Feedback Control as MAPE-K loop in Autonomic Computing*

Eric Rutten[1], Nicolas Marchand[2], and Daniel Simon[3]

[1] INRIA, Grenoble, France,
eric.rutten@inria.fr, https://team.inria.fr/ctrl-a/members/eric-rutten
[2] CNRS, GIPSA-lab, Grenoble, France, Nicolas.Marchand@gipsa-lab.fr,
http://www.gipsa-lab.fr/~nicolas.marchand
[3] INRIA Sophia Antipolis & LIRMM Montpellier, France,
daniel.simon@inria.fr, http://www2.lirmm.fr/~simon

Abstract. Computing systems are becoming more and more dynamically reconfigurable or adaptive, to be flexible w.r.t. their environment and to automate their administration. Autonomic computing proposes a general structure of feedback loop to take this into account. In this paper, we are particularly interested in approaches where this feedback loop is considered as a case of control loop where techniques stemming from Control Theory can be used to design efficient safe, and predictable controllers. This approach is emerging, with separate and dispersed effort, in different areas of the field of reconfigurable or adaptive computing, at software or architecture level. This paper surveys these approaches from the point of view of control theory techniques, continuous and discrete (supervisory), in their application to the feedback control of computing systems, and proposes detailed interpretations of feedback control loops as MAPE-K loop, illustrated with case studies.

Keywords: Autonomic managers, administration loops, control theory

1 Feedback loops in computing systems

1.1 Adaptive and reconfigurable computing systems

Computing systems are becoming more and more dynamically reconfigurable or adaptive. The motivations for this are that, on the one hand, these systems should dynamically react to changes on their environment or in their execution platform, in order to improve performance and/or energy efficiency. On the other hand, complex systems are too large to continue being administrated manually and must be automated, in order to avoid error-prone or slow decisions and manipulations.

This trend can be observed at very diverse levels of services and application software, middle-ware and virtual machines, operating systems, and hardware architectures. The automation of such dynamical adaptation manages various aspects such as

* This work has been partially supported by CNRS under the PEPS Rupture Grant for the project API: https://team.inria.fr/ctrl-a/members/eric-rutten/peps-api and by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

computing and communication resources, quality of service, fault tolerance. It can concern small embedded systems like sensors networks, up to large-scale systems such as data-centers and the Cloud. For example, data-centers infrastructures have administration loops managing their computing resources, typically with energy-aware objectives in mind, and possibly involving management of the cooling system. At a lower level, FPGA-based architectures (Field-Programmable Gate Arrays) are hardware circuits that can be configured at run-time with the logics they should implement: they can be reconfigured dynamically and partially (i.e. on part of the reconfigurable surface) in response to environment or application events; such reconfiguration decisions are taken based on monitoring the system's and its environment's features.

1.2 Autonomic computing

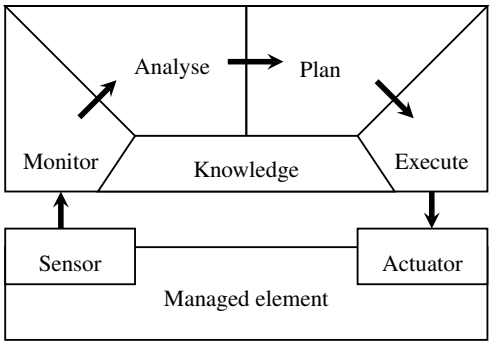

Fig. 1. The MAPE-K autonomic manager for administration loop.

Autonomic computing [50] proposes a general feedback loop structure to take adaptive and reconfigurable computing into account. In this closed loop, systems are instrumented with monitors of sensors, and with reconfiguration actions or actuators; these two have to be related by a control and decision component, which implements the dynamic adaptation policy or strategy. The loop can be defined as shown in Figure 1 with the MAPE-K approach, with sub-components for Analysis of Monitored data, Planning response actions, Execution of these actions, all of them based on a Knowledge representation of the system under administration. Autonomic computing has now gained a large audience [54].

Such autonomic loops can be designed and developed in many different ways, relying on techniques from e.g. Artificial Intelligence. However, an important issues remains in that it is generally difficult to master the behavior of the automated closed-looped systems with precision.

1.3 Need for control

We are therefore particularly interested in an approach where this MAPE-K loop is considered as a case of a control loop. Then, techniques stemming from control theory can be used to design efficient, safe, and predictable controllers. Control theory provides designers with a framework of methods and techniques to build automated systems with well-mastered behavior. A control loop involves sensors and actuators that are connected to the process or "plant" i.e., the system to be controlled. A model of the dynamical behavior of the process is built, and a specification is given for the control objective, and on these bases the control is derived. Although there are approaches to the formal derivation of software from specifications, such as the B method [3], this methodology is not usual in Computer Science, where often a solution is designed directly, and only then it is analyzed and verified formally, and the distinction between the process and its controller is not made systematically.

We observe that this approach of using Control Theory methods and techniques for computing systems, although well identified [44, 77], is still only emerging. Works are scattered in very separate and dispersed efforts, in different areas of the field of reconfigurable or adaptive computing, be it at software or architecture level, in communities not always communicating with each other. Some surveys are beginning to be offered [35], some offering a classification [66], or concentrating on Real-Time computing systems [22, 7] but a synthesis of all these efforts is still lacking. The community begins to structure itself, notably around workshops focused specifically on the topic e.g., Feedback Computing [21].

There exist related works in the community on Software Engineering for self-adaptive systems presenting approaches to integrate Control Theory, typically with continuous models [34] : here we focus on the MAPE-K loop as a target for Control techniques. Also, some works exist considering different approaches to discrete control, in the sense of considering events and states (see Section 3.3, [56]) related to planning techniques from Artificial Intelligence e.g., [28, 70, 17] or reactive synthesis in Formal Methods or Game Theory e.g., [29, 13]. A wide overview is given in another chapter of this book [56]. In this paper we make the choice to focus in more detail on approaches from the Control Theory community, based on continuous models and on the supervisory control of Discrete Event Systems [69, 18, 73].

Our point in this paper is to contribute to relating on the one hand, the general notion of control loop in Autonomic Computing, and on the other hand, design models and techniques stemming specifically from Control Theory. In doing so, we perform choices and do not pretend to exhaustivity. In particular, we do not include in our scope techniques from different approaches and communities like, e.g., Artificial Intelligence or Formal Methods, even though they may have features not covered here.

Indeed several layers and different flavors of control must be combined to fully handle the complexity of real systems. This already lead a long time ago to hierarchical control, e.g. [5], where low level (i.e. close to the hardware) fast control layers are further coordinated by slower higher level management layers. Besides the different bandwidth between layers, it also happens that different control technologies must be combined. For example, it is often observed in robot control architectures that low level control laws, designed in the realm of continuous control, are managed by a control

actions scheduler based on discrete events systems, such as in [2]. Both the continuous and discrete time control designs described in the next sections are example of building blocks to be further used in hierarchical control for Autonomic Computing.

1.4 Outline.

This paper proposes interpretations of the MAPE-K loop from Autonomic Computing in terms of models and techniques from Control Theory. We first consider this from the point of view of continuous control in Section 2, starting with the classical PID up to more elaborate nonlinear and event-based control.

Discrete control is then considered in Section 3, where discrete event systems are modeled by transition systems e.g., Petri nets or labelled automata.

Then some illustrative case studies are presented in Section 4, showing how these concepts can be put into practice in the framework of real-world computing systems.

Finally, Section 5 provides discussions and perspectives.

2 Continuous control for autonomic computing

2.1 Brief basics of continuous control

The basic paradigm in control is *feedback*, or *closed-loop* control, as depicted in Figure 2. The underlying idea is that control signals are continuously computed from the *error signal*, i.e. the difference between the actual behavior of the plant (measured by its outputs) and its desired behavior (specified by a reference signal). The loop is closed through the controlled plant, and control actions are computed indefinitely at a rate fast enough with respect to the process dynamics [9], [8].

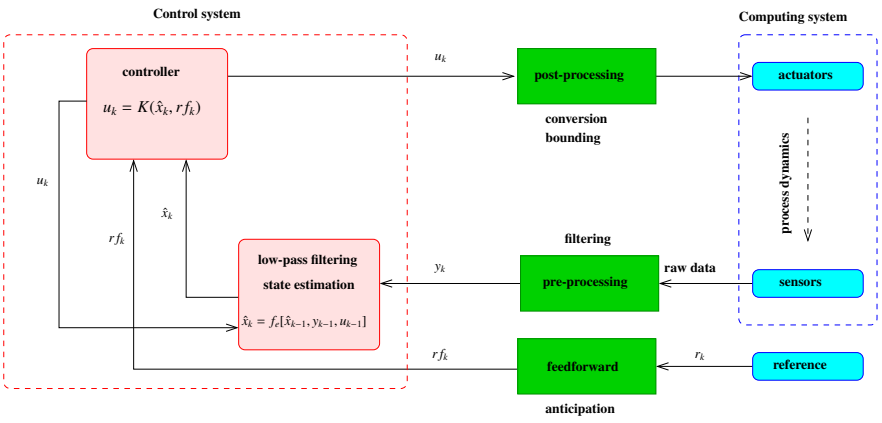

Fig. 2. The control loop for continuous control.

The behavior of the controlled process, whatever its nature (e.g. electro-mechanical or chemical for physical devices, but also digital for schedulers, databases and other numerical components) is never perfectly known. It is known only through a dynamic *model* which aims to capture the main characteristics of the process. It is most often given as of a set of difference equations where the continuous time is sampled (usually periodically) at instants $t_k, t_{k+1}, ...$:

$$\begin{aligned} \mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k), \qquad x(t=0) = x_0 \\ \mathbf{y}_k &= g(\mathbf{x}_k) \end{aligned} \tag{1}$$

Here $\mathbf{x}$ is the state vector of the process able to describe its behavior over time, x_0 is its value at the initialization of the system. $\mathbf{y}$ is a vector of outputs measured on the process via sensors and **f(.)** and **g(.)** are respectively the state evolution function and the output function of the plant model. $\mathbf{u}$ is the vector of control signals sent to the actuators, it is repetitively computed by a controller, e.g. by a state feedback as:

$$\mathbf{u}_k = K(\hat{\mathbf{x}}_k, \mathbf{r}_k) \tag{2}$$

where $\hat{\mathbf{x}}$ is an estimate of the state vector, $\mathbf{r}$ is a set of reference signals to be tracked and **K(.)** is a function (which can be static or dynamic) of the state (or outputs) and of the reference signals.

For example, considering the control model of a web server, the system state may gather the number of admitted requests and an estimate of the CPU load, the observed output vector may gather a measure of the CPU activity filtered over some time window and the observed response time, and the control input can be an admission controller. The control objective can be the result of the minimization of a Quality of Service criterion gathering the rejection rate and the response time of the server under constraint of CPU saturation avoidance. Note that most often the variables of interest are not directly available from simple measurements, and that the system state must be reconstructed and smoothed using signal processing and filtering from the raw sensors outputs.

Remember that (1) is a model of the plant, i.e. an abstraction of reality where the structure is simplified and the value of the parameters is uncertain. Therefore, an open-loop control, using the inverse of the model to compute control inputs from the desired outputs, inevitably drifts away and fails after some time. Compared with open-loop control (which would rely on an utopian perfect knowledge of the plant), the closed-loop structure brings up several distinctive and attractive properties:

Stability Briefly speaking, a system is said stable if its output and state remain bounded provided that its inputs and disturbances remain bounded (formal definitions, such as BIBO stability, Lyapunov stability and others, can be easily found thorough the control literature). It is a crucial property of a control system, ensuring that the trajectory of the controlled plant is kept close enough to the specified behavior to satisfy the end-user's requirements. A distinctive property of feedback controllers is that, if they are well designed and tuned, they are able to improve the stability of marginally stable plants, and even to stabilize naturally unstable systems, e.g., modern aircrafts. Anyway the stability of a control system must be carefully assessed, as poor design or tuning can drive the system to unstability;

Robustness The control actions are repeated at a rate which is fast compared with the system dynamics, hence the increments of tracking errors due to imperfect modeling are small, and they are corrected at every sample. Indeed, besides the main directions of the dynamics, the model does not need to capture the details of the process dynamics. Most often, poorly known high frequencies components are discarded, and exact values of the parameters are not needed. Therefore, feedback is able to provide precise and effective control for systems made of uncertain components.

Tracking and performance shaping As the controller amplifies and adjusts the tracking error before feeding the actuators, it is able to shape the performance of the controlled plant. For example, the closed-loop system can be tuned for a response faster than the open-loop behavior, and disturbances can be rejected in specified frequency bands. Thanks to robustness, tracking objectives can be successfully performed over a large range of input trajectories without need for on-line re-tuning;

2.2 The MAPE-K loop as a continuous control loop

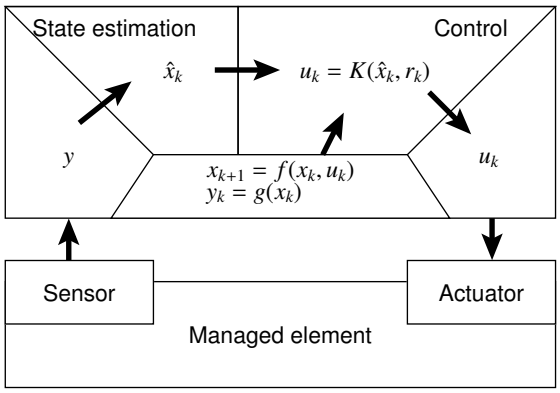

Fig. 3. The continuous control loop as a MAPEK diagram

The MAPE-K description corresponding to the model (1) with control (2) is as shown in Figure 3. The *Monitor* phase of the MAPE-K loop corresponds to the sampling of the system, typically, it defines the frequency at which the data must be acquired. It is usually related to sampling theory (Shannon theorem). The *Analyse* phase is in our description represented by the hat over x. This notation is commonly used to denote that the exact value of x is not known, either because of noise that requires a filtering action or because it can not be measured directly from the system. It is for instance the case of energy consumption that must be estimated using other variables like CPU or disk usage. The analyse phase would in that case include the signal estimation/reconstruction or more simply the filtering. The *Plan* phase corresponds to the computation of the control law using the Knowledge of the system held in the model. Finally, the *Execute*

phase consist in changing the value of the actuator at a frequency which is most often identical to the sampling frequency of the monitor phase.

2.3 Continuous feedback computing

Let us now examine how feedback can be applied to computing system administration, resource management or network management. Although feedback control was first developed to control physical devices like steam engines, factory lines or various kind of vehicles [9], the closed-loop concept has been adapted for the control of digital devices such database servers, e.g. [60] and [44], or real-time schedulers as in [62]. However, compared with usual control applications, the nature of the controlled process deeply differs in the case of control for computing devices, and the usual components of the control loops must be adapted for the particular technology.

Models Usual models for the control of continuous process are given as a set of differential equations, which are further discretized as difference equations for numerical control purpose. In contrast, at a detailed level, digital objects can be often described by large FSMs which are not well suited for closed-loop control. However, thanks to robustness, a feedback-control compliant model only needs to capture the essential of the plant dynamics. For example, computing devices can be approached by "fluid modeling" where, e.g., flow equations describe flows of input requests and levels in tanks represent the state of message queues [44]. Using such abstractions leads to quite simple difference models, where for example queues behave as integrators and provide the basic dynamics in the model. Besides metrics related to performance, as computation loads or control bandwidth, some relevant properties of a software are related to reliability. For example, the failure probabilities of software components may lead to a reliability formal model given as a Discrete Time Markov Chain, further used for the design of a predictive feedback control law [33]. Some control related modeling issues are detailed in another chapter of this book [56].

Sensors Sensors provide raw measurements from the controlled process, they are provided by hardware or software probes in the operating system or in the application code. Basic measurements record the system activity such as the CPU use, deadlines misses or response times of individual components. Raw measurements are gathered and pre-processed to provide compound records and quality of service clues to the controller. Note that the CPU use is always of interest, as CPU overloading is a permanent constraint. However, it is only meaningful when estimated over windows of time : the size of these measurement windows and associated damping filters provide a second main source of dynamics in the plant model.

Actuators In software control, the actuators are provided by function calls issued by the operating system, or by other software components with enough authority. Scheduling parameters such as clock rates, deadline assignments and threads priorities can be used to manage multitasking activities. Admission control can be used to handle unpredictable input request flows. The frequency scaling capability of modern chips is also an effective tool to optimize the energy consumption due to high speed calculations in mobile devices.

Controllers Potentially all the control algorithms found in the existing control toolbox can be adapted and used for the control of digital devices [66]. Most often, thanks to the usually simple dynamic models considered for software components, simple and cheap controllers can be effective as detailed in section 2.4. Anyway some more complex control algorithms have been worked out to better handle the complexity of QoS control for computing systems (section 2.5).

2.4 Basic control

PID (Proportional, Integral, Derivative) control algorithms are widely used, as they are able to control many single input/single output (SISO) systems through a very simple modeling and tuning effort. In that case the control input u is written as a function of the error signal -that is the difference between a desired output and its measure on the real system- $e(t) = r(t) - y(t)$ as:

$$u(t) = Ke(t) + \frac{K}{T_i}\int_0^t e(\tau)d\tau + KT_d\frac{d}{dt}e(t) \tag{3}$$

Here the proportional term $K \cdot e(t)$ controls the bandwidth and rising time of the control loop, the derivative term $KT_d\frac{d}{dt}e(t)$ damps the oscillations and overshoots and the integral term $\frac{K}{T_i}\int_0^t e(\tau)d\tau$ nullifies the static errors, that is the value of the error $e(t)$ when t goes to the infinite.

Indeed, the ideal continuous PID must be discretized for implementation purpose. For example, using a backward difference method (with T_s the sampling period) yields

$$u_k = u_{k-1} + K[e_k - e_{k-1}] + \frac{K \cdot T_s}{T_i}e_k + \frac{K \cdot T_d}{T_s}[e_k - 2e_{k-1} + e_{k-2}] \tag{4}$$

The MAPEK diagram of the PID controller then follows as in Figure 4. Tuning the PID is made using the knowledge of the system. This knowledge can take the form of a state space or transfer function model but can also reside in an empirical tuning of the parameter of the PID controller.

2.5 Advanced modeling and control

Besides PID controllers which have been used in pioneering works (e.g. [59]), other simple/linear control schemes were also implemented in the context of computer science. [44] is an emblematic example of a black-box modeling in order to derive a controller using classical linear control theory aiming to maximize the efficiency of Lotus Notes. Other linear approaches were also implemented for periods rescaling [19] or to control elasticity of distributed storage in cloud environment [65]. All linear systems share the same superposition property and can be analyzed and assessed using well established mathematical tools. Unfortunately, their use to real systems that are most of the time non-linear is possible only on a limited range of the state space for which linearization is meaningful.

Indeed, many classical non-linearities of computer systems (limited range for variables, limited bandwidth, etc.) can hardly be taken into account only with linear tools

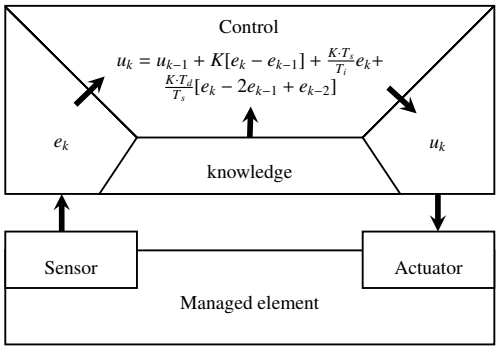

Fig. 4. The PID continuous control loop as a MAPEK diagram

[34]. In particular, optimizing a computing system operation may need to intentionally load the actuators (i.e. the CPUs) until saturation, rather than avoiding the actuators limits as in the linear control approach. Therefore, in addition to the linear control theory, the control toolbox now contains a rich set of advanced control algorithms, which have been developed over years to supplement the shortage of simple linear control in specific cases.

For example, early models of servers were simple linear fluid models and the corresponding linear controller as [44]. However, handling trashing in servers needs to model the overhead due to parallel operations : the resources needed by a server to serve requests is not proportional to the number of requests. Non-linear models and control are needed in that case detailed in section 4.2 [62].

In another case study [67], handling the static input and output non-linearities of a software reservation system is made by the combination of linear and non-linear blocks in a Hammerstein-Wiener block structure. Then, the corresponding QoS controller is designed in the predictive control framework. Note that even when these more elaborated non-linear models are considered, the resulting controllers remain simple with a very small run-time overhead and a moderate programming effort.

Other non-linear, switched, hybrid, hierarchical and cascaded schemes were implemented on various computing systems (see for instance [76, 78] and the references therein).

Indeed it appears that, considering the quite simple dynamics of the controlled system, the time devoted to modeling is by far larger than the time devoted to control design. Models well suited for control purpose must be simple enough to allow for the synthesis of a control algorithm, while being able to capture the essence of the system behavior. Typically, modeling for the control of autonomic computing systems needs to consider trade-offs between the control objectives and the cost needed to reach them through the execution of parallel activities running on shared hardware and software resources [55].

For example, it has been shown in [61] that a game theoretic framework allows for the decoupling between the resource assignment and the quality setting, finally leading to a resource manager with a linear time complexity in the number of applications running in parallel. Once the resources and concurrent activities has been suitably modeled, the control decisions can be implemented as a hierarchy of layered controllers ranging from the control of elementary components up-to QoS optimization, e.g., as in [57].

Finally, the execution cost of the controller itself must be considered. Traditionally control systems are time triggered, allowing for a quite simple stability analysis in the framework of periodic sampling. However, the choice of the triggering period is an open issue, as reactivity needs fast sampling leading to a high computing cost. However, fast reactions are not always necessary, for example in case of slowly varying workloads. To avoid wasting the computing resource, the event-based control paradigm has been proposed (e.g. [75]). With this approach, the controller is activated only when some significant event acting on the system triggers an event detector.

3 Discrete control for autonomic computing

3.1 Brief basics of supervisory control of Discrete Event Systems

Amongst the different approaches to discrete control (see Sections 1.3 and 3.3, [56]) this Section focuses on and technically details the supervisory control of Discrete Event Systems [69, 18]. Figure 5 shows a control loop for the case of discrete control with a memorized state, a transition function, and a supervisory controller obtained by discrete controller synthesis.

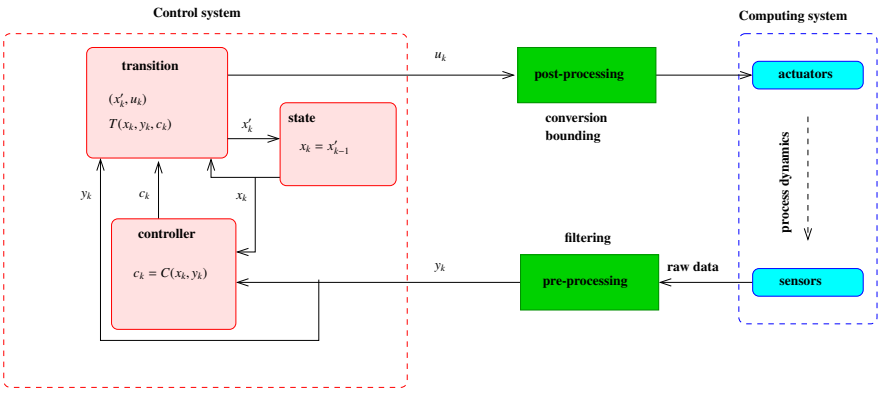

Fig. 5. The control loop for discrete control.

The characterization of Discrete Event Systems [18] is given by the nature of the state space of the considered system: when it can be described by a set of discrete values, like integers, or vectors of Booleans, and state changes are observed only at

discrete points in time, then such transitions between states are associated with events. In this section we very briefly and informally summarize some essential notions.

The modeling of *sequences* of such events, like the sequence of values, at time k, of y_k and u_k in Figure 5, can be approached by *formal languages*. They enable to specify structure in the sequences of events, representing possible behaviors of a system, or desired behaviors in the interaction with a system. Operations on languages can help composing them, or making computations on the represented behaviors.

Automata are formal devices that are capable of representing languages, in the graphical and intuitive form of state and transition graphs, also called transition systems, or Finite State Machines (FSM). As shown in Figure 5, they involve two main features. On the one hand, there is a memorizing of a *state*, the current value x_k resulting from the previous transition at $k-1$ (with an initial value xi at time 0). On the other hand, is a *transition function* T computing the next value of the state x'_k in function of the current observed value y_k (we do not yet distinguish controllable variables c) and current state x_k. It can also compute values u_k that can be used to send commands to the controlled system. Figure 5 also shows possible pre-processing between raw data and y_k, or post-processing between u_k and concrete actions, e.g corresponding to implementation-specific filters.

$$\begin{array}{rcl} (\mathbf{x'}_k, \mathbf{u}_k) & = & T(\mathbf{y}_k, \mathbf{x}_k) \\ \mathbf{x}_k & = & \mathbf{x'}_{k-1} \\ \mathbf{x}_0 & = & \mathbf{xi} \end{array} \tag{5}$$

Such automata can be associated with properties pertaining to their general behavior e.g., determinism, reactivity, or more specific like reachability of a state, and manipulated with operations e.g., parallel or hierarchical composition. The transitions are labelled by the events which are recognized when they are taken. Such automata-based models are precisely the basic semantic formalism underlying reactive systems and languages [43, 11]. Related models in terms of transitions systems also include *Petri nets*, where the transitions are connecting places which are associated with tokens: the marking of the set of places by present tokens defines the state of the Petri net. The transitions can be labelled by events and their sequences define languages. The relationship with automata is given by the graph of reachable markings of the net. *Analysis* of such transition systems is made possible by algorithmic techniques exploring the reachable states graph in order to check typically for safety properties (e.g., using model checking techniques and tools), or concerning diagnosis of the occurrence of unobservable events from the observations on the behavior of a system.

Control of transition systems has then been defined as the problem of restricting the *uncontrolled behaviors* of a system. The latter can be described by an automaton G, control restricts its behavior so that it remains in a subset of the language of G, defined by a control objective, describing the desired behavior. The notion of supervisory control of discrete event systems has been introduced [69], which defines a supervisor that can inhibit some transitions of G, called *controllable* (controllability of a system can be

partial), in such a way that, whatever the sequences of uncontrollable events, these controllable transitions can be taken in order for the desired behavior to be enforced, and the undesirable behavior avoided. Typical desired behaviors, or control objectives, in the supervisory control approach are safety properties : deadlock avoidance, or invariance of a subset of the state space (considered good). A specially interesting property of the supervisor is it should be restricting only the behaviors violating the desired objectives, or in other terms it should be *maximally permissive*. It can be noted that this important property is possible in this approach, whereas it is not considered or defined in approached dealing with more expressive goals such as liveness. As shown in Figure 5, the resulting synthesized controller C gives values to controllable variables c, which are part of the parameters of the transition function T:

$$\begin{array}{rcl} (\mathbf{x'}_k, \mathbf{u}_k) & = & T(\mathbf{y}_k, \mathbf{c}_k, \mathbf{x}_k) \\ \mathbf{c}_k & = & C(\mathbf{y}_k, \mathbf{x}_k) \\ \mathbf{x}_k & = & \mathbf{x'}_{k-1} \\ \mathbf{x}_0 & = & \mathbf{xi} \end{array} \tag{6}$$

Tools available to users who wish to apply such automated controller synthesis techniques, adopting the approach of supervisory control for Discrete event Systems, include: TCT, based on languages models and theory [74]; Supremica, related to the manufacturing languages of the IEC standard [4]; SMACS, which achieves Controler Synthesis for Symbolic Transition Systems with partial information [47]; Sigali, which is integrated in the synchronous reactive programming environments [63] , and in the compiler of the BZR language [23]. A new tool, ReaX, extends the expressivity to Discrete Controller Synthesis for infinite state systems, and treats logic-numeric properties [12].

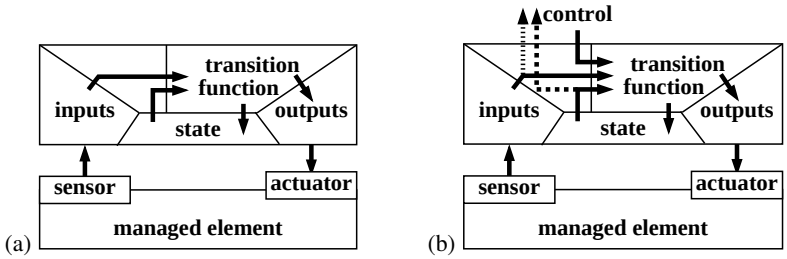

Fig. 6. The discrete control loop as a MAPEK diagram. (a): simple automaton-based manager; (b): exhibiting observability and controllability.

3.2 The MAPE-K loop as a discrete supervisory control loop

In the general framework for autonomic computing shown in Figure 1, discrete control can be integrated as shown in Figure 6(a): it instantiates the general autonomic loop

with knowledge on possible behaviors represented as a formal state machine, and planning and execution as the automaton transition function, with outputs triggering the actuator. As evoked in previous Section, the models used in supervisory control of DES enable to address properties on the order of events or the reachability of states, with tool-equipped techniques for verification (e.g. model checking) and especially Discrete Controller Synthesis (DCS). The latter is automated and constructive, hence we use it for the logic control of autonomic systems, encapsulated in a design process for users experts of systems, not of formalisms.

In the autonomic framework, in order to support coordination of several autonomic managers by an upper layer, some additional observability can be obtained by having additional outputs, as shown by dashed arrows in Figure 6(b) for a FSM autonomic manager, exhibiting (some) of the knowledge and sensor information (raw, or analyzed); this can feature state information on the autonomic manager itself or of managed elements below. At that level, additional inputs can provide for controllability by an external coordinator.

3.3 Discrete feedback computing

As was noted by other authors, while classical control theory has been readily applied to computing systems [44], applying Discrete Control Theory to computing systems is more recent. One of the earliest works deals with controlling workflow scheduling [71]. Some focus on the use of Petri nets [46, 45, 58] or finite state automata [68].

In the area of fault-tolerant systems, some works [53, 15] present notions similar to control synthesis, not explicitly considering uncontrollables. In that sense, it resembles more open-loop control, considering the internals of a computing system, to which we prefer closed-loop control, taking into account events from its environment.

A whole line of work focuses on the computing systems problem of deadlock avoidance in shared-memory multi-threaded programs. These work rely on the literature in Discrete Control Theory concerning deadlock avoidance, which was originally motivated by instances of the problem in manufacturing systems. [73] is a programming language-level approach, that and relies upon Petri net formal models, where control logic is synthesized, in the form of additional control places in the Petri nets, in order to inhibit behaviors leading to interlocking. The Gadara project elaborates on these topics [72]. They apply Discrete Control internally to the compilation, only for deadlock avoidance, in a way independent of the application. Other works also target deadlock avoidance in computing systems with multi-thread code [10, 30].

Another kind of software problem is attacked by [37, 38]: they consider run-time exceptions raised by programs and not handled by the code. Supervisory control is used to modify programs in such a way that the un-handled exceptions will be inhibited. In terms of autonomic computing, this corresponds to a form of self-healing of the system. Applications of the Ramadge and Wonham framework to computing systems can also be found concerning component-based systems reconfiguration control, enforcing structural as well as behavioral properties [51], and more generally adaptive systems, as one of the decision techniques in a multi-tier architecture [28].

In an approach related to reactive systems and synchronous programming, discrete controller synthesis, as defined and implemented in the tool Sigali, is integrated in a pro-

gramming language compiler. [27] describes "how" compilation works, with modular DCS computations, performing invariance control. This language treats expression of objectives as a first class programming language feature. The programming language, called BZR, is used in works concerning component-based software [16]. It is extended to handle logico-numeric properties, by replacing, in the modular architecture of the compiler, Sigali with the new tool ReaX [12]. Other previous work related to the synchronous languages involved some separate and partial aspects of the problem, testing the idea in the framework of a more modest specialized language [25], and particular methods and manual application of the techniques [36], and elaborating on the articulation between reactive programs and DCS [64, 6, 24], as well as application to fault-tolerance [39, 31].

As noted above, some other related work can be found in computer science and Formal Methods, in the notions of program synthesis. It consists in translating a property on inputs and outputs of a system, expressed in temporal logics, into a lower-level model, typically in terms of transition systems. For example, it is proposed as form of liberated programming [41] in a UML-related framework, with the synthesis of StateChart from Live Sequence Charts [42, 52]. Other approaches concern angelic non-determinism [14], where a non-deterministic operator is at the basis of refinement-based programming. These program synthesis approaches do not seem to have been aware of Discrete Control Theory, or reciprocally: however there seems to be a relationship between them, as well as with game theory, but it is out of the scope of this paper.

Also, interface synthesis [20] is related to Discrete Controller Synthesis. It consists in the generation of interfacing wrappers for components, to adapt them for the composition into given component assemblies w.r.t. the communication protocols between them.

4 Case studies

4.1 Video decoding and DVFS

Energy availability is one of the main limiting factors for mobile platforms powered by batteries. Dynamic Voltage and Frequency Scaling (DVFS) is a very effective way to decrease the energy consumption of a chip, by reducing both the clock frequency and the supply voltage of the CPUs when high computation speeds are not necessary. Many chips used in embedded or mobile systems are now fitted with such capabilities, and the computing speed is adapted on-the-fly thanks to some estimated computing performance requirement.

Using feedback loops is an effective way to robustly adapt the chip computing speed even if the incoming computation load is hard to predict, as in the example described in [32]. The problem is to minimize the energy consumption of a H.264 video decoder by using the lowest possible computing speed able to decode the frames with respect to the output display rate (25 frames/sec).

The computing speed is adapted thanks to the control architecture depicted in Figure 7a). At low level, a computing speed controller -integrated in silicon- drives the DVFS

hardware with frequency and Vdd set points (see [32] for details). It is driven from estimates of the needed computation load (i.e. the number of CPU cycles) and decoding deadline for the incoming frame. These estimates are themselves computed by an outer frame control loop.

Measurements of decoding execution times (Figure 7b) show that, between noisy and almost flat segments, the decoding times exhibit sharp and unpredictable isolated peaks when switching between plans. Therefore, rather than trying to compute any prediction, the estimation of the next frame computation load $\hat{\Omega}_{i+1}$ can be simply taken equal to the last one, i.e. Ω_i, recorded by the instrumentation inserted in the H.264 decoder. Even better, it can be provided by smoothed past values through a low pass filter :

$$\hat{\Omega}_{i+1} = \alpha\hat{\Omega}_{i-1} + (1-\alpha)\Omega_i \tag{7}$$

where $0 \leq \alpha < 1$ controls the filter damping.

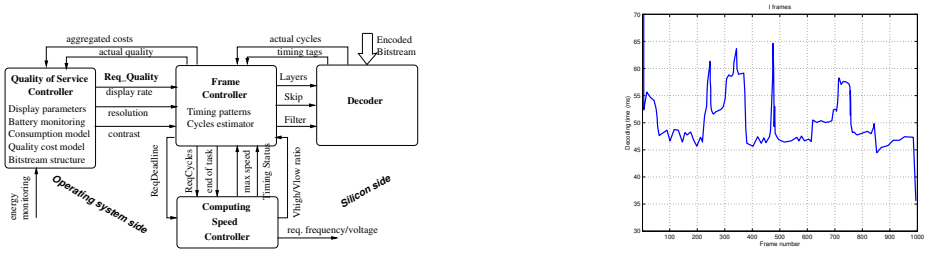

Fig. 7. a) Control architecture – b) Frames computation times

This rough estimate is used by the frame controller to compute the ideal deadline for the incoming frame using a simple proportional controller :

$$\Delta_{r_{i+1}} = \tau_{i+1} + \beta\,\delta_i, \quad 0 < \beta \leq 1 \tag{8}$$

where δ_i is the observed overshoot for the last decoded frame. Indeed this controller aims at driving the end-of-computation of frame $i+1$ towards τ_{i+1}, which is the theoretic timing of the periodic video rate.

Despite the apparently overly simple computing load model (7), the very simple and low cost frame controller (8) is able to regulate the decoding timing overshoot to very small values (Figure 8a), thus keeping an fluid display rate. Computing a penalty function based on the viewing quality (Figure 8b) shows that using these elementary feedback loops allow both for a better viewing quality and up-to 24 % energy saving compared with the uncontrolled decoding case.

Hence, this example show that even very simple control loops with negligible computation overheads, if carefully designed, may have a very positive impact on an embedded system adaptiveness and robustness against a poorly modelled environment.

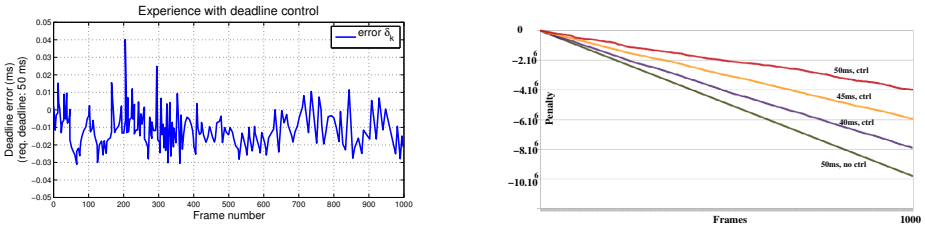

Fig. 8. a) Deadline with control – b) Video viewing penalty function

4.2 Server provisioning

A classical technique used to prevent servers from thrashing when the workload increases consists in limiting client concurrency on servers using admission control. Admission control has a direct impact on server performance, availability, and quality of service (QoS). Modeling of servers and feedback control of their QoS has been one of the first application domain targeted by feedback scheduling, first using linear models [60], [44]. However, it appears that to handle trashing, the model must accurately capture the dynamics and the *nonlinear* behavior of server systems, while being simple enough to be deployed on existing systems.

Based on numerous experiments and identification, a nonlinear continuous-time control theory based on fluid approximations has been designed in [62]. It is both simple to use and able to capture the overhead due to the parallel processing of requests responsible for trashing (Figure 9a).

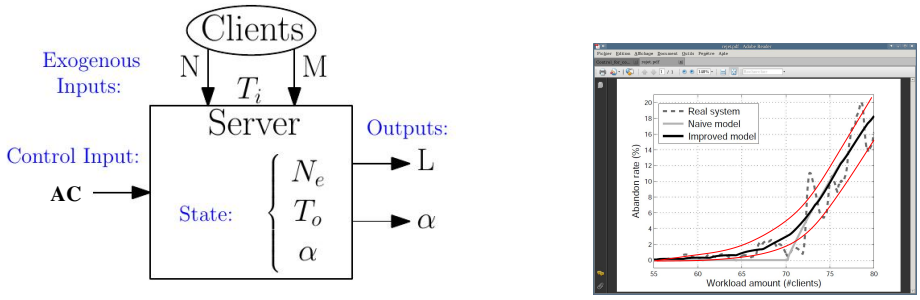

Fig. 9. a) Fluid model – b) rejection rate

The request queue is considered as a fluid tank receiving client request flows M and N and emitting a served requested flow with latency L for the served requests. The system state is defined by the number of concurrently admitted requests N_e, the server throughput T_0 and the rejection rate α. The modelling effort leads to the following model for the input/output latency:

$$L(N_e, M, t) = a(M,t)N_e^2 + b(M,t)N_e + c(M,t) \tag{9}$$

where the latency L is a non-linear function of the number of N_e, of the server mix load M and of continuous time t.The rejection rate is given by

$$\dot{\alpha}(t) = -\frac{1}{\Delta}\left(\alpha(t) - \frac{N_e(t)}{AC(t)} \cdot \left(1 - \frac{T_o(t)}{T_i(t)}\right)\right) \tag{10}$$

with Δ the sampling rate, T_i the input flow and AC the admission control value.

Then two control laws could be derived for different control objectives:

- $AC = \frac{N_e}{1+\gamma_L(L-L_{\max})}$ maximizes the availability of the server, i.e. minimizes the rejection rate;
- $AC = \frac{\alpha N_e}{\alpha-\gamma_\alpha(\alpha-\alpha_{\max})}$ maximizes the performance, i.e. minimizes the latency for the admitted requests.

These simple control laws are cost effective and easy to tune, as they both use a single tuning parameter γ_L or γ_α.

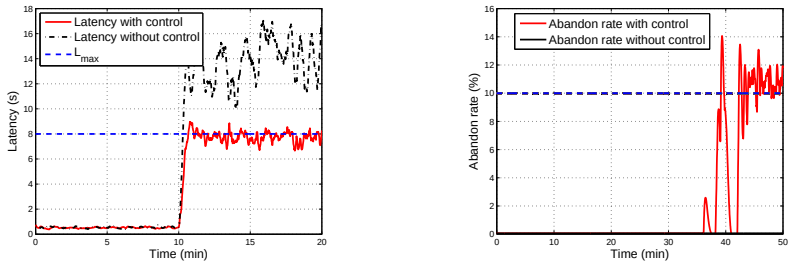

Fig. 10. a) Latency control – b) Rejection rate control

Despite their simplicity, using these simple controllers allows for an efficient on-line management of an Apache web server. For example, Figure 10 show that the rejection rate can be kept close to a desired goal, or that the latency of the served requested can be regulated around the requested value.

Even more important, these controllers –with negligible computing cost– turns the nature of the system into a safer behavior. Indeed, trashing is automatically avoided even in the case of huge overloads with no need for an operator to manually re-tune the AC parameters.

4.3 Coordination of multiple autonomic administration loops

Real autonomic systems require multiple management loops, each complex to design, and possibly of different kinds (quantitative, synchronization, involving learning, ...). However their uncoordinated co-existence leads to inconsistency or redundancy of action. Therefore we apply discrete control for the interactions of managers [40]. We validate this method on a multiple-loop multi-tier system.

Controllable managers as seen in Figure 6(b) can be assembled in composites, where the coordination is performed in a hierarchical framework, using the possibilities offered by each of them, through control interfaces, in order to enforce a coordination policy or strategy. We base our approach on the hierarchical structure in Figure 11: the top-level AM coordinates lower-level ones.

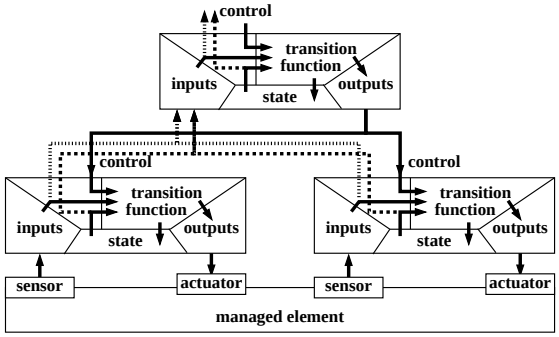

Fig. 11. Autonomic coordination for multiple administration loops.

We consider the case study of the coordination of two administration loops for the management of a replicated servers system based on the load balancing scheme. Self-sizing addresses resource optimization, and dynamically adapts the degree of replication depending on the CPU load of the machines hosting the active servers. Self-repair addresses server recovery upon fail-stop failure of a machine hosting a single or replicated server. Co-existence problems occur when failures trigger incoherent decisions by self-sizing : The failure of the load balancer can cause an under-load of the replicated servers since the latter do not receive requests until the load balancer is repaired. The failure of a replicated server can cause an overload of the remaining servers because they receive more requests due to the load balancing. A strategy to achieve an efficient resource optimization could be to (1) *avoid removing a replicated server when the load balancer fails*, and (2) *avoid adding a replicated server when one fails*.

Figure 12 shows the automata modelling the behaviors of the managers, in the BZR language [26], abstracted to the relevant activity information. In the right of the Figure, the self-sizing manager is composed of three sub-automata. In brief, the two external ones model the control of the adding (resp. removal) of servers, with `disU` (resp. `disD`), which, when true, prevent transitions where output `add` (resp. `rem`) triggers operations. The center one models the behaviors in reaction to load variation, for which all detail is available elsewhere [40]. In the left of the Figure are self-repair managers for the load balancer (LB) and the replicated servers (S). The right automaton concerns servers, and is initially in `OkS`. When `failS` is true, it emits repair order `rS` and goes to the `RepS` state, where `repS` is true. It returns back to `OkS` after repair termination (`Sr` is true). Repair of the LB is similar. The automata in Figure 12 are composed in order to have the global behavior model, and a contract specifies the coordination policy. The policies (1) and (2) in Section 4.3 are enforced by *making invariant, by control*

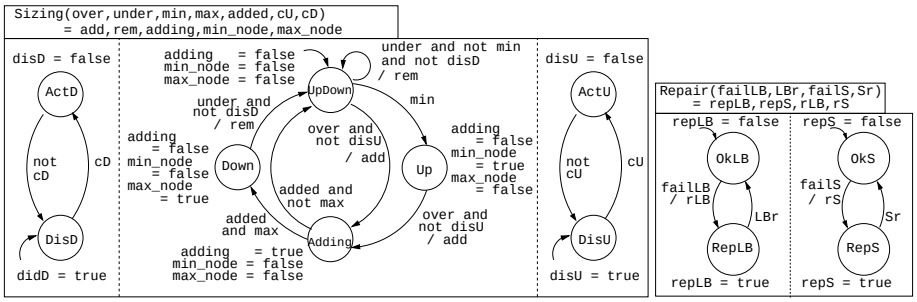

Fig. 12. Managers models : self-sizing (left) and self-repair (right).

upon the controllable variables `Cu, Cd`, the subset of states where the predicate holds : `((repLB => disD) and (repS => disU))`

This controller was validated experimentally on a multi-tier system based on Apache servers for load balancing (with a self-repair manager) and replicated Tomcat servers (with both self-sizing and self-repair managers), with injection of workloads and failures to which the system responded properly, without overreacting, according to the objective.

5 Conclusions and perspectives

We propose a discussion of the problem of controlling autonomic computing systems, which is gaining importance due to the fact that computing systems are becoming more and more dynamically reconfigurable or adaptive, to be flexible w.r.t. their environment and to automate their administration. We observe that one approach consists of using Control Theory methods and techniques for computing systems : although it is well identified [44], it is still only emerging, and works are scattered in separate areas and communities of Computer Science.

We aim at conveying to Computer Scientists the interest and advantages of adopting a Control Theory perspective for the efficient and predictable design of autonomic systems. Compared with open-loop, closed-loop control provides adaptability and robustness, allowing for the design of fault-tolerant systems against varying and uncertain operating conditions. However, there still is a deep need for research in the problems of mapping from high-level objectives in terms of Quality of Service (QoS) or Service Level Objectives (SLO) and abstract models towards lower-levels effective actions on the managed systems. In the area of Computing Systems research, there is an important topic in the design of architectures so that they are made controllable [48], as self-aware software (adaptive, reflective, configuring, repairing...) needs explicitly built-in sensing and acting capabilities [49]. On the other side, the kind of models usual in Control Theory must be totally reworked to be useful for computing systems, and this is a research challenge for the Control community. Also, an important issue is that complex systems will involve multiple control loops, and their well-mastered composition and coordination to avoid interferences is a difficult and hardly tackled question [1].

One lesson learned in this work is that the open problems are concerning both Control Theory and Computer science, and that solutions going beyond simple cases require active cooperation between both fields. As noted by other authors e.g., [33], this bi-disciplinary field is only beginning, and the problems involved require competences in Control, as well as expertise in the computing systems. There is a costly investment in time to build common vocabulary and understanding, for which the MAPE-K loop offers a common ground, and this investment opens the way for better controlled autonomic computing systems, safer and optimized.

References

1. Tarek Abdelzaher. Research challenges in feedback computing: An interdisciplinary agenda. In *8th International Workshop on Feedback Computing*, San Jose, California, 2013.
2. Ahmed Soufyane Aboubekr, Delaval Gwenaël, Roger Pissard-Gibollet, Éric Rutten, and Daniel Simon. Automatic generation of discrete handlers of real-time continuous control tasks. In *IFAC World Congress 2011*, Milano, Italie, August 2011.
3. J.-R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, New York, NY, USA, 1996.
4. K. Akesson. Supremica http://www.supremica.org/.
5. James Sacra Albus, Anthony J Barbera, and Roger N Nagel. *Theory and practice of hierarchical control*. National Bureau of Standards, 1980.
6. K. Altisen, A. Clodic, F. Maraninchi, and E. Rutten. Using controller synthesis to build property-enforcing layers. In *European Symposium on Programming*, volume 2618 of *LNCS*, pages 126–141, Warsaw, Poland, April 2003.
7. Karl-Erik Årzén, Anders Robertsson, Dan Henriksson, Mikael Johansson, H. Hjalmarsson, and Karl Henrik Johansson. Conclusions of the ARTIST2 roadmap on control of computing systems. *ACM SIGBED (Special Interest Group on Embedded Systems) Review*, 3(3), July 2006.
8. Karl J. Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2nd edition, 2015. http://www.cds.caltech.edu/~murray/amwiki/index.php/Main_Page.
9. Karl Johan Åström and Björn Wittenmark. *Computer-Controlled Systems*. Information and System Sciences Series. Prentice Hall, third edition, 1997.
10. A. Auer, J. Dingel, and K. Rudie. Concurrency control generation for dynamic threads using discrete-event systems. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 927 –934, 30 2009-oct. 2 2009.
11. A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages twelve years later. *Proc. of the IEEE, Special issue on embedded systems*, 91(1):64–83, January 2003.
12. Nicolas Berthier and Hervé Marchand. Discrete Controller Synthesis for Infinite State Systems with ReaX. In *12th Int. Workshop on Discrete Event Systems*, WODES '14. IFAC, May 2014.
13. Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, May 2012.
14. R. Bodik, S. Chandra, J. Galenson, D. Kimelman, N. Tung, S. Barman, and C. Rodarmor. Programming with angelic nondeterminism. In *Principles of Programming Languages, POPL*, pages 339–352, January 2010.

15. Borzoo Bonakdarpour and Sandeep S. Kulkarni. On the complexity of synthesizing relaxed and graceful bounded-time 2-phase recovery. In *Proceedings of FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009*, pages 660–675, 2009.
16. Tayeb Bouhadiba, Quentin Sabah, Gwenaël Delaval, and Eric Rutten. Synchronous control of reconfiguration in fractal component-based systems: a case study. In *Proceedings of the ninth ACM international conference on Embedded software*, EMSOFT'11, pages 309–318, Taipei, Taiwan, October 2011.
17. Victor Braberman, Nicolas D'Ippolito, Jeff Kramer, Daniel Sykes, and Sebastian Uchitel. Morph: A reference architecture for configuration and behaviour self-adaptation. In *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, CTSE 2015, pages 9–16, New York, NY, USA, 2015. ACM.
18. C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, 2008.
19. A. Cervin, J. Eker, B. Bernhardsson, and K.E. Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems*, 23(1–2):25–53, July 2002.
20. A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang. Synchronous and bidirectional component interfaces. In *Computer Aided Verification*, volume 2404 of *LNCS*, pages 414–427, Copenhagen, Denmark, July 2002.
21. Feedback Computing. http://www.feedbackcomputing.org/.
22. Control for Embedded Systems Cluster. Roadmap on control of real-time computing systems. Technical report, EU/IST FP6 Artist2 NoE, 2006.
23. G. Delaval. Bzr http://bzr.inria.fr.
24. G. Delaval. Modular distribution and application to discrete controller synthesis. In *International Workshop on Model-driven High-level Programming of Embedded Systems (SLA++P'08)*, Budapest, Hungary, April 2008.
25. G. Delaval and E. Rutten. A domain-specific language for multi-task systems, applying discrete controller synthesis. *J. on Embedded Systems*, 2007(84192):17, January 2007.
26. G. Delaval, É. Rutten, and H. Marchand. Integrating discrete controller synthesis into a reactive programming language compiler. *Discrete Event Dynamic Systems*, 23(4):385–418, December 2013.
27. Gwenaël Delaval, Hervé Marchand, and Éric Rutten. Contracts for modular discrete controller synthesis. In *ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2010)*, pages 57–66, Stockholm, Sweden, April 2010.
28. Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. Hope for the best, prepare for the worst: Multi-tier control for adaptive systems. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 688–699, New York, NY, USA, 2014. ACM.
29. Nicolás D'ippolito, Victor Braberman, Nir Piterman, and Sebastián Uchitel. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Trans. Softw. Eng. Methodol.*, 22(1):9:1–9:36, March 2013.
30. Christopher Dragert, Juergen Dingel, and Karen Rudie. Generation of concurrency control code using discrete-event systems theory. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 146–157, New York, NY, USA, 2008. ACM.
31. E. Dumitrescu, A. Girault, H. Marchand, and E. Rutten. Multicriteria optimal discrete controller synthesis for fault-tolerant tasks. In *Proc. of the 10th IFAC Int. Workshop on Discrete Event Systems (WODES'10)*, Sept. 2010.
32. Sylvain Durand, Anne-Marie Alt, Daniel Simon, and Nicolas Marchand. Energy-aware feedback control for a H.264 video decoder. *International Journal of Systems Science*, page 16, August 2013.

33. Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *Proc. 26th IEEE/ACM Int. Conf. Automated Software Engineering (ASE 2011)*, pages 283–292, 2011.
34. Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, New York, NY, USA, 2014. ACM.
35. Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro Vittorio Papadopoulos, Suprio Ray, Amir M. Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. Software engineering meets control theory. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15, pages 71–82, Piscataway, NJ, USA, 2015. IEEE Press.
36. A. Gamatié, H. Yu, G. Delaval, and E. Rutten. A case study on controller synthesis for data-intensive embedded systems. In *Proceedings of the 6th IEEE International Conference on Embedded Software and Systems (ICESS'2009)*, HangZhou, Zhejiang, China, May 2009.
37. Benoit Gaudin and Paddy Nixon. Supervisory control for software runtime exception avoidance. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, C3S2E '12, pages 109–112, New York, NY, USA, 2012. ACM.
38. Benoit Gaudin, Emil Iordanov Vassev, Patrick Nixon, and Michael Hinchey. A control theory based approach for self-healing of un-handled runtime exceptions. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 217–220, New York, NY, USA, 2011. ACM.
39. Alain Girault and Eric Rutten. Automating the addition of fault tolerance with discrete controller synthesis. *Int. journal on Formal Methods in System Design*, 35(2):pp. 190–225, october 2009. http://dx.doi.org/10.1007/s10703-009-0084-y.
40. Soguy Mak-Karé Gueye, Noël de Palma, and Eric Rutten. Coordination control of component-based autonomic administration loops. In *Proceedings of the 15th International Conference on Coordination Models and Languages, COORDINATION,* 3-6 June, Florence, Italy, 2013.
41. D. Harel. Can programming be liberated, period? *Computer*, 41(1):28–37, 2008.
42. D. Harel, H. Kugler, and A. Pnueli. Synthesis revisited: Generating statechart models from scenario-based requirements. In *Formal Methods in Software and Systems Modeling*, volume 3393 of *LNCS*, pages 309–324, 2005.
43. D. Harel and A. Pnueli. On the development of reactive systems. In *Logic and Models of Concurrent Systems, NATO*. Springer-Verlag, 1985.
44. J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE, 2004.
45. M. Iordache and P. Antsaklis. Concurrent program synthesis based on supervisory control. In *2010 American Control Conference*, 2010.
46. M. V. Iordache and P. J. Antsaklis. Petri nets and programming: A survey. In *Proceedings of the 2009 American Control Conference*, pages 4994–4999, 2009.
47. Gabriel Kalyon and Tristan Le Gall. Smacs http://www.smacs.be/.
48. Christos Karamanolis, Magnus Karlsson, and Xiaoyun Zhu. Designing controllable computer systems. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, HOTOS'05, pages 9–9, Berkeley, CA, USA, 2005. USENIX Association.
49. J. Kephart. Feedback on feedback in autonomic computing systems. In *7th International Workshop on Feedback Computing*, San Jose, California, 2012.
50. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, January 2003.

51. Narges Khakpour, Farhad Arbab, and Eric Rutten. Supervisory controller synthesis for safe software adaptation. In *12th IFAC - IEEE International Workshop on Discrete Event Systems, WODES, Cachan, France, May 14-16*, 2014.
52. H. Kugler, C. Plock, and A. Pnueli. Controller synthesis from LSC requirements. In *Fundamental Approaches to Software Engineering, FASE'09, York, UK, March 22-29*, 2009.
53. S. S. Kulkarni and A. Ebnenasir. Automated synthesis of multitolerance. In *DSN '04 Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 209–, 2004.
54. Philippe Lalanda, Julie A. McCann, and Ada Diaconescu. *Autonomic Computing - Principles, Design and Implementation*. Undergraduate Topics in Computer Science Series, Springer, 2013.
55. Mikael Lindberg and Karl-Erik Årzén. Feedback control of cyber-physical systems with multi resource dependencies and model uncertainties. In *31st IEEE Real-Time Systems Symposium*, San Diego, California, USA, November 2010.
56. Marin Litoiu, Mary Shaw, Gabriel Tamura, Norha M. Villegas, Hausi A. Müller, Holger Giese, and Eric Rutten. What can control theory teach us about assurances in self-adaptive software systems? In Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese, editors, *Software Engineering for Self-Adaptive Systems*, volume (this volume) of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2016.
57. Marin Litoiu, Murray Woodside, and Tao Zheng. Hierarchical model-based autonomic control of software systems. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.
58. Cong Liu, A. Kondratyev, Y. Watanabe, J. Desel, and A. Sangiovanni-Vincentelli. Schedulability analysis of petri nets based on structural properties. In *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pages 69 –78, june 2006.
59. C. Lu, J. Stankovic, T. Abdelzaher, G. Tao, S. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *Real-Time Systems Symposium*, december 2000.
60. C. Lu, J.A. Stankovic, S.H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling and algorithms. *Real-Time Systems Journal, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 23(1/2):85–126, July 2002.
61. Martina Maggio, Enrico Bini, Georgios Chasparis, and Karl-Erik Årzén. A game-theoretic resource manager for RT applications. In *25th Euromicro Conference on Real-Time Systems, ECRTS13*, Paris, France, July 2013.
62. Luc Malrait, Sara Bouchenak, and Nicolas Marchand. Experience with ConSer: A system for server control through fluid modeling. *IEEE Transactions on Computers*, 60(7):951–963, 2011.
63. H. Marchand. Sigali http://www.irisa.fr/vertecs/Logiciels/sigali.html.
64. H. Marchand, P. Bournai, M. Le Borgne, and P. Le Guernic. Synthesis of discrete-event controllers based on the signal environment. *Discrete Event Dynamic Systems: Theory and Applications*, 10(4):325–346, October 2000.
65. M Amir Moulavi, Ahmad Al-Shishtawy, and Vladimir Vlassov. State-space feedback control for elastic distributed storage in a cloud environment. In *ICAS 2012, The Eighth International Conference on Autonomic and Autonomous Systems*, pages 18–27, 2012.
66. T. Patikirikorala, A. Colman, J. Han, and L. Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012, Zurich, Switzerland*, 2012.
67. T. Patikirikorala, L. Wang, A. Colman, and J. Han. Hammerstein Wiener nonlinear model based predictive control for relative QoS performance and resource management of software systems. *Control Engineering Practice*, 20:49–61, 2012.

68. V.V. Phoha, A.U. Nadgar, A. Ray, and S. Phoha. Supervisory control of software systems. *Computers, IEEE Transactions on*, 53(9):1187 – 1199, sept. 2004.
69. P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987.
70. Daniel Sykes, Domenico Corapi, Jeff Magee, Jeff Kramer, Alessandra Russo, and Katsumi Inoue. Learning revised models for planning in adaptive systems. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 63–71, Piscataway, NJ, USA, 2013. IEEE Press.
71. C. Wallace, P. Jensen, and N. Soparkar. Supervisory control of workflow scheduling. In *Advanced Transaction Models and Architectures Workshop (ATMA), Goa, India*, 1996.
72. Y. Wang, H.K. Cho, H. Liao, A. Nazeem, T. Kelly, S. Lafortune, S. Mahlke, and S. A. Reveliotis. Supervisory control of software execution for failure avoidance: Experience from the gadara project. In *Proc. of the 10th IFAC Int. Workshop on Discrete Event Systems (WODES'10)*, Sept. 2010.
73. Y. Wang, S. Lafortune, T. Kelly, M. Kudlur, and S. Mahlke. The theory of deadlock avoidance via discrete control. In *Principles of Programming Languages, POPL*, pages 252–263, Savannah, USA, 2009.
74. W. M. Wonham. TCT http://www.control.utoronto.ca/cgi-bin/dlxptct.cgi.
75. Feng Xia, Guosong Tian, and Youxian Sun. Feedback scheduling: An event-driven paradigm. *ACM SIGPLAN Notices*, 42(12):7–14, Dec 2007.
76. Christos A Yfoulis and Anastasios Gounaris. Honoring slas on cloud computing services: a control perspective. In *Proceedings of the European Control Conference*, 2009.
77. Xiaoyun Zhu. Application of control theory in management of virtualized data centres. In *Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, Paris, France, 2010.
78. Xiaoyun Zhu, Zhikui Wang, and Sharad Singhal. Utility-driven workload management using nested control design. In *American Control Conference, 2006*, pages 6–pp. IEEE, 2006.