheduler in terms of events. In our RA, specifically, we are interested in two kinds of events. First, we look into events triggered by tasks, the *system calls*. The second kind of event relates to *interruptions*. We observed that two events are necessary for scheduling simulation: (*i*) `TASK_END`, a system call that indicates that the executing task has finished; and (*ii*) `IRQ_SCHED`, a periodic interruption that calls the scheduler at the end of each time slice (the time given by the scheduler to the execution of tasks). System calls and interruptions become events in the simulation.

1) Task Model: The task model corresponds to the representation of tasks within the simulator. We assume that task models include the necessary information for the simulation of the `TASK_END` system call, that is, that must be possible to calculate the time in which the task ends. Note that the representation of tasks is a requirement of the algorithms instead of a requirement of the simulator. Other system calls can be included in the simulator. For example, if modeling for resource-aware scheduling [10], a system calls such as `TASK_SLEEP` may come in hand, permitting other tasks to run until the required resources are available.

2) Interruption Model: The interruption model corresponds to the emulation of interruption events and their characterization. The goal of the scheduler is to release tasks so that tasks will not miss their deadline. The scheduler often rely on interruption. For instance, the scheduling interruption `IRQ_SCHED` dictates the periodic call to the scheduler. Other interruptions may be required if considering hardware interruption, e.g. `IRQ_IN` and `IRQ_OUT` (synchronous read/write to peripherals).

B. System Model and Scheduling Algorithm

The system model corresponds to the control of states of tasks within real systems and is often implemented through lists of tasks. Our system model assumes a system to be a set of interconnected lists of tasks $Q = \{q_1, q_2, ...q_n\}$, where $|Q| \geq 2$ always holds because the system must distinguish between running and idle tasks. More lists can be added as necessary. A graph $G = V \times E$ represents the transition system (movement of tasks between lists), where E is the set of edges and V is the set of vertices. Finally, the scheduling algorithm (function) $\Phi : Q \rightarrow Q$ sorts one of the lists such that the next task to execute is on top.

C. Simulation Model

The simulation model is the method used by the simulator to generate the simulation trace. Simulation trace is the result of a simulation session. It usually includes the name of events and annotations indicating the time they would occur. For uni-processor systems, no two events can occur at the same time. Consequently, there must be an order of precedence that determines which event executes first in case of a conflict — failing in resolving conflicts between events results in non-determinism.

D. Performance Model

A performance model is a tool for achieving the assessment of simulation performance. This model must include the timing evaluation of events, as well as the scheduling algorithm. The goal of the performance model is to predict how much time the simulation will take and assert simulation determinism.

IV. ORCA RT-Bench Tool

ORCA RT-Bench[1] is an open-source scheduling simulator tool written in C++ and developed to validate our reference architecture. For simplicity, we limit the tool's scope to the simulation of hard real-time applications in uni-processed systems.

A. Task Model

We assume an application be a set of hard-real time, independent, periodic tasks $T = \{t_0, ...t_n\}$ running in a uni-processed system. Each task is a 3-tuple $t_i =< p, c, d >$, where p is the period, c is the capacity, and d is the deadline, all them expressed in terms of discrete time units (u). This model is a simplification over other models presented in books [11]–[13] and implemented within the Linux kernel [14].

Our tool implements a file parser in which applications are described as directed graphs. Each node represents one task, which we label with the information required by our task model (period, capacity, and deadline), added to some extra information (e.g., task name). Edges are unused in this work. However, we are aware of approaches that consider task dependency and real-time communication [15], so we keep this structure for future use. Our parser also allows for more information to be added to the file without much effort. Figure 2 shows an example of an application description file.

[1]https://github.com/andersondomingues/orca-rt-bench

```
1  [nodes]
2  #id task   capacity   deadline   period
3  01  T1     1          4          4
4  02  T2     2          5          5
5  03  T3     5          20         20
```

Fig. 2. Example of an application description file depicting an application taken from [12].

B. System Model

The system model corresponds to a set of lists of tasks, and a transition system coordinates the movement of tasks between lists. Our simulator implements three lists: (*i*) running, (*ii*) ready, and (*iii*) blocked, as shown in Figure 3. The behavior of the transitioning system assumes task preemption and is briefly discussed below.

1) Scheduling: The *running* list stores the only executing task in the system if any. Every time a task leaves the running list, another one (can be the same task) must take its place. The scheduling algorithm has to sort the ready list and move one task from the ready list to the running list. If no suitable task exists, an idle task is artificially introduced until at least one task arrives at the ready list.

2) Preemption: The running task resides in the running list until the next scheduling event (in real systems, determined by a scheduling interruption) or until it finishes, whichever happens first. If the scheduling interruption happens before the task end, that task is moved back to the ready list.

3) Sleeping: If the running task ends before the scheduling interruption, that task is moved on the blocked list.

4) Awakening: Tasks in the blocked list move to the ready list as soon as they reach their next period window.

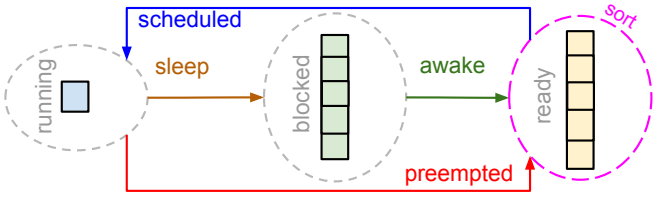

Fig. 3. System model implemented as an interconnected list system. Arrows indicate the direction in which tasks can move between lists. The sorting algorithm (scheduling) applies only to the ready list.

C. Simulation Model

The implemented simulation model relies on the discrete-event simulation [7, 16, 17]. In this model, events are pushed into a list and sorted by their release time (priority queue). Events occur instantaneously, allowing the simulation to skip the remaining simulation time until the next event. For this reason, simulating a discrete-event system is indeed faster than executing any operating system kernel or scheduler. We represent events as 3-tuples $e =< t, f, T >$, where t is the release time of that event, f is an activation function, and T is a period increment. For periodic events, T adds to t at

each successive release of e (period). The activation function $f : L \rightarrow L$ corresponds to the movement of tasks in the system model, where L is the set of all lists in that model.

A simulation (1) is represented by a 5-tuple, where t_0 is the initial simulation time (usually equals to zero), n is the maximum simulation time, P is a priority-queue of events, E is the set of all events, and g is the computation function. The next state of the simulation can be achieved by applying g over the current state. Successive applications of g generate a trace of computation steps, ending when $t_m \geq n$, at the m^{th} step. We have applied a similar simulation model in our previous work [7].

$$SIM =< t_0, n, P, E, g > \mid SIM' = g(SIM) \qquad (1)$$

The simulation engine, a component of our simulator, implements the g function as an algorithm, which takes the current state of the simulation as input and updates that state, generating the next state. This operation repeats until the end of the simulation.

D. Performance Model

Our performance model (2) can estimate the time taken to simulate a task set for a given hyper-period. Our estimation relies on the fact that we can predict the effort of the simulation model if we know the task set *a priori*. We must calculate the number of events to be simulated for that task set and multiply the resulting value for the effort of simulating a single event. In this case, our simulator must be deterministic, and the effort to simulate one event must be constant.

$$(O(Sort_n) + k) \times \left(\sum_{i}^{n} \frac{HP}{P_i} + \frac{HP}{time\ slice} \right) \qquad (2)$$

Our simulator implements a routine that generates a trace of computation steps for a given initial simulation state. In that routine, the effort to simulate one event is constant (k), corresponding to moving the running task from the running list to another list and moving one task from the ready list to the running list. Before moving the one task back to the running list, the scheduling algorithm sorts the ready list. The cost of sorting the ready list is bound to the execution of the underlying sorting algorithm, which we denote $O(Sort_n)$, where n is the number of tasks in the ready list. A conservative approach would take n as the number of tasks in the whole system, which is always greater than the size of the ready list. In a few words, the effort of calling the scheduler once is given by $O(Sort_n) + k$.

The last component of our performance model corresponds to the number of simulated events, that is, the number of calls to the scheduler during the hyper-period (HP), which corresponds to the "*the smallest interval of time after which the periodic patterns of all tasks is repeated*" [18]. The number of events must be at least $\frac{HP}{time\ slice}$, as the simulator calls the scheduler at the end of each time slice (`IRQ_SCHED`). Finally, we must consider the invocation of the scheduler at the end of each task, equals to $\sum_{i}^{n} \frac{HP}{P_i}$, where n is the number of tasks in the model and P_i is the period of the i^{th} task.

E. Scheduling Algorithms and Other Features

Our tool implements the following scheduling algorithms: Deadline Monotonic (DM), Earliest Deadline First (EDF), Least Laxity First (LLF), Least Slack Time (LST), and Rate Monotonic (RM) [12, 19]. One may select the algorithm by entering the corresponding acronym of the algorithm as a parameter, e.g., `-EDF` selects the Earliest Deadline First algorithm. At the startup, the simulation engine selects the proper scheduling algorithms, that is, the algorithm sorting the ready list.

F. Application Interface

Due to performance reasons, our simulator runs in console mode, producing a *trace file* as output. The trace file stores a list of events produced during the simulation, indicating the time in which tasks enter and leave the running list and their absolute deadline. Our trace file can serve as input for two visualization tools: KProfiler[2] and ORB_KProfiller[3]. The former is a tool for visualizing system events for the HellfireOS operating system. Calling our tool with the `-kprofiler` parameter format the trace file to match the input of KProfiler. The latter, ORB_KProfiller, is a front end to our tool, capable of interactively generate information on the simulation, e.g., number of missed deadlines, schedulability tests, multiple simulation charts. Figure 4 displays ORB_Kprofiller. Figure 5 shows an example of trace file.

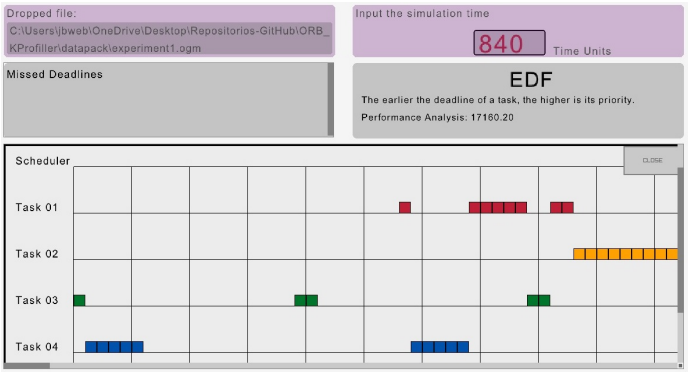

Fig. 4. ORB_Kprofiller interface depicting a simulation trace. Horizontal axis represents time, while vertical axis represents tasks. Performance estimation result (pink) and schedulability test (gray) are shown at the right-upper corner.

V. Performance Model Validation

This Section validates the performance model presented in Section IV-D. The goal is to assert simulation determinism, which is key for applications such online-scheduling (simulator as part of another scheduler), and edge computing (simulation is performed at the processing node). We simulated seven applications, 30 times each, collecting their execution

[2]https://github.com/sjohann81/hellfireos/tree/master/usr/kprofiler
[3]https://github.com/bennoXav/ORB_KProfiller

	#id	#adl	#start	#end
2	1	30	1	2
3	1	30	2	3
4	1	30	3	4
5	1	30	4	5
6	2	35	5	6

Fig. 5. Trace file reporting the result of a simulation. The id field matches the one in the input file. Other fields are presented in discrete time units.

time. Table I shows the characterization of applications, the simulated hyper-period, and their mean execution time. The hyper-period is given by $HP = lcm(T_p)$, where T_p is the set of all periods of all tasks in application T, and lcm is the least common multiplier function. We arbitrarily choose EDF as the scheduling algorithm for the experiment, and the value of $O(Sort_n)$ depends only on the number of tasks of each application.

To eliminate noise during the experiment, we configured the simulator to collect the time tags (in milliseconds) in which the first and last events left the event queue for each run. Subtracting both tags give us the amount of time spent by the simulator to process the simulation events, ignoring file manipulation and startup routines in the process.

TABLE I
CHARACTERIZATION OF APPLICATIONS A TO G

App.[1]	Task	Period	Cp.[2]	HP[3]	Mean Exec. Time (ms)
A	T1	90	1	1260	3.406
	T2	4	2		
	T3	21	5		
B	T1	4	1	1540	8.438
	T2	14	2		
	T3	28	7		
	T4	10	1		
	T5	44	11		
C	T1	10	2	3600	11.689
	T2	12	2		
	T3	16	2		
	T4	18	2		
	T5	20	2		
	T6	200	2		
D	T1	90	10	6300	42.372
	T2	60	12		
	T3	105	19		
	T4	50	25		
	T5	150	5		
E	T1	30	5	50400	218.885
	T2	35	9		
	T3	45	15		
	T4	100	10		
	T5	800	40		
F	T1	24	8	840	3.737
	T2	30	10		
	T3	7	2		
G	T1	64	8	960	4.192
	T2	80	10		
	T3	20	2		
	T4	30	5		
	T5	60	20		

[1]application, [2]capacity, [3]hyper-period
*period, capacity and hyper-period expressed in discrete time units (u)

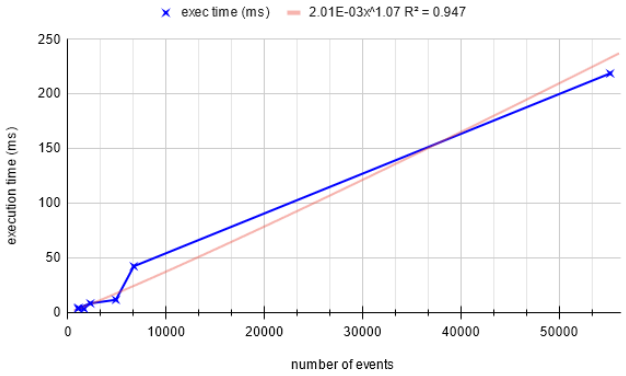

Fig. 6. Approximation of the collected data to a power series function. Points are sorted by execution time (ascending).

A. Results

As discussed in Section IV-D, the performance of our simulator is bound to the number of simulated events. When simulating shorter time slices, the number of calls to the scheduler (`IRQ_SCHED`) increases. Consequently, the number of events to be simulated increases, degrading the performance of the simulation. The worst performance is achieved when time slice is $1u$. For the experiment, we applied a normalized time slice of $1u$ for all applications.

Execution time increases linearly to the number of events. This is true as long the number of tasks remains the same. The time taken to simulate a single event depends only on the number of tasks in the ready list. We observe that the ratio $\frac{ET}{HP}$ grows linearly on the number of events, where ET is the execution time of the simulation. Figures 6 and 7 show the approximation of collected data to a power series and linear functions, respectively. The average time to simulate one event roughly approximates 0.00448ms. Please note that this time is bound to the performance of the machine in which the experiments were executed, as well as the configuration of the installed tool-chain and compilation scripts.

The ratio $\frac{ET}{e}$, where e is the number of simulated events, presents the same linear behavior as the $\frac{ET}{HP}$ ratio. In this case, for a constant number of events, the time to simulate an application grows linearly to the number of tasks. This is true as long as the time slice is $1u$ and $HP = e$. We omit the charts as they would be trivially similar to the ones shown in Figures 6 and 7.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented a reference architecture for scheduling simulators for uni-processed systems. Our contributions include the building blocks of our architecture and its models. To validate our RA, we developed a simulation tool named ORCA RT-Bench. The tool implements the building blocks of our architecture. From the results, we showed that our performance model holds for our tool. Below we enumerate some research opportunities and further improvement of the proposed RA.

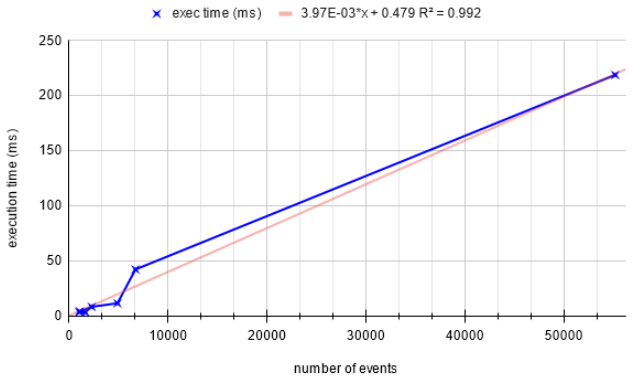

Fig. 7. Approximation of the collected data to a linear function. Points are sorted by execution time (ascending).

A. Future Work

Our RA can be further extended to support other task models. However, we did not develop an interface to support such a feature. In the future, we intend to create a meta-model where multiple task models can co-exist within the tool, supplying a larger class of applications, including heterogeneous systems.

The performance of our tool takes into consideration the execution of the tool in a single-threaded environment. We intend to extend our tool to simulate a single system using multiple processors, which we achieved in the past for non real-time systems [7]. We also intend to modify our performance model to support the analysis of multi-threaded simulation.

We intend to improve the usability features of ORB_Kprofiller, adding interactive simulation (e.g., pause and recording), system-level sensing emulation (for resource-aware simulation), and simulation statistics (e.g., energy characterization and consumption estimation) to the tool. We also intend to study the applicability of the tool with undergraduate students.

Finally, our RA is directed to the simulation of scheduling algorithms for uni-processed systems. It is of our interest to support distributed scheduling in multi-processed systems in the future. Other potential features include kernel time emulation and dynamic application admission in the system.

REFERENCES

[1] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory and Practice*. Addison-Wesley, 2007, p. 119.

[2] B. J. Coleman, "Lookahead scheduling in a real-time context: Models, algorithms, and analysis," Ph.D. dissertation, College of William & Mary – Arts & Sciences, 2004.

[3] B. DasGupta and M. A. Palis, "Online real-time preemptive scheduling of jobs with deadlines," in *Approximation Algorithms for Combinatorial Optimization*. Springer Berlin Heidelberg, 2000, pp. 96–107.

[4] C. Yaashuwanth and R. Ramesh, "Web-enabled framework for real-time scheduler simulator (a teaching tool)," in *International Conference on Computer Research and Development*, 2010, pp. 826–830.

[5] A. Casile, G. Buttazzo, G. Lamastra, and G. Lipari, "A scheduling simulator for real-time distributed system," *IFAC Workshop on Distributed Computer Control Systems*, vol. 31, no. 32, pp. 161–167, 1998.

[6] RETIS Lab, "RTSim Home Page." [Online]. Available: http://rtsim.sssup.it/

[7] A. R. P. Domingues, "Orca: A self-adaptive, multiprocessor system-on-chip platform," Master's thesis, School of Technology, PUCRS, 2020.

[8] A. Manacero, M. B. Miola, and V. A. Nabuco, "Teaching real-time with a scheduler simulator," *Frontiers in Education Conference. Impact on Engineering and Science Education*, pp. 15–19, 2001.

[9] Parallel and Distributed Systems Lab (GSPD) at Universidade Estadual Paulista (UNESP), ":: RTsim Homepage." [Online]. Available: https://www.dcce.ibilce.unesp.br/spd/rtsim/english/index.php

[10] M. Tillenius, E. Larsson, R. M. Badia, and X. Martorell, "Resource-aware task scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 1, Jan. 2015.

[11] R. Williams, Ed., *Real-Time Systems Development*. Oxford: Butterworth-Heinemann, 2006.

[12] P. A. Laplante and S. J. Ovaska, *Real-Time Operating Systems*. John Wiley & Sons, Ltd, 2011, ch. 1-3, pp. 79–147.

[13] A. S. Berger, Ed., *Debugging Embedded and Real-Time Systems*. Newnes, 2020.

[14] M. Kerrisk, "sched(7) — linux manual page." [Online]. Available: https://man7.org/linux/man-pages/man7/sched.7.html

[15] G. Al-Kadi and A. S. Terechko, "A hardware task scheduler for embedded video processing," in *High Performance Embedded Architectures and Compilers*, A. Seznec, J. Emer, M. O'Boyle, M. Martonosi, and T. Ungerer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 140–152.

[16] G. S. Fishman, *Discrete-Event Simulation – Modeling, Programming and Analysis*, 1st ed. Springer Science+Business Media New York, 2001.

[17] E. Kofman, A. Muzu, and B. Zeigler, *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*, 3rd ed. Elsevier Academic Press, 2019.

[18] I. Ripoll and R. Ballester-Ripoll, "Period selection for minimal hyperperiod in periodic task systems," *IEEE Transactions on Computers*, vol. 62, no. 09, pp. 1813–1822, sep 2013.

[19] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. Springer Publishing Company, Incorporated, 2011.