SMART Computing Systems: Sensing, Modelling, Actuating, Regulating, and Tuning

Martina Maggio[1], Alessandro V. Papadopoulos[1,2] and Alberto Leva[2]

[1]Department of Automatic Control
Lund University, Sweden
{martina.maggio,alessandro.papadopoulos}@control.lth.se

[2]Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
{papadopoulos,leva}@elet.polimi.it

ABSTRACT

Control-based decision mechanisms are nowadays exploited as a viable tool to obtain reliability and adaptivity in complex computing systems. However, due to the complexity of these systems, a lot of effort is often devoted to figuring out what strategy is the best fit for the problem. This leads to *ad hoc* solutions that are perfect for a specific issue but are not keen to be reused. In this position paper, some reasoning is presented on one approach that can be followed when dealing with a control problem in complex computing systems. The framework is named SMART, which stands for Sensing, Modelling, Actuating, Regulating and Tuning. The considerations discussed herein come from experiences with feedback scheduling and feedback "application-aware" resource allocation. Relevant case studies are referenced for the interested reader.

Categories and Subject Descriptors

D.2.10 [**Design**]: Methodologies; I.2.8 [**Problem Solving, Control Methods, and Search**]: Control theory

General Terms

Design, Performance

Keywords

Resource allocation, Feedback control, Design approaches

1. INTRODUCTION

In recent years, a significant convergence can be observed between the need for "self adaptation" capabilities, strongly evidenced in the computer science community, and the possible responses coming from methodologies developed in the control domain [4]. Examples where controllers are introduced to close loops in computing systems can be found in various research areas, however general solutions and design frameworks are hard to find. A few exceptions are [5,10]. The idea behind this paper is to investigate the general methodology to build "controllable" computing systems, a matter that often requires redesigning part of the system. The methodology that will be presented in the following can be applied to a lot of different problems. In some of them this approach has already been applied, for example task scheduling [7] and application-aware resource allocation [9]. In this latter case, one may think about designing an application specific allocator, however, the goal is to be as general as possible and a strong theory is required to pursue this objective.

To further motivate the need for generality, it is worth evidencing where it is usually missing. The first place is the "sensing" part of the computing/control system, as most available solutions in fact rely on architecture-specific performance measurements. Think for example to a specific web server controller that measures the number of users that are active in the system or the number of connections that are kept alive. The type of measures that are obtained from that are definitely application specific and are not suitable for another class of applications, say multimedia manipulators. Also, one could think about reading an operating system value in a Unix environment or in a different one. Often, these operations are done with different functions and not necessarily the same information is available in both cases. This leads to fragility and jeopardises portability to other systems, if not even the capability of correctly quantifying the required metrics. To justify this claim, think about the case when the progress of an application towards its goal is to be evaluated. In such a scenario, measuring the number of instructions executed in a period of time does not tell whether those instructions were doing useful work or just spinning on a lock. Similarly, measuring CPU utilisation or cache miss rates has drawbacks and does not allow for generalisation.

The second place is the "actuation" part. Also in this case, most solutions are hand-crafted for a particular computing platform, and exert their actions on the phenomenon to be controlled with an extremely wide variety of mechanisms, from hardware-based timing to system calls, and more. One example could be dropping connections in a web server, that is not portable to other platforms or class of applications. The clarify, the claim here is that it should be possible to generalise all these specific solutions, without losing the benefits of their level of detail. Abstraction is needed to deal with actuators and is required to cast different knobs into a unitary method.

The third place is the "modelling" of the mentioned core phenomenon. Also here, available solutions seem to be more related to the application-specific sensing and actuation poli-

cies than to a methodologically grounded description. As a consequence, the design of the control law is far from being easy. Often, this design requires black box modelling, which means treating the system as an unknown cloud of dust that needs to be sorted in some sense. Almost the totality of the contributions are tied to specific problems and not keen to solution reuse, examples being [1–3, 6]. On the contrary, the proposed framework inherently fosters a structured, unitary, and reusable control system design. Intuitively, this can lead to worse performance in specific solutions where a specifically designed strategy is able to perform better. However, when generalisation is needed, following the framework allows for great advantages.

2. THE SMART DESIGN FRAMEWORK

SMART originates from the idea of building components of computing systems as controllers. In such a way, one does not need to make "adaptive" an existing component but simply designs the component itself as a controller. A clear example is a scheduler that is made self-adaptive by choosing the parameter of a policy, for example the quantum of a round robin. One could do that and try to formally assess the behaviour of the system, proving stability and some convergence properties. However, replacing the entire scheduler with a controller, modelled in terms of equations, allow to obtain much more in terms of formal guarantees. This is in general true in the case of resource allocators, that are inherently "controllers" in nature.

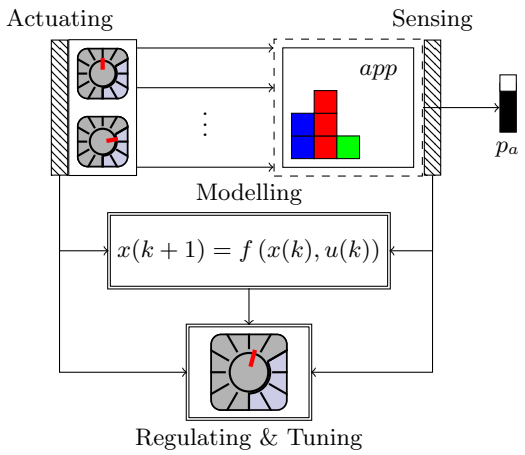

Figure 1: The SMART Framework.

Figure 1 shows the *rationale* of SMART, which is a clear division of the system components in "sensing, modelling and actuating" ones, to separate the problems as much as possible. As can be seen, the figure represents an application that is received some amount of resources. Noticing that the picture is completely general, for simplicity it is now cast into a specific application case. Suppose, in fact, that the blue resource is CPU time, the red resource is memory pages and the green resource is disk space. During its life, the application will require different amounts of these resources that are managed by knobs or actuators. The interfaces, depicted by blocks filled a with north west lines pattern, transform the behaviour of the application into some performance measure p_a and the action chosen by a regulator into operating system resource quantities, such as memory and disk space. Moreover, modelling is necessary to define the "expected relationship" between the actuator values and the performance measure. Grey box macro models are here used to capture the controlled phenomena without including unnecessary system parts. By unnecessary we here mean those parts that can be excluded from the original system when the control components are redesigned. Recalling the scheduler example, the controller with adaptive parameters is some part that the designer can get rid of if the scheduler is now designed as a feedback controller from the very beginning. No adaptation is required in this case, since the adaptation mechanism is the nature of the controller itself. If this modelling phase is carried out thoroughly, the "M" part - i.e., the model to be used for designing the control law - quite often turns out to be very simple.

Starting from the model – and not needing the interfaces anymore, therefore decoupling the two phases – a controller can be designed, using methods that are suitable for the specific problem. For example, a deadbeat controller can be used whenever a fast response is needed and disturbance rejection is not of primary importance. Similarly, a proportional integral controller can be implied when the steady state error should be driven to zero and a model predictive controller is handful whenever a complex situation has to be dealt with. Many techniques can be employed to build the regulator that better fits the problem. Having designed and parametrised that law, the "RT" part is obtained, while the mentioned system partition directly reflects in the "S" and "A" ones.

2.1 Interfacing with the computing system

In computing systems, actuators (in the control engineering jargon, the entities that physically act on the object to be controlled) are extremely heterogeneous and sometimes do not even correspond to any well defined physical entity. An actuator could be the memory allocated to a specific process, the number of cores or the CPU time the process is obtaining, the CPU frequency, the nice number that drive the scheduling algorithm, and so forth. In some cases, operating an actuator directly means modifying some physically meaningful quantity, but this is not always true. For example, the nice number has an extremely indirect action on the running processes, being in fact little more than an abstraction. Also, the efficacy of actuators is extremely variable, and virtually impossible to predict. If an application switches from a CPU-bound to a memory-bound behaviour, the efficacy of the "number of cores" actuator will drop. The variability of actuators' efficacy occurs at a time scale dictated by the application code, and sometimes the processed data, in such a way that only the application itself could possibly notify the control system of which actuator is effective and which is not. Envisaging such an application role is however quite unrealistic. It is necessary to design controllers in the absence of any information on a very uncertain actuator efficacy.

A second peculiarity refers to sensors. Contrary to most other control domains, measurements are practically error-free, but their relationship with the used performance metrics is sometimes quite indirect. As a result, the way the system is "instrumented" is crucial. Suppose, for example, that the performance metric is the amount of CPU time allotted

to an application. By introducing suitable timestamping operations, one can measure the time elapsed between two chosen events with virtually no error. However, there is no certainty that all the measured time was really allotted to the application, unless it was in a critical section, as interrupts might have intervened. To eliminate such problems, one must introduce "sensors" at all the involved levels, i.e., in the example, both at the system and the application levels. Moreover, and again concerning sensing, in classical control problems, once a control-relevant quantity is identified, setting up the sensing mechanism for it is mostly a matter of technology. Static and dynamic errors may have to be dealt with, but in no sense the *nature* of the measured quantity is questioned. For example, if a temperature is to be controlled, no doubt (unless in an irrelevant minority of cases) it is among the quantities to measure. Depending on the particular problem, a thermocouple or a resistive or an infrared sensor could be convenient, and there may be precision and response speed issues, but the *role* and the *meaning* of that temperature in the control problem is totally unambiguous.

In computing system problems, and particularly in the case of resource allocation, things are far more blurred. The control problem may not naturally lead to identify the quantities to be measured, and their dependence on the quantities to be controlled may not be unambiguous. Suppose for example to employ a cache miss sensor that provides the amount of misses an application is experiencing. Obviously enough, the measurement provided by that sensor is related to the application speed but it is definitely *not* a direct measure of it.

2.2 A multi-level physics

Elaborating on the ideas above, a third peculiarity emerges. In computing systems, at least two time scales are evidenced. A "micro" scale is at the instruction level, where most of the unpredictability resides. The other is at the control level, where what is measured is typically the averaged result of the micro-scale behaviour over conveniently defined time spans (for example, the throughput of a server in the last second). Most reasonably, one should model the system at this second time scale, but this has to be done in the absence of any intermediate layer between this and the micro-scale one. Accepting a suggestive parallel, in temperature control the micro scale would be that of molecular motion, the macro scale that of a suitable control-to-output relationship, but there is an intermediate modelling level in the form of energy balance equations, that encapsulate the micro level in a way suitable for the macro one. In other words, in classical domains, "macro-physics" helps modelling the system in a control-keen manner. In computing systems, the equivalent of macro-physics simply needs inventing. If one accepts to do that, this in general leads to interpretable models, in the sense that parameters have a well-defined counterpart in the computing system. A deeper discussion on the exquisitely modelling side of the problem can be found in [8].
In classical problems, sensors are naturally related to the intermediate macro physics level, the same where control-oriented equations reside. In computing systems, the opposite is true: the intuitively conceived sensors are at the lowest or the highest level, both unfit to produce models suitable for a simple control design. The proposed framework, in its most conceptual nature, is an attempt to systematically fill this gap.

2.3 Sensors and actuators design

In the following $\mathbf{s}$ shall denote a set of N_s sensor measurements s_i in a set S_i; formally

$$\mathbf{s} = \{s_i \in S_i | S_i \subseteq \mathbb{R}, i = 1, \ldots, N_s\} \tag{1}$$

and $\mathcal{S}$ shall be a map that provides a suitable performance metric (vector) y from the sensors data

$$y = \mathcal{S}(\mathbf{s}). \tag{2}$$

In one word, in computing systems, the sensing problem is "how to have sensor measurements at the right level". As for actuating, the problem is in somehow dual with respect to sensing. In the former context, more "sensors" than performance metrics are typically present. In the latter, one has to map some synthetic control actions on a tendentiously larger number of "physical" actuators. In the following $\mathbf{a}$ shall denote a set of N_a actuators signals a_i in a set A_i; formally

$$\mathbf{a} = \{a_i \in A_i | A_i \subseteq \mathbb{R}, i = 1, \ldots, N_a\} \tag{3}$$

and $\mathcal{A}^{-1}$ shall be a map between the control signal to the manipulated variables $\mathbf{a}$

$$\mathbf{a} = \mathcal{A}^{-1}(u). \tag{4}$$

To allow a straightforward use of simple control techniques, it is additionally required that as many synthetic control actions as performance metrics be present.

2.4 Modelling and controlling

The considerations made so far naturally lead us, for the modelling phase, to describe the controlled system with the block diagram of Figure 2. Suppose for simplicity that we

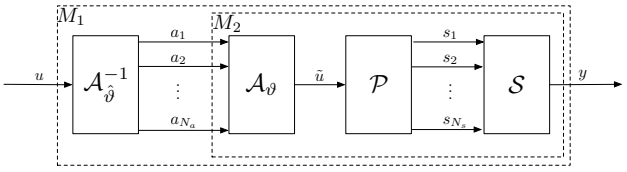

Figure 2: The process is neatly isolated from the other components of the system. M_1 and M_2 depict two partitions of the system where identification techniques could be applied.

are dealing with a scalar performance metric, termed y. Block $\mathcal{S}$ represents the way y is obtained from the low-level sensors s_i. Block $\mathcal{P}$ represents the core phenomenon to be controlled. The key point here is that $\mathcal{P}$ is supposed to have a *single* input $\widetilde{u}$, that can be interpreted as the superposition of all the actuators' effects, combined together by block $\mathcal{A}$. The assumption that any combination of actuator values a_i can be described by a single value of $\widetilde{u}$ is definitely strong, but can in principle be accepted if $\mathcal{A}$ is assumed to depend on a time-varying parameter vector ϑ, whence the subscript in $\mathcal{A}_\vartheta$. If ϑ is estimated reliably enough, the scheme can be complemented with a block $\mathcal{A}^{-1}_{\hat{\vartheta}}$ capable of selecting, for a given desired u, one of the combination of values a_i that produces $\widetilde{u}$ by means of $\mathcal{A}_\vartheta$. It is then possible to reason

on the problem like if the input to the controlled system was u. Three different approaches could be envisioned when reasoning about the problem. First, it is possible to treat the series of $\mathcal{A}_\vartheta$, $\mathcal{P}$ and $\mathcal{S}$ as a Multiple Input Single Output (MISO) System. The model of this system, denoted with M_2 in Figure 2, can be identified with a black box strategy. This solution often leads to very complex models, such as the one learned with an Artificial Neural Network in [1]. As discussed before, the efficacy of a single actuator is mainly time-varying and off-line identification techniques are not able to provide reliable results. A second possibility is to choose $\mathcal{A}^{-1}$ according to an heuristic and keep it fixed, resorting to identification to build a model of the $u \mapsto y$ relationship, denoted with M_1 in Figure 2. When dealing with the specific problem, this approach will be explored but it will be shown that in general, too complex models are obtained. From these complex models, building a control solution is difficult, and the generality of it is unquestionably poor, since a very complex model captures the dynamics of the instance used to get identification data more than a flexible model. The third alternative, that in the presented case study proved successful, is to consider the series of $\mathcal{A}_\vartheta$, $\mathcal{P}$ and $\mathcal{S}$ as a MISO system, but identifying such a system with a *grey box* approach. In so doing, $\mathcal{A}_\vartheta$ is assumed static and $\mathcal{P}$ dynamic and linear time invariant, based on the introduced (control) time scale considerations. When a model is successfully identified with this approach, usually resulting in a low order $\mathcal{P}$, it is possible to use the identified $\mathcal{A}_\vartheta$ to design $\mathcal{A}_{\hat{\vartheta}}^{-1}$. This makes the $u \mapsto y$ relationship still time-varying, but simplifies the use of adaptive control techniques.
From the so obtained model, the control has to be designed. Having isolated the process core phenomenon from the other parts of the system, the design phase is simpler than it would be without such a partition. In relevant cases, even simple control strategies (e.g., PID-based) could be beneficial. However, in the most general case, time-varying behaviours may occur and alter the controlled system dynamics. To cope with this situation, a "tuning" or adaptation block is to be considered. This block adapts the regulator parameters according to the actual behaviour of the process. Hence, in such a case, a control policy suitable for adaptive control is necessary, e.g., Model Predictive Control (MPC) ones. The resulting control scheme is shown in Figure 3, where $\mathcal{R}$ is the regulator and $\mathcal{T}$ the tuning block.

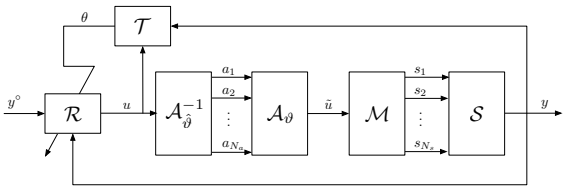

Figure 3: The SMART control scheme.

3. CONCLUSION AND FUTURE WORK

A framework to design computing system components as feedback controllers was presented, named SMART as it divides the design flow into Sensing, Modelling, Actuating, Regulating, and Tuning. The strength of SMART is twofold. First, the control law is not directly coded as an algorithm, rather is obtained as the unambiguous implementation of a dynamic model. Second, the identification of what is the core phenomenon to be controlled leads to inherently simple modelling and control techniques, thus to an easier design and assessment. Those two aspects are distinctive with respect to classical approaches, and the obtained results show their effectiveness [9]. One of the main advantages of the framework is that the generality of the resulting control system is greater than any specific solution that could be envisioned, this means that introducing actuators and different sensors would not require to change the structure of the control problem. SMART was developed within a long-term research, aimed at a deeper use of control-theoretical methods in the realisation of computing systems' components. From a practical standpoint, SMART formalises the mentioned idea of not only "closing loops around systems as they were originally built", but of intervening in the design itself of said systems.

4. REFERENCES

[1] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proc. of the 41st MICRO*, pages 318–329, 2008.

[2] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *Proc. of the Int. Conf. on Autonomic Computing*, pages 36–43, 2004.

[3] P. Geurts, I. El Khayat, and G. Leduc. A machine learning approach to improve congestion control over wireless computer networks. In *Proc. of the 4th IEEE Int. Conf. on Data Mining*, pages 383–386, 2004.

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

[5] C. Karamanolis, M. Karlsson, and X. Zhu. Designing controllable computer systems. In *Proc. of the 10th conference on Hot Topics in Operating Systems*, pages 9–15, 2005.

[6] C. Lu, J. A. Stankovic, and S. H. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23:85–126, 2002.

[7] M. Maggio, F. Terraneo, and A. Leva. Task scheduling: a control-theoretical viewpoint for a general and flexible solution. *ACM Transactions on Embedded Computing Systems*, May 2012.

[8] A. V. Papadopoulos, M. Maggio, and A. Leva. Control and design of computing systems: what to model and how. In *Proc. of the 7th Int. Conf. of Mathematical Modelling, MATHMOD'12*, 2012. (to appear).

[9] A. V. Papadopoulos, M. Maggio, S. Negro, and A. Leva. General control-theoretical framework for online resource allocation in computing systems. *IET Control Theory & Applications*, (accepted for publication) 2012.

[10] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *Proc. of the 22 nd Int. Conf. on Distributed Computing Systems (ICDCS'02)*, pages 301–306, 2002.