A Multiprocessor Real-Time Scheduling Simulation Tool

Arnoldo Díaz-Ramírez, Dulce K. Orduño
Department of Computer Systems
Instituto Tecnológico de Mexicali
Av. Tecnológico s/n Col. Elías Calles
Mexicali, B.C., México 21376
{adiaz, korduno}@itmexicali.edu.mx

Pedro Mejía-Alvarez
Department of Computer Sciences
CINVESTAV-IPN
Av. Instituto Politécnico Nacional 2508, Col. Zacatenco
México, D.F. 07360
pmalvarez@delta.cs.cinvestav.mx

***Abstract*—Real-time systems have relied on multiprocessor architectures since some time ago. However, the proliferation of multi-core architectures, which have multiple processing units on a single chip, has increased the interest on such systems. As a consequence, scheduling techniques for multiprocessor architectures have received considerable attention in recent years. In the last decade, a lot of scheduling algorithms have been proposed. In many cases, the least upper bound for these algorithms is about 50% of the whole system utilization, which is very pessimistic. One alternative to verify the schedulability of a real-time task set with a larger system utilization on a multiprocessor system, is through the use of exhaustive simulation. In this paper, we introduce *RealtssMP*, a tool to perform scheduling analysis and simulation of multiprocessor real-time scheduling algorithms. The proposed tool has shown to be helpful in the evaluation of the performance of existing and new scheduling algorithms. Moreover, it can be be used as an educational tool.**

***Index Terms*—real-time scheduling, real-time multiprocessor scheduling, real-time scheduling simulation**

I. Introduction

A real-time system is defined by Burns and Wellings as an information processing system which has to respond to externally generated input stimuli within a finite and specified period: the correctness depends not only on the logical result but also on the time it was delivered; the failure to respond is as bad as the wrong response [11]. Examples of real-time systems include digital control, command and control, signal processing, and telecommunication systems [26]. A real-time system is comprised of several tasks. Most of the real-time requirements can be expressed onto the tasks deadlines, arrival times and execution times. The use of efficient scheduling algorithms, supported by accurate schedulability analysis techniques, is required to guarantee the accomplishment of the temporal constraints of the tasks in the system.

While the scheduling problem for uniprocessor systems has been widely investigated for decades, there are still many open problems regarding the schedulability analysis of multiprocessor systems. As pointed out by Liu in his seminal paper [25], few of the results obtained for a single processor generalize directly to the multiple processor case: bringing in additional processors adds a new dimension to the scheduling problem.

The approaches for multiprocessor scheduling can be categorized in *partitioned* and *global*. In the partitioned scheme, each task is allocated to a single processor, dividing the multiprocessor scheduling problem into a *task allocation problem* and a *uniprocessor scheduling problem*. In contrast, the global scheme allows the tasks to migrate from one processor to another at run-time. The scheduling algorithms for both approaches can be further categorized into three classes according to the way the priorities can change: *fixed task-priority*, *fixed job-priority* and *dynamic priority* [17].

For uniprocessor scheduling, the *Earliest Deadline First* algorithm is optimal for fixed-job priority. In the case of multiprocessor scheduling, a number of global dynamic scheduling algorithms are known to be optimal (*Pfair* scheduling and its variants). However, since they introduce excessive overhead caused by frequent preemption and migration, their implementation can be impractical. For multiprocessor scheduling algorithms based on fixed task-priority and fixed job-priority, the maximum possible utilization bound is of about 50%, which is very pessimistic.

To investigate the schedulability of a real-time task set on a multiprocessor system, a schedulabilty test may be used. However, for task sets with a large system utilization, one alternative is given by the use of exhaustive simulation. In this paper, we introduce *RealtssMP*, a tool to perform scheduling analysis and simulation of multiprocessor real-time scheduling algorithms. The proposed tool has shown to be helpful in the evaluation of the performance of existing and new scheduling algorithms. Moreover, it can be be used as an educational tool.

The document is organized as follows. In section II we briefly introduce real-time multiprocessor scheduling notation and theory. In section III we discuss the related work. Section IV introduces *RealtssMP*, our simulation and analysis tool. Finally, the conclusions and future work are described in Section V.

II. Real-Time Multiprocessor Scheduling

The purpose of the multiprocessor real-time scheduling is to execute a set of n real-time tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ on a set of m processors $P = \{P_1, P_2, \ldots, P_m\}$. A task is usually a thread or a process within an operating system. The parameters that define a task are: its execution time C_i, its period T_i, and its deadline D_i. The majority of the real-time scheduling theory focuses on the *periodic* and *sporadic* task models. In both models, each task is composed of an infinite sequence of instances called *jobs*. The period T_i of a periodic task τ_i is a fixed time interval between release times of consecutive jobs in τ_i. In contrast, the period of a sporadic task τ_i represents the minimum inter-arrival time between consecutive jobs. The task execution time C_i is the maximum execution time of all the jobs of τ_i. Any job in τ_i that is released at time *t* must complete at most D_i time units after *t*; that is, it must complete within the interval $(t, t+D_i]$. We use $H(\tau)$ to denote the least common multiple of T_i, for *i = 1, 2,..., n*. A time interval of length $H(\tau)$ is called the *hyperperiod* of the task set τ.

The utilization factor of task τ_i is defined as $u_i = {C_i}/{T_i}$. The utilization factor of a task set τ is the sum of the utilization of each task in τ; that is, $U = \sum_{i=1}^{n} {C_i}/{T_i}$. The multiprocessor utilization U_s of a task set executing on a multiprocessor system comprised of *m* processors is defined as $U_s = {U}/{m}$.

The task deadlines can be classified as *implicit*, *constrained* and *arbitrary*. If the task deadlines are implicit, they are equal to their periods ($D_i = T_i$). If the task deadlines are constrained, they are less than or equal to their periods ($D_i \leq T_i$). If the task deadlines are arbitrary, they may be less than, equal to, or greater than their periods.

There are two problems that the multiprocessor scheduling is aimed to solve: the *allocation* problem and the *priority* problem [17]. In the allocation problem, the scheduler must decide on which processor a task should execute. In the priority problem, the scheduler must determine when, and in what order with respect to the jobs of other tasks, should each job execute.

The scheduling of real-time tasks on multiprocessors can be carried out under the *partitioned* scheme or under the *global* scheme [13]. In the partitioned scheme, no migration is permitted, whereas in the *global* scheme migration is permitted. If *task-level migration* is used, the jobs of a task may execute on different processor, but each job can only execute on a single processor. If *job-level migration* is allowed, a single job can migrate to and execute on different processors. However, parallel execution is not permitted; that is, no job of any task can be executed at the same time on more than one processor.

When a task has a single fixed priority applied to all its jobs, it is referred as *fixed task priority*. The *Rate Monotonic* scheduling in an example of this. If the jobs of a task may have different priorities, but each job has a single static priority, it is referred as *fixed job priority*. The *Earliest Deadline First* scheduling is an example of this priority assignment. When a single job may have different priorities at different instant times, it is referred as *dynamic priority*, as in the *Least Laxity First* scheduling.

If there exist a scheduling algorithm that can schedule all possible sequences of jobs that may be generated by the task set on a given system without missing any deadlines, then the task set is said to be *feasible* with respect to that system. A task set is *schedulable* according to a given scheduling algorithm if none of its jobs miss any deadline.

A *schedulability test* is a mathematical condition that is used to verify if a task set is schedulable when scheduled with a given scheduling algorithm. A schedulability test is said to be *sufficient*, with respect to a scheduling algorithm and a system, if all the task sets that are considered as schedulable according to the schedulability test are in fact schedulable. A schedulability test is said to be *necessary* if all the task sets that are considered unschedulable according to the schedulability test are in fact unschedulable. A schedulability test is *exact* if it is both sufficient and necessary.

In the last decade, scheduling techniques for multiprocessor architectures have received considerable attention. As a consequence, a lot of scheduling algorithms and schedulability tests have been proposed [16] [7] [9]. However, in many cases, the maximum utilization bound for the scheduling algorithm is too pessimistic. For instance, for periodic task sets with implicit deadlines, even though the maximum possible utilization bound for global dynamic priority scheduling is *m* [6], for all other classes of multiprocessor scheduling algorithms, the maximum utilization bound is ${(m+1)}/{2}$ [1].

Besides its use as schedulability test, *worst-case utilization bounds* have been widely used as a performance metric. The worst-case utilization bound U_A for a scheduling algorithm A is defined as the minimum utilization of any task set that is just schedulable according to the algorithm. If the utilization of the task set is increased, it may be unschedulable according to that algorithm. In contrast, there are no task sets with total utilization less than or equal to U_A that are unschedulable according to the algorithm A. In this case, the worst-case utilization bound can be used as a sufficient schedulability test.

Another alternative to study the performance of a scheduling algorithm, or to verify the schedulability of a task set, is through the use of simulation. For instance, simulation can be helpful to determine the number of context switches and migrations when comparing different scheduling algorithms. Similarly, it can be used to verify if a task set misses any deadline, showing if it is unschedulable. Even though simulation in general cannot be used to prove the schedulability of a task set, it can be used as a sufficient condition for unschedulability.

The task set hyperperiod $(0, H(\tau))$ is a feasibility interval for implicit and constrained deadline synchronous periodic task sets, when scheduled by a deterministic and *memoryless* algorithm, such as global EDF [15]. For these algorithms, the schedulability of the task set can be verified by checking if the schedule generated misses any deadline in $(0, H(\tau))$. In these cases, the use of a simulation and analysis tool can be very helpful.

III. RELATED WORK

Although the real-time scheduling community has provided a vast amount of results over the past 30 years, only a few simulation and analysis tools have been proposed, and it seems that some of them are not maintained anymore.

In [5], Audsley *et al.* proposed a collection of CASE tools for analyzing and simulating the behavior of hard real-time safety-critical applications. The proposed simulator, called STRESS, included a simulation language to specify both the system environment and the task parameters. It included a graphical front-end for control and display, some feasibility tests and support for multiprocessing and networking. Since the simulator was built upon a simulation language, it imposed a limit in the scope of scheduling algorithms that could be simulated.

De Vroey *et al.* proposed in [18] a language for defining scheduling algorithms for hard real-time systems and a tool to simulate the behavior of such systems on a predefined task set. Their proposal was similar to STRESS but presented an extended language in order allow the simulation of newly devised scheduling policies.

Kramp *et al.* proposed FORTISSIMO in [23]. It was also based on STRESS. Their proposal was not a ready-to-run application but an open framework to facilitate the development of tailor-made real-time scheduling simulators for multiprocessor systems. FORTISSIMO was not a simulation application but provided a basic infrastructure to build a real-time scheduling simulator.

In [29] Manacero *et al.* introduced RTSIM, a set of real-time scheduling simulator tools aimed primarily to be used as a teaching tool. It is a collection of programming libraries written in C++ for simulating real-time control systems. RTSIM is still an active project and its code has been recently released as open source.

Jakovljevic *et al.* presented in [22] a research focused on the application of object-oriented language (Java) in the development of real-time system simulation. They described advantages and disadvantages of Java, and gives a critical overview of necessary modifications to make Java an acceptable choice for real-time systems.

Gonzalez-Harbour *et al.* introduced MAST in [21], which is a model for representing the temporal and logical elements of real-time applications. It provides some schedulability tests and algorithms for fixed priority scheduling. MAST also provides sensitivity analysis, to determine how far or close is the system from meeting its timing requirements.

The CICLIC tool was introduced in [33] by Yepez *et al.*, which help system designers to create a cyclic scheduler automatically. The design mechanism of CICLIC is based on algorithms of exhaustive search that use heuristic rules in optimizing searching paths.

Singhoff *et al.* introduced in [30] the Cheddar project, which is a set of open-source tools aimed to help system designers to automatically apply real-time schedulability theory. It includes tools to evaluate the performance of scheduling algorithms and to apply schedulability tests. The Cheddar project has been focused on AADL, an architecture language to model system and software real-time architectures.

The LITMUS project is Linux-based testbed, which was developed for empirically evaluating multiprocessor real-time scheduling algorithms [12]. LITMUS is a soft real-time extension of the Linux kernel with focus on multiprocessor real-time scheduling and synchronization. The Linux kernel was modified to support the sporadic task model and modular scheduler plug-ins. Both partitioned and global scheduling is supported.

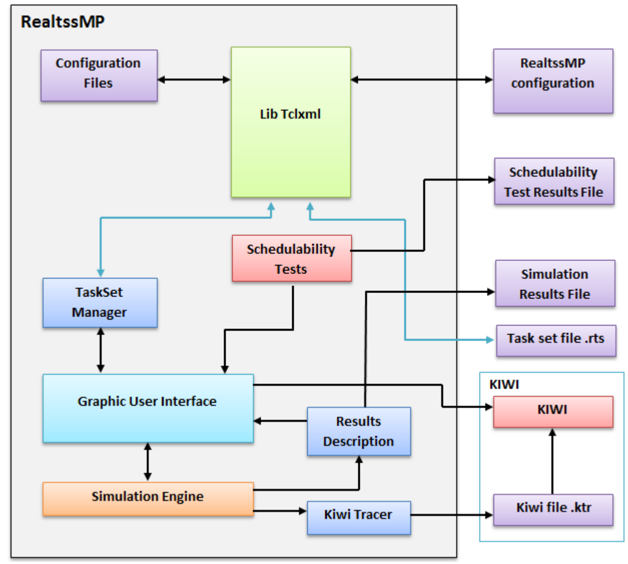

Figure 1. Architecture of RealtssMP

Among the commercial tools, TimeWiz from TimeSys Corporation [14] is an integrated design tool for modeling, analyzing, and simulating the predicted and simulated performance and timing behavior of real-time embedded systems. Rapid RMA from Tri-Pacific Software, Inc. [32] is a set of Rate Monotonic Analysis (RMA) tools to allow designers to test software models against various design scenarios and evaluate how different implementations might optimize the performance of their systems. However, these commercial tools are targeted to only some specific executive environments.

IV. RealtssMP

The proliferation of multi-core architectures, which have multiple processing units on a single chip, has increased the interest on multiprocessor real-time systems. The intensive research in this field has generated many interesting results in recent years. However, there are still many important research challenges. In this context, the availability of an analysis tool can be very helpful in the study of multiprocessor scheduling algorithms.

RealtssMP is an open-source analysis and simulation tool, aimed to support the study of uniprocessor and multiprocessor real-time scheduling algorithms. It can be used to analyze and simulate the predicted performance of real-time systems. Similarly, it can be used to study the schedulability of a task set. RealtssMP is a re-written from scratch version of the uniprocessor simulation tool introduced in [19]. It is intended to be flexible. Its modular design allows the integration of additional scheduling algorithms seamlessly. New scheduling algorithms can be added as modules written in TCL, C or C++ and is not limited by a particular simulation language. It can be executed in many operating systems, such as Linux or Windows.

Fig. 1 shows the architecture of the RealtssMP tool. It is important to note that all data sent to the tool or produced by it is XML formatted. We have defined a set of XML schemes for

```xml
<?xml version="1.0" encoding="utf-8" ?>
- <TaskSet>
    <Name>TaskSet5</Name>
  - <Task>
      <Id>1</Id>
      <Phase>0</Phase>
      <Period>7</Period>
      <Wcet>2</Wcet>
    </Task>
  - <Task>
      <Id>2</Id>
      <Phase>0</Phase>
      <Period>3</Period>
      <Wcet>1</Wcet>
    </Task>
  - <Task>
      <Id>3</Id>
      <Phase>0</Phase>
      <Period>5</Period>
      <Wcet>1</Wcet>
    </Task>
  - <Task>
      <Id>4</Id>
      <Phase>0</Phase>
      <Period>10</Period>
      <Wcet>9</Wcet>
    </Task>
  - <Task>
      <Id>5</Id>
      <Phase>0</Phase>
      <Period>12</Period>
      <Wcet>1</Wcet>
    </Task>
</TaskSet>
```


Figure 2. Definition of a task set

this purpose, that are managed by the *RealtssMP Configuration* module. For instance, a user can define the categorization of the scheduling algorithms and schedulability tests in the *Configuration Files*, using the appropriate schema. When the tool executes, the *Graphical User Interface* module will show the available algorithms and tests using that categorization. The list of available scheduling algorithms and schedulability tests, along with their parameters, are also defined in special *Configuration Files*. The RealtssMP GUI was written in the TCL language.

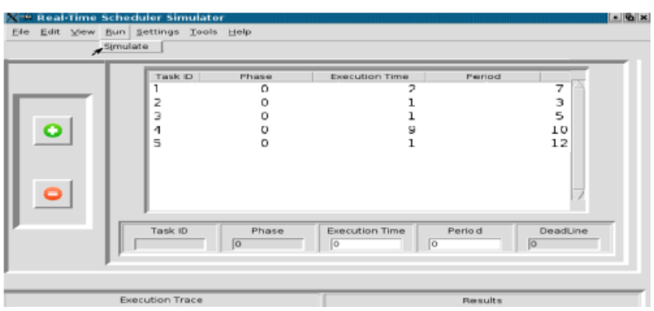

Figure 3. RealtssMP Graphical User Interface

A task set must be defined to study and evaluate the performance of a scheduling algorithm, or to apply a schedulability test. The task set parameters may be declared in an external file, using the appropriate XML schema, or can be generated using the RealtssMP tool and further saved into a file. In Fig. 2 we can observe the definition of a task set using our XML schema. Similarly, Fig. 3 shows the parameters of the same task set using the *Graphical User Interface*. Once the task set is defined, the user selects the scheduling algorithm, the number of processor and the simulation time. If the simulation

Figure 4. Execution chronogram of two processors

time is set to zero, the tool calculates the hyperperiod for the task set and uses it as the simulation time.

RealtssMP is a multi-threaded application. For every processor defined in the simulation setup, a new thread is created. This is helpful to reduce the simulation time, specially if a multi-core or mult-processor computer is used to conduct the simulation experiment.

After the simulation time has elapsed, some scheduling information is shown in the *Graphical User Interface*. The information includes the system utilization, task response times, the number of preemptions, number of migrations, number of missed deadlines, etc. If a schedulability tests is executed, the results are shown in the GUI. A trace file is also generated, which is used by the Kiwi program. RealtssMP is fully integrated with Kiwi [20], a graphic application which displays tasks execution trace logs. The results of the simulations are generated in a kiwi-compatible format in order to be displayed properly with the use of this tool. After the simulation has been executed, the user can analyze the behavior of the tasks set once it has been scheduled with the chosen scheduling policy. Fig. 4 shows the execution chronogram of the task set shown in Fig. 2, when scheduled using the partitioned algorithm *EDF Best-Fit* using two processors.

The use of the simulator involves three steps most of the times: definition of parameters, simulation and results analysis. In the first stage the user defines the parameters of the tasks set. The parameters that can be defined are period, worst-case execution time, phase and deadline.

We have already integrated some scheduling algorithms into RealtssMP, besides the uniprocessor algorithms already implemented in the previous version. The list of scheduling algorithms include partitioned RM and EDF scheduling using First-Fit, Next-Fit, Best-Fit and Worst-Fit heuristics, along with Decreasing-Utilization variants [28] [27], the R-Bound-MP algorithm [24], the RM-ST and RM-GT algorithms [10], and the R-Bound-MP-NFR algorithm [4].

Some global scheduling algorithms are also implemented, as the global RM and global EDF algorithms, the EDF-US[ζ]

algorithm [31], the TkC algorithm [2], the fpEDF algorithm [8], the adaptiveTkC algorithm [3], the RM-US[m/(3m-2)] algorithm [1], just to mention a few.

Similarly, many schedulability tests for uniprocessor and multiprocessor scheduling have been implemented in RealtssMP.

V. Conclusions and Future Work

Real-time multiprocessor scheduling have received considerable attention in recent years. However, there are still many important open research challenges. In this paper, we introduced RealtssMP, an open-source analysis and simulation tool, aimed as a tool to study, evaluate and simulate the performance of real-time multiprocessor systems. The proposed tool is flexible enough, allowing the seamlessly integration of additional scheduling algorithms and schedulability tests. RealtssMP is fully integrated with Kiwi, a visualization tool used to evaluate the execution chronograms of the task sets. RealtssMP has shown to be helpful in the evaluation of the performance of existing and new scheduling algorithms, and as an effective educational tool.

As future work, we will integrate scheduling algorithms for shared resources, and sensitivity analysis techniques.

References

[1] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, pages 193–202, London, UK, Dec 2001. IEEE Computer Society.

[2] Andersson, B. and Jonsson, J. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In *Proceedings of the 7th IEEE International Conference on Real-Time Computing Systems and Applications*, pages 337–346, Cheju Island, South Korea, Dec 2000.

[3] Andersson, B. and Jonsson, J. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. In *Proceedings of the 21st IEEE Real-Time Systems Symposium, WiP Session*, pages 53–56, Orlando, Fl, USA, Nov 2000.

[4] Andersson, B. and Jonsson, J. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 33–40, Porto, Portugal, July 2003.

[5] Audsley, N.C., Burns, A., Richardson, F.M, and Wellings, A.J. Stress: a simulator for hard real-time systems. *Software - Practice and Experience*, 24(6):543–564, June 1994.

[6] S. Baruah, N.K. Cohen, C.G. Plaxton, and D.A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, Jun 1996.

[7] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. Improved multiprocessor global schedulability analysis. *Real-Time Systems*, 46(1):3–24, Sep 2010.

[8] Baruah, S. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784, Jun 2004.

[9] Konstantinos Bletsas and Björn Andersson. Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound. *Real-Time Systems*, 47(4):319–355, Jan 2011.

[10] Burchard, A., Liebeherr, J., Oh, Y., and Son, S.H. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12):1429–1442, Dec 1995.

[11] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley Longmain, April 2009.

[12] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. Litmus: A testbed for empirically comparing real-time multiprocessor schedulers. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123, Rio de Janeiro, Brazil, Dec 2006. IEEE Computer Society.

[13] S. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms*, pages 30–1 – 30–19. CRC Press LLC., 2004.

[14] TimeSys Corporation. *The Concise Handbook Of Real-Time Systems Version 1.1*. 2000.

[15] L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 397–404, Prague, Czech Republic, 2006.

[16] Robert I. Davis and Alan Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, Jan 2011.

[17] Robert I. Davis and Alan Burrns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 2011.

[18] de Vroey, S., Goossens, J., and Hernalsteen, C. A generic simulator of real-time scheduling algorithms. In *Proceedings of the 29th Annual Simulation Symposium*, pages 242–249, New Orleans, LA, April 1996.

[19] Arnoldo Diaz-Ramirez, Ruben Batista, and Oskardie Castro. Realtss: a real-time scheduling simulator. In *Proceedings of the 4th IEEE International Conference on Electrical and Electronics Engineering*, pages 165–168, Mexico City, Sep 2007.

[20] Agustin Espinoza. Kiwi user guide. Technical report, Universidad Politecnica de Valencia, 2003. Available on-line at http://www.dsic.upv.es/users/ia/sma/tools/kiwi/index.html.

[21] M. Gonzalez-Harbour, J.J. Gutierrez Garcia, J.C. Palencia Gutierrez, and J.M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Proceeding of the 13th Euromicro Conference on Real-Time Systems*, pages 125–134, Delft , Netherlands, Jun 2001.

[22] G. Jakovljevic, Z. Rakamaric, and D. Babic. Java simulator of real-time scheduling algorithms. In *Proceedings of the 24th International Conference on Information Technology Interfaces*, volume 1, pages 411–416, Cavtat, Croatia, June 2002.

[23] Kramp, K., Adrian, M., and Koster, R. An open framework for real-time scheduling simulation. *Lecture Notes in Computer Science*, 1800:766+, 2000.

[24] Lauzac, S., Melhem, R., and Mosse, D. An efficient rms admission control and its application to multiprocessor scheduling. In *Proceedings of the IEEE 1st Merged International Symposium on Parallel and Distributed Processing*, pages 511–518, Orlando, Fl, USA, April 1998.

[25] C. L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary 37-60*, II:28–31, 1969.

[26] Liu, J.W.S. *Real-Time Systems*. Prentice Hall, 2000.

[27] Lopez, J.M., Diaz, J.L., and Garcia, D.F. Utilization bounds for edf scheduling on real-time multiprocessor systems. *Real-Time Systems*, 28(1):39–68, Oct 2004.

[28] Lopez, J.M., Garcia, M., Diaz, J.L., and Garcia, D.F. Utilization bounds for multiprocessor rate-monotonic scheduling. *Real-Time Systems*, 24(1):5–28, Jan 2003.

[29] Manacero, E., Miola, M.B., and Nabuco, V.A. Teaching real-time with a scheduler simulator. In *Proceedings of 31st ASEE/IEEE Frontiers in Education Conference*, pages 15–19, Reno, NV, October 2001.

[30] F. Singhoff, A. Plantec, P. Dissaux, and J. Legrand. Investigating the usability of real-time scheduling theory with the cheddar project. *Real-Time Systems*, 43(3):259–295, Nov 2009.

[31] Srinivasan, A. and Baruah, S. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, Oct 2003.

[32] Inc. Tri-Pacific Software. Rapid rma tutorial. Available on-line at http://www.tripac.com/rschedtutorial.

[33] J. Yepez, J. Guardia, M. Velasco, J. Ayza, R. Castane, P. Marti, and J.M. Fuertes. Ciclic: A tool to generate feasible cyclic schedules. In *Proceedings of the IEEE International Workshop on Factory Communication Systems*, pages 399–404, Torino, Italy, Sep 2006.