Object-oriented simulation of preemptive feedback process schedulers

Martina Maggio*, Alberto Leva
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
`{maggio,leva}@elet.polimi.it`
*PhD student at the Dipartimento di Elettronica e Informazione

Abstract

Based on recent research, very simple discrete-time control structures can be used to synthesise preemptive process schedulers for multitasking systems within a rigorous system-theoretical formalism. Doing so virtually eliminates any heuristics, and allows for a methodologically grounded analysis and assessment of the achieved performances. This paper introduces a Modelica library for the above purpose, at present still under development, and illustrates its use with some tests.

Keywords: Feedback scheduling; Multitasking systems; Preemptive systems.

1 Introduction

Many problems related to computing systems are being recognised and tackled as *control* problems [4]. A notable example is that of process scheduling in multitasking (not necessarily real-time) computing systems. The role of the process scheduler in such systems is to allocate the CPU usage to the running processes, so as to guarantee properties like fairness, responsiveness, and so forth [6]. Feedback-based techniques have been applied to the scheduling problem [2, 3, 8] to deal with uncertainty and disturbances, such as the behaviour of the processes, and the availability of the resources they may require.

In virtually the totality of the feedback scheduling literature, however, the idea is concisely to "close some loop around an *existing* scheduler". Since the object to be controlled (the "plant" to stick to the standard terminology) includes said scheduler, in the above context the term "actuators" takes the specific meaning of "having the feedback controller assign the values of some scheduling parameters" like queue lengths, priority variations, and so on, while the term "sensors" refers to measurements of the required properties, such as the processes' CPU utilisation [1].

Moreover, existing schedulers are conceived by their designers in terms of algorithms and data structures (i.e., the way *computer* scientists think of the word "model") and not of equations (i.e., the way the same word is thought of by *control* scientists). Modeling those schedulers is thus generally complex, but above all it is highly unnatural with formalisms that allow for powerful and simple analysis and synthesis tools. Some attempts were made to devise a model for such a scheduler (as well as other hardware and software components) in Modelica [9]; that work however does not include feedback policies.

This work is part of a wider research that takes completely different an attitude. Instead of acting on the scheduler already present in the considered system, the idea here is to replace that scheduler completely. And correspondingly, instead of writing a model to reflect the scheduling algorithm, the *modus operandi* is to have that algorithm emerge from the digital realisation of a controller model—a perspective shift indeed.

In other words, in this research schedulers are designed exactly in the same way as a feedback controller is synthesised, so as to allow expressing the specifications by means of the usual concepts of set point tracking, load disturbance rejection, and so on. Thanks to the adopted formalism—that of discrete-time *linear* discrete-time dynamic systems, as will be briefly justified later on—the above concepts can be given a *quantitative* meaning, which is a novelty of this research with respect to other approaches to the same problem, where qualitativeness and heuristics play a central (albeit often tedious for the designer) role.

The aim of this manuscript, within the mentioned research, is to present a small Modelica library (at present in its first version and under continuous development) aimed at helping the designer of a scheduling policy assess the behaviour of said policy by means of simulations representing the *on line* system behaviour under suitably chosen load conditions. This is another peculiarity of this work, since the great majority of the available literature concentrates on *off line* schedulability analysis issues [5].

2 The modelling formalism

To set up a process scheduler in the same way as a feedback regulator is designed it is first necessary to select a control-theoretical formalism that admits a clear separation between the "plant" and the "controller". The necessity of such a separation *de facto* rules out the use of discrete-event models. In fact, in the discrete-event control context, defining an open-loop process model almost inherently calls for specifying the desired behaviour in terms of constraints only—think for example of the supervisory control framework—while if said behaviour is more naturally expressed (also) as a desired sequence of events, then only top/down approaches (where the model of the plant and the controller live jointly right from the beginning, however) allow to guarantee some formal property for the closed-loop system.

In addition, in the formalism to be chosen here, the modelled objects have to admit a direct, *non ambiguous* realisation as algorithms. This is a further argument to avoid discrete-event frameworks such as queue networks and Petri nets for our purposes (a promising work on the matter however is [7]), because their inherently asynchronous nature requires to specify "something else" in order to turn a model into an algorithm—think, for example, of evolution rules or similar ideas.

The next step is to define a correct partition between the plant and the controller. The matter is addressed, in this work, in the context of a single-processor system, and of negligible context switch durations; both hypotheses can be relaxed at an acceptable cost, but keeping them in for now eases the treatise. In this context, a physical separation between controller and plant is extremely cumbersome to figure out, but a time-based partition between them is on the contrary very natural. In fact, the (one) CPU is either executing some of the scheduled processes, or the scheduling algorithm. The complete system can thus be viewed as a discrete-time (not sampled-signals, however) one, with a time index being related to the scheduler interventions.

It is now necessary to state what is to be meant for (the model of) "the plant in open loop". In the classical applications of the control theory, think for example of the process or motion control domains, doing so is trivial (at least conceptually). The plant (model) is a system of differential and/or algebraic equations, stemming essentially from the underlying physics, where the inputs (the controller actions) are thought of as exogenous signals. Here, the time-based model partition comes into play: the model of the plant in open loop is a discrete-time system that receives as inputs the results of the "controller" algorithm execution, and returns the results of the "plant" algorithm execution.

The last step, and another peculiarity of the chosen modelling formalism, is that *anything else* but the scheduler action is treated here as an *exogenous* disturbance. This may appear to be a limitation, since for example a resource request is not exogenous at all for the computing system composed of the running processes (meaning that it can be somehow predicted, for example). However, although not exogenous for the running processes (the plant), such a fact is exogenous *for the scheduler* (the controller). Adopting such an attitude, we can take profit of the extremely powerful idea of "disturbance" as thought of in the control theory, i.e., as one of the fundamental reasons for the necessity of feedback.

Consider a single-processor multitasking system with a preemptive scheduler; let N be the number of processes to schedule, that we assume for now constant (some words on the matter will be spent later on). Let the column vectors $\tau_p(k) \in \Re^N$, $\rho_p(k) \in \Re^N$, $b(k) \in \Re^{n(k)}$ and $\delta b(k) \in \Re^{n(k)}$, $1 \leq n(k) \leq N \forall k$ represent, respectively,

- the total CPU times *actually* allocated to the processes up to the beginning of the k-th scheduling round, thus defining (as anticipated) the meaning of k,
- the times to completion at the beginning of the k-th scheduling round for the processes that have a duration assigned (elements corresponding to processes without an assigned duration will be $+\infty$, therefore allowing for the presence of both batch processes and interactive ones),
- the bursts assigned by the scheduler to the processes at the k-th scheduling round,
- the disturbances possibly acting on the scheduling action during the k-th scheduling round,

where $n(k)$ is the number of processes that the scheduler considers at each round (traditionally constant and equal to one—an example of aprioristic constraint that in principle can be relaxed). Denoting by t the total time actually elapsed from the system initialisation, the simplest plant model one can conceive is then

$$\left\{\begin{array}{rcl} \tau_p(k) & = & \tau_p(k-1)+S_\sigma b(k-1)+\delta b(k-1) \\ t(k) & = & t(k-1)+r_1\tau_p(k-1) \\ \rho_p(k) & = & \max\left(\rho_p(k-1)-S_\sigma b(k-1)-\delta b(k-1),0\right) \end{array}\right. \qquad (1)$$

where r_1 is a row vector of length N with unit elements, and $S_\sigma \in \Sigma$ a $N \times n(k)$ switching matrix with elements equal to 0 or 1 and only one 1 per column, assigning the elements of $b(k)$ to the correct processes. Notice that, being $n(k)$ bounded, set Σ is finite for any given N.

As for variations of the process pool, a newly arriving one simply requires to increase N by one, add one zero at the end of τ and one duration (or one $+\infty$) at the end of ρ, and add one row to S_σ. Incidentally, the row position orders the processes by arrival time, which is of interest for some existing and already used scheduling policies. Similarly, altering the "grouping" of the process pool in sub-pools, for example in view of a multilevel scheduling, means acting on $n(k)$.

In synthesis, under the sole limitation (to be possibly relaxed in the future) that n be constant, adding (and obviously removing) processes from the pool simply means formulating a new model in the form (1), that is initialised from the last state of the previous one in a straightforward way. Since process arrivals or terminations are events that occur on a time scale much longer than that of the scheduling task, we concentrate in this manuscript on the constant pool case.

Notice that the first two equations in (1) form a linear, switching discrete-time dynamic system, apart from the obvious input saturation constraint given by the impossibility of negative bursts. The third one is conversely nonlinear, but given the role of disturbances in the adopted framework, a process reaching termination before exhausting its burst is simply modelled as a negative disturbance element on that burst—then the process is of course removed from the pool, see above. Recalling that the relevant fact is here that disturbances are exogenous to the scheduler only, one can therefore safely treat the model as the linear switching one

$$\left\{\begin{array}{rcl} \tau_p(k) & = & \tau_p(k-1)+S_\sigma b(k-1)+\delta b(k-1) \\ t(k) & = & t(k-1)+r_1\tau_p(k-1) \\ \rho_p(k) & = & \rho_p(k-1)-S_\sigma b(k-1)-\delta b(k-1) \end{array}\right. \qquad (2)$$

apart from the mentioned saturation issue.

3 Schedulers as controllers

3.1 Classical scheduling policies

If conditions are imposed to n and/or S_σ, some very common existing scheduling policies are represented by the chosen formalism entirely. For example

- $n = 1$ and a N-periodic S_σ produce all the possible Round Robin (RR) policies having the (scalar) $b(k)$ as the only control input, and obviously the pure round robin if $b(k)$ is kept constant,
- $n = 1$ and a S_σ chosen so as to assign the CPU to the process with the minimum row index and a ρ_p greater than zero produces the First Come First Served (FCFS) policy,
- $n = 1$ and a S_σ giving control to the process with the minimum ρ_p yields the Shortest Remaining Time First (SRTF) policy,
- $n = 1$ and a S_σ that switches according to the increasing order of the initial ρ_p vector produces the Shortest Job First (SJF) policy (notice that this is the same as SRTF if no change to the process pool occurs).

It is important to observe that in the list above the classical "actuators" of the previous works for the mentioned policies are evidenced, but here as entities in a neat system-theoretical framework. Also the state variables of the scheduler, that in the previous works is part of the plant, are clearly defined (examples are that required to switch S_σ in a periodic manner, or to store the initial ρ_p). The chosen formalism hence allows to include also rules that appear very far from it, e.g. owing to the presence of queues. A notable example is the so called selfish round robin (SRR), that is represented by $n = 1$ and S_σ depending on a controller state variable representing the time spent by the corresponding process waiting for the CPU.

However, it is worth further stressing that in evidencing the actuators, constraints had to be imposed on the "pure" plant model, therefore *including in that model something that is actually control, not plant.* Moreover, the resulting "plant including the scheduler" (model) is not only switching but apparently nonlinear, which is *not* true for (2) (we mention the input saturation issue here for the last time, as there are plenty of methods to deal with it while reasoning for the control synthesis in a linear context).

Removing the above mentioned constraints yields therefore two benefits. First, it is a generalisation of current scheduling policies, in a sense that is now characterised. Second, it does not put into the plant model any element that is *de facto* control, and as a result leads to a plant model that can be treated as linear—and in any case, regarding also possible futire extensions, is in nature much simpler than any other one aiming at represent also partd of the control.

3.2 New policies within the same framework

So far, it was shown that the simple modelling framework adopted here can represent classical, well known scheduling policies as variations (better, *restrictions* or specialisations) of a *single* discrete-time dynamic system. The question is then immediately what can be done if *different* specialisations are imposed to the general model. In the opinion of the authors, such an exploration opens a way toward the design of schedulers as dynamic systems, the mentioned "specialisations" qualifying (sub)classes of schedulers in a system-theoretical (not algorithmic) taxonomy.

To illustrate the idea at the present state of the research, two control strategies will be explored in the following, that use a specialisation of the model (2) different from the ones proposed above to replicate existing policies, and by the way introduce control structures that are widely used and very well assessed in other contexts than scheduling.

Let $\tau_r(k) \in \Re$ represent the *actual* duration of the k-th round and let the model not take into account the CPU time required by each process, dealing with process termination and insertion with re-initialisation as will be explained later on. Furthermore, let the S_σ switching signal be an ordered sequence that starts from the first process and proceeds to last one, and consistently be $n = N$. To deal with the possibility of non activating a process (e.g. because the scheduler knows that it has not all the needed resources to execute) the given burst can be zero. In other words, let the "specialisation" affect only the switching signal, so that the result be an LTI discrete-time system that can be controlled by an LTI control structure designed with well assessed methods—maybe the most interesting class of schedulers to study, for sure the simplest from the control-theoretical standpoint. Doing so the model becomes

$$\left\{ \begin{array}{rcl} \tau_p(k) & = & b(k-1) + \delta b(k-1) \\ \tau_r(k) & = & r_1 \tau_p(k-1) \\ t(k) & = & t(k-1) + r_1 \tau_p(k-1) \end{array} \right. \qquad (3)$$

and some remarks are in order.

- Model (3) is linear and time-invariant. Of course negative bursts are not allowed, but that is simply a matter of input saturation, and can be tackled with a number of techniques. Crudely speaking, the control literature has been managing controller design in that way with linear models for decades, so it can be assumed that there is no need here for anything more complex.
- Similarly, each $b + \delta b$ element cannot be negative. This rigorously means that the disturbance is bounded in a time-varying manner, which is however irrelevant for the controller design as far as the disturbance is taken as totally exogenous (i.e., assuming that $b + \delta b$ *could in principle* be negative one considers a set of disturbances wider than the real one, to the apparent validity confirmation of the devised solutions).
- The scheduler is acting not at each process activation, but only once per round. Clearly some b elements can be zero, meaning that not all the process will actually run. Since the proposed scheme allows to control the round duration, system responsiveness issues are implicitly addressed.

The proposed control scheme has the nested loop structure of figure 1. Let τ_r° be the required scheduling round duration, and let

$$\theta_p^\circ \in \Re^N, \qquad \theta_{p,i}^\circ \geq 0, \quad \sum_{i=1}^{N} \theta_{p,i}^\circ = 1 \qquad (4)$$

be the vector containing the required CPU time fractions to be allotted to each process.

First, consider the closed-loop system (CL_1 in figure 1) having as set point the desired CPU times consumed by each process in the current round, i.e., $\tau_p^\circ(k)$, and as controlled variable the CPU times actually consumed in the same round, i.e., $\tau_p(k)$; the control variable is the burst vector $b(k)$, while $\delta b(k)$ is a (vector) load disturbance.

By choosing R_p as a diagonal integral regulator with gain k_{pi}, i.e., $A_{R_p} = C_{R_p} = I_N$, $B_{R_p} = k_{pi} I_N$, $D_{R_P} = 0_{N \times N}$, one makes CL_1 a (diagonal) system the $2N$ eigenvalues of which are N times the couple $0.5 \mp \sqrt{0.25 - k_{pi}}$. More control on those eigenvalues could be achieved with slightly more complex a structure for R_p, but the choice adopted here is adequate for this work, where the scheduling algorithm complexity needs keeping to a minimum.

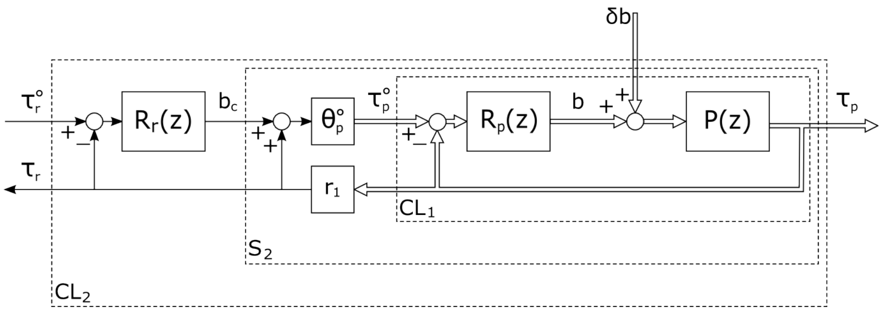

Figure 1: The proposed scheme.

If τ_p° were chosen as $\theta_p^\circ \tau_r^\circ$, then CL_1 would control both the CPU distribution and the round duration, but there would be two problems. First, the dynamics of those two controls would be ruled by the same eigenvalues, which can be inadequate in some cases: for example, one may want the CPU distribution to move smoothly from one situation to another, but the round duration to respond very quickly to its set point, e.g. because a change of that set point means that a greater system responsiveness is needed immediately. Second, and more serious, should some process be blocked, the round duration set point could not be attained.

Consider therefore the system denoted in figure 1 by S_2. Its dynamic matrix is

$$A_{S_2} = \begin{bmatrix} 0_{N\times N} & C_{R_p} \\ \theta_p^\circ r_1 - I_N & A_{R_p} \end{bmatrix} \tag{5}$$

With the chosen R_p, the $2N$ eigenvalues of (5) are 0 and 1 (with multiplicity 1 each) and $N-1$ times the couple $0.5 \mp \sqrt{0.25 - k_{pi}}$. Notice that said eigenvalues do not depend on θ_p°. The SISO system with input b_c and output τ_r seen by R_r in figure 1 has thus the transfer function

$$\frac{T_r(z)}{B_c(z)} = \frac{k_{pi}}{z(z-1)} \tag{6}$$

and in the following two choices are proposed for R_r.

The overall system (CL_2 in figure 1) has the dynamic matrix

$$A_{CL_2} = \begin{bmatrix} D_{R_p}(\theta_p^\circ(r_1 - D_{R_r} r_1) - I_N) & C_{R_p} & D_{R_p}\theta_p^\circ C_{R_r} \\ B_{R_p}(\theta_p^\circ(r_1 - D_{R_r} r_1) - I_N) & A_{R_p} & B_{R_p}\theta_p^\circ C_{R_r} \\ -B_{R_r} r_1 & 0_{N\times N} & A_{R_r} \end{bmatrix} \tag{7}$$

The simplest idea is to choose $R_r(z)$ as a purely proportional controller, i.e., $R_r(z) = k_{rp}$. In this case the $2N$ eigenvalues of A_{CL_2} are those of A_{S_2} with couple (0,1) replaced by $0.5 \mp \sqrt{0.25 - k_{pi}k_{rp}}$. This choice will be termed the "I+P" one from now on, with evident meaning.

In the case of a constant required CPU distribution, the I+P scheme can assign the dynamics of τ_p and τ_r in a (partially) independent manner, having R_r act by means of an additive correction b_c to the round CPU times set point as computed based on the actual round duration. If, conversely, a variable CPU distribution is to be considered, the same scheme can be viewed as a linear switching system with switch signal θ_p°. However, its eigenvalues do not depend on the switching signal, and in force of well known results there surely exists a finite dwell time ensuring the exponential stability of the scheme as switching system.

The I+P scheme is apparently the computationally lightest choice, allows for the simple stability statement above, also permits to give the CPU distribution and the round duration controls different dynamics, but still has the (only) drawback that the round duration control is lost if a process stays blocked (the following examples illustrate the fact). In this scheme the closed-loop transfer function from τ_r° to τ_r is

$$\frac{T_r(z)}{T_r^\circ(z)} = \frac{k_{pi}k_{rp}}{z^2 - z + k_{pi}k_{rp}} \tag{8}$$

thus allowing for a simple choice of the parameters (see the examples later on).

If one cannot guarantee that no persistent process blockings arise, it is advisable to select for R_r a PI structure, i.e., $R_r(z) = k_{rr}(z - z_{rr})/(z-1)$, that can recover such a situation, and leads to what will be termed the "I+PI" scheme. In this case the $2N+1$ eigenvalues of CL_2 have a long expression omitted for brevity, but still do not depend on the switching signal if a variable required CPU distribution has to be assumed. Hence, the same stability considerations above apply. In the I+PI case the closed-loop transfer function from τ_r° to τ_r is

$$\frac{T_r(z)}{T_r^\circ(z)} = \frac{k_{pi}k_{rr}(z - z_{rr})}{z^3 - 2z^2 + (1 + k_{pi}k_{rr})z - k_{pi}k_{rr}z_{rr}} \tag{9}$$

and the parameter choice is just slightly more articulated (again, refer to the following examples).

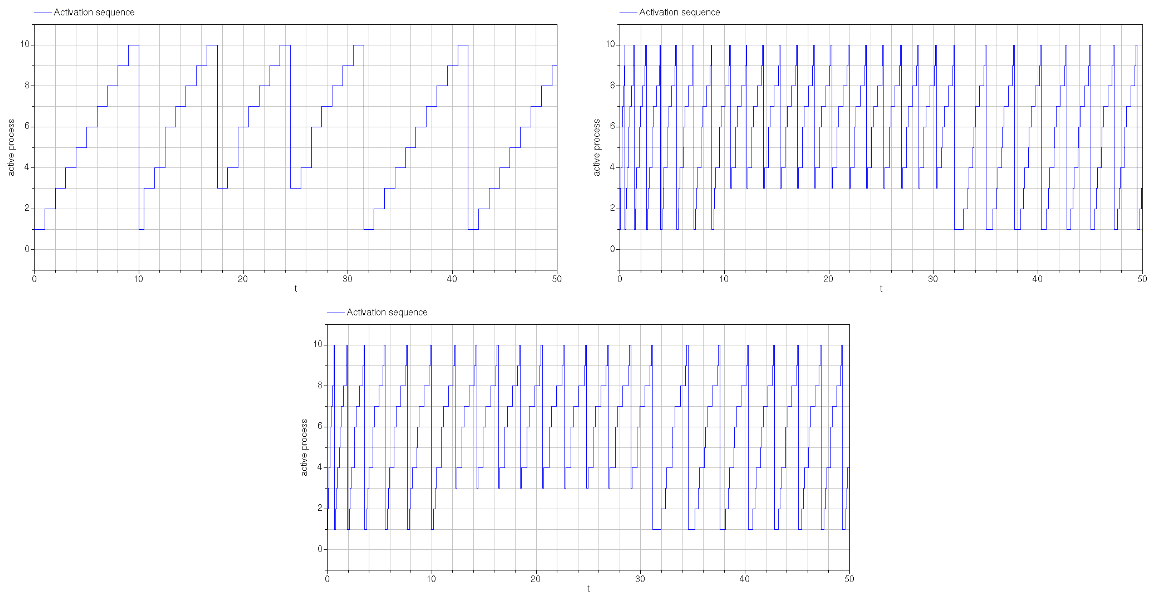

Figure 2: Activation sequence, in the first figure the round robin sequence is presented, in the second one the I+P and in the third one the I+PI.

4 The SkedSim library

The objects to be modelled are the `process`, the `pool` of processes, the `scheduler`, and the complete system (the `world` in our terminology). Since dynamic allocation of objects is not permitted, the pool will simply be a vector of processes, i.e., a fixed number of processes *possibly* being scheduled is allocated, and each of them can be in the scheduling round or not at any given instant.

The process does not *de facto* accomplish anything for the purpose of this research, that is centered on the scheduler's operation; therefore the process is simply a time delay realised by a convenient use of the modelica `when` clause, taking control of the CPU for a time equal to the allotted burst plus a certain variability, and counting the elapsed time by trivially setting the derivative of a convenient continuous variable to one or zero depending on the process having or not the CPU.

The scheduler is in fact the object determining how the switching signal is managed, hence the place where different "actuation" policies are realised. In this manuscript a "round robin" scheduler suffices for all the presented policies, the difference between the standard round robin and the I+P or I+PI schemes residing only in the way the bursts are computed at the beginning of each round.

Thus, each type of scheduler can further specialise its operation by calling a particular *scheduling function*, that implements the specific way of determining the quantities needed to turn an actuation policy into a complete scheduling mechanism. Such a partition between "actuation" and scheduling is consistent with the proposed way of classifying the policies, which in turn corresponds to the system-theoretical idea of viewing them as particular cases of a switching signal, the taxonomy residing precisely in how the switching signal is managed.

Given all the above, the library organisation is very simple and intuitive. To avoid a lengthy treatise, in the following an excerpt of the `process` modelica code is reported.

```
model Process
  ProcessPort p;              // Process/scheduler I/F
  input Boolean blocked;      // Variable for blockings
  discrete Real startTime;    // Last activation time
  Real gCpuPerc;              // Global cpu percentage
  Real gCpuTime;              // Global cpu time
  Real rCpuTime;              // Last round cpu time
algorithm
  when edge(p.activation) then
    p.running := true;
    startTime := time;
    reinit(rCpuTime, 0);
  end when;
  when time>=startTime+p.burst or
       p.burst<=0 or (pre(p.running) and p.blocked)
       then
       p.running := false;
  end when;
equation
  blocked = p.blocked;
  gCpuPerc = if time<=0 then 0 else gCpuTime/time;
  der(gCpuTime) = if p.running then 1 else 0;
  der(rCpuTime) = if p.running then 1 else 0;
  p.gCpuTime = gCpuTime;
  p.rCpuTime = rCpuTime;
end Process;
```

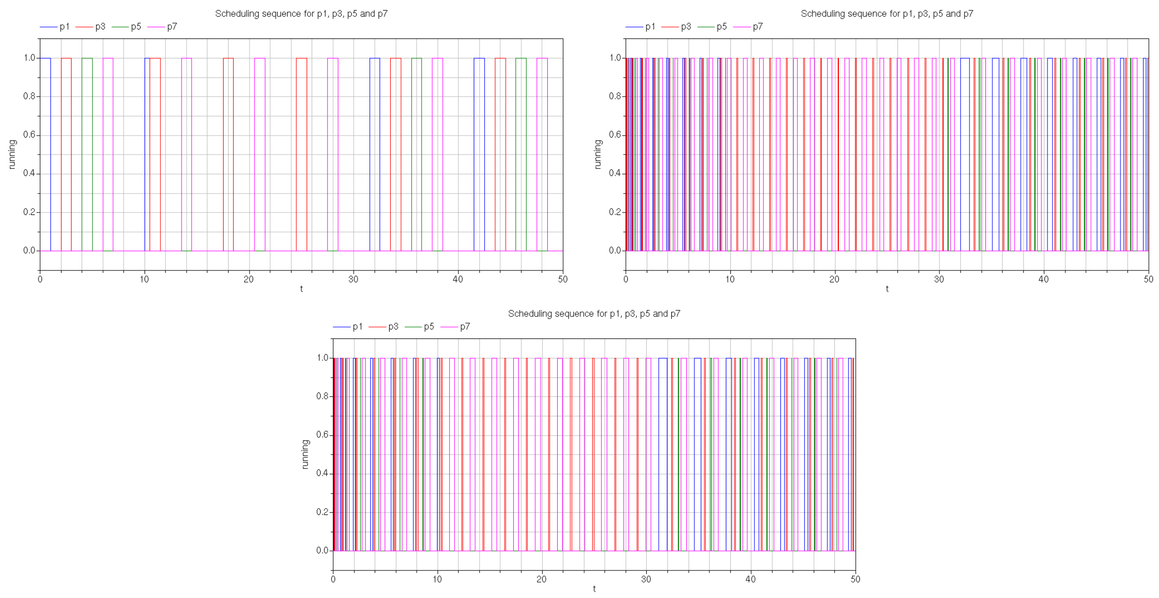

Figure 3: Running time for processes 1, 3, 5 and 7, in the first figure the round robin sequence is presented, in the second one the I+P and in the third one the I+PI.

Moreover,the relevant (algorithm and equation) section of the `scheduler` is shown below.

```
algorithm
 when edge(not_running[previous]) or initial() then
      activation[previous]:=false;
      if previous==nProcesses then
         bursts := Schedulers.SchedulerFunctions
                   .IplusP
                   (nProcesses,SP_Tr,alfa,Tp,bursts);
         // ...or any other scheduling function
         previous := 1;
         roundDuration := time - startTime;
         startTime := time;
         for i in 1:nProcesses loop
           CPUPercPerRound[i]:=if roundDuration<=0
           then 0
           else Tp[i]/roundDuration;
         end for;
       else
         previous:=previous+1;
       end if;
       while bursts[previous]<=0 loop
         previous:=previous+1;
         if previous>nProcesses then
            bursts := Schedulers.SchedulerFunctions
                      .IplusP
                      (nProcesses,SP_Tr,alfa,Tp,bursts);
            // ...same as above
            previous:=1;
            roundDuration := time - startTime;
            startTime := time;
            for i in 1:nProcesses loop
                CPUPercPerRound[i]:=if roundDuration<=0
                then 0
                else Tp[i]/roundDuration;
            end for;
         end if;
       end while;
 activation[previous]:=true;
 end when;

equation
```

```
for i in 1:nProcesses loop
    not_running[i] = not pre(running[i]);
end for;
```

As can be seen, the library organisation and the meaning of the various models are consistent with the specific way of addressing the scheduling problem proposed in this research, and take profit of the possibility (typical of modelica) of mixing algorithmic modeling and asynchronous events. The SkedSim library will be made available to the scientific community under the terms of the GPL license as soon as possible.

5 Simulation examples

This section presents some simulation examples, comparing a standard round robin policy with the two proposed feedback ones, that share the actuation scheme but encompass a more powerful burst computation mechanism. The simulations are conducted with ten running processes. Processes 1, 2 and 5 block from time 10.5 to time 30.5. The round duration set point is 2 for the I+P and I+PI schemes (there is no equivalent in the standard, open-loop one) while the desired percentage distribution is $\{0.1, 0.05, 0.05, 0.2, 0.01, 0.09, 0.2, 0.2, 0.05, 0.05\}$. The quantum for the round robin scheduling is 1 time unit.

Figure 2 shows the activation sequence (i.e., the order in which the scheduler makes the processes run).

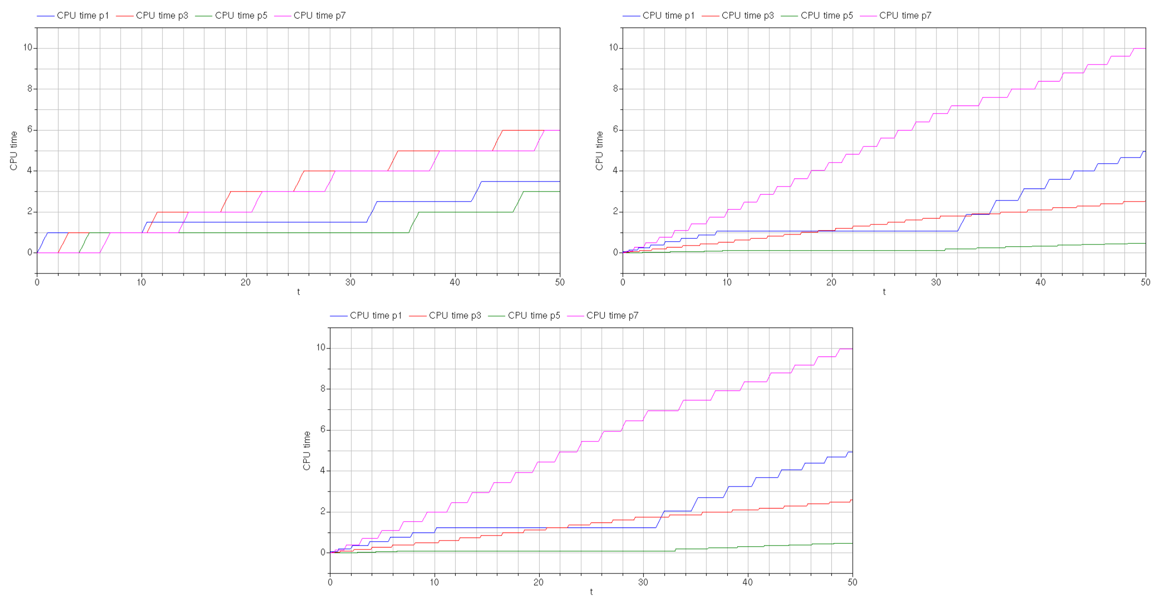

Figure 4: CPU time allotted to processes 1, 3, 5 and 7, in the first figure the round robin sequence is presented, in the second one the I+P and in the third one the I+PI.

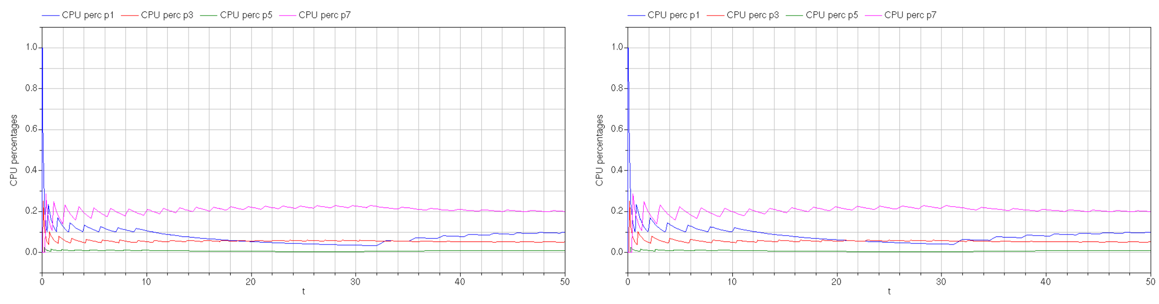

Figure 5: CPU percentages to processes 1, 3, 5 and 7, in the first figure the I+P sequence is presented, in the second one the I+PI.

Figure 3 shows the `running` signal taken from the processes. One can see that in the round robin example the first process was activated at time 10 with a quantum of a time unit but at time 10.5 blocks, releasing therefore the CPU, that is given to process three (the second is blocked too). In figure 4 the times allotted to processes 1, 3, 5 and 7 is shown; the comparison between the round robin and the feedback policies is self-explanatory (the feedback policies allows for a “controlled” distribution, while in the round robin policy no adaptation is possible).

Finally, figures 5 and 6 depict the CPU percentage distribution. The first one shows the distribution over time, while the second one reports the distribution within the current round. It can be seen that even if the percentage per round when the processes are blocked is zero, in the long run the gap is filled thanks to the feedback strategy.

6 Conclusions and future work

In this work, very simple discrete-time control structures were used to synthesise preemptive process schedulers for multitasking systems within a rigorous system-theoretical formalism. In particular, in this paper a Modelica library for the above purpose, at present still under development, was presented, and its use was illustrated with some tests.

Based on the work done until now, the modelica language has shown some advantages when it comes to modelling computing systems. First of all, it allows to integrate discrete dynamics and events (like signals from the system components) with continuous evolution (i.e., the processes’ execution seen in the real world time). Along the same line, thinking of future extensions, not only scheduling-related control strategies can be seamlessly implemented, but it is also possible to test their validity in the case they have

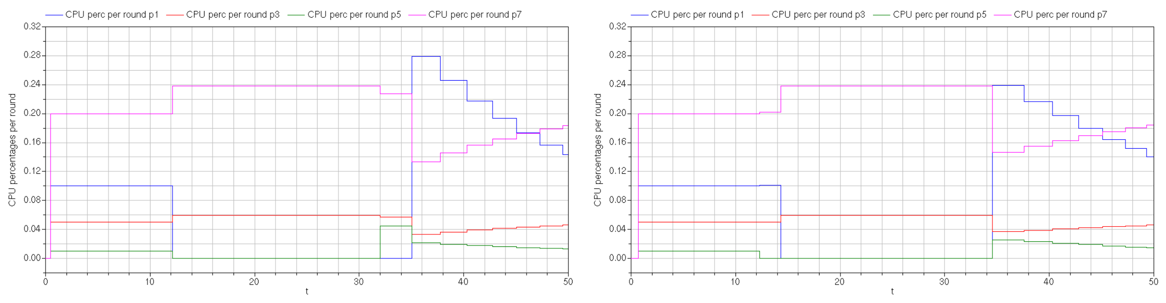

Figure 6: CPU percentages per round to processes 1, 3, 5 and 7, in the first figure the I+P sequence is presented, in the second one the I+PI.

to be used for realising control systems connected to the physical world (as is typically the case for embedded real-time ones, a field that has been devoted much research but is in some sense lateral with respect to this work).

However, the present research has also highlighted some limitations of the modelica use for the chosen purpose. For example, it would be of great interest in this work to investigate the scheduling algorithm and time performance, analysing how much time is spent for the scheduler execution, and therefore having a clue of the expectable system performances. For well known motivations the modelica language is not conceived for such a kind of analysis, however.

For the reasons summarised above, future directions for the presented research still need some reasoning and discussion to be envisaged clearly, but given the positive remarks above, other attempts will certainly be done to overcome the mentioned limitations and employ modelica for the purpose sketched out herein.

References

[1] T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. *IEEE Control Systems Magazine*, 23, 2003.

[2] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole. Analysis of a reservation-based feedback scheduler. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 71–80, 2002.

[3] Ashvin Goel, Molly H. Shor, Jonathan Walpole, David Steere, and Calton Pu. Using feedback control for a network and cpu resource management application. In *Proceedings of the 2001 American Control Conference*, volume 4, pages 2974–2980, Arlington, VA, USA, 2001.

[4] J.L. Hellerstein, Y. Diao, S. Parekh, and D.M. Tilbury. *Feedback Control of Computing Systems*. Wiley, September 2004.

[5] J.C. Palencia and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *In Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[6] M. Pinedo. *Scheduling Theory, Algorithms, and Systems*. Springer, third edition edition, July 2008.

[7] O.H. Roux and A.M. Déplanche. A t-time petri net extension for real time-task scheduling modeling. *European Journal of Automation*, 36, 2002.

[8] David C. Steere, Molly H. Shor, Ashvin Goel, Jonathan Walpole, and Calton Pu. Control and modeling issues in computer operating systems: resource management for real-rate computer applications. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 3, pages 2212–2221, Sydney, NSW, Australia, 2000.

[9] F. Wagner, L. Liu, and G. Frey. Simulation of distributed automation systems in modelica. In *Proceedings of the 6th International Modelica Conference*, Bielefeld, Germany, 2008.