# MODELING REAL-TIME SCHEDULERS FOR USE IN SIMULATIONS THROUGH A GRAPHICAL INTERFACE

Fernanda F. Peronaglio
Dept of Computer Science and Statistics
São Paulo State University - UNESP
S. J. do Rio Preto, Brazil
fernandaperonaglio@gmail.com

Aleardo Manacero
Dept of Computer Science and Statistics
São Paulo State University - UNESP
S. J. do Rio Preto, Brazil
aleardo@ibilce.unesp.br

Renata S. Lobato
Dept of Computer Science and Statistics
São Paulo State University - UNESP
S. J. do Rio Preto, Brazil
renata@ibilce.unesp.br

Roberta Spolon
Dept of Computing
São Paulo State University - UNESP
Bauru, Brazil
roberta@fc.unesp.br

## ABSTRACT

Simulators of real-time systems are usually oriented to the evaluation of how different schedulers perform under different loads. Existing simulators can be split into two groups, one where scheduling algorithms are native to the simulator and one where they have to be programmed as part of the model. RTsim falls in the first group, enabling the simulation of some selected algorithms. In this paper, we present an extension to RTsim, enabling the simulation of non-native scheduling algorithms through the insertion of a description of their policies. With this extension, its user will be able to provide an algebraic description of the scheduler into a graphical interface, which creates Java code for the given algorithm. That code can be added to RT-sim's library in order to simulate the algorithm. Examples of the application of this extension are presented, as well as an evaluation of the policies generated.

**Keywords:** real-time scheduling simulation, scheduling algorithm models, GUI modeling.

## 1 INTRODUCTION

Real-time systems comprise time-constrained applications where the instant when results are presented can be as much important as their correctness. Real-time applications are usually found in critical environments, where delays in the processing can create hazardous conditions, such as crashes, overheating machinery and so on (Liu 2000). Therefore, the assurance that all processes will terminate inside their time restrictions is the main goal for real-time systems developers.

The development of real-time applications involves two different aspects: the actual application and the selection of a proper scheduling algorithm. While the former depends mostly of the real environment, which constrains what can be done, the latter presents the chance for adapting the processing environment to the tasks that have to be processed. To identify the best option to schedule real-time tasks, one has to evaluate several algorithms, with different goals and restrictions. This evaluation is usually performed through simulation in order to avoid hazardous events.

Several simulators of real-time schedulers have been proposed, such as RTsim (actually there are several simulators named this way, even aiming at quite different applications, such as power generation), Realtss, STRESS and Simso, which are described in the next section. The goal of these simulators is the evaluation of different scheduling policies applied to real-time tasks. The simulators can be grouped into two categories following the approach used to provide the scheduler. A first group provides a set of native policies that can be simulated. The obvious advantage of this approach is that the analyst using the simulator has to worry only about the load (application) that will be simulated. Its disadvantage is that one can simulate only the native policies. The second group circumvents this problem by demanding that every scheduler has to be programmed in order to be simulated. While this is more flexible, it demands that the analyst learn how to program a scheduler in the language adopted by that specific simulator.

Therefore, the development of a simulator that is flexible and easy-to-use is an interesting line of work. It is interesting because a flexible simulator allows for the evaluation of many more scheduling policies than the "native schedulers" approach provide. It is also interesting because an easy-to-use simulator is more attractive than one where the user has to write programs in order to simulate something. The new version of RTsim (**R**eal-**T**ime **sim**ulator) (Manacero, Miola, and Nabuco 2001), presented in this paper, incorporates flexibility while keeping its easy-to-use approach when new schedulers have to be evaluated. The approach used to add new scheduling policies is based in graphical interfaces, where the user will provide the ordering criteria for the scheduler through one or more relations and equations. This information is used to automatically create a Java class and its methods that can be executed by the simulator.

In the remaining sections we describe RTsim and how the capability of adding new schedulers was incorporated into it. We also present results of this new functionality as well as conclusions about its effectiveness.

## 2 RELATED WORK

Simulation of real-time scheduling algorithms can be performed through several tools. A thorough literature review in this topic is beyond the scope of this paper, so we focus in few historic tools and more recent proposals. The simulators presented here are separated according to the approach used to formulate the schedulers, as defined in the Introduction, starting with tools that demand that the user write the scheduling code.

One of the first simulators proposed was STRESS (Audsley, Burns, Richardson, and Wellings 1994), aimed to evaluate hard critical systems. STRESS was built using a specific, internally defined, language to program the schedulers, what complicated its use, besides all graphical interfaces provided to execute the simulations. It must be mentioned that several tools were proposed following the ideas from STRESS, such as the FORTISSIMO framework (Kramp, Adrian, and Koster 2000).

Recently presented, SimSo (Ch'eramy, D'eplanche, and Hladik 2013) also demands scheduler codification using Python language. An important advantage of SimSo is its formulation as a web-based tool, allowing the availability of a large set of previously programmed schedulers.

In the class of simulators with native schedulers we find Realtss (Diaz, Batista, and Castro 2007), which provides some general policies, such as Rate Monotonic, Earliest Deadline First and Deadline Monotonic. Realtss was originally devised as a tool for teaching scheduling algorithms and also enables the insertion of new schedulers through Tcl, C or C++ codes.

Another tool that aims teaching schedulers is AURTSS (Yaashuwanth and Ramesh 2010). As SimSo, it is web-based and was developed using the LabVIEW framework (from National Instruments Co.), allowing the simulation of classic policies (RM, DM, EDF, e.g.).

RTsim (Casile, Buttazzo, Lamastra, and Lipari 1998) is another simulator that have native schedulers. It is implemented in C++ and allows the introduction of new algorithms through the insertion of libraries written in C++. The graphical interfaces are simple but it demands that the description of the system load be made by script files. Although this approach gives a bigger flexibility in the loads that can be applied to the simulation, it demands an extra knowledge (and effort) from its user.

From these tools one can see that none provides an interface where new schedulers can be easily formulated. Several simulators provide native algorithms and a mechanism to write code to simulate a new one. They differ in the language used to write that code, ranging from python to built-in specification languages. The approach proposed here avoids programming through the use of graphical interfaces that guide the policy specification.

## 3 RTSIM

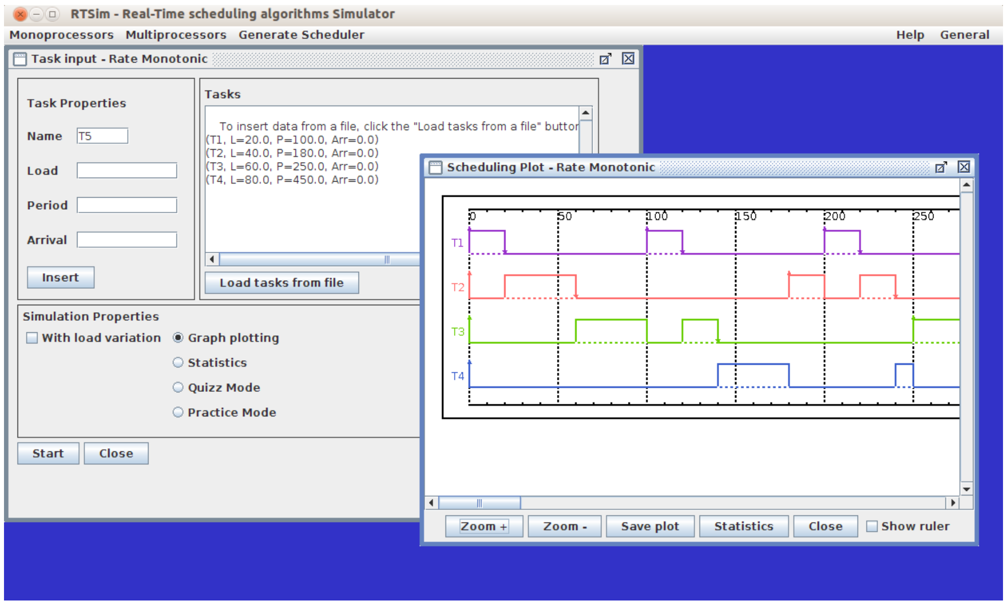
RTsim (Manacero, Miola, and Nabuco 2001) is a simulator of real-time schedulers devised as a helper to teach real-time systems. Its original prototype was implemented in 1997 and included five single-processor schedulers: Rate Monotonic, Sporadic Server, Deferrable Server, Priority Exchange and Priority Ceiling protocols. The initial tests showed that it would be an interesting tool to help students learning those schedulers, that have been performed regularly in the undergraduate course of Real-Time Systems offered in the Computer Science major at Unesp. Several upgrades have been introduced since then, such as the inclusion of some distributed schedulers, a thorough reengineering towards object-oriented programming and Java, and components oriented to teaching, such as a practice mode interface.

RTsim's current version includes the following native algorithms (the last three for multiprocessor systems):

- Rate Monotonic (periodic tasks)
- Deferrable Server (aperiodic tasks)
- Sporadic Server (aperiodic tasks)
- Priority Exchange Protocol (concurrent tasks)
- Priority Ceiling Protocol (concurrent tasks)
- Myopic (distributed systems)
- Bidding (distributed systems with migration)
- Focused Bidding (distributed systems with migration)

The simulator can be used in three ways, where the user decides the appropriate path during the process of providing data for the simulation. These ways are:

- **Graphic output:** where a gantt chart provides CPU occupation by each task. The interface allows zooming and the use of a ruler to see scheduling details. When multiprocessor systems are being simulated, the output window provides a selector for specific machines. In this option it is also possible to verify some statistics about the execution, such as missed deadlines or average delays.
- **Statistical output:** where statistical data is presented through some tables that can be viewed either for specific tasks or globally.
- **Learning and Test modes:** where the user, that is, a student, will input the scheduling for a given set of tasks and the simulator will guide that (if in Learning mode) or "grade it" (if in Test mode).



Name T5

To insert data from a file, click the "Load tasks from a file" buttor
(T1, L=20.0, P=100.0, Arr=0.0)
(T2, L=40.0, P=180.0, Arr=0.0)
(T3, L=60.0, P=250.0, Arr=0.0)
(T4, L=80.0, P=450.0, Arr=0.0)

☐ With load variation
[✓ Graph plotting] Statistics Quizz Mode Practice Mode

Figure 1: Input and Output windows used for the Rate Monotonic algorithm.

Besides the output variations, all data input is done by different GUIs. For example, in Figure 1 we can see the window to enter periodic tasks in the Rate Monotonic algorithm (in the upper left) and the graphic output produced by a set of four periodic tasks (right side). It is possible to see in the input window that the user chooses the "Graph Plotting" output and the parameters of the simulated tasks, such as task T1 with a Load of 20 time units, Period of 100 time units and first arrival at 0. In the output window one can see the little arrows marking the arrival and completion of each task and the buttons to show statistics, zooming and plot saving.

The input of the tasks can be done manually by the insertion of each task and related parameters or by reading an input file. Although this file follows a fixed format, that format can be recovered after a first run since the simulator creates such files from the data inserted to each simulation. Another file created during every simulator's run is a trace of the scheduling, allowing for a post-mortem analysis of the schedule.

Although the use of RTsim for learning is, by itself, rewarding, it would be interesting to expand its use to real-time developers. To do so it is necessary to enable the introduction of new scheduling policies into RTsim, which can be done through two different mechanisms: adding new native schedulers or enabling an user to write new schedulers in the simulator's language. Besides the first approach could be easily done, it would not reach completeness of algorithms. Therefore, the second approach is more attractive, although it will depend on the user's capabilities. In the next section we describe an extension to RTsim that enables the insertion of new algorithms simply by its algebraic definition, avoiding the need for writing code.

## 4 AUTOMATING THE ADDITION OF SCHEDULERS INTO RTSIM

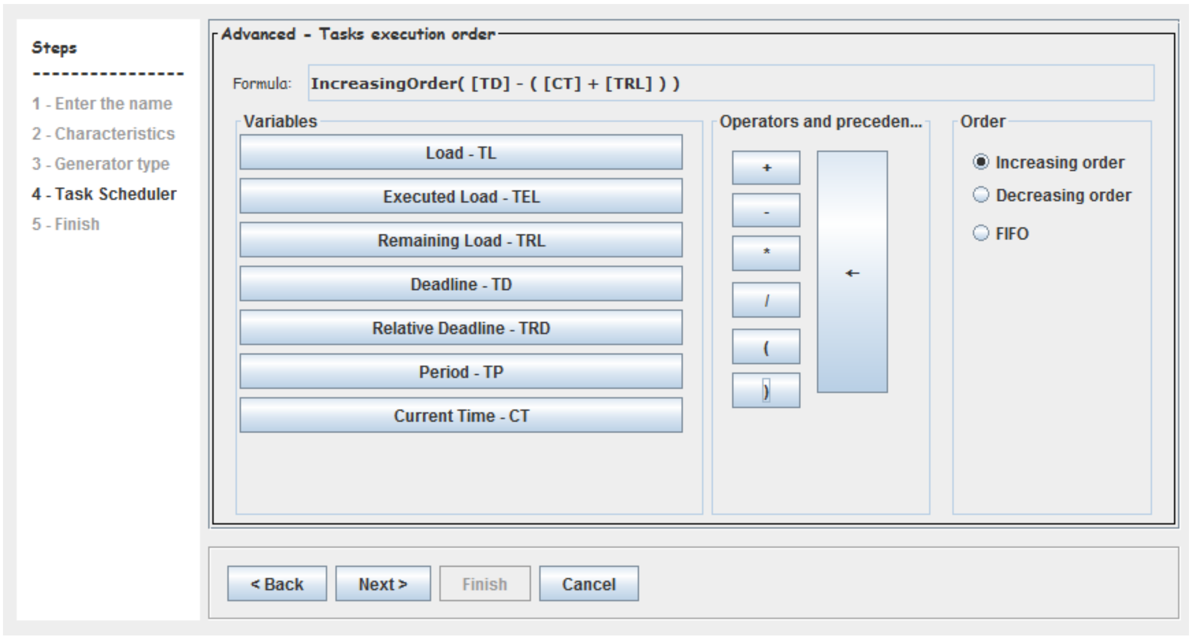
In order to enable an user to define a new scheduling algorithm without code-writing it is necessary to develop a graphical interface that automates the process of code generation. This component has to provide means to define the scheduling policy and to translate that definition into a piece of source code. Our approach to perform these tasks was inspired in another simulator, the iSPD (Manacero, Lobato, Oliveira, Garcia, Guerra, Aoqui, Menezes, and Da Silva 2012), which simulates grid and cloud systems.

In iSPD there is a component to generate metaschedulers through their mathematical definition. However, that generator could not be migrated to RTsim in a straightforward form. The migration was impossible because the scheduled tasks in iSPD are any event occurring in a distributed system, including data packets and virtual machine instantiations. Indeed, the simulation engine in iSPD simply considered all events as something that will take some time to be completed and will be moved forward to another server in the network. In RTsim the events are real-time tasks that have to be executed, and the simulation engine have different mechanisms to deal with different types of tasks (periodic, aperiodic, concurrent, etc). Therefore, to create a similar component to RTsim we had to evaluate which parameters are relevant to real-time tasks and how they could be used to formulate a scheduling policy.

An extensive study over real-time scheduling policies proposed in the literature led to the identification of some parameters that are mostly commonly found in the specification of a policy. At the current version of RTsim the parameters that can be used to formulate a scheduler, and their definitions, are:

- Period: time interval until the next occurrence of a periodic task;
- Load: processing time needed to complete a task;
- Executed Load: processing time already consumed by the task;
- Remaining Load: processing time still to be executed to complete a task;
- Relative Deadline: maximal time interval allowed for the task completion;
- *(Absolute)* Deadline: actual instant marking the maximal time allowed for completion;
- Current Time: actual clock value during a scheduling decision.

These parameters are shown in Figure 2, alongside the buttons used to write and define the equation that describes the scheduling policy (field "Formula"). In order to specify a policy the user will have to write its algebraic definition using the parameters and operators buttons. This equation is displayed in the top part of the window, such as the equation appearing in Figure 2, which defines the Least Slack-Time algorithm.



Formula: IncreasingOrder( [TD] - ( [CT] + [TRL] ) )

Order: [✓ Increasing order] Decreasing order FIFO

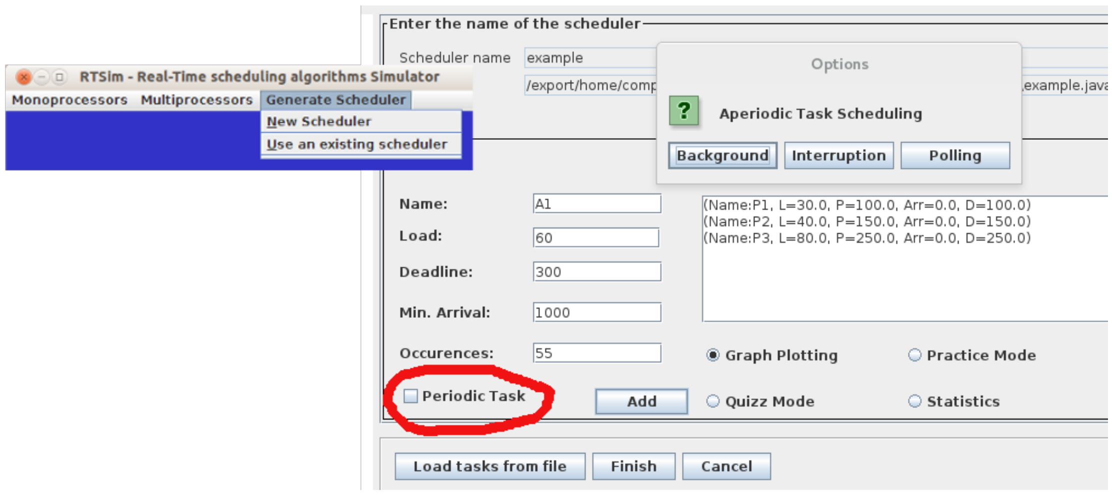
Figure 2: GUI to formulate the algebraic definition of a scheduler.

The code for the scheduler is created after the definition of the scheduler's equation. At this moment RTsim creates two files, one containing a text description of the scheduling equations, and other containing a Java class with methods and objects needed to simulate the scheduling policy. The creation of the Java class is made by an interpreter written specifically for the grammar designed for the policies' equations. This interpreter checks for the syntax correctness and a code generator uses the abstract syntax tree just built to fill in the methods for that class.

To run a simulation with the new scheduler the user will have to call/open the saved text file. This is done in the "Generate Scheduler" tab in the main RTsim's window (seen in the left side of Figure 3). After selecting the option for using an existing scheduler, the user is prompted with a list of schedulers and has to provide the data relative to the tasks that will be simulated. In the folder where these files are stored the user will find, in following runs of RTsim, files storing tasks previously used in a simulation.

To enter task data one has to define if it is necessary to define if it is periodic or aperiodic, as shown in Figure 3. In that figure it is possible to see the tasks already defined (periodic tasks P1, P2 and P3, with their characteristics), as well the definition of the aperiodic task A1 (the not-tagged box inside the red circle indicates that it is not periodic). While periodic tasks are scheduled by the equation previously entered, for aperiodic tasks it is also necessary to define how these tasks will be served (the pop-up in Figure 3). Currently there are three different approaches to execute aperiodic tasks:

1. Background time, when it occupies the CPU whenever it is idle;
2. System interrupts, when it preempts the CPU as soon as the aperiodic occurs and runs that task until its completion;
3. Polling Server, where a virtual periodic task is created, having a specific load and period (the server's capacity and period) in order to accommodate the execution of aperiodic tasks, whenever the polling server's remaining load is different from zero and the task occurred before the server's instantiation terminates.



Scheduler name: example

/export/home/comp ... example.java

Name: A1

Load: 60

Deadline: 300

Min. Arrival: 1000

Occurences: 55

[ ] Periodic Task

(Name:P1, L=30.0, P=100.0, Arr=0.0, D=100.0)
(Name:P2, L=40.0, P=150.0, Arr=0.0, D=150.0)
(Name:P3, L=80.0, P=250.0, Arr=0.0, D=250.0)

[✓ Graph Plotting] Practice Mode Quizz Mode Statistics

Figure 3: Interfaces for tasks definition (right side) and scheduler's choice (left).

## 5 EVALUATION OF SCHEDULER GENERATION

In order to validate the scheduler generator we conducted several tests, checking the generation of different policies, including those that were already implemented as native schedulers. The policies presented here were selected because they represent very well known policies, making easier to describe them. It is important to state that the gantt charts presented here are not the only output produced by RTsim. A new scheduler produces the same statistics produced by the native algorithms, including missed deadlines or relative delays.

## 5.1 Least Slack-Time First - LST

LST is a well known dynamic algorithm, designed for periodic tasks. In this policy tasks are ordered considering that the task with the smallest slack-time will be allocated first, that is, the algorithm prioritizes the task which completion will be closest to its deadline. The algebraic formulation of LST can be seen in Figure 2, while a run of that algorithm, using the tasks presented at Table 1, is seen in Figure 4.

Table 1: Tasks set used in the simulation of algorithms for periodic tasks.

| Task | Load | Period | Deadline |
|---|---|---|---|
| P1 | 20 | 100 | 100 |
| P2 | 40 | 180 | 180 |
| P3 | 60 | 250 | 250 |
| P4 | 80 | 450 | 450 |

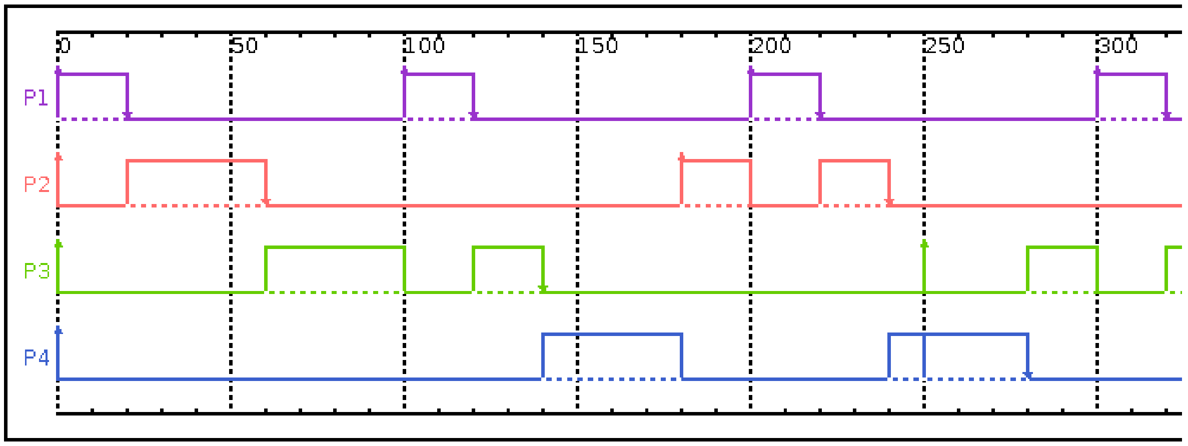


Figure 4: Schedule produced by a run of the generated LST.

Looking carefully at Figure 4 it is possible to identify two instants where the policy acts preempting the running process to allocate the CPU to one with higher priority. These preemptions occur at times 100 and 200, when:

- Time=100: Task P3 is running and a new instance of P1 occurs. At this time step, the priorities are recalculated. The slack-time for P1 is 80 time units (200 - 100 - 20) while it is 130 time units (250 - 100 - 20) for P3, so P3 is preempted and P1 gets the CPU;
- Time=200: At this moment task P2 is executing and has a slack-time of 140 time units (360 - 200 - 20) when a new instance of P1 occurs (slack-time of 80) and receives the CPU again.

The same, expected, behavior was observed throughout the simulation. All subsequent runs of the created scheduler produced the results that would be produced by a conventional implementation of the LST algorithm.

## 5.2 Deadline Monotonic - DM

Another classical algorithm not native in RTsim is the Deadline Monotonic. Its definition is also simple, since the fixed priority is the relative deadline of the tasks (`TRD` buttom in Figure 2). Its behavior is shown in Figure 5. In that output it is possible to observe that the relative priorities do not change during the execution. This implied in a different decision in T=250, where task P4 continues executing in LST and is interrupted in DM.

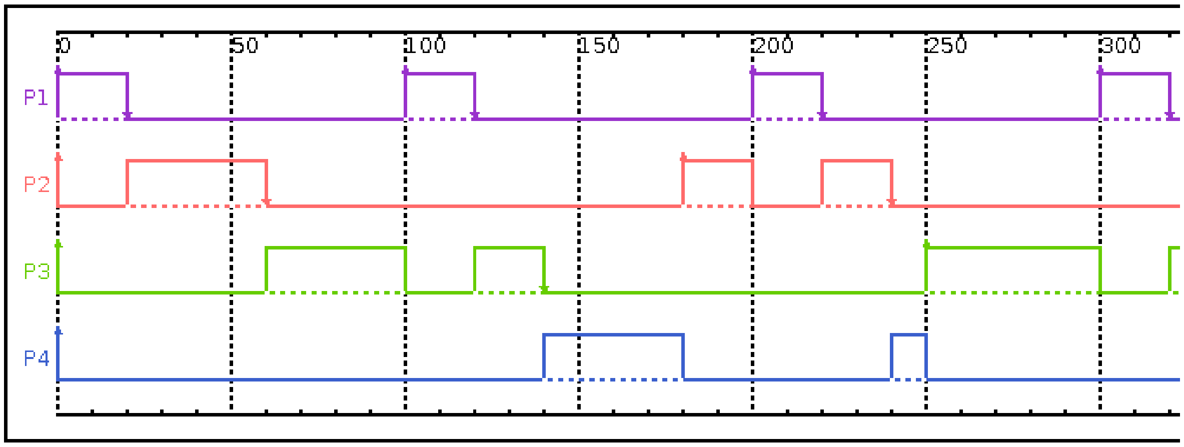


Figure 5: Schedule produced by the Deadline Monotonic algorithm (tasks from Table 1).

## 5.3 Earliest Deadline First - EDF

Another algorithm not present in RTsim is the Earliest Deadline First, which is also applied to periodic tasks and has a dynamic behavior (as the LST). The resulting schedule for the set of tasks listed in Table 1 is presented in Figure 6. It is observable that the results from EDF and LST are equivalent for this set of tasks, and both are different from DM. Although DM and EDF use deadline as the ruling parameter, it must be understood that DM is static while EDF is dynamic. Therefore, while P3 has always higher priority than P4 in the DM algorithm, this may change during the execution of the EDF algorithm. As shown in Figure 6, before the first instance of P3 terminates, it has priority over P4. This changes at T=250, when P4 has its deadline in t=450, and P3 has its deadline in T=500, getting postponed by P4.

## 5.4 Polling Server

In the previous section it was described that aperiodic tasks can be managed either by a polling server, interrupts, or by using CPU idle times. To evaluate how a scheduler using one of such approaches can be generated we present a scheduler using a polling server. To simulate it it was used the set of tasks listed in Table 2. It is important to notice that when a polling server is used, RTsim considers that aperiodic tasks, when occurring, will use the CPU time reserved for the server. Therefore it is necessary to “design” the

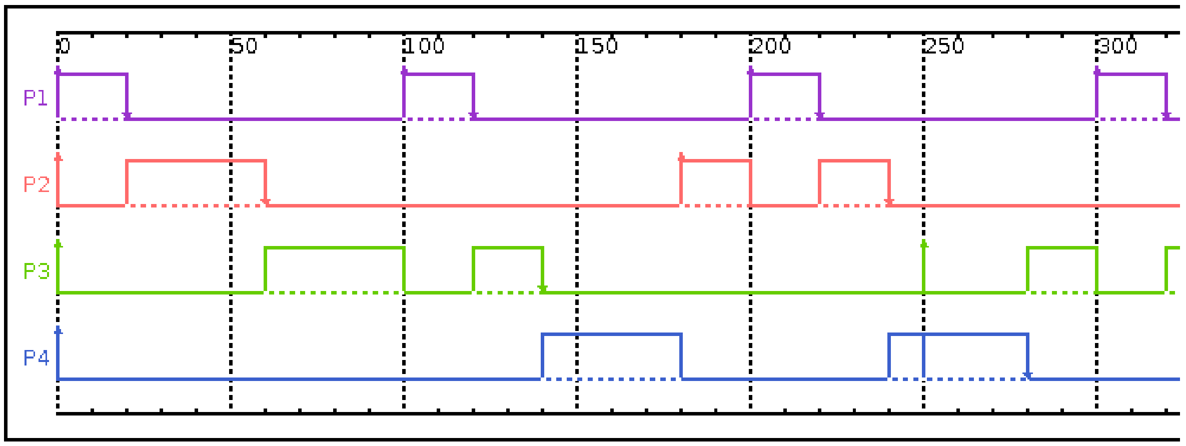


Figure 6: Resulting schedule produced by the Earliest Deadline First (tasks from Table 1).

server, that is, to define its period and load. The periodic tasks, including the polling server, are scheduled according to the policy equation introduced during the algorithm definition.

Table 2: Tasks set used in the simulation of algorithms for aperiodic tasks.

| Task | Load | Period | Deadline | Occurrences |
|---|---|---|---|---|
| P1 | 30 | 100 | 100 | - |
| P2 | 15 | 120 | 200 | - |
| A1 | 10 | - | 200 | 10 |
| A2 | 10 | - | 400 | 15 |
| A3 | 40 | - | 200 | 40, 120 |

In the test presented here the polling server was represented by a periodic task with a load of 30 time units and a period of 120 time units. In Figure 7 one can observe that tasks A1, A2 and A3 occurs at specific times and are served only while there are remaining load from the server, being interrupted otherwise (as it happens at T=150, when task A3 leaves the CPU after consuming the 30 time units of the server's load).

In order to show the convenience of enabling an easy approach to model and evaluate schedulers we present results achieved with a policy that uses CPU idle times. The application of this policy over the same set of tasks, presented in Figure 8, shows that it produced a better schedule for the aperiodic tasks. This happened because this specific case has a very light load from periodic tasks. Therefore, using a polling server unnecessarily delays the execution of aperiodic tasks in this case, but it must be observed that different loads would result in quite different scenarios for these algorithms.

## 6 CONCLUSIONS

The component introduced to RTsim allowed for the simulation of different scheduling policies, expanding the simulator's flexibility. The component is composed by a small set of graphical interfaces, which can be easily manipulated to create a Java class to simulate a given policy. The only restriction is that one should be able to define the policy algebraically.

Besides the need for an algebraic formulation the approach introduced here is interesting since it avoids the need for code writing that is present in other simulators. Indeed, even considering that a real-time systems developer or student is a programming literate, writing few dozen of lines using partially known

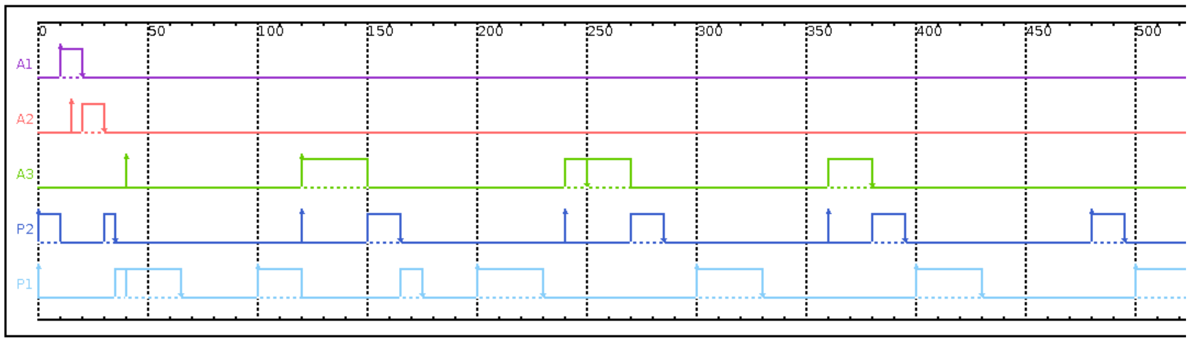


Figure 7: Results from the application of Polling Server for aperiodic tasks.

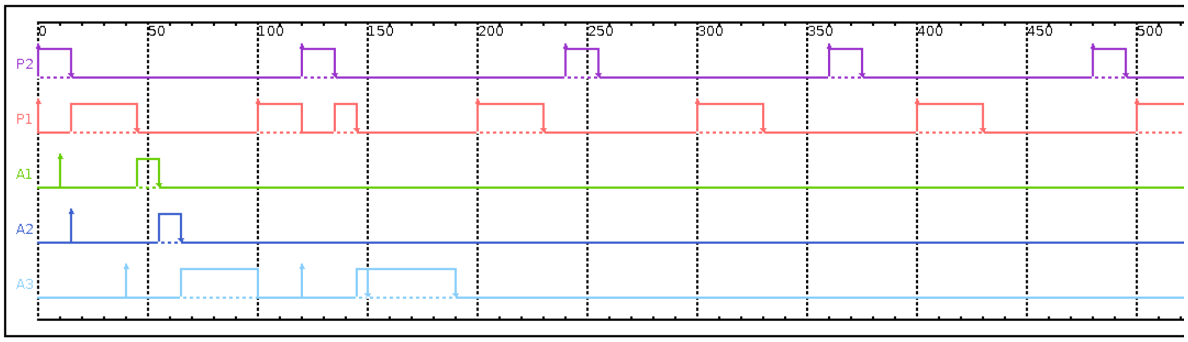


Figure 8: Results from the use of Background time for aperiodic tasks.

data structures is more complex than writing an equation. For example, Simso documentation presents a version of EDF algorithm using 27 lines of python code with several calls for classes and objects defined elsewhere. The same algorithm can be created in RTsim simply defining an equation saying to order by the lowest deadline.

The scheduling policies that cannot be currently generated by RTsim are those that have a combination of decisions to perform the task allocation. At this time this means algorithms that work in two stages, such as the myopic algorithm for multiprocessor systems, but these are very rare for real-time systems since they usually imply in an extra processing load. Therefore, most of schedulers can be defined by relatively simple equations, which can be easily transformed in Java code by RTsim's new component.

All tests that were performed confirmed that the creation of a new scheduler for simulation is quite simple. All schedulers created produced the correct results, what is not surprising since the scheduler were algebraically defined. The correctness of the results was verified by manual execution of the algorithms for the sets of tasks that were tested.

Future work with this component involves the addition of interfaces to define task interactions, such as those present in tasks sharing resources (mutual exclusion). We are also working in a methodology that would allow for the specification of combinations of policies and scheduling in stages.

To conclude, the possibility to simulate new algorithms with a small effort to define them is very interesting, allowing to use RTsim as a performance analysis tool. The use of graphical interfaces is easier and more time effective than writing small pieces of code, as done by other simulators. With this new component RTsim becomes more attractive to both developers and students analyzing/learning scheduling algorithms.

## ACKNOWLEDGMENTS

Authors want to acknowledge the support from Unesp, for local resources, and from CNPq, for undergraduate research scholarships.

## REFERENCES

Audsley, N., A. Burns, M. Richardson, and A. Wellings. 1994. "Stress: A Simulator for Hard Real-Time Systems". *Software-Practice and Experience* vol. 24 (6), pp. 543–564.

Casile, A., G. Buttazzo, G. Lamastra, and G. Lipari. 1998, Oct. "Simulation and tracing of hybrid task sets on distributed systems". In *Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No.98EX236)*, pp. 249–256.

Ch'eramy, M., A.-M. D'eplanche, and P.-E. Hladik. 2013. "Simulation of Real-Time Multiprocessor Scheduling with Overheads". In *Intl Conf on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2013)*, pp. 5–14. INSTICC.

Diaz, A., R. Batista, and O. Castro. 2007. "Realtss: a real-time scheduling simulator". In *Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on*, pp. 165–168. IEEE.

Kramp, T., M. Adrian, and R. Koster. 2000. "An Open Framework for Real-Time Scheduling Simulation". In *Parallel and Distributed Processing, 15 IPDPS 2000 Workshops, Cancun, Mexico, May 1-5, 2000, Proceedings*, pp. 766–772.

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

Manacero, A., R. S. Lobato, P. H. M. A. Oliveira, M. A. B. A. Garcia, A. I. Guerra, V. Aoqui, D. Menezes, and D. T. Da Silva. 2012. "iSPD: An Iconic-based Modeling Simulator for Distributed Grids". In *Proceedings of the 45th Annual Simulation Symposium*, ANSS '12, pp. 5:1–5:8. San Diego, CA, USA, Society for Computer Simulation International.

Manacero, A., M. B. Miola, and V. A. Nabuco. 2001. "Teaching real-time with a scheduler simulator". In *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No.01CH37193)*, Volume 2, pp. T4D–15–19 vol.2. IEEE.

Yaashuwanth, C., and R. Ramesh. 2010. "Web-Enabled Framework for Real-Time Scheduler Simulator (A Teaching Tool)". In *Computer Research and Development, 2010 Second International Conference on*, pp. 826–830. IEEE.

## AUTHOR BIOGRAPHIES

**FERNANDA F. PERONAGLIO** finished her BSc in Computer Science from Unesp in 2016. She is currently enrolled as a MSc student in the same university, under the supervision of Dr. Manacero, continuing her work in the simulation of scheduling algorithms.

**ALEARDO MANACERO** is an Associate Professor within the Department of Computer Science and Statistics at Unesp at Rio Preto. He got his BSEE (87), MSc(91) and PhD (97) in Electrical Engineering, all from State University of Campinas, Unicamp. He is with Unesp since 1990, where he started as a Lecturer. He was also a Visiting Professor at University of Oregon in 2011.

**RENATA S. LOBATO** is an Associate Professor within the Department of Computer Science and Statistics at Unesp at Rio Preto. She got her BSc (91), MSc(94) and PhD (01) in Computer Science, all from University of Sao Paulo, USP. She is with Unesp since 2004, where she started as an Assistant Professor. Prior to working at Unesp she was with the Computing Department of Federal University of Mato Grosso do Sul.

**ROBERTA SPOLON** is an Associate Professor within the Department of Computing at Unesp at Bauru. She got her BSc from Unesp (91), and MSc (94) and PhD (99) in Computer Science, both from University of Sao Paulo, USP. She is with Unesp at Bauru since 1992, where she started as a Lecturer.