Control and Design of Computing Systems: What to Model and How

Alessandro Vittorio Papadopoulos * **Martina Maggio** **
Alberto Leva *

* *Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34/5, 20133 Milano, Italy. (e-mail: {papadopoulos,leva}@elet.polimi.it).*
** *Lund University, Department of Automatic Control, Ole Römers väg 1, SE 223 63 Lund, Sweden (email: martina.maggio@control.lth.se).*

Abstract. The application of feedback control to computing systems is a promising research area, but has to date been hindered by the almost unanimously perceived complexity in creating control-oriented system models. Computing systems are in fact considered very hard to describe with dynamic models allowing for simple and powerful control design tools, so that complex ones need bringing in to the detriment of efficiency and result assessment. In this work a novel approach to the modelling of computing systems is proposed, in a view to explain and partially avoid such complexity, by capturing only their relevant dynamics with the simplest possible models. The approach is shown to work at least on two relevant case studies, so that a significant generality can be inferred from it, being able to reproduce the relevant parts of the system's behaviour and paving the way to control design and synthesis.

Keywords: Computing systems; feedback control; scheduling; resource allocation.

1. INTRODUCTION

The complexity of many computing system functionalities is nowadays abruptly increasing. To give just one example, consider the Linux scheduler. In the Kernel version 2.4.37.10 (September 2010) all of its code was contained in a single file of 1397 lines. In version 2.6.39.4 (August 2011) the scheduler code is spread among 13 files for a total of 17598 lines. Other examples could be given, but are omitted for space limitations.

Indeed, when such "explosions" are experienced, the overall design approach is to be somehow reconsidered. Observing the matter from a modelling-oriented standpoint, and not limiting the scope to the scheduler example, it can be noticed that hardly any computing systems functionality has been conceived and developed based on a dynamic model of some physical phenomenon to be controlled.

In the scheduler case, to stick to the example, the phenomenon is how the CPU is distributed among the running tasks, depending on control actions (the allotted timeslices) and exogenous disturbances (task blockings, resource contentions, and so on).

The situation just sketched has quite clear historical reasons. Suffice to say that, while in any other context controlled objects can be modelled based on physical (first) principles, this is not the case for computing systems, because there the "physics" is created by the system designer him/herself.

In the absence of a modelling framework, system design is carried out directly in an algorithmic setting, leaving the engineer without any means to assess its behaviour in the sense that term is given in the system and control theory domain.

While such a *scenario* could to date be tolerated, given the mentioned complexity rate increase, it cannot be assured that said tolerability will carry over to the future.

In fact, as "more physics" is created, the absence of a rigorous dynamic description of it may sooner or later pose intractable problems as for its governance. As a consequence, rigorous – and possibly simple – modelling frameworks to ground system design upon are needed.

The main message this paper wants to convey, is that if one accepts to re-design part of said system, such a framework can be found by (usefully) limiting the model scope to describing the real physical phenomenon on which the addressed aspects of the system behaviour depend. If this is done, surprisingly simple formalisms can be used—a noticeable example indeed of process/control co-design, and a relevant step forward with respect to previous research as presented e.g. in Hellerstein et al. (2004).

This paper concentrates on the modelling side of the problem, by showing the ideas above at work. Some words are spent on the consequent advantages in terms of system (and control) design, leaving however the matter to other works.

2. THE QUEST FOR A PHYSICS

At the very core of any computing system behaviour there is some strictly physical phenomenon. For example, in the case of an operating system scheduler, that phenomenon has the form $accCPU(k) = accCPU(k-1) + burst(k) + disturbance(k)$, where $accCPU$ is the CPU time accumu-

lated by a task, *burst* is the CPU timeslice allotted to the task, and *disturbance* accounts for any difference between $burst(k)$ and the actual CPU use by the task. A similar model can be obtained considering the present state of an application's progress toward its final goal which depends on that at the last resource arbitration instant and on the allotted resources at that instant; other examples can be found but the obtained models are almost invariantly very simple.

In any case, what happens ultimately depends on extremely fine-grained facts, down to the detailed behaviour of any single assembler instruction and electronics transient. This makes computing systems different from most other objects to model, since in them, the fine-grained one is often the only physical level that can be rigorously defined. In thermal controls, for example, one can avoid treating fine-grain phenomena (in that case, molecular motions) since there exist suitable macro-physic entities (e.g., temperature or enthalpy) that allow to write rigorous balances (e.g., of energy) to base dynamic models upon.

In the development of computing systems, in addition, no set of "first principles" has *de facto* ever been sought. Sticking again to the scheduling example, *action* policies are typically defined as "give the CPU to the task with the earliest deadline" by foreseeing their effect in some nominal conditions (for a schedulable task pool, doing so there will be no misses).

In the addressed domain, in other words, there is classically no distinction among the behaviour of the system in the absence of such actions, the desired behaviour of the same system, and the way actions are to be determined based on the above. There is no evidence, in other words, of the fundamental elements of a (control-oriented) modelling process.

Deepening the analysis, one may object that many works deal with computing system control, and do use control-theoretical methodologies. This is true, but virtually all of them take the computing system *as is* and close loops around it (e.g., aiming at a certain CPU distribution by altering task deadlines). Doing so however requires to model the core phenomenon *plus all the "created physics" around it* (e.g., the existing deadline-based scheduler).

In the authors opinion, the presence of such "unconsciously created" physics is a major reason for the complexity of most computing systems' models, at least as far as the ultimate scope of said models is to design parts of those systems in the form of controllers. To circumvent the problem, one should thus in the first place evidence the core phenomenon, i.e., that part of the system behaviour that really relies on physics and cannot be altered (the examples later on will clarify). Most often, modelling that phenomenon is enough to describe the system in a view to suitably control it. In some cases, in addition, the so obtained models will be natively (almost) uncertainty-free, making control design and assessment very straightforward. In other cases, there may be relevant uncertainty, or – in other words – some aspects of the system behaviour will not admit a clear physical interpretation. In such cases, the advice is to figure out some convenient grey box description based on qualitative considerations on input-output relationships. As will be shown, this approach generally leads to more complex but still tractable models: control design may be correspondingly harder, but still there will be the possibility of a rigorous assessment.

In the following, some examples are shown of how the proposed approach leads to dynamic models of computing system components that can successfully serve the evidenced needs, while being very simple and thus suitable for powerful and rigorous analysis and control result assessment.

3. EXAMPLES AND APPLICATIONS

3.1 A unified framework for task scheduling

This section shows how the task scheduling in a preemptive single-processor system can be fully treated having as model class that of discrete-time dynamic systems, in some cases even linear and time-invariant. A few words are also spent on the natural attitude of said modelling formalisms to scale up towards, for example, multicore or multiprocessor contexts, where any other modelling formalism and design approach do experience severe difficulties.

Consider a single-processor multitasking system with a preemptive scheduler, preemptive meaning that the scheduler can interrupt the current task and substitute it with another one. Let N be the number of tasks to schedule. Define the *round* as the time between two subsequent scheduler intervention. Let the column vectors $\tau_p(k) \in \mathbb{R}^N$, $\tau_r(k) \in \mathbb{R}$, $\rho_p(k) \in \mathbb{R}^N$, $b(k) \in \mathbb{R}^{n(k)}$ and $\delta b(k) \in \mathbb{R}^{n(k)}$, $1 \leq n(k) \leq N \, \forall \, k$ represent, respectively,

- the CPU times *actually* allocated to the tasks in the k-th round,
- the time duration of the k-th round,
- the *times to completion* (i.e., the remaining CPU time needed by the task to end its job) at the beginning of the k-th round for the tasks that have a duration assigned (elements corresponding to tasks without an assigned duration will be $+\infty$, therefore allowing for the presence of both batch and interactive tasks),
- the *bursts*, i.e., the CPU times allotted by the scheduler to the tasks at the beginning of the k-th round,
- the disturbances possibly acting on the scheduling action during the k-th round (for example because one of the tasks release the CPU before its burst has expired or because of an interrupt management amidst the task operation),

where $n(k)$ is the number of tasks that the scheduler considers at each round. In the traditional scheduling policies $n(k)$ is constant and equal to one—an example of aprioristic constraint that in principle can be relaxed, maybe resulting in better performances. Denote by t the total time actually elapsed from the system initialisation.

A very simple model for the phenomenon of interest is then

$$\begin{cases} \tau_p(k) = S_\sigma b(k-1) + \delta b(k-1) \\ \tau_r(k) = r_1 \tau_p(k-1) \\ \rho_p(k) = \max\left(\rho_p(k-1) - S_\sigma b(k-1) - \delta b(k-1),\, 0\right) \\ t(k) = t(k-1) + \tau_r(k) \end{cases} \tag{1}$$

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. The elements of S_σ are zero or one, and each column contains at most one element equal to one. Matrix S_σ determines which tasks are considered in each round, to the advantage of generality (and possibly for multiprocessor extensions). Notice that, since $n(k)$ is bounded, the set Σ is finite for any N.

Several scheduling policies can be described with the presented formalism, by merely choosing $n(k)$ and/or $S_\sigma(k)$. For example

- $n = 1$ and a N-periodic S_σ with
$$S_\sigma(k) \neq S_\sigma(k-1), \quad 2 \leq k \leq N \tag{2}$$
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,
- generalisations of the RR policy are obtained if the period of S_σ is *greater* than N, and (2) is obviously released,
- $n = 1$ and a S_σ chosen so as to assign the CPU to the task 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 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 task pool occurs),
- $n = 1$ and a S_σ selecting the task with the minimum ρ_p yields the Shortest Remaining Time First (SRTF) policy.

The capability of model (1,2) to reproduce the mentioned policies is shown in Figure 1, in the case of $n(k) = 1$, $N = 5$, and $S_\sigma(k)$ chosen as described above.

In all these policies, the core phenomenon can be noticed in the form $\tau_p(k) = S_\sigma b(k-1) + \delta b(k-1)$. Also the "added physics" can be noticed, as the algorithm used to select $n(k)$ and/or $S_\sigma(k)$.

If one attempts to model both things together, to close the loop around the existing scheduler, then switching systems must be brought into play.

If, on the contrary, one models the core phenomenon only, and treats all the rest as part of the controller, the single and trivial equation just written is enough. Notice that here in modelling the core phenomenon no uncertainty is present, nor is there any measurement error, since the only required operation is to read the system time.

Based on model (1) one can thus abandon "non control theoretical" (and often not even closed-loop) choices of S_σ as in the examples just sketched, and synthesise schedulers as controllers with very simple blocks, for example of the PI or Model Predictive Control type (Leva and Maggio, 2010; Maggio et al., 2012).

3.2 A unified framework for resource allocation

This section shows that, also in the case of resource allocation, a core phenomenon can be identified and modelled. In this case, however, uncertainties are generally present but if one installs additional sensors in the system so as to measure exactly what pertains to the core phenomenon, the resulting models are still much simpler and reliable than those obtained by attempting to describe the system as is.

The resource allocation problem consists in dynamically modifying the amount of system resources (memory, bandwidth, number of computing units and so forth) allotted to an application, in such a way the said application progress towards its goal at the desired rate. For example, one may want a video encoder to process exactly 30 frames per second, despite different amount of computational resources needed by the individual frames, and the overall system load. Quite intuitively, the progress rate – that in this work is measured in WorkLoad Units (WLU) per second – is defined on a per application basis (e.g., for a video encoder it could be the completion of one frame).

In most cases, however, a measure of the mentioned progress rate is not available, since usually hardware performance counters are used Kufrin (2005); Sprunt (2002). The relationship between the progress rate and typically measured quantities is another clear example of added physics – or better, in this case, physics that should not be in the control loop – as the core phenomenon is here "how the progress rate dynamically reacts to resources".

On a time scale suitable for evaluating (and possibly controlling) an application behaviour, the effect of allotting more or less resources to it is practically instantaneous. However, the efficacy of a given resource on the application progress may vary over time. For example, if an application is presently executing operations that do not require parallelism, the effect of allotting more computational units is modest. Similar considerations hold for memory or other resources.

Contrary to the remark above, the time scale of resource-to-performance effects is almost invariantly comparable to that suitable for monitoring and controlling.

Therefore, if one accepts to introduce a progress rate measurement, it turns out that many relevant problems can be treated with discrete time nonlinear dynamic systems of simple structure, obtained with a grey box approach.

For example, when the resources to allot are computational units c and clock frequency f while the application progress rate pr is measured with the Application Heartbeats framework (Hoffmann et al., 2010), a vast campaign of experiments and data analysis indicated that a suitable model is

$$\begin{aligned} pr(k) &= p \cdot pr(k-1) + \\ &+ (1-p) \cdot (k_c\, c(k-1)^{\alpha_c} + o_c)(k_f\, f(k-1)^{\alpha_f} + o_f) \end{aligned} \tag{3}$$

where parameter $p \in [\,0, 1\,)$ is essentially related to the sampling time used for the performance measurements, thus not application specific; the other (time varying) parameters account for resource response of the application. Note that (3) contains a nonlinear static (multi-)input characteristic cascaded to a linear dynamics, in accordance with the idea that the control time scale is very slow

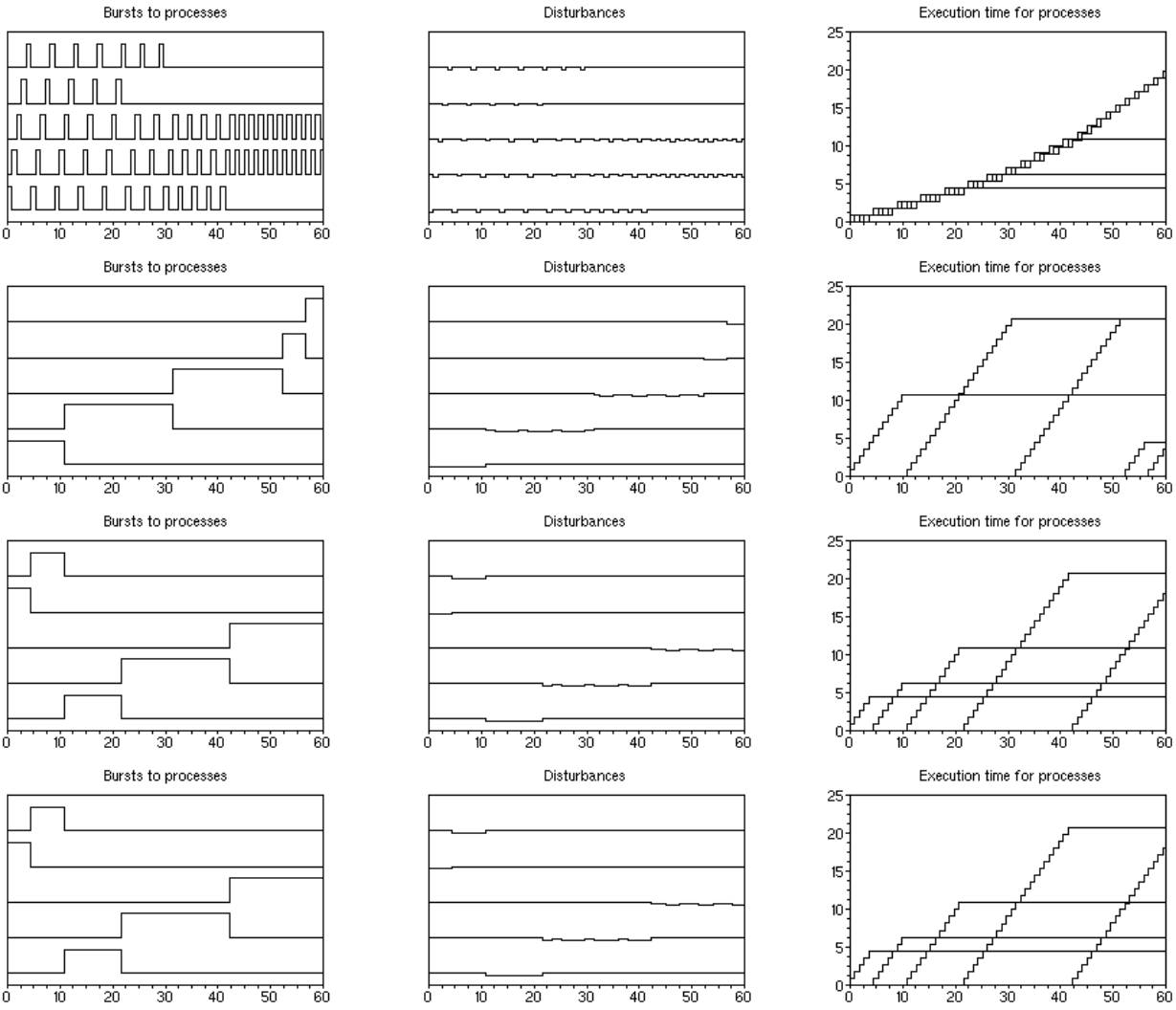


Figure 1. Capability of the presented *single* model of reproducing classical scheduling policies such as RR, FCFS, SJF, and SRTF.

with respect to the actuation one, and complexity resides essentially in the actuators' influence on the process.

In fact, in most of the addressed situations, parameter p (the discrete-time pole) typically takes low values in the 0–1 interval, indicating that at the control time scale, the action of actuators is nonlinear but practically instantaneous. Some exceptions may arise for example when some actuating action requires to negotiate resources with the operating system, e.g. posting requests that may be fulfilled at a time scale comparable to that of control, but nonetheless the modelling hypotheses introduced hold reasonably true in all the cases of interest, and in most of them the system to be controlled actually behaves as a nonlinear static one cascaded to a pure one-step delay.

As a further possible objection, then, application behaviour variabilities can be present and depend on many factors, including for example the processed data, so the proposed modelling approach may not seem very useful.

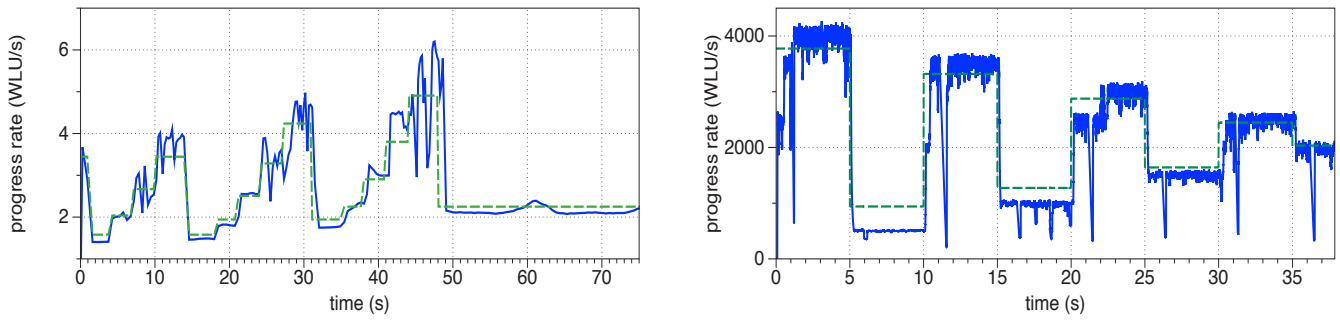
To explain why on the contrary it is, consider the following. With a sufficiently wide (yet in general affordable) number of profiling tests, one can obtain range and rate bounds for parameter variations.

By generating parameter behaviours based on that information, one can then simulate a potentially infinite number of possible application behaviours in much less time than the same number of real runs would require, which is very useful in a view to synthesise and assess controllers.

Notice that attempting to do the same thing with classical black-box identification applied to linear models – a widely used approach – is for the problem at hand less effective, as such models are structurally inadequate, and any order selection procedure would eventually produce very complex structures.

Needless to say, reverting for a moment to control, the simplicity of (3) – once that model was tested for the capability of actually replicating application runs – suggests correspondingly simple regulators, contrary to what one would conclude based on standard black-box models.

Coming to some examples referring to benchmark applications, Figure 4(a) shows the `bodytrack` measured progress



(a) `bodytrack` (b) `vips`

Figure 2. Collected data from the specified software application (solid blue line) and simulation with the grey-box identified model (dashed green line).

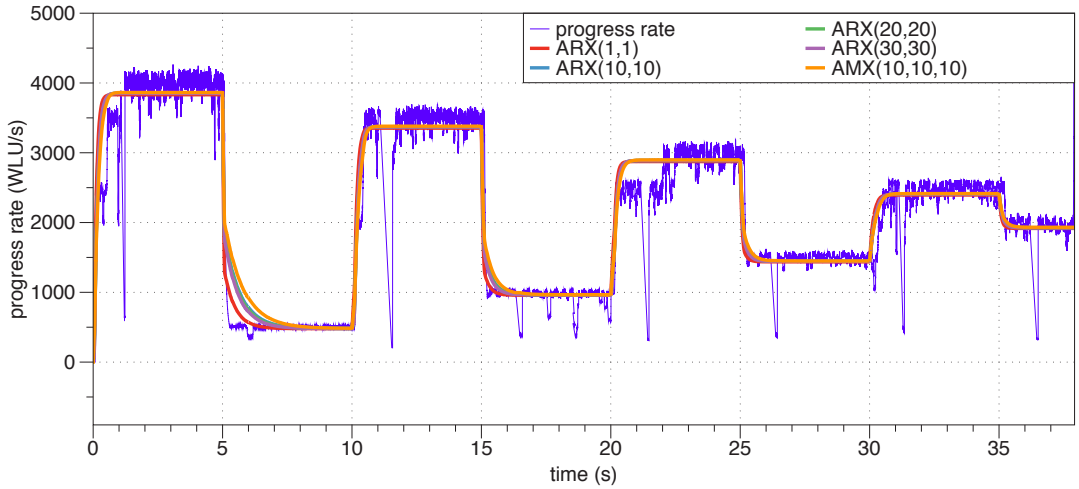


Figure 3. Identification results for the vips software application with different model structures, namely $ARMAX(10,10,10)$, $ARX(30,30)$, $ARX(20,20)$, $ARX(10,10)$, and $ARX(1,1)$.

rate and the one estimated with the identified simulation model (3) for a particular run, where the parameters' behaviour was obtained by means of an Extended Least Squares procedure.

Figure 3 conversely shows the outcome of model (3) together with some ARX (AutoRegressive with eXogenous input) and $ARMAX$ (AutoRegressive and Moving Average with eXogenous input) ones for a run of the `vips` application. The data used for the identification are denoted in black, the simulation results of the $ARMAX(10,10,10)$ in violet, the $ARX(30,30)$ in light blue, the $ARX(20,20)$ in red, the $ARX(10,10)$ in green and the $ARX(1,1)$ in blue.

The `bodytrack` and `vips` applications are taken from the PARSEC benchmark suite. The *rationale* behind the suite, together with its use, is presented in Bienia et al. (2008), to which the interested reader is referred for details.

Figure 4(a) illustrates that (3) is actually capable of replicating the data, by catching main variabilities and trends in a way suitable for control design—its sole purpose here.

Figure 3 also suggests that $AR(MA)X$ models are not keen to capture the relevant application behaviour. In fact, if one tries to identify the same data with the Matlab Identification toolbox, performing an order selection for the $ARX(n_a, n_b)$ model, the result is that the identification procedure tries to give to the model as much higher an order as it can, indicating that the *structural* choice is not adequate.

For completeness, the grey box model (3) used in the presented examples, denoting with $\hat{\vartheta}$ the estimated parameter vector, is parametrised for `bodytrack` as

$$\hat{\vartheta}_{vips} = \begin{bmatrix} k_c \\ \alpha_c \\ o_c \end{bmatrix} = \begin{bmatrix} 258.75388 \\ 1.1930687 \\ 681.67218 \end{bmatrix}, \tag{4}$$

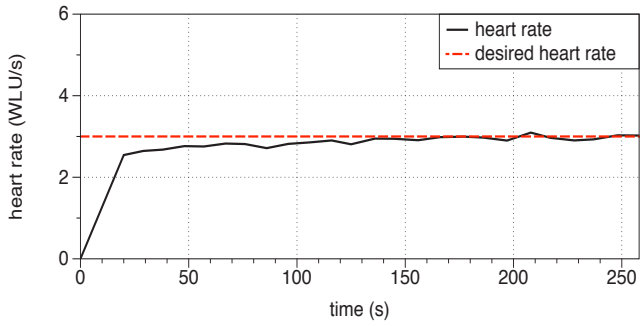
and for `vips` as

$$\hat{\vartheta}_{bodytrack} = \begin{bmatrix} k_c \\ \alpha_c \\ o_c \\ k_f \\ \alpha_f \\ o_f \end{bmatrix} = \begin{bmatrix} 0.1931659 \\ 1.613834 \\ 3.5964752 \\ 2.3736936 \\ 0.1609101 \\ -1.9965658 \end{bmatrix}. \tag{5}$$

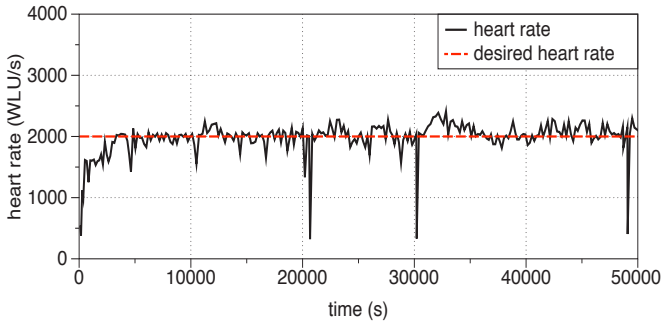
In addition, by introducing a fit measure, the obtained models can be ranked. In the example proposed herein, said fit expression is set to

$$\left[1 - \frac{\|Y - \hat{Y}\|_2}{\|Y - \bar{Y}\|_2}\right] \cdot 100 \tag{6}$$

and Table 1 shows the obtained results in the `vips` case.



(a) `bodytrack`



(b) `vips`

Figure 4. Experimental control results with `bodytrack` and `vips`: the application progress (heart) rate is required to attain a specified set point value.

Model	Delay	Best Fits
$ARMAX(10, 10, 10)$	1	62.24
$ARX(30, 30)$	9	61.63
$ARX(20, 20)$	9	61.53
$ARX(10, 10)$	9	61.36
$ARX(1, 1)$	1	58.98

Table 1. Results obtained with the Matlab Identification Toolbox for the `vips` application with various model structures.

Notice that, starting from the system insight induced by the grey box model, successful adaptive control could be achieved with an $ARX(1, 1)$ structure.

To end this section, although this paper focuses on modelling and not on control, just a minimum example is presented on what can be achieved in that respect. Figure 4 shows experimental results with both the `bodytrack` and the `vips` applications, when their progress – measured via HeartBeats – is regulated by an adaptive predictive controller based on the presented model.

As can be seen, the required set point is well attained also in the presence of application behaviour's variations, thus proving the effectiveness of the underlying modelling approach.

4. CONCLUSIONS AND FUTURE WORK

In this work a novel approach to the modelling of computing systems was proposed, in a view to capture their relevant dynamics with the simplest possible models, grounded on some "physical" principles.

The two cases just shown are only a small example of the model usefulness improvement yielded by the idea of isolating the core phenomenon.

Although the focus of this work is not set on control, and in accordance with that only a very short example on the matter was shown, undoubtedly control is maybe the main reason why the proposed approach should be adopted. In fact, starting from the core phenomenon and its desired behaviour is indeed a mean for streamlining both control and system design, to the advantage of a better operation of their compound.

The so envisaged *scenario* is quite neat an example of process/control co-design. In the opinion of the authors, the characteristics of the computing system domain make it a privileged arena for co-design, and it is quite curious that so promising a matter has not been exploited to date.

Of course much further work is required, but in any case an innovative attempt was here made to circumvent one of the main obstacles for co-design success.

Along this research line, future developments can be foreseen as the application of the presented ideas to other computing system problems, like for example bandwidth allocation, and their exploitation through the application of simple (thus computational lightweight) control techniques.

REFERENCES

Bienia, C., Kumar, S., Singh, J.P., and Li, K. (2008). The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*.

Hellerstein, J.L., Diao, Y., Parekh, S., and Tilbury, D.M. (2004). *Feedback Control of Computing Systems*. Wiley.

Hoffmann, H., Eastep, J., Santambrogio, M.D., Miller, J.E., and Agarwal, A. (2010). Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, 79–88. ACM, New York, NY, USA. URL `http://doi.acm.org/10.1145/1809049.1809065`.

Kufrin, R. (2005). Measuring and improving application performance with perfsuite. *Linux Journal*, 2005, 4–10.

Leva, A. and Maggio, M. (2010). Feedback process scheduling with simple discrete-time control structures. *Control Theory Applications, IET*, 4(11), 2331–2342. doi:10.1049/iet-cta.2009.0260.

Maggio, M., Papadopoulos, A.V., and Leva, A. (2012). On the use of feedback control in the design of computing system components. *Asian Journal of Control*. doi: 10.1002/asjc.509. (in press).

Sprunt, B. (2002). The basics of performance-monitoring hardware. *Micro, IEEE*, 22(4), 64–71. doi: 10.1109/MM.2002.1028477.