Chapter 2
Self-Adaptation of Software Using Automatically Generated Control-Theoretical Solutions

Stepan Shevtsov, Danny Weyns, and Martina Maggio

Abstract Control theory has contributed a set of foundational techniques to handle "change" at runtime in software applications. These techniques however have fundamental limitations as well: (i) they require the development and understanding of mathematical models; (ii) synthesizing solutions is often done on a per-problem basis, discouraging flexibility and generality. Software engineering, as a discipline, has always aimed at finding reusable and modular solutions. The combination of the desire to apply formally grounded control-theoretical principles and reuse existing solutions has motivated research on the topic of *automatically generated control solutions*. This research aims at designing control strategies in an automated way from data that qualifies the given problem at hand. This chapter provides an overview of the research topic of automatically generated control-theoretical solutions, explaining the key research contributions and paving the way for future research.

2.1 Introduction

Software applications need, more than ever, to be able to deal with "change" [30, 41]. Software needs to be continuously available, which in turns requires that developers treat change as a first-class concern in the complete life cycle of the application development, operation, and maintenance. Software applications are nowadays expected to deal seamlessly with different types of change, such as resource fluctuations [37], component failures [44], requirement modifications [6, 49], different user preferences [43], and much more [1, 2, 9, 14, 27, 42]. Often,

S. Shevtsov (✉) · D. Weyns
Linnaeus University, Växjö, Sweden
KU Leuven, Leuven, Belgium
e-mail: stepan.shevtsov@lnu.se; danny.weyns@kuleuven.be

M. Maggio
Lund University, Lund, Sweden
e-mail: martina.maggio@control.lth.se

© Springer Nature Singapore Pte Ltd. 2019

Y. Yu et al. (eds.), *Engineering Adaptive Software Systems*,
https://doi.org/10.1007/978-981-13-2185-6_2

these changes are not predictable at design time, requiring software to execute with incomplete knowledge and face new challenges during operation [50, 53]. Consequently, software engineering researchers are experimenting with new solutions that can handle change at runtime without incurring into penalties, slowdown, and downtime. Generally speaking, the software built to deal with change is often called "self-adaptive" [15, 17, 51], for the ability to modify its own behavior and adapt to the current execution conditions.

Continuous- and discrete-time control theory[1] has been identified as a promising approach to design self-adaptive software [10, 18, 26, 56]. However, the wide adoption of control-theoretical solutions in the design of self-adaptive systems has been limited by a number of factors.

First and foremost, continuous- and discrete-time control solutions often require a "physical" model of the object to be controlled. In the case of low-level resources – such as CPU, memory, and network bandwidth – researchers have proposed models that attempt to capture the phenomena of interest [3, 20, 55] with a precision sufficient to perform adaptation. However, it is very difficult to extract control-theoretical (i.e., equation-based) models for the behavior of software applications. This has been one of the main reasons why several researchers have argued that applying control theory to adapt the higher-level software elements is a more complex problem [4, 11, 22]. Other reasons are the diversity and interplay of requirements and the need for instrumenting software to obtain measurements from sensors and enacting the system through actuators [12, 28]. Second, the models often become complicated, calling for elaborate solutions from the mathematical perspective. Finally, since appropriate and accurate models are so difficult to write, existing control-based approaches are often tailored for a particular problem, while software engineers usually aim for reusable solutions. These observations have been recently confirmed by a systematic study on control-theoretical software adaptation, highlighting the shortcomings of the existing ad hoc control-theoretical solutions [47].

As a response to these shortcomings, researcher aimed at automatically generating control solutions. These solutions are general enough to tackle a variety of problems, trading off the optimality that could be reached by tailored solutions. The code for these general solutions can be automatically generated based on observations and data from the software application that should be controlled. Simple linear models describing the software behavior are automatically extracted from the data and used – at runtime – to synthesize a control solution. This chapter gives an overview of the state of the art of the research in automatically generated control strategies for software applications and outlines promising paths for future work.

The remainder of this chapter is structured as follows. Section 2.2 provides a brief background on automatically generated control-theoretical adaptation of software.

[1] In this chapter, we restrict ourselves to continuous- and discrete-time control [8, 54]. Discrete event systems are out of our scope.

In Sect. 2.3, we delve into details discussing the differences among the proposed solutions. Finally, Sect. 2.4 outlines a number of challenges for future research, and Sect. 2.5 draws some conclusions.

2.2 Background

This section explains the basic principle behind automatically generated control-theoretical solutions and its use for self-adaptation.

The overall objective of automatically generated control-theoretical adaptation is the simplification of the software design process. The aim of these strategies is to provide the software engineer with the advantages of a control-theoretical design, without the need for in-depth control expertise. The main advantage of control-theoretical solutions is the presence of *formal guarantees* [24]. If mastered correctly, the use of the knowledge coming from control theory allows for certified and verifiable solutions, where desired properties can be guaranteed *by design*. For example, with control theory it is possible to precisely calculate the amount of disturbance the system can withstand or to prove that the system will not overconsume resources in changing external conditions.

Figure 2.1 shows a typical control-theoretical feedback loop that is used in self-adaptive software systems. Reading the figure from left to right, the *Goal* represents a particular level of software quality that should be achieved by self-adaptation. The Goal is often specified as a setpoint, i.e., a certain value of a nonfunctional requirement, such as a specific service failure rate or response time. Using the setpoint and the *Measured Output* value for the same software quality, an *Error* is calculated as $Setpoint - MeasuredOutput$, where the *-1* block indicates that the Measured Output value should be subtracted. The *Feedback Controller* uses the Error in order to compute the *Control Signal*, a value or a vector of values that effect the *Software System*. If designed correctly, the Control Signal will result in a Measured Output that is equal or very close to the Goal value. The *Disturbances*, such as changing availability of resources or component failures, affect the software behavior at runtime. So one of the main purposes of control strategies is to neglect the effect of Disturbances on the system.

Historically, many manually generated control strategies used the typical feedback loop shown in Fig. 2.1. The automated strategies have two main differences

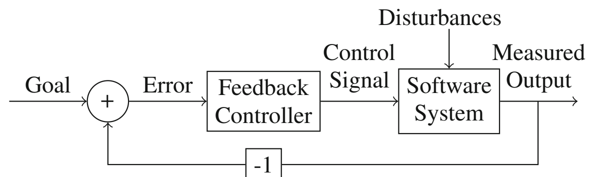

Fig. 2.1 A typical control-theoretical feedback loop

from these solutions. First, the automated strategies require certain conditions to be satisfied and the availability of specific software functions:

- The developer that wants to generate and use the control strategy should have access to the software system, which should be working and on which experiments should be done and data must be collected – the data is used in an automated way to build a model of the software that can be used for control purposes;
- The developer should be able to qualify, quantify, and measure the requirements that must be satisfied on the system – these requirements are then translated into goals and objectives that the controller will try to achieve;
- The developer must provide access to a set of sensors that get reliable data about the quantifiable objectives (e.g., measure the response times of a cloud application);
- The developer must provide access to a set of actuators (tunable parameters of the system) that can be used during runtime to modify the behavior of the software application (e.g., the percentage of rejected requests, or different implementations of the same functionality).

Second, the Feedback Controller is created automatically. Namely, the automated solution starts by running experiments on the software application, changing the values of the actuators according to predefined patterns and measuring the values of the goals in the tested configurations. With this data, the solution generates a mathematical model of the software using system identification [34].[2] Finally, this model is used to synthesize a controller that provides guarantees on certain system properties. The controller – synthesized in form of equations and subsequently in form of a code block – adapts the behavior of the software changing the values of the actuators to achieve the given goals. The resulting controller is often tunable – some parameters have default values that can be changed to alter the behavior of the controller itself. For example, parameters can be used to exploit the trade-off between robustness to disturbances and speed of convergence. The software engineer can select these parameters based on experience and on the specific execution conditions.

2.3 Automated Control-Theoretical Software Adaptation

This section outlines the research progress in self-adaptation of software using automatically generated control-theoretical solutions. We discuss five different research problems that have been explored. Figure 2.2 gives an overview of the

[2]Other model synthesis techniques can be used to produce system model. But historically, automated approaches used system identification as it is fast and approximates software well enough for controllers to work.

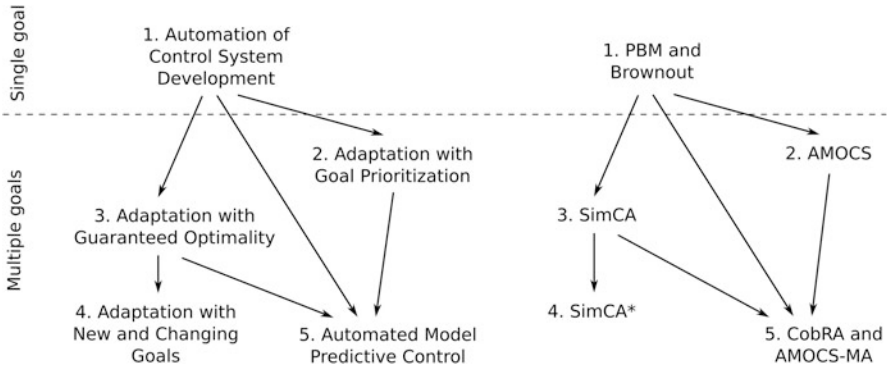

Fig. 2.2 Research in automated control-theoretical software adaptation: progress steps (left) and approaches (right)

research steps and shows representative approaches for each step. The arrows in the figure show the contribution of each step/approach to the following efforts.

The initial research was primarily targeting the automation of a control solution development. Based on prior experience with control of software applications, some generalization arose and led to the introduction of the Push-Button methodology (PBM) [22]. At the same time, a similar method called Brownout [33] was applied in a specific software domain, cloud applications. The next clear research goal has been the extension of automated methodologies to support *multiple adaptation goals* simultaneously, e.g., to achieve a specific performance level and minimize cost at the same time. The first proposed extension has been the Automated Multi-objective Control of Software (AMOCS) approach [23], followed by the Simplex Control Adaptation (SimCA) [45]. SimCA tackled the problem of multi-objective adaptation by combining controllers with the simplex optimization algorithm in a hierarchical structure. Then, SimCA* [48] introduced components that adjust the adaptation mechanism at runtime, to deal with new types of goals and changes in the set of adaptation goals (e.g., adding a new goal, removing a goal). Finally, the use of *Model Predictive Control* (MPC) was investigated. In this approach, the controller acts based on the current feedback from the software but uses the model of its own behavior to predict the software evolution. The fully automated MPC-based approach is called Automated Multi-objective Control of Software with Multiple Actuators (AMOCS-MA) [36].

The main properties of all automated control-theoretical adaptation approaches are listed in Table 2.1; these approaches will be discussed in detail in the following sections.

Table 2.1 Automated control-theoretical adaptation approaches

Approach	Inputs (goals)	Main pros	Main cons
Brownout, PBM	1-setpoint	Automation, guarantees	Handles only one goal
AMOCS	n-setpoint, 1-optimization	Multiple goals and prioritization	Suboptimal adaptation decisions
SimCA	n-setpoint, 1-optimization	Guarantees + optimality	Setpoints, needs knowledge about some of the system parameters
SimCA*	n-setpoint, n-threshold, 1-optimization	Handles new types of goals and goal changes at runtime	Needs knowledge about some of the system parameters
AMOCS-MA	n-setpoint, 1-optimization	Guarantees + optimality, does not need system knowledge, flexible computation time	Sensitive to disturbances and model inaccuracies

2.3.1 Automation of Control System Development

Control-theoretical approaches were first used in software adaptation more than a decade ago [1, 2, 14]. However, most of these approaches aim to solve a specific problem at hand. Therefore, new problems would require modifications or even replacement of a control system, which in turn requires expertise in control theory, extra resources, and effort. To overcome this concern, researchers have studied the ways to automate the entire process of control system development from the model synthesis to the formal analysis of guarantees. This became the first step of research on applying automatically generated control-theoretical solutions in software adaptation.

The representative of the first step of research are Brownout [33] and PBM [22]. Both these approaches are based on the same underlying principles (creating a first-order model from data and controlling that first-order model using pole placement). Brownout is applied to the more confined domain of cloud computing applications and is tailored to the specific problem of capacity shortages. Because of this, Brownout achieves – on its own problem – better performance than the application of the PBM controller without any modifications. We provide details on both of these approaches below.

2.3.1.1 Brownout

The main idea behind Brownout [33, 35] is to apply the principles of graceful degradation to cloud applications using control theory. Cloud applications behave according to the request-response paradigm, with clients issuing requests and a certain number of replicas of the same application providing the according responses. When producing the response to the user requests, it is often possible

to identify a part of the response that is the mandatory to display and a part of the response that would provide a better user experience and increase revenues, but is not mandatory. In the case of a travel agency website, the mandatory part of the response is the flight search, while additional optional information are car rental locations and hotel suggestions. Clearly, the application owner wants to provide the additional information, but not at the expense of losing a customer. Brownout divides the response into the two mentioned parts and measures the response time to determine how much percentages of the optional content should be served. This percentage is called the *dimmer* value. The goal of brownout is to have as big dimmer as possible, i.e., to show as much optional content as possible, without penalizing response times.

Brownout assumes that the cloud application behaves according to a simple first-order linear model, where the value of the 95th percentile of the response time τ_{95} varies depending on the dimmer value as follows:

$$\tau_{95}(k) = \alpha\,\theta(k-1) + \delta\tau_{95}(k), \tag{2.1}$$

where $\theta(k)$ is the dimmer value; $\alpha(k-1)$ is a time-varying coefficient that depends on the computing platform and can be estimated; $\delta\tau_{95}(k)$ is a disturbance, interfering with the nominal system's behavior; and k is the discrete time instance.

Based on the model (2.1), the following controller is then synthesized using loop shaping [8]:

$$\theta^{*}(k) = \theta(k-1) + \frac{1-p_b}{\hat{\alpha}(k)} \cdot e_{\tau 95}(k) \tag{2.2}$$

where $\hat{\alpha}(k)$ is an estimate of $\alpha(k)$ obtained with a recursive least squares (RLS) filter, p_b is a controller parameter called pole, and $e_{\tau 95}(k)$ is the error between the desired 95th percentile of the response time $\bar{\tau}_{95}(k)$ and the actual value. The pole p_b can be used to trade the speed of controller convergence for robustness to model perturbations. The analysis of the brownout closed loop allows to prove a number of properties, such as system stability and zero steady-state error. However, this proof is subject to how well the model (2.1) approximates the behavior of the cloud application.

Brownout uses a single actuator (the dimmer value) to achieve a single goal, specified in terms of a setpoint for the response time statistic. The control strategy in Brownout can be greatly improved, and many follow-ups were devised. For example, an event-based version of the brownout paradigm [19] explores a similar cloud problem but controls the server queue length. Furthermore, extensions that include brownout load balancing were considered [21, 32]. They demonstrate that state-of-the-art load balancers which use response times as a measure for determining where to send requests do not work with brownout-aware applications. This is a natural limitation as the brownout controller can satisfy only a single goal and therefore cannot form a multi-objective control strategy with other controllers.

Brownout was designed specifically for cloud applications, so strictly speaking, it is not a generally applicable solution. However, it is important to include Brownout in this work as it became the first building block for development of automated control-theoretical adaptation. The generally applicable Push-Button methodology, discussed in the following section, is based on the same principles and shares many elements with Brownout.

2.3.1.2 Push-Button Methodology

The PBM methodology [22] works in a way similar to Brownout but *goes beyond a single goal and a single actuator. Also, it introduces the idea of identifying the model online. Unlike in Brownout, where model is pre-determined, PBM builds a model directly from the data received by running experiments on the software and produces a controller for this model*. Figure 2.3 shows the two phases of the methodology: model building and controlling.

The input required by PBM from a software engineer is a method to set the actuator value and a method to collect measurements about the system goal. Based on this input, PBM first produces a linear model $\mathcal{M}$ of the software:

$$\mathcal{M}: \; y(k) = \alpha(k-1) \cdot u(k-1) \tag{2.3}$$

where the input u is the value of the actuator, the output y is the effect of the actuator on the goal, the parameter α is a time-varying coefficient that is determined during model building by feeding different input values as u and measuring the resulting outputs y, and k is a discrete time instance.

After the model building, the controller synthesis phase automatically generates a proportional-integral controller C that works on the model $\mathcal{M}$ and adapts the software.

$$C: \; u(k) = u(k-1) + \frac{1-p_b}{\alpha} \cdot e(k) \tag{2.4}$$

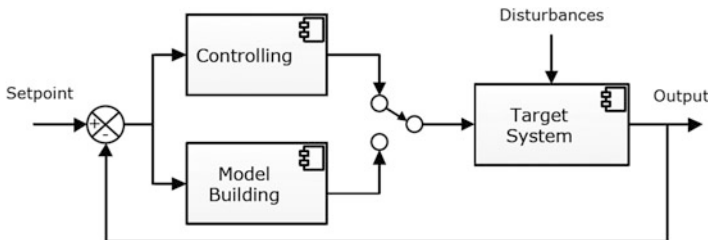

Fig. 2.3 The two operational phases of PBM

The controller has one parameter, p_b, that has the same role that it had in the Brownout controller. More guidelines on how to tune the controller parameter p_b are available in [22].

To address model inaccuracies and small perturbations during software operation, the value of α is updated at runtime. In case of critical changes (e.g., a software component failure), PBM restarts the model building phase and regenerates the controller.

2.3.2 Adaptation with Goal Prioritization

In order to automatically create control solutions for more practical problems, researches have studied the ways to address multiple adaptation goals simultaneously. The first automated approach that offered control-based multi-objective software adaptation was AMOCS [23]. *This approach extends the methodology behind PBM to use multiple actuators and multiple controllers in a cascaded structure*; see Fig. 2.4.

AMOCS works as follows. The set of available actuators $\mathcal{A} = \{a_1, \ldots, a_m\}$ is partitioned to reach the set of goals $\mathcal{G} = \{g_1, \ldots, g_n\}$, where $m \geq n$, i.e., the system should have more actuators than goals. The goals are added into the set $\mathcal{G}$ according to their priority order, forming the chain $< g_1, g_2, \ldots g_n >$, where g_1 is the most important goal and g_n is the least important one. All goals, except the last one, are specified as setpoint values to be achieved by the adaptation. The last goal g_n is always the optimization of a specific value (e.g., maximization of profit, minimization of cost). $\mathcal{A}_i$ denotes the subset of actuators used to achieve the goal g_i. AMOCS assumes that every actuator is used:

$$\bigcup_{i \in \{1 \ldots n\}} \mathcal{A}_i = \mathcal{A}, \tag{2.5}$$

and each actuator is assigned to a single goal only:

$$\forall i, j \in \{1 \ldots n\}, i \neq j \implies \mathcal{A}_i \cap \mathcal{A}_j = \emptyset, \tag{2.6}$$

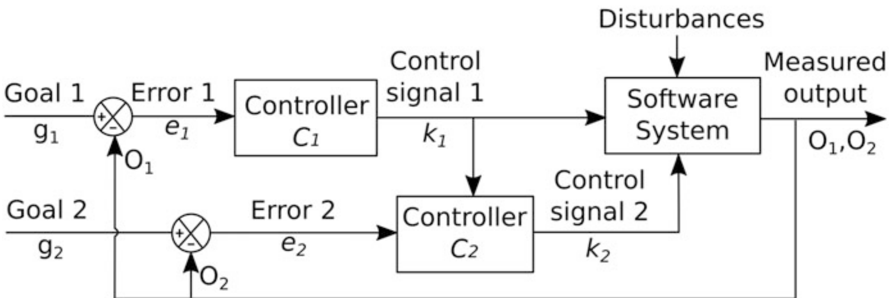

Fig. 2.4 A self-adaptive software with AMOCS (for 2 goals)

A first instance of PBM controller C_1, see (2.4) for a controller description, is then used to translate the discrete set of configurations of all the actuators $\mathcal{A}_1$ related to the first goal g_1 into a single configuration that satisfies this goal. This configuration is then sent in the form of control signal k_1 to the software system and to the second instance of PBM controller C_2,which tries to achieve the second goal g_2 with the available actuators $\mathcal{A}_2$ and operating conditions. The resulting configuration is sent to software as control signal k_2. If goals g_1 and g_2 are not related, the control signal k_1 will still be received by controller C_2, but it will not affect the reachability of the goal g_2.

In this controller chain, only the first goal is guaranteed to be stable, while the stability of the others depend on the disturbances and on the control values set by the previous controllers in the chain. In other words, the goal g_2 is guaranteed to be reached only if control signal k_1 allows to reach it. The last optimization requirement is reached to the best of the chain ability; hence there is no guarantee for the solution optimality. Despite the lack of formal guarantees, the experiments with AMOCS show that the chain of controllers behaves well in a variety of different scenarios and can successfully handle multiple goals of a setpoint type.

2.3.3 Adaptation with Guaranteed Optimality

Guided by the need for stronger adaptation guarantees in systems with multiple goals, the research explored new ways to automatically build the control system. The approach resulting from these efforts is called Simplex Control Adaptation (SimCA) [45]. *SimCA combines PBM with the simplex optimization method, utilizing the advantages of both approaches*. SimCA finds a system configuration that satisfies multiple goals, reaches optimality with respect to an additional goal, achieves robustness to environmental disturbances and measurement inaccuracy, and provides control-theoretical adaptation guarantees. To that end, SimCA runs on-the-fly experiments on the software in an automated fashion, builds a set of linear models of the software at runtime, creates a set of tunable PI controllers that operate on these models and independently compute control signals for each of the goals, and combines controller outputs using the simplex method to adapt the system. Figure 2.5 schematically shows the primary building blocks of SimCA.

SimCA builds a self-adaptive system in three phases executed during system operation:

1. In the *Identification* phase, n linear models of the controlled system are built. SimCA uses multiple instances of the PBM model $\mathcal{M}$, where each model $\mathcal{M}_i$, $i \in [1, n]$, is responsible for one goal s_i. Similar to PBM, each model is automatically learned at runtime by running the experiments on the software (see Sect. 2.3.1 for details). As in PBM, the model $\mathcal{M}_i$ automatically adjusts at runtime according to changes in the system behavior.

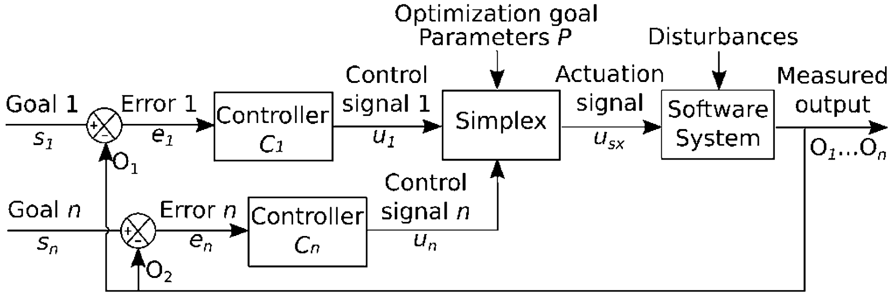

Fig. 2.5 A self-adaptive software with SimCA

2. In the *Controller Synthesis* phase, SimCA constructs a set of n controllers; each controller C_i is responsible for the i-th goal. C_i calculates the control signal $u_i(k)$ at the current time step k depending on the previous value of control signal $u_i(k-1)$, model coefficient α_i, parameter pole p_i, and the error $e_i(k-1)$, with $e_i = s_i - O_i$. Similar to PBM, p_i is used to tune the controllers and trade off different system properties.

$$u_i(k) = u_i(k-1) + \frac{1-p_i}{\alpha_i} \cdot e_i(k-1) \qquad (C_i)$$

3. In the *Operation* phase, the set of controllers effectively perform control. Each controller C_i manages one goal s_i, rejects disturbances acting on the according output $O_i(k)$, and provides an output signal $u_i(k)$. SimCA combines the signals $u_i(k)$ from all the controllers and uses the simplex method to calculate the actuation signal u_{sx} that drives the system toward an output that satisfies all adaptation goals.

   Generally, the simplex method allows to find an optimal solution to a linear problem written in the standard form:

$$\max\{c^{\mathrm{T}}x \mid Ax \leq b; x \geq 0\} \qquad (2.7)$$

   where x represents the vector of variables (to be determined), c and b are vectors of (known) coefficients, A is a (known) matrix of coefficients, and $(\cdot)^{\mathrm{T}}$ is the matrix transpose [16].

   In SimCA each equation, except the last one, represents a goal s_i to be satisfied. The last equation ensures that the system selects a valid actuation signal by constraining the values that can be taken by elements of the vector x, e.g., $x \geq 0$. The control signals $u_i(k)$ produced during the control phase replace constants b, whereas matrix A and vector c^{T} are substituted with the monitored parameters $\mathcal{P}(k)$ of the system. The goal of simplex is to find a proper actuation signal u_{sx}, i.e., vector x.

Note that SimCA uses a simplex variant with equalities ($Ax = b$) in order to prevent simplex from changing the effect of control signal $u_i(k)$ on the output signal $O_i(k)$. Instead, simplex is responsible for seamless translation of control signals $u_i(k)$ to actuation signal u_{sx}. This allows to provide the entire set of *control-theoretical guarantees*, including stability, absence of overshoot, tunable settling time, and robustness to disturbances. A major advantage of SimCA over approaches from the previous research steps is that simplex guarantees solution optimality, meaning that all the system goals are guaranteed to be achieved. An interested reader may refer to [45] for further details.

A follow-up work [46] compares SimCA with an architecture-based ActivFORMS approach using a simulated service-based system. The study shows that both approaches can deal with multiple goals and provide guaranteed solution optimality. However, SimCA achieves better results in the presence of runtime changes as it does not rely on data verified at design time. Except optimality, the two adaptation approaches offer different guarantees. The design of SimCA adaptation mechanism allows to formally prove the properties of underlying system and guarantee that they will hold at runtime independent of the system parameters. ActivFORMS, on the other hand, can guarantee the functional correctness of the implementation of the adaptation algorithm, such as the absence of erroneous states and correct interaction between adaptation components.

2.3.4 Adaptation with New and Changing Goals

One interesting research line for automated methodologies and for control methodologies in general is the selection and support of types of adaptation goals. The previously developed automated approaches had two major drawbacks. First, they addressed goals specified either in the form of particular setpoint values to be achieved by the system (S-goal) or values to be optimized (O-goal), while many software systems need to address a threshold goal that keeps a value above/below a threshold (T-goal). A typical example is limiting the response time of a Web server. Approaches such as described in [31, 33, 38] solve this problem either by optimizing the response time (O-goal) or by defining a setpoint for response time that the controller should guarantee (S-goal), when the actual requirement is to keep response time lower than a certain threshold. Second, the previously developed approaches did not provide support for changing the set of system requirements during operation, which requires on-the-fly adjusting, activation, and deactivation of adaptation goals. Changing requirements are important in practice, e.g., to deal with drastic changes in the system or its environment that may require the system to change from one set of requirements to another.

In order to address the two mentioned concerns, the SimCA approach (see Sect. 2.3.4) was reworked and upgraded into SimCA* [48]. *Compared to original*

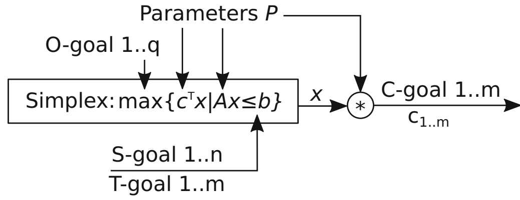

Fig. 2.6 Goal Transformation phase of SimCA*

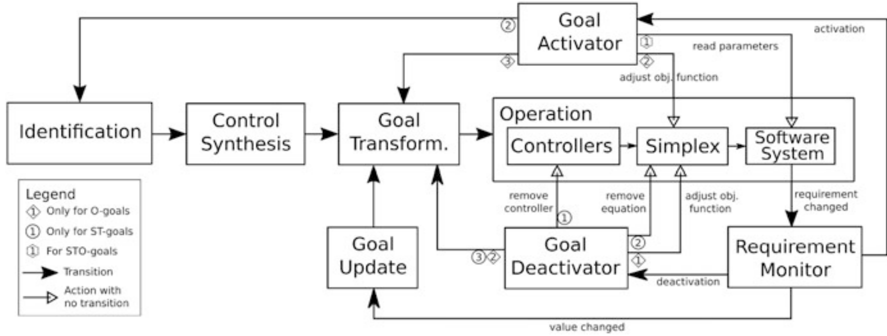

Fig. 2.7 Dealing with requirement changes in SimCA*. Numbers in circles/diamonds show the sequence of actions

SimCA, the new approach includes an additional Goal Transformation phase (Fig. 2.6) *and the necessary mechanisms to support changing system requirements by activating/deactivating goals* (Fig. 2.7).

The *Goal Transformation phase* of SimCA* is performed between the Controller Synthesis and Operation phases. The purpose of this phase is to transform T-goals into goals that can be controlled by the original SimCA controller (C_i). As such, the approach uses simplex, where each equation in the system (2.7), except the last one, represents an S-goal or T-goal to be satisfied (see Fig. 2.6). Equalities are used for S-goals, while inequalities are used for T-goals. The last equation ensures that the system selects a valid solution, the vector x, by the means of constraints, e.g. $x \geq 0$. The goal of simplex is to find such vector x that satisfies all system goals; the details of how simplex finds such a solution can be found in the linear programming literature [16]. Knowing the vector x, each T-goal is transformed into a controller goal (C-goal) c_i as follows: $c_i = \mathcal{P}_i(k) * x$. The resulting C-goal represents a particular value of a corresponding T-goal. For example, a T-goal that should keep a value below a threshold will be transformed into a C-goal with a value that is equal to the lowest possible value of the goal below that threshold that satisfies all other requirements. All the C-goals and the original S-goals are then used by controllers (C_i) in the usual Operation phase described in Sect. 2.3.4.

In order to address the *changing system requirements*, SimCA* is equipped with a Requirement Monitor, Goal Activator, and Goal Deactivator components; see Fig. 2.7. The Requirement Monitor triggers the corresponding adaptation component after any system requirement is changed. The Goal Activator first reads

the relevant parameters $\mathcal{P}$ related to the activated goal. Then, in case of O-goal activation, it inserts $\mathcal{P}$ into the objective function c^{T} of simplex, performs a Goal Transformation (described above), and proceeds to standard Operation phase. In case of S- or T-goal, the Goal Activator triggers a standard Identification phase for the new goal, which is followed by Controller Synthesis, Goal Transformation, and Operation. The Goal Deactivator removes the according elements of the adaptation mechanism. Namely, when an S- or T-goal is deactivated, the corresponding controller is removed together with the equation responsible for the goal being deactivated. When an O-req is deactivated, the corresponding variables are removed from the objective function c^{T} of simplex. After that, the Goal Deactivator always triggers a Goal Transformation adapting the configuration of the control system to the new set of requirements, after which the system returns to standard Operation.

2.3.5 Automated Model Predictive Control

The scope of applicability of the first multi-objective control solutions is limited in different ways. For example, SimCA cannot prioritize goals or use infinite sets of values for the actuators, while AMOCS produces suboptimal solutions. *To eliminate these limitations, researchers have studied the application of automated model predictive control (MPC)* – a technique based on the optimization of a cost function and on the prediction of a future outcome of the adaptation. Generally, in control theory, MPC is considered particularly well suited for multi-objective problems with optimization, because all the interdependencies between actuators and goals are taken into account simultaneously, achieving a truly optimal solution.

The first research effort that identifies automated MPC as a potential multi-objective control strategy for self-adaptive systems is [5]. However, it lacks details and does not provide any analysis of guarantees. In the same research line – again for a specific problem, but with a general overlook – CobRA [7] provides a framework to reason about MPC and its application to computing systems. Although the model in CobRA has to be generated manually and fed to the system, the solution of the MPC problem is general with respect to the involved quantities. The paper only provides an example of the framework application, which also requires extensive manual tuning in order to tailor the equations to a specific problem. Although formal guarantees are not discussed in CobRA, it is possible to prove that they hold to the extent that the model allows. PLA [38, 39] is based on similar principles that CobRA. It uses a model of the environment and of the software to determine the best strategy to be followed using a model checker with the ability of looking into the future expectations for the system. CobRA and PLA have been compared [40] showing similar results but a different runtime behavior. The authors conclude that the concrete approach should be picked based on the problem at hand. For example, CobRA suits more for continuous inputs, while PLA works better with discrete control.

Finally, a fully automated model predictive control strategy was developed as a part of AMOCS-MA approach [36]. Similar to other automated solutions, AMOCS-MA starts with a model building phase. The following model S is synthesized:

$$S = \begin{cases} x(k+1) = A \cdot x(k) + B \cdot \Delta a(k) \\ O(k) = C \cdot x(k) \end{cases} \tag{2.8}$$

where k is a discrete time instance, $O(k)$ is the vector of all system outputs at time k, $\Delta a(k)$ is the control signal containing values of all actuators, $x(k)$ is the current system state, $x(k+1)$ is the next system state, and A, B, and C are the matrices of coefficients obtained with model learning by running experiments on the software at runtime. One of the AMOCS-MA advantages is that it reduces the model learning time by using special input signals in the model building phase; see details in [36]. As in other automated approaches, the model S is updated according to runtime changes that appear in the software system.

The model S is used by an MPC controller to minimize the following cost function, which handles all S-goals and O-goals:

Minimize $\Delta a(k+i-1)$, with $i = 1 \dots L$ in:

$$\sum_{i=1}^{L} \left\langle \sum_{j=1}^{p} q_j \cdot [O_j(k+i) - g_j(k+i)]^2 + \sum_{l=1}^{m} r_l \cdot \Delta a_l(k+i-1)^2 \right\rangle \tag{2.9}$$

Subject to: model S (2.8) and additional $\Delta a(k)$ constraints (see [36])

where k is a discrete time instance, L is the number of discrete time instances in the future used for predicting software behavior, p is the number of goals, q_j is the weight of goal j (allows goal prioritization), $O_j(k+i)$ is the predicted measured output of goal j at the i-th step in future, $g_j(k+i)$ is the value of goal j at the i-th step in future (this value is constant if goals do not change at runtime), m is the number of actuators, r_l is the weight of actuator l (allows actuator prioritization), and $\Delta a_l(k+i-1)$ is the predicted change in the value of actuator l at the $i-1$-th step in future.

As the controller depends on the model (2.8), it requires information about the system state $x(k)$. However, it is problematic to measure the system state $x(k)$ directly, so it is estimated instead. To accomplish this, AMOCS-MA uses a Kalman filter that computes an estimate $\hat{x}(k)$ of the state $x(k)$ based on the previous control signal $\Delta a(k-1)$, the measured outputs $O_j(k)$, prediction error, and a number of other parameters.

Using the estimate $\hat{x}(k)$, the MPC controller solves (2.9) and produces an optimal plan of control actions for the future i steps: $\Delta a(k+i-1)$, with $i = 1 \dots L$. The plan $\Delta a(k+i-1)$ contains particular values of all actuators at time instance $(k+i-1)$. However, AMOCS-MA uses only the first action of the plan, i.e., $\Delta a(k)$ is applied to software; see Fig. 2.8.

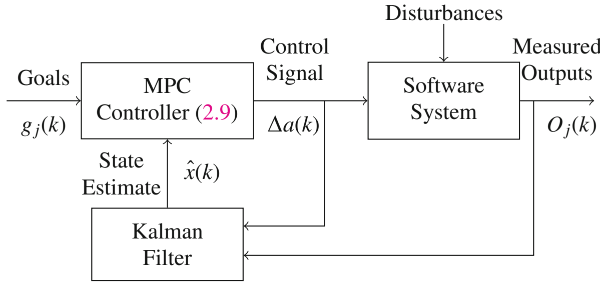

Fig. 2.8 A self-adaptive system with AMOCS-MA

The controller (2.9) guarantees stability, zero steady-state error, and minimal settling time by design. It also guarantees the optimality of a cost function specified by the user. This function has tunable weights for the system goals q_j and actuators r_l, allowing to trade off different system properties, e.g., to prioritize response time over cost.

2.4 Challenges

The analysis of automated control-theoretical adaptation solutions showed the use of various controllers, from hierarchical adaptive PI control (SimCA) to model predictive control (AMOCS-MA). However, most of these approaches use the PBM model (2.3) or its variations. Indeed, one of the key points behind this line of research is the difficulty in finding generic models that describe software applications and their behavior. Although the usual software models – architectural models and UML descriptions – are a very good reference to understand how the control code interfaces with the rest of the software application, they are not suitable for the control design process. To design a controller, there is usually a need to understand how the quantities that should be controlled are influenced by the actuators that one has available. Depending on the modeling effort that the software engineer is willing to do, the control strategies can be more or less effective:

- PLA [38, 39] and Brownout [33], for example, use explicit modeling of both the software behavior and the environment. Explicit modeling goes a long way for improving the performance of the control strategy that can be perfectly tailored for a new scenario using the given knowledge. Generally speaking, when an explicit model is available, the spectrum of results that it is possible to obtain is much wider, opening up possibilities and allowing for more precise results.
- SimCA [45] and SimCA* [48] lift some of the requirements on the modeling side. While no explicit disturbance model is written, the system parameters specified in the Simplex algorithm are part of prior knowledge that is given to the control strategy and that the controller does not have to identify based on experiments.

- The PBM [22], AMOCS [23], and AMOCS-MA [36] approaches use implicit modeling requiring a very limited effort from the software engineer. The engineer should only specify the actuators and sensor and possibly some weights that are unrelated to the model itself but specify the properties of controller and how to reach the goals. Despite the lack of modeling needs from the software engineer, these approaches still build a representation of the software in the form of equations in their model building phase. The synthesized model is then used to create a controller.
- Advances in control theory have recently unveiled a new set of methods denoted model-free control [13, 25, 29]. Model-free control synthesis does not build a model of the system to be controlled but only uses data to optimize a control strategy. To date, model-free control has not been applied to software and could open possibilities for performance improvement and to tackle the complexity of software systems in an automated way.

Apart from using the same type of model, all the automated approaches discussed in this chapter synthesize centralized control solutions deployed on a single software product. Such approaches are not suitable for systems where communication between components is limited or very costly. A recent work on architecture-based adaptation [52] introduced a number of patterns for designing decentralized adaptation solutions, where controllers make independent decisions but have some kind of interaction. The automated control solutions may definitely benefit from this and similar efforts, as they provide means to adapt an entirely new class of software systems.

2.5 Conclusions

Throughout the recent years, the automatically generated control-theoretical solutions have made a huge progress. Starting from addressing a single adaptation requirement, these solutions can now handle multiple goals of different types, deal with addition or removal of system requirements on the fly, or even adapt based on the predicted software evolutions. In this chapter, we listed the key research steps that led to such progress and highlighted the main approaches representing each of the steps. Surely, the automated approaches have limitations. For example, they use simple models that are not always accurate, and they are less effective in specific scenarios than controllers finely tuned for those scenarios. However, the main advantage of automated control comes from these limitations: simple models in combination with a generally applicable controller allow to build a control-based self-adaptive system without involvement of a control expert.

As for the future of automated control-based solutions, the research efforts can be aimed in two directions. First, as the scope of applicability and practical effectiveness of existing solutions is often unclear, these solutions should be tested in the industrial settings. Second, the researchers could use more state-of-the-art practices, such as model-free control or decentralized adaptation.

References

1. Abdelwahed, S., Kandasamy, N., Neema, S.: Online control for self-management in computing systems. In: Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, pp. 368–375 (2004)
2. Abdelzaher, T.F., Shin, K.G., Bhatti, N.: Performance guarantees for web server end-systems: a control-theoretical approach. IEEE Trans. Parallel Distrib. Syst. **13**(1), 80–96 (2002)
3. Abdelzaher, T., Stankovic, J., Lu, C., Zhang, R., Lu, Y.: Feedback performance control in software services. IEEE Control Syst. **23**(3), 74–90 (2003)
4. Andersson, J., de Lemos, R., Malek, S., Weyns, D.: Modeling dimensions of self-adaptive software systems. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems, pp. 27–47. Springer Berlin/Heidelberg (2009). https://doi.org/10.1007/978-3-642-02161-9_2
5. Angelopoulos, K., Papadopoulos, A.V., Mylopoulos, J.: Adaptive predictive control for software systems. In: Proceedings of the 1st International Workshop on Control Theory for Software Engineering, CTSE 2015, Bergamo, pp. 17–21. ACM (2015)
6. Angelopoulos, K., Souza, V.E.S., Mylopoulos, J.: Capturing variability in adaptation spaces: a three-peaks approach. In: Johannesson, P., Lee, M.L., Liddle, S.W., Opdahl, A.L., Pastor López, Ó. (eds.) Proceedings of Conceptual Modeling: 34th International Conference, ER 2015, Stockholm, 19–22 Oct 2015, pp. 384–398. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-25264-3_28
7. Angelopoulos, K., Papadopoulos, A.V., Silva Souza, V.E., Mylopoulos, J.: Model predictive control for software systems with cobra. In: Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS '16, Austin, pp. 35–46. ACM, New York (2016). https://doi.org/10.1145/2897053.2897054
8. Åström, K.J., Murray, R.M.: Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press, Princeton (2008)
9. Babu, S.: Towards automatic optimization of mapreduce programs. In: SoCC, pp. 137–142. ACM, New York (2010)
10. Brun, Y., et al.: Engineering self-adaptive systems through feedback loops. Lect. Notes Comput. Sci. **5525**, 48–70 (2009)
11. Brun, Y., Desmarais, R., Geihs, K., Litoiu, M., Lopes, A., Shaw, M., Smit, M.: A design space for self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, 24–29 Oct 2010 Revised Selected and Invited Papers, pp. 33–50. Springer, Berlin/Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_2
12. Cai, K.Y., Cangussu, J., DeCarlo, R.A., Mathur, A.: An overview of software cybernetics. In: Eleventh Annual International Workshop on Software Technology and Engineering Practice, Amsterdam, pp. 77–86 (2003)
13. Campi, M.C., Savaresi, S.M.: Direct nonlinear control design: the virtual reference feedback tuning (VRFT) approach. IEEE Trans. Autom. Control **51**(1), 14–27 (2006)
14. Cangussu, J.A.W., Cooper, K., Li, C.: A control theory based framework for dynamic adaptable systems. In: Proceedings of the 2004 ACM Symposium on Applied Computing SAC '04, Nicosia, pp. 1546–1553. ACM, New York (2004). https://doi.org/10.1145/967900.968209
15. Cheng, B.H., et al.: Software engineering for self-adaptive systems: a research roadmap. In: Cheng, B.H.C. (eds.) Software Engineering for Self-Adaptive Systems, LNCS, vol. 5525. Springer, Berlin/New York (2009)
16. Dantzig, G.B., Thapa, M.N.: Linear Programming 1: Introduction. Springer, New York (1997)
17. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. Lecture Notes in Computer Science, vol. 7475. Springer, Berlin/Heidelberg (2013)

18. de Lemos, R., Garlan, D., Giese, H.: Software Engineering for Self-Adaptive Systems: Assurances, Dagstuhl Seminar 13511 (2013)
19. Desmeurs, D., Klein, C., Papadopoulos, A., Tordsson, J.: Event-driven application brownout: reconciling high utilization and low tail response times. In: 2015 International Conference on Cloud and Autonomic Computing (ICCAC), Cambridge, pp. 1–12 (2015)
20. Diao, Y., Gandhi, N., Hellerstein, J., Parekh, S., Tilbury, D.: Using MIMO feedback control to enforce policies for interrelated metrics with application to the apache web server. In: NOMS 2002. 2002 IEEE/IFIP Network Operations and Management Symposium, Florence, pp. 219–234 (2002)
21. Durango, J., Dellkrantz, M., Maggio, M., Klein, C., Papadopoulos, A., Hernandez-Rodriguez, F., Elmroth, E., Arzen, K.E.: Control-theoretical load-balancing for cloud applications with brownout. In: 2014 IEEE 53rd Annual Conference on Decision and Control (CDC), Los Angeles, pp. 5320–5327 (2014)
22. Filieri, A., Hoffmann, H., Maggio, M.: Automated design of self-adaptive software with control-theoretical formal guarantees. In: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, Hyderabad, pp. 299–310. ACM (2014)
23. Filieri, A., Hoffmann, H., Maggio, M.: Automated multi-objective control for self-adaptive software design. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, pp. 13–24. ACM, New York (2015). https://doi.org/10.1145/2786805.2786833
24. Filieri, A., Maggio, M., Angelopoulos, K., D'Ippolito, N., Gerostathopoulos, I., Hempel, A., Hoffmann, H., Jamshidi, P., Kalyvianaki, E., Klein, C., Krikava, F., Misailovic, S., Papadopoulos, Alessandro, V., Ray, S., Shariﬂoo, Amir, M., Shevtsov, S., Ujma, M., Vogel, T.: Software engineering meets control theory. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Florence (2015). https://hal.inria.fr/hal-01119461
25. Fliess, M., Join, C.: Model-free control. Int. J. Control. **86**(12), 2228–2252 (2013)
26. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. Wiley, New York (2004)
27. Herodotou, H., Babu, S.: Profiling, what-if analysis, and cost-based optimization of mapreduce programs. PVLDB **4**(11), 1111–1122 (2011)
28. Hoffmann, H., Eastep, J., Santambrogio, M.D., Miller, J.E., Agarwal, A.: Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In: Proceedings of the 7th International Conference on Autonomic Computing, ICAC '10, Reston, pp. 79–88. ACM, New York (2010). https://doi.org/10.1145/1809049.1809065
29. Hou, Z., Jin, S.: Data-driven model-free adaptive control for a class of MIMO nonlinear discrete-time systems. IEEE Trans. Neural Netw. **22**(12), 2173–2188 (2011)
30. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
31. Kihl, M., Robertsson, A., Wittenmark, B.: Performance modelling and control of server systems using non-linear control theory. In: J. Charzinski, R.L., Tran-Gia, P. (eds.) Providing Quality of Service in Heterogeneous Environments Proceedings of the 18th International Teletraffic Congress – ITC-18, Teletraffic Science and Engineering, vol. 5, pp. 1151–1160. Elsevier (2003). http://www.sciencedirect.com/science/article/pii/S1388343703802640
32. Klein, C., Papadopoulos, A., Dellkrantz, M., Durango, J., Maggio, M., Arzen, K.E., Hernandez-Rodriguez, F., Elmroth, E.: Improving cloud service resilience using brownout-aware load-balancing. In: 2014 IEEE 33rd International Symposium on Reliable Distributed Systems (SRDS), Nara, pp. 31–40 (2014)
33. Klein, C., Maggio, M., Årzén, K.E., Hernández-Rodriguez, F.: Brownout: building more robust cloud applications. In: Proceedings of the 36th International Conference on Software Engineering, Hyderabad, pp. 700–711. ICSE 2014. ACM (2014)
34. Ljung, L.: System Identification: Theory for the User. Prentice-Hall, Inc., Upper Saddle River (1986)

35. Maggio, M., Klein, C., Årzén, K.E.: Control strategies for predictable brownouts in cloud computing. In: IFAC Proceedings Volumes, vol. 47, pp. 689–694 (2014)
36. Maggio, M., Papadopoulos, A.V., Filieri, A., Hoffmann, H.: Automated control of multiple software goals using multiple actuators. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2017, Paderborn, pp. 373–384. ACM, New York (2017). https://doi.org/10.1145/3106237.3106247
37. Mars, J., Tang, L., Hundt, R., Skadron, K., Soffa, M.L.: Bubble-up: increasing utilization in modern warehouse scale computers via sensible co-locations. In: Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-44, Porto Alegre, pp. 248–259. ACM, New York (2011). https://doi.org/10.1145/2155620.2155650
38. Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B.: Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2015, Bergamo, pp. 1–12. ACM, New York (2015). https://doi.org/10.1145/2786805.2786853
39. Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B.: Efficient decision-making under uncertainty for proactive self-adaptation. In: 2016 IEEE International Conference on Autonomic Computing (ICAC), Würzburg, pp. 147–156. IEEE (2016)
40. Moreno, G.A., Papadopoulos, A.V., Angelopoulos, K., Cámara, J., Schmerl, B.: Comparing model-based predictive approaches to self-adaptation: CobRA and PLA. In: Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS '17, Buenos Aires, pp. 42–53. IEEE Press, Piscataway (2017). https://doi.org/10.1109/SEAMS.2017.2
41. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime software adaptation: framework, approaches, and styles. In: Companion of the 30th International Conference on Software Engineering. ICSE Companion '08, Leipzig, pp. 899–910. ACM, New York (2008). https://doi.org/10.1145/1370175.1370181
42. Rizvandi, N., Taheri, J., Zomaya, A.: On using pattern matching algorithms in mapreduce applications. In: ISPA, pp. 75–80 (2011)
43. Sayyad, A.S., Menzies, T., Ammar, H.: On the value of user preferences in search-based software engineering: a case study in software product lines. In: Proceedings of the 2013 International Conference on Software Engineering. ICSE '13, San Francisco, pp. 492–501. IEEE Press, Piscataway (2013). http://dl.acm.org/citation.cfm?id=2486788.2486853
44. Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., Wilkes, J.: Omega: flexible, scalable schedulers for large compute clusters. In: Proceedings of the 8th ACM European Conference on Computer Systems. EuroSys '13, Prague, pp. 351–364. ACM, New York (2013). https://doi.org/10.1145/2465351.2465386
45. Shevtsov, S., Weyns, D.: Keep it simplex: satisfying multiple goals with guarantees in control-based self-adaptive systems. In: 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering. FSE 2016, Seattle (2016)
46. Shevtsov, S., Iftikhar, M.U., Weyns, D.: SimCA vs ActivFORMS: comparing control- and architecture-based adaptation on the TAS exemplar. In: Proceedings of the 1st International Workshop on Control Theory for Software Engineering, CTSE 2015, Bergamo, pp. 1–8. ACM, New York (2015). https://doi.org/10.1145/2804337.2804338
47. Shevtsov, S., Berekmeri, M., Weyns, D., Maggio, M.: Control-theoretical software adaptation: a systematic literature review. IEEE Trans. Softw. Eng. **44**(8), 784–810 (2018)
48. Shevtsov, S., Weyns, D., Maggio, M.: Handling new and changing requirements with guarantees in self-adaptive systems using SimCA. In: Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS '17, Buenos Aires, pp. 12–23. IEEE Press, Piscataway (2017). https://doi.org/10.1109/SEAMS.2017.3
49. Silva Souza, V.E., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness requirements for adaptive systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS '11, Waikiki, pp. 60–69. ACM, New York (2011). https://doi.org/10.1145/1988008.1988018

50. Souza, V.E.S., Lapouchnian, A., Mylopoulos, J.: (Requirement) evolution requirements for adaptive systems. In: Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS ’12, Zurich, pp. 155–164. IEEE Press, Piscataway (2012). http://dl.acm.org/citation.cfm?id=2666795.2666820
51. Weyns, D.: Software engineering of self-adaptive systems: an organised tour and future challenges. In: Cha, S., Taylor, R.N., Kang, K.C. (eds.) Handbook of Software Engineering. Springer, Cham (2018)
52. Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., Göschka, K.M.: On patterns for decentralized control in self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, 24–29 Oct 2010 Revised Selected and Invited Papers, pp. 76–107. Springer, Berlin/Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_4
53. Weyns, D., Bencomo, N., Calinescu, R., Camara, J., Ghezzi, C., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J.M., Malek, S., Mirandola, R., Mori, M., Tamburrelli, G.: Perpetual assurances for self-adaptive systems. In: de Lemos, R., Garlan, D., Ghezzi, C., Giese, H. (eds.) Software Engineering for Self-Adaptive Systems III: Assurances, Lecture Notes in Computer Science, vol. 9640. Springer, Cham (2017)
54. Wittenmark, B., Åström, K., Årzén, K.E.: Computer control: an overview. Technical report (2002)
55. Zhu, X., Wang, Z., Singhal, S.: Utility-driven workload management using nested control design. In: American Control Conference, Minneapolis, p. 6 (2006)
56. Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., Shin, K.: What does control theory bring to systems research? SIGOPS Oper. Syst. Rev. **43**(1), 62–69 (2009). https://doi.org/10.1145/1496909.1496922