Automated Multi-objective Control for Self-Adaptive Software Design

Antonio Filieri
University of Stuttgart
Stuttgart, Germany
filieri@
informatik.uni-stuttgart.de

Henry Hoffmann
University of Chicago
Chicago, USA
hankhoffmann@
cs.uchicago.edu

Martina Maggio
Lund University
Lund, Sweden
martina@
control.lth.se

ABSTRACT

While software is becoming more complex everyday, the requirements on its behavior are not getting any easier to satisfy. An application should offer a certain quality of service, adapt to the current environmental conditions and withstand runtime variations that were simply unpredictable during the design phase. To tackle this complexity, control theory has been proposed as a technique for managing software's dynamic behavior, obviating the need for human intervention. Control-theoretical solutions, however, are either tailored for the specific application or do not handle the complexity of multiple interacting components and multiple goals. In this paper, we develop an automated control synthesis methodology that takes, as input, the configurable software components (or knobs) and the goals to be achieved. Our approach automatically constructs a control system that manages the specified knobs and guarantees the goals are met. These claims are backed up by experimental studies on three different software applications, where we show how the proposed automated approach handles the complexity of multiple knobs and objectives.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Design—*Methodologies*; I.6.5 [**Computing Methodologies**]: Model Development—*Modeling methodologies*

General Terms

Design, Experimentation, Theory, Performance, Reliability

Keywords

Adaptive software, control theory, dynamic systems, non-functional requirements, run-time verification.

1. INTRODUCTION

Self-adaptation is a first-class property of modern software systems. Adaptive software incorporates monitoring, decision-making, and actuation to maintain reliable behavior despite sudden, unpredictable changes like application workload fluctuations and hardware failures. Adaptive software detects such changes in the operating environment, determines how to respond, and implements the response with an actuation phase. Many methodologies have arisen for designing and implementing such *adaptive* or *autonomic* software systems [1–3].

Dynamic, feedback driven adaptation has been studied in *control theory* for decades, where rigorous engineering techniques adapt physical systems while providing formal guarantees of their behavior; *e.g.,* cruise control in an automobile [4]. Recent research has applied control theory to adaptive software systems, ensuring software behaves predictably in dynamic environments (for examples see the survey [5]). Control theoretic solutions are particularly attractive for software systems with strict requirements in unpredictable environments because control techniques permit formal analysis of a system's dynamic behavior.

The drawback of using control theory in software is that software engineers must master both their application domains and control systems. Indeed, using control theory requires (1) understanding all the *actuators*, or software components that can be changed during runtime, (2) developing a suitable mathematical model relating these components to observable feedback, and (3) implementing a control strategy based on the model and the desired behavior [6]. While the first point is typically easy for the software developer, the other two items require deep background in control science and might be not applicable for all the software adaptation problems [6].

Recent work proposed addressing points (2) and (3) by fully automating the construction of a mathematical model and the synthesis of a suitable controller for software systems, putting control techniques in the hands of non-experts [7]. That approach addresses systems having a single quantitative goal and a single actuator. It produces a controller using two phases: *learning* and *synthesis*. The former conducts a systematic exploration of the actuator and builds a mathematical model relating this actuator to measurable changes in the software. The synthesis phase uses the model to construct a controller that is formally guaranteed to meet the goal, if feasible. Additional enhancements make this approach robust in the face of unpredictable external changes; *e.g.,* , those in third-party components or services. Despite the variety of problems that can be tackled with this approach [7], its guarantees are limited to only a single quantitative goal. Software systems requiring guarantees in multiple dimensions (*e.g.,* energy and performance) cannot make use of this approach.

Automatic synthesis of controllers for multiple goals is usually challenging and requires human intervention. A general approach handling multiple goals and multiple actuators has been proposed [8]. To deal with conflicting goals, this approach uses a cascade control schema where higher priority goals are pursued before lower priority ones. This approach assumes perfect knowledge of

the system at design time and requires continuous knobs to be discretized, possibly leading to an overwhelming number of control actions. To find the best configuration within a large search space, an optimization problem is formulated, but the approach relies on heuristic solutions to avoid scalability problems [8]. Recent work, however, has shown that the effectiveness of such heuristics is system dependent; *i.e.,* they are not general and not portable across systems [9].

We address this need to generalize support for multiple goals by extending our previous automated framework to synthesize controllers to manage requirements for multiple non-functional quantitative properties simultaneously by synthesizing cascade controller systems given a prioritization of the control goals. We address shortcomings in prior work by handling both discrete and continuous actuators and by finding exact solutions to optimization problems that translate control signals into knob settings.

The methodology we propose assumes the software system is a equipped with multiple actuators and is required to satisfy multiple quantitative goals. Each actuator is associated with a control *knob* (or control variable) that may affect multiple quantitative dimensions at the same time. After an initial (possibly rough) discretization of continuous knobs, the controller dynamically refines the discretization between relevant values to increase control accuracy at runtime. No constraints are imposed on the actual (unknown) relationship between knob and goal, which can be linear or nonlinear.

Multiple actuators affecting multiple dimensions create an exponential explosion of possible software configurations, which might limit the scalability of prior work. The key insight of our approach is to overcome this explosion by exploiting the *ranking* of the non-functional properties (or goals) under control to design an efficient control allocation [4] where multiple controllers operate in cascade on subsequently smaller decision spaces. To this end, we partition the knobs depending on the goals they affect and decide on each partition separately, whenever possible. This can reduce the number of knobs configurations involved in each separate decision.

Furthermore, we replace the heuristic solution of the internal optimization problems with an exact solution based on the analysis of the dual problem. This analysis allows to efficiently reduce the number of configurations that has to be considered for finding an optimal solution by several orders of magnitude. For example, in a radar processing case study (see Sec. 5.2) this analysis reduces the possible number of actuator combinations to consider from over 2 million to just under 2 thousand – three orders of magnitude reduction.

In the proposed approach, the controllers are dynamically synthesized based on the control actions selected for higher-ranked goals. The only requirement is that at least one problem dimension must be *free*; *i.e.,* it will be minimized (or maximized) without any guarantees on its value.

Our methodology supports two ranking schemes. The first is *user-defined* — users explicitly define the priority of different goals. This schema resembles the elicitation of primary (mandatory) requirements and secondary (desirable) requirements. If some goals cannot be achieved, the controllers first meet the highest ranked one, and then the subsequents, coming as close as possible on infeasible goals. The second scheme is an *automatic* ranking where the control system orders the goals to maximize the number which can be achieved. As in prior work [7], continuous learning mechanisms are applied to keep the system model updated at runtime and maintain guarantees despite possible changes.

The methodology in this paper greatly extends the generality of prior approaches to automated control design, moving existing effort much closer to the requirements necessary for deployment in a real system with multiple constraints, all of which must be met despite unpredictable and uncertain execution environments. At the same time, the proposed approach requires no prior knowledge of the software under control nor special mathematical skills, making control theoretic solutions available to non-experts who must guarantee multiple aspects of a programs behavior.

We implement the proposed methodology and obtain results in three case studies. In the first, we manage performance, security and energy for encrypted communications on a mobile device. In the second, we manage hardware resources and software configuration to achieve accurate and timely target localization in a cyber-physical radar system. In the third we design a dynamic binding mechanism for a service-oriented system to guarantee reliability, response time, and cost effectiveness. These examples demonstrate the broad applicability of our proposed automated methodology.

The rest of the paper is organized as follows. Section 2 discusses prior work. Section 3 presents some control background. The proposed methodology is discussed in Section 4 and evaluated in Section 5. Section 6 concludes the paper.

2. RELATED WORK

Many modern software systems are *self-adaptive* [10, 11]; *i.e.,* they select at runtime the best configuration to achieve specific non-functional requirements like reliability or response time. There are many examples, from compiler-based support for alternative implementations [12–14] to the exploitation of dynamic knobs for power and energy management [15–17]. Hardware architectures can be dynamically adjusted to target execution speed and much more [18–20]. In High Performance Computing it is common to adapt a running application; *e.g.,* tuning FFTs for graphics processing units [21]. Considerable effort has been devoted to MapReduce, which exposes many configurable parameters [22–24].

Self-adaptive techniques are also prominent in industry. For example, companies like IBM [25] developed the IBM Touchpoint Simulator and the K42 Operating System [26]. Oracle produced the Automatic Workload Repository [27] and Intel the RAS Technologies for Enterprise [28].

Formal methods are often used to build self-adaptive systems because of their ability of providing mathematical guarantees on both the effectiveness and the dependability of the adaptation mechanism [29]. Among those methods, control theory [6, 30–32] has been recognized by the software engineering community as a solution to meet quality of service requirements despite unpredictable changes in the execution environment. Several recent surveys capture the current state-of-the-art applying control-theory to software applications [5, 33], as well as highlighting the main criticisms of early approaches [3]. Examples control delays for web servers [34], manage data centers [35], allocate resources [36–38], tune operating systems [39–41], minimize energy [42], and coordinate across the system stack [43]. These strategies adapt tunable knobs that can be identified either offline or at runtime [44].

The use of control theory in software engineering, however, is still in a preliminary stage. It is difficult to develop accurate control models for software because strong mathematical skills are needed to deal with the complex non-linear dynamics of real systems [45, 46]. These difficulties result in control strategies that solve a particular problem and operate in particular conditions, but do not generalize. This paper's goal is to increase the generality of control solutions by introducing a methodology that automatically constructs control systems for software adaptation. The proposed methodology requires little prior knowledge, instead adopting a "push-button" approach that maintains the formal guarantees

offered by traditional control approaches, but requires little mathematical background.

The first general and automated solution was recently proposed [7]. The solution was based on very simple qualitative equation-based models. While complex and precise quality models have been used in the past to enable design-time optimizations [47], such complexity is a drawback for runtime adaptation, due to its overhead [48]. Despite its generality with respect to problems and environments, the solution proposed in [7] suffers from restrictions and limitations. This methodology is, in fact, only applicable to single-input single-output systems, where the application developer must identify one single actuator to be changed and the methodology guarantees only a single non-functional requirement; *e.g.,* response time. In this paper, we further generalize this previous work to obtain an automated solution capable of dealing with multiple objectives and actuators simultaneously.

3. BACKGROUND

This section connects software engineering and control theoretic terminology to provide the necessary background for following the methodology presented in Sec. 4.

Applying control theory requires (1) the ability to measure the quantitative property under control and (2) some desired values for these properties. These quantitative properties are referred to as the *goals* of the system and are related to non-functional requirements like energy consumption, response time, and reliability. For example, the measured response time of a web server can be of 500 ms, while its desired value is below 1 s. In this paper, we assume it is possible to measure every objective that the control system should fix and that the system developer assigned some desired values for the quantities under control. The desired value, in control terms, is called the *setpoint*. A software systems will have multiple non-functional requirements, which we refer to as the dimensions of software behavior, or simply *dimensions* for brevity. The current measurement of the system status in a particular dimension is a *feedback signal*.

Control also requires adjustable system components, called control *knobs* or *actuators*. These actuators should affect the measured quantitative properties and allow the system to reach the desired setpoints. The software can have multiple knobs and multiple setpoints. However, the methodology proposed here requires the number of knobs to be greater than or equal to the number of dimensions under control.

The controller chooses the knobs' settings to drive the feedback signal to the goals, a process called *setpoint tracking*. If the controller has to counteract external factors that could interfere with behavior, it is doing *disturbance rejection*. If it has to act also in presence of unreliable measurement, *e.g.,* noisy feedback, this calls for *robustness* to inaccuracies.

These requirements for the control system can themselves be mapped into specific quantifiable properties. Specifically, *stability* refers to convergence to an equilibrium — with a well designed controller, the setpoint. Also, one can decide how the system reaches the equilibrium point. *Overshooting* means that the measured feedback may be higher than the goal for some time. Avoiding overhsoots avoids penalties; *e.g.,* violating user requirements or service level agreements. The time required to reach the equilibrium point is the *settling time*. The system can also be *robust*, in control theoretical terms, and converge to the setpoint despite quantifiable errors in the control model.

It is obviously desirable to design a stable system that avoids overshooting, has low settling time, and high robustness. The main advantage of a control-theoretical approach is that these properties are formally guaranteed on the system's model. Thus, it is possible to determine when the system behaves as expected and, when it does not, how it will converge and what the values of the measured quantities will be.

4. METHODOLOGY

This section presents our methodology for controlling multiple dimensions of software behavior using multiple actuators. We assume no prior knowledge about the effect of these actuators on the system. We do assume that each knob has a *nominal* value, which is the knob's value when it is not used to control any dimension. When all the knobs are set to their nominal value, the i-th dimension d_i takes value b_i.

The term *configuration* denotes a specific set of values for the available knobs. For example, given the set of knobs $\mathcal{K} = \{k_1, k_2, k_3\}$ a possible configuration is $\{k_1 = 3, k_3 = 0\}$, where k_2 is not assigned and can be set to any of its feasible values.

4.1 Control Strategy

We propose a generalized feedback control strategy to guarantee the behavior of the software system in multiple dimensions. The new approach builds on prior work that controlled performance, power, and application accuracy [8] by (1) extending the control strategy to any set of non-functional quantitative goals, (2) automating the controller's design, (3) solving internal optimization problems exactly, and (4) incorporating continuous control knobs.

Such multi-dimensional control is a difficult problem because decisions made to adjust behavior in one dimension will likely affect others. To overcome these complications we impose an ordering on dimensions of control. Given an ordering, our approach applies control based on *rank*, where higher-ranking dimensions are controlled before lower-ranking ones. For example, if performance is ranked highest, and energy is lower-ranked, the controller will first tune knobs to satisfy the performance goal, then determine a set of knobs that affect energy but not performance, and finally tune those knobs to meet the energy goal.

We support two different ordering schemes. The first allows user-specified *priorities*. A user may be concerned more about software's performance than its accuracy. Performance will be higher ranked and the control strategy will adjust for performance first, using any knob. Knobs which do not affect performance are used to control accuracy. The second ordering scheme supports *feasibility*. In this scheme, the controller automatically ranks dimensions based on available actuators to achieve as many goals as possible. For example, if three of the ten available actuators are necessary to constrain the performance behavior and five of them would be needed to address accuracy goals, the controller chooses performance as a primary dimension, to leave as many actuators as possible free to control accuracy.

The ordering scheme, whether priority or feasibility based, is the key insight to our approach. At runtime, the control strategy selects the actuators in the order of dimension ranking. After applying the control values for one dimension, we estimate the effects on the subsequent dimensions, remove the already constrained knobs from the pool of available ones and execute the controller to optimize for the new dimension. Adapting the control strategy during runtime, based on decisions made for higher-ranking dimensions, allows us to account for the dependence between dimensions. The estimation strategy bases the decision on actual data about how the system is reacting to changes in the previously controlled dimensions, adjusting prior knowledge about the mentioned dependence.

Figure 1 shows an example of the proposed feedback control strategy managing two dimensions: accuracy and performance. In

this example, the highest ranking dimension is accuracy; performance is lower ranking. At runtime, our approach first collects the current goals $g_a(t)$ and $g_p(t)$ and feedback measurements $f_a(t)$ and $f_p(t)$, in the accuracy (a) and performance (p) dimensions at time t.

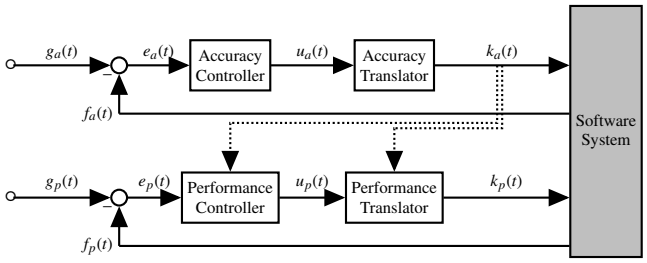

Figure 1: Block diagram for controlling accuracy and performance.

The controller then computes the *errors* in both dimensions, the differences between the goals and current values $e_a(t)$ and $e_p(t)$. The control system produces $u_a(t)$, a signal indicating how to change accuracy to compensate for the measured error. The control signal $u_a(t)$ is passed to a translator, which determines the set of possible knobs values that achieves the desired accuracy and among those, the ones that allow for optimal performance. The predicted performance of this configuration is used to modify a second control system that produces $u_p(t)$, a signal that drives the performance error $e_p(t)$ to zero. This signal is then translated to a disjoint set of knobs that achieves $u_p(t)$ with minimal interference on any other dimension. At the next time step, feedback and goals are measured and new control signals are calculated.

4.2 Controller Synthesis

This section presents a formal description of the proposed controller. For clarity, we first present an approach for goals in two dimensions, and then extend it to handle more. This approach is based on classical control synthesis techniques [4] but extends them to handle a general set of problems with a general set of knobs. In classical control synthesis, the knobs and the goals are known in advance and the synthesis is carried out to find a suitable controller for the specific case and the specific system model to be used. Our approach, instead, focuses on achieving generality and tackles the problem of having a set of knobs and goals to match.

The approach has five phases. The first selects the lead dimension. For user-specified priorities, this phase is skipped and user priorities become input for the second phase. Subsequent phases are the control of the lead dimension, its translation into knob settings, the control of the subordinate dimension, and its translation. If more dimensions are added, the last two steps are repeated for each additional dimension. We first focus on discrete (or discretized) knobs and discuss the dynamic refinement continuous knob discretization later in this section.

4.2.1 Selecting the Lead Dimension

If the controller does not receive user-specified priorities, the first step is to select the lead dimension. Given a set of N dimensions $\mathcal{D} = \{d_1, d_2, \ldots d_N\}$, the lead dimension is the one that can be controlled using the fewest actuators. A set of m knobs $\mathcal{K} = \{k_1, k_2 \ldots, k_m\}$ does not affect dimension d_i if

$$\forall \vec{c} \in C, \quad d_i(\vec{c}) \simeq b_i, \tag{1}$$

where $C = \{\mathcal{K}_1 \times \mathcal{K}_2 \times \cdots \times \mathcal{K}_m\}$; $\mathcal{K}_a$ with $a \in \{1 \ldots m\}$ the domain of the a-th knob, or the set of values that the a-th knob can assume; $d_i(\vec{c})$ is the value of dimension d_i due to the effect of the configuration $\vec{c}$; b_i is a baseline measurement of dimension d_i. In other words, for any possible combination of actuator values (the Cartesian product of the set of admissible values for each knob), the behavior is unchanged.

With two dimensions, we would like to find d_{lead}, the primary dimension, and d_{sub}, the secondary one. We start by finding the set $\mathcal{K}_1$ of knobs affecting the first dimension, d_1 and the set $\mathcal{K}_2$ affecting the second one, d_2. From these two sets we select the lead dimension as the one which is influenced by a smaller number of knobs.

$$\mathcal{K}_1 = \{k \mid d_1(\vec{c}) \neq b_1\} \tag{2}$$

$$\mathcal{K}_2 = \{k \mid d_2(\vec{c}) \neq b_2\} \tag{3}$$

$$d_{lead} = \text{argmin}_{d_i} \mid \mathcal{K}_i \mid \tag{4}$$

$$d_{sub} = \text{argmax}_{d_i} \mid \mathcal{K}_i \mid \tag{5}$$

Eqns. 2 and 3 determine the set of knobs affecting the goal dimensions, while Eqn. 4 chooses the lead dimension as the set with minimal cardinality. The remaining one is the secondary dimension. To control the lead dimension, fewer actuators are used. This means that more knobs are available for subsequent dimensions.

4.2.2 Controlling the Lead Dimension

Let $g_{lead}(t)$ and $f_{lead}(t)$ denote the goal and feedback in the lead dimension at time t. Control for the lead dimension eliminates the error $e_{lead}(t) = g_{lead}(t) - f_{lead}(t)$, by computing a control signal $u_{lead}(t)$ based on its prediction of the next feedback measurement $f_{lead}(t+1)$:

$$f_{lead}(t+1) = b_{lead} \cdot u_{lead}(t) + dist_{lead}(t), \tag{6}$$

where b_{lead} is the baseline behavior in the lead dimension, *i.e.,* its behavior with all the knobs configurations set to their nominal value. $dist_{lead}(t)$ represents a transient disturbance. For example, if the lead dimension is performance, b_{lead} is the performance with the software in its default configuration. $f_{lead}(t)$ represents the measured performance at time t. $u_{lead}(t)$ represents the speedup (over baseline) the translator should achieve at time t, and $dist_{lead}(t)$ represents a momentary disruption in performance (*e.g.,* due to a page fault).

Given $e_{lead}(t)$, our methodology finds $u_{lead}(t)$ using a *deadbeat* controller, based on the model defined by Eqn. 6. The approach followed is the same as proposed in [7], using zero as value for the pole, *i.e.,* imposing that the controller is as fast as possible in making the measured value converge to its specified goal. The control signal $u_{lead}(t)$ then becomes

$$u_{lead}(t) = u_{lead}(t-1) + \frac{e_{lead}(t)}{b_{lead}}. \tag{7}$$

The disturbance $dist_{lead}(t)$ disappears from Eqn. 7. This is not surprising, since $dist_{lead}(t)$ models any fast, transient change that is not under control and affects f_{lead}. The fast transient disturbances cannot be controlled in any way, unless a precise model of their action exists. When the disturbance does not immediately disappear, its action is slowly included into the model via a change in the baseline b_{lead} that is kept updated at runtime. Thus, a temporary page fault receives no correction, but a sudden lack of memory that persists for some time is compensated for, due the feedback signal itself. For now, we assume to know how b_i varies, for each dimension, during the execution of the application. This assumption will be relaxed in Sec. 4.3.

4.2.3 Translating the Lead Dimension

Once the control signal $u_{lead}(t)$ is computed, it must also be actuated. To this end, we must select specific settings for the knobs in the set $\mathcal{K}_{lead}$ (Eqns. 2–5), to obtain the desired control signal.

To convert the continuous control signal into a configuration for the available knobs, our approach schedules configurations for a window of τ time units, with τ sufficient for the next feedback signal to reflect the effects of the schedule. The schedule is a list ℓ_{lead} of couples $(\vec{c_w}, \tau_w)$, where c_w represents a specific configuration and τ_w denotes a time to spend in that configuration. The configurations in the list are sequentially applied until the end of the window τ. Configurations are chosen so that the average behavior over the time window is equal to the control signal and the effect of the schedule on the subordinate dimension is minimized. The resulting optimization problem is:

$$\underset{\ell_{lead}}{\text{optimize}} \quad \sum_{(\vec{c_w},\tau_w)\in\ell_{lead}} d_{sub}(\vec{c_w})\cdot\tau_w \tag{8}$$

subject to

$$\sum_{(\vec{c_w},\tau_w)\in\ell_{lead}} \frac{\tau_w}{\tau}\cdot\frac{d_{lead}(\vec{c_w})}{b_{lead}} = u_{lead}(t) \tag{9}$$

$$\sum_{(\vec{c_w},\tau_w)\in\ell_{lead}} \tau_w = \tau \tag{10}$$

$$\forall(\vec{c_w},\tau_w)\in\ell_{lead},\ \nexists k_i\in\vec{c_w},\ k_i\notin\mathcal{K}_{lead} \tag{11}$$

Here, the word optimize stands for either minimize or maximize, depending on the specific dimension. For example, optimizing power consumption minimizes it, while optimizing reliability maximizes it. This formulation assures the subordinate dimension is optimized (Eqn. 8), the control signal for the lead dimension is realized (Eqn. 9), and the total time does not exceed the time before the next feedback measurement (Eqn. 10). It also guarantees that every configuration included in the schedule uses only knobs belonging to $\mathcal{K}_{lead}$ (Eqn. 11).

While mathematical optimization problems are, in general, expensive to solve, the particular structure of this problem lends itself to a cost-effective solution. This problem has two non-trivial constraints (Eqns. 9 and 10), and the other constraints confine solutions to have non-negative components. The geometric structure of this problem implies that the optimal solution will have at most two configurations in ℓ_{lead}, *i.e.,* only two of the coefficients τ_w in Eqn. 8 will be nonzero [49]. A simple solution that returns the true optimal, then, would divide the set of C_{lead} of all possible configurations of the knobs in $\mathcal{K}_{lead}$ (the Cartesian product of all sets of potential values for the selected knobs) into two subsets:

$$C_o = \{c\in C_{lead} \mid d_{lead}(\vec{c}) \geq u_{lead}(t)\} \tag{12}$$

$$C_u = \{c\in C_{lead} \mid d_{lead}(\vec{c}) \leq u_{lead}(t)\} \tag{13}$$

Eqn. 12 finds the set of all the configurations that exceed u_{lead}, while Eqn. 13 selects the configurations that approach the desired control signal from below. Given these two sets, we can search over all pairs where one entry in the pair is drawn from C_o and one entry is drawn from C_u.

This algorithm is guaranteed to return an optimal solution to the problem, but it executes in $O(\mid C^2 \mid)$ time. This exhaustive search can be straightforwardly parallelized (*e.g.,* in with a map-reduce strategy). For very large configuration spaces, it is possible to further reduce the complexity by creating a table of buckets for each dimension (essentially a hash table), where each bucket represents a range of behavior in that dimension. We employ this approach for large C, confining the search to the small number of configurations that map to the same bucket, requiring $O(1)$ operations when the table is large enough to avoid bucket collisions. Finally, an analysis of the dual problem of the (mixed integer) linear optimization in Eqns. 8–11 can lead to a dramatic reduction of the search space by pruning out the configurations dominated by others toward the identification of the optimal point [50]. A deeper discussion of the possible optimization strategies is beyond the scope of this paper. Notably, state of the art optimization tools have a variety of them built in and can be used off-the-shelf, provided the user knows how to structure the problem.

The lead dimension is now controlled with a set of knobs that is as small as possible, leaving the other knobs free for other dimensions.

4.2.4 Controlling the Subordinate Dimension

Having selected $\vec{c_o}$ and $\vec{c_u}$, the proposed approach calculates $\hat{b}_{sub}(t)$, an estimate of how the choice for the lead dimension will affect the subordinate one:

$$\hat{b}_{sub}(t) = b_{sub}\cdot\frac{d_{sub}(\vec{c_o})\cdot\tau_{\vec{c_o}} + d_{sub}(\vec{c_u})\cdot\tau_{\vec{c_u}}}{\tau} \tag{14}$$

We use Eqn. 14 and the integral-control-law to calculate a control signal that eliminates the error $e_{sub}(t) = g_{sub}(t) - f_{sub}(t)$ in the subordinate dimension.

$$u_{sub}(t+1) = u_{sub}(t) + \frac{e_{sub}(t)}{\hat{b}_{sub}(t)} \tag{15}$$

The value of $u_{sub}(t)$ is then passed to the translator to obtain the remaining knob configuration.

4.2.5 Translating the Subordinate Dimension

For stability, $u_{sub}(t)$ must be translated into knob settings without affecting the behavior of d_{lead}. From Eqns. 2–5, we know the two sets of knobs $\mathcal{K}_{lead}$ and $\mathcal{K}_{sub}$ affecting the lead and subordinate dimensions. The controller then computes the set $\mathcal{K}_{valid}$ affecting only the subordinate one. If $\mathcal{K}_{valid}$ is the empty set, the problem is not feasible and the system reports that it is not possible to achieve the subordinate dimension's goal without compromising that of the lead dimension.

$$\mathcal{K}_{valid} = (\mathcal{K}\setminus\mathcal{K}_{lead})\cap\mathcal{K}_{sub} \tag{16}$$

Let d_{free} be the dimension for which there is no goal. Then the signal $u_{sub}(t)$ is translated to knob configurations that optimize d_{free} by computing a schedule ℓ_{sub} of couples $(\vec{c_s}, \tau_s)$, solving the following optimization problem

$$\underset{\ell_{sub}}{\text{optimize}} \quad \sum_{(\vec{c_s},\tau_s)\in\ell_{sub}} d_{free}(\vec{c_s})\cdot\tau_s \tag{17}$$

subject to

$$\sum_{(\vec{c_s},\tau_s)\in\ell_{sub}} \frac{\tau_s}{\tau_{sub}}\cdot\frac{d_{sub}(\vec{c_s})}{b_{sub}} = u_{sub}(t) \tag{18}$$

$$\sum_{(\vec{c_s},\tau_s)\in\ell_{sub}} \tau_s = \tau_{sub} \tag{19}$$

$$\forall(\vec{c_s},\tau_s)\in\ell_{sub},\ \nexists k_i\in\vec{c_s},\ k_i\notin\mathcal{K}_{valid} \tag{20}$$

where τ_{sub} represents the sampling time of the subordinate loop, which can be different than the lead one. Eqn. 17 optimizes the free dimension, Eqn. 18 ensures the control signal is realized, Eqn. 19 ensures the time window is respected, and Eqn. 20 ensures that configurations come from Eqn. 16. This optimization problem is solved using the same optimal algorithm as presented previously for the lead dimension.

Extension to More Dimensions: The process above can be extended to an arbitrary set of dimensions. Instead of a lead and subordinate dimension, the methodology ranks dimensions. The highest rank dimension is equivalent to the lead dimension in the above. For each subsequent dimension, the process described in the last two steps is applied, substituting the set of all the used knobs $\mathcal{K}_{lead}$ in Eqn. 16. This set is computed as the union of $\mathcal{K}_{lead}$ with all the already prescribed $\mathcal{K}_{sub}$.

In the hash table implementation of the above, control for each dimension takes $O(1)$ time. Therefore, applying this process to multiple dimensions takes $O(N)$ time, where N is the number of dimensions under control.

Discretization refinement The controller schema introduced in this section identifies a sequence of discrete configurations $\vec{c_i}$ approximating the enforcement of a continuous reference u_i provided by a deadbeat abstract controller. Whenever a continuous knob k_c assumes different values between two consecutive configurations in the sequence, the transition between them is linearly smoothed over one or more steps for the continuous knob k_c, while the discrete knobs follow the original plan.

In practice, assuming at time i the plan requires a transition between the two configurations $\vec{c_{...i}}$ and $\vec{c_{i+1...}}$, a one step smoothing of k_c would replace this step change with a subsequence $\vec{c_{...i-1}}, \vec{c_i}, \vec{c_{i+1...}}$, where the value of k_c at time i is $k_c^i = (k_c^{i-1} + k_c^{i+1})/2$. This refinement explore a new configuration for knob k_c, smoothing the discretization level.

This linear smoothing can be also extended over more steps. However, since no assumptions have been made on the actual function that is being discretized, it is possible that the new values for k_c make the control plan deviate from its expected behavior, e.g., in presence of a nonlinear behavior around those values. For this reason, the length of the smoothed transition should be kept short when nonlinear behaviors are expected. Nonetheless, the possible deviations are only transitory for the current τ actuation steps, while providing additional information for the next control decision.

To avoid an unnecessary growth of the discretized configuration space for control knobs, it is a good practice to introduce a maximum resolution threshold, such that smoothing is only enforced when the difference between two subsequent values of a continuous knob k_c is larger than such threshold. This threshold also bounds the error incurred through discretization of continuous knobs.

4.3 Learning and Runtime Adaptation

We have assumed the values $d_i(\vec{c})$ for each dimension i and each valid configuration $\vec{c}$ are known. However, in many cases these values might be unknown or subject to runtime changes. In control theory, updating the model parameters is called *system identification*. We review some existing techniques [4], and propose guidelines to find the best identification method for a specific problem.

A naive solution applies only a statistical estimator to learn, and update, each value $d_i(\vec{c})$ depending on $\vec{c}$. Several such approaches have been proposed both in control theory and in software engineering [51–53]. Despite its simplicity, this approach is realistic only for small numbers of configurations because of the prohibitively large number of samples needed to guarantee convergence [54].

Surrogate models approximate software behavior as a function of knob configurations. Examples are radial basis function, spline models, or Gaussian processes, such as the popular Kriging models [55–57]. These models may require fewer samples for a suitable approximation the software behavior, but their increased computational complexity may reduce the reaction time of the controllers.

Whenever possible, a computationally efficient parametric model is preferred. Such models define families of possible behaviors; the objective of learning then reduces to finding the best parameter assignment to describe the behavior of the system with respect to each non-functional requirement dimension. Several parametric models are already used in software engineering, especially to reason about reliability and performance, [48, 51, 58].

Furthermore, parametric models are usually easy to keep updated with online tracking mechanisms such as Bayesian estimation [52], Kalman filtering [51], or other techniques for statistical learning [59]. This has a twofold benefit: (1) the model tracks changes in the system and (2) the quality of the estimates is continuously improved while the system is running, overcoming possible inaccuracies in the information collected during an initial learning phase. In the third case study on quality-driven dynamic binding, we fit and continuously update a parametric model based on incremental estimation and quasi-Montecarlo sampling.

4.4 Discussion and Formal Assessment

Our methodology uses a control theoretic runtime decision engine to adapt a running application in response to unpredictable events. This control theoretic approach allows formal analytical assessment. Such analysis is based on the assumption that software behavior is *bounded*; *i.e.,* an uncontrolled application cannot continually increase or decrease its performance, power consumption, or output accuracy. Bounded also implies that every situation can be recovered. This is not always the case with control strategy, where a signal could indefinitively grow, therefore leading to instability that cannot be addressed. Having bounded inputs and outputs simplifies the analysis and the formal assessment of the system. We analyze here the properties mentioned in Sec. 3: stability, overshooting, and settling time.

Stability: To study the convergence of the system, the time-based quantities can be converted to their frequency domain counterparts using the Z-transform [4], a frequency domain representation of a discrete time control signal. In particular, to assess the system stability, we consider the closed loop system Z-transform, and determine if the poles of said Z-transform lie in the unit circle [4]. The controlled system is composed of multiple cascade loops (one for each dimension under control). The first closed loop system, controlling the lead dimension, is stable by design. In fact, given the controller synthesis method (we generate a deadbeat controller – in control terms this means generating the controller that is less robust to noise but brings the desired signals as close as possible to their setpoints, as fast as possible [4]) its Z-transform is $1/z$, therefore there is only one pole, at zero. The analysis of the subsequent subordinate loop is more complicated, since some of the signals depend on previous loops, and are thus time-varying. In principle, these loops should be analyzed as a switching system, where some signals are rapidly changed from one value to another. However, it is possible to make the simplifying assumption that the loops corresponding to the dimensions controlled beforehand (*i.e.,* the lead and the subsidiaries with higher priority) have already stabilized to their goals. In this case, the analysis becomes straightforward, since again the deadbeat nature of the control strategy guarantees a closed loop Z-transform of $1/z$. [4] contains an exhaustive description of deadbeat controllers' stability properties.

To guarantee that the loops for higher-ranked dimensions are already stabilized, the time constant τ_{sub} that appears in Eqn. 18 should be long enough. More precisely, $\tau_{sub} \geq 2\tau$ where τ is the one used in Eqn. 9. For each additional subordinate dimension, the sampling time of the controller should be increased to twice the value of the previous loop. This guarantees that the values set by the previous loop have already settled to their regime values. The control signal of such systems is seen as a disturbance from the

subsequent ones in the chain. To guarantee the stability in face of disturbance one could do a robustness analysis and verify what is the maximum amount of change that the subordinate dimensions could tolerate. The Z transform function of the closed loop, augmented with the disturbance, has the same poles as the original one, so the stability property is preserved whenever the goals are feasible. The augmented settling time is therefore a sufficient condition. In principle, it could be relaxed with a switching system analysis. Such analysis, however, is system-dependent and unsuitable for an automated control strategy. It is common in practice to select the sampling period of each loop multiples of one another, so that the sufficient condition for system stability is fulfilled.

Overshoot: The deadbeat controller converges to the set point without overshoot if the system operates under perfect information. However, short overshoots are expected because of transient disturbances that cannot be canceled without additional knowledge on the system's behavior; *e.g.,* the effects of an outlier reported by the monitors will be reduced by the integral action of the control, though it may be too large to be fully compensated [4]. However, the fast controller reaction (the Z-transform has only one pole, at zero) guarantees overshoots are quickly acted upon and canceled by the control strategy [4]. Indeed, the experimental results in the following section show that the system can be found in overshoot conditions, but they are promptly lowered by the controller action.

Settling Time: The settling time of the closed loop system is by definition the settling time of the slowest loop in the system. Assuming the sampling strategy indicated above is employed to guarantee the system's stability ($\tau_{sub} = 2\tau$), the settling time of the overall system is given by $2^{n-1} \cdot \tau$, where n is the number of dimensions under control. Clearly, one can select τ to be as small as possible, for faster convergence. However, the choice of τ is limited by the fact that the control action taken at time t should have a measurable effect at time $t + \tau$. This last assumption is unavoidably application dependent. For example, changing the distribution policy for a load balancer can be enforced in fraction of seconds, while starting a virtual machine in the cloud may take a potentially unpredictable time. τ has to be greater than the maximum actuation time in the application, and may limit the applicability of the proposed control approach when faster reactions are desired. Including an online estimation procedure, also, may extend the time that the system needs to settle, since the estimator settling time should also be taken into account before correct information about the system become available. For every loop, one should consider the time that the corresponding estimation strategy takes to converge as part of the total convergence time.

Optimality of the translation: The optimality of the translation from the continuous reference value to a sequence of τ discrete configurations is subject to several assumptions. If the knobs are all discrete, the only way to guarantee the optimality of the translation is to know the effects of every possible configuration. The exhaustive exploration of such (finite) configuration space is often infeasible and replaced by a systematic or randomized exploration of a smaller subspace (as described in Section 4.3). This introduces an approximation of the optimal solution that should be taken into account when implementing a specific application. This observation extends naturally to discretized continuous functions, where the quality of the finite discretization may not capture all possible nonlinearities in the approximated continuous function. These lack of information may lead to sub-optimal plans, though the stability of the system is not compromised since only known configurations will be enforced, whose effect are known.

The problem of discrete knowledge can be overcame when an analytical model of the system is available. In such case the optimization problems can be straightforwardly restated taking into account the actual function relating knobs to goals, however this esulates from the scope of this paper.

The use of online learning techniques introduce additional uncertainty. Indeed, until the estimators converged, the decisions might be transitory biased. As side effect, a configuration might be not enforced on the base of wrong knowledge, preventing the gathering of additional information and in turn the slower convergence of the estimators. The initial learning phase can be leveraged to reduce this risk, while at runtime it is possible force the periodic exploration of configurations that have not been visited for a while. If a parametric model of the system is available, the goal of learning and online updating moves to the estimation of unknown model parameters (see Sections 4.3). In such case the risk of outdated or not converged estimates is compensated by the additional knowledge of the model structure.

Knobs: Finally, besides the possible feasibility limitations due to a bad prioritization schema discussed in Section 4.1, another limitation of the proposed strategy is that the number of controlled dimensions can never exceed the number of knobs available in the system. Notice that the controlled dimensions do not include the free one (or possibly more than one, if several dimensions can evaluated through a utility function) that is used for the optimization in Eqn. 17.

5. EXPERIMENTAL EVALUATION

We present three case studies that evaluate our methodology and its ability to automatically manage multiple quantitative non-functional properties for a software system. In our first study, we create an adaptive encryption system for mobile communication. In the second, we present a cyberphysical managing a radar infrastructure required to provide timely and accurate target localization. The final study deals with an adaptive web infrastructure, which balances load between different heterogeneous servers offering the same service. For each study, we describe the knobs used to control the system, the dimensions managed, and the results of deploying the adaptive software system to respond to dynamic events. The secure mobile system responds to changes in user goals controlling the encryption algorithm settings, the radar case manages both the hardware and the application to achieve the target localization quality, and the load balancer operates at application level responding to changes in server reliability, performance, and cost. The first case study has only discrete knobs, while the others both continuous and discrete ones.

5.1 Secure Mobile Communication

Mobile systems are a natural match for our methodology. They are limited by battery life, making energy is a primary concern. Further, their applications are primarily interactive, so predictable performance is essential. In this scenario, we add an additional non-functional property: security, as users might want a certain level of privacy for their communication. To accommodate such users, we apply our methodology to create an adaptive system managing performance, energy, and security on a mobile device using the Advanced Encryption Standard (AES) [60] for privacy. AES encodes 128 bit data blocks using keys of size 128, 192, or 256 bits. We create an adaptive encryption system which dynamically selects the block size and processor frequency to manage performance, energy, and security.

Controllable Dimensions:

- **Performance**: 128 bit blocks encoded per second. We measure this directly from the application.

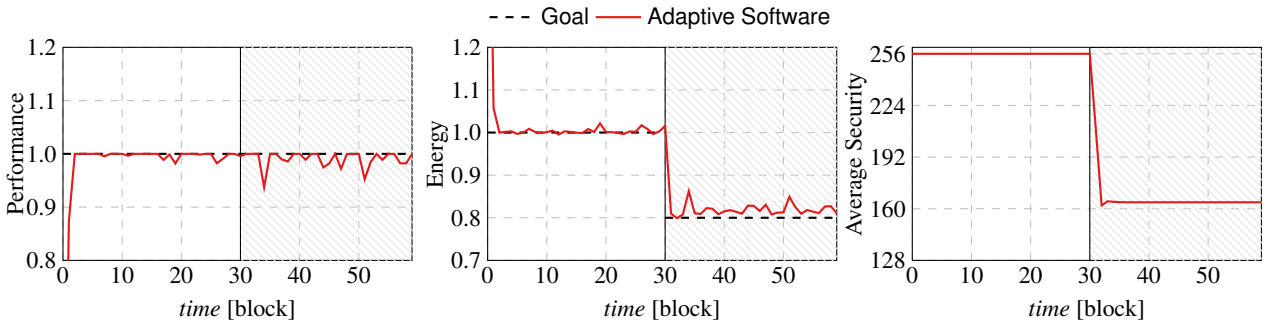

Figure 2: Secure Mobile Communication results: normalized performance and energy consumption, and average security over time.

- **Energy**: joules per block. Energy is measured using the hardware performance counter on our test platform.
- **Security**: average key size over 1024 blocks. This value is read directly from the application.

Knobs: We manage two different knobs for this adaptive software application. The first is based on the hardware and controls the energy and performance tradeoff. The second is based on the AES application and controls the performance and security tradeoff. The mobile system is a Sony VAIO SVT11226CXB Tablet with a dual core Intel Haswell processor. The processor supports eleven clock speed settings, ranging from 600 MHz to 1.501 GHz. Each system configuration shows different tradeoffs between performance and power consumption. Here, therefore, the clock speed affects performance and energy simultaneously.

Our application is based on OpenAES [61]. It supports 3 key sizes each with a different tradeoff between security and performance. The 256 bit key size is the most secure and slowest. The 192 bit key size increases performance by ~ 18%, and the 128 bit key size is ~ 40% faster.

Adapting to Changing Users Needs: We demonstrate how the security application constructed with our methodology responds to changing user goals. Fig. 2 shows the performance, energy, and security for our adaptive application over time (where time is measured in the application progress, *i.e.,* encoded blocks). The performance and energy are normalized to the default setting, *i.e.,* when the actuators have their default values.

We show the system reacting to a change in user goals. This change represents a shift from operating on wall power (where energy is not an issue) to battery power (where energy consumption becomes paramount). Initially, the goal is to obtain real-time performance (represented by 1 in the first plot of Fig. 2) and maximum security, which means using a key size of 256 bits). At time 30, we switch from wall power to battery and set new goals: the application should still maintain real-time performance but reduce energy consumption to 80% of the default value, increasing battery life by 25%. The areas corresponding to the two different goals are marked in the plot by a white background a striped pattern background. Notice that we are using two knobs to control two dimensions, while optimizing in the third one.

The initial goals are highest security and real-time performance, and they switch to the same performance with less energy consumption. As shown in the figure, the performance goal is maintained throughout the application's execution — with some minor disturbances due to random system activity. When, at block 30, the goals change, the control system makes energy consumption immediately drop to the desired value. After a minor degradation, performance quickly returns to real-time, by reducing security.

5.2 Radar Signal Processing

Radar signal processing is another example application which must balance the competing demands of multiple quantitative non-functional properties, this time in a cyberphysical system. Signal processing applications are composed of individual kernels, each of which might support a tradeoff between the computational complexity of the kernel and the accuracy of the result. The accuracy of the radar (*i.e.,* its ability to detect targets in noise) will be a function of the composition of the accuracy of these individual kernels. As with our previous examples, energy consumption is a major concern in many platforms. For example, in an autonomous vehicle, the vehicle's range will be determined by its energy consumption. A fielded radar system will have goals in all of these dimensions: performance ensures that targets are detected in time, accuracy ensures targets are detected in noise, and energy determines mission lifetime. This case study demonstrates the methodology proposed in this paper, but here accuracy/performance tradeoffs are determined by a combination of continuous and discrete knobs.

Controllable Dimensions:

- **Performance**: radar pulses processed per second. This value is measured directly from the application.
- **Power**: Watts per pulse. This is measured using a hardware power meter available on our test platform.
- **Accuracy**: Signal to noise ratio for detected targets. The application reports these values.

Knobs: We use our methodology to control several different knobs at both the system and application level. Our hardware platform is a 32-core Intel Xeon E5-2690 processor running Linux 3.2.0. The processor supports hyper-threading and has 16 different speeds including TurboBoost. We read total system power from a WattsUp power meter. Like our prior example, there are three system-level knobs that alter the performance and energy tradeoffs. The first uses the `cpufrequtils` package to control clockspeed (the highest setting actually turns control over to the hardware by enabling TurboBoost). Higher clockspeeds increase performance at the cost of increased power consumption. The second adaptation uses thread affinity to reduce the number of cores actively performing computation. The processor supports power-gating, so reducing core usage will reduce both power consumption and performance. The final adaptation is to idle the processor to take advantage of the low-power idle state. This adaptation supports racing to idle, where the system attempts to complete work as quickly as possible and maximize idle time. Idling will not increase power consumption, but can decrease it for a decrease in performance. The total number of system configurations is 512 (counting the use of idle states, the number of configurations is effectively infinite).

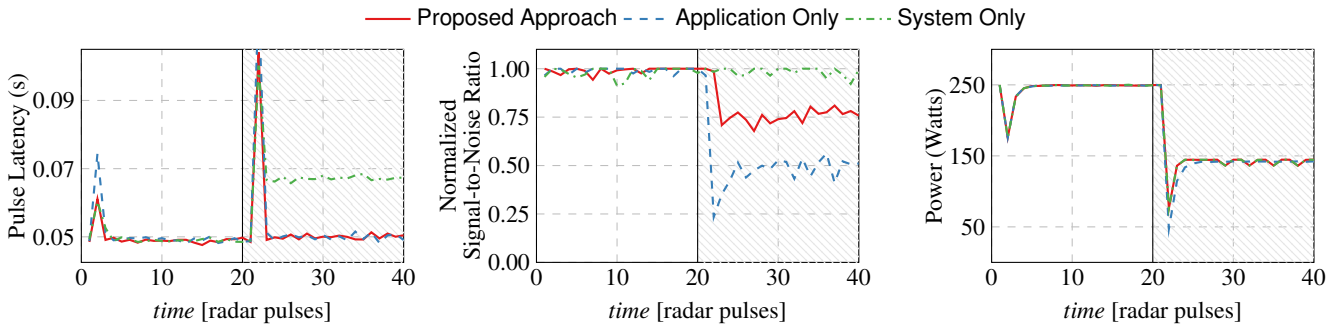

Figure 3: Radar results: performance, energy consumption and accuracy over time.

We use an existing radar benchmark, which represents a generic processing chain for a phased array sensor [62]. The radar supports application-level tradeoffs that can reduce signal to noise ratio (accuracy) in exchange for increased performance. There are a number of parameters affecting the radar's accuracy/performance tradeoffs. Two of these are discrete parameters: the strength of an initial low-pass filter, which has 16 settings, and the number of beams (number of directions to search simultaneously), which can be set from 8 to 96, in multiples of 8. The third parameter is effectively continuous. It is the number of range bins (the resolution in distance from the radar). This value can be anywhere from 1 to 9000. While it is possible to model this as a discrete variable with 9000 settings, it is impractical, so we treat it as a continuous variable.

The total number of possible configurations considering both application and system is greater than 1 billion. It is simply impossible to build empirical models of every possible combination of knob settings. Thus, this case study further tests our methodology's ability to approximate the effect control of one property will have on another (learning is indeed limited to a grid sampling of 2 million configurations; at runtime, thanks to the analysis of the dual optimization problem described in [50], the search space is reduced to just 2 thousand, reducing the runtime computational overhead).

Adapting to Changing Mission Requirements: To demonstrate the benefit of our proposed approach, we consider an autonomous vehicle in the field performing radar processing. During its mission, the vehicle gets a change in requirements. Initially, the system was deployed with a latency and accuracy goal: it was to meet a target latency (1/20th of a second per radar pulse) and maximize the signal to noise ratio. Halfway through this mission, the vehicle receives a new set of goals: it must still maintain the same latency, but now has to increase its mission lifetime (reducing power to 145 W).

To demonstrate the power of this paper's proposed approach we compare to two other approaches. The first adapts only the application. The second adapts only the system. We measure the performance (as latency), power, and accuracy for each of these three approaches. The results are shown in Fig. 3. The left chart shows latency, the middle shows power, and the right shows accuracy. The x-axis of each chart shows time measured in radar pulses.

The results demonstrate the power of our technique. The system-level approach reduces power, but exceeds the target latency by almost 30%. The application-level approach meets the target latency and power, but sacrifices accuracy (reducing signal to noise ratio to just 50% of the default. The approach in this paper provides the best outcome: it meets the latency and power goals, but when the mission changes it reduces signal to noise ratio to 75% of the default, a considerable savings over the the application-level approach.

5.3 Multi-objective Service Dynamic Binding

Our final example is the problem of multi-objective dynamic binding in the context of Service Oriented Architecture (SOA). Dynamic binding flexibly assigns abstract service interfaces to concrete implementations, possibly provided by third parties and is one of the principal means to adapt the behavior of SOAs [63]. In this example, an abstract service interface can delegate an incoming request to one out of three third-party services. Although functionally equivalent, each of the three services provides a distinct reliability and performance depending on the paid service level agreement. The measurable performance depends on the service level and external factors including the underlying communication and execution infrastructure and the workloads of the three services.

The controller selects which service should process any incoming request and, for each service, the service level for each time step. The controller is synthesized to achieve the desired reliability and performance, while minimizing the cost due the selection of higher service levels. We considered the problem of dynamic binding in previous work, controlling only a single objective with a single knob [7, 64]. Here we provide a solution to the multi-objective problem.

Controllable Dimensions:

- **Reliability**: the probability of processing an request without exceptions. We estimate this from the counts of forwarded requests and thrown exceptions each time step.
- **Performance**: average response time. The binder measures its perceived end-to-end service time for each forwarded request and averages over a time step.
- **Cost**: depending on the service level requested for the three services. Each service reports this through an API.

Knobs: The controller can decide a service level from 1 to 5 for each of the three services and the distribution of the incoming requests by setting two probabilities p_1 and p_2. In particular, the probability of selecting the first service is p_1, while the second service has probability $(1-p_1)\cdot p_2$, and, consequently, the third service is selected with probability $(1-p_1)\cdot(1-p_2)$. The domain of both p_1 and p_2 is continuous and incrementally discretized up to a maximum resolution of 0.01 (leading to at most $1,275,125$ configurations to be potentially explored).

Each service s_i, has a measurable reliability r_i, performance coefficient t_i, and cost coefficient c_i. Also, the controller can decide a service level l_i. For every request, a fair coin is flipped in our implementation, to decide whether it will raise an exception or not according to p_i; the response time for each request is sampled from an exponential distribution with mean $t_i/(l_i^2)$, *i.e.,* the time required to process the request is an inverse quadratic function of the service level; the cost of processing an incoming request is $c_i \cdot l_i$.

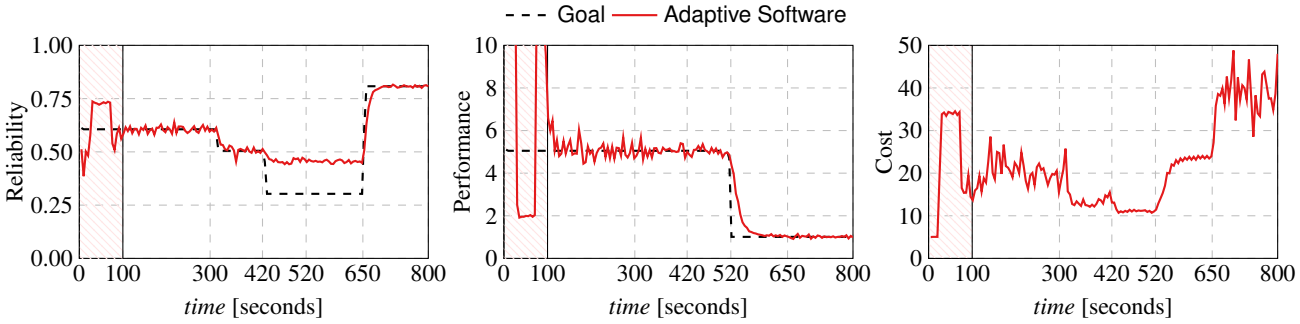

Figure 4: Service Dynamic Binding results: reliability, performance and cost over time.

The nominal values r_i, t_i, and c_i are not known by the controller, which can only measure the time it takes to process a request it forwards to s_i, how many of such requests are successful or failed, and how much it cost to process each request. In order to quantify the values $d_j(\vec{c})$ needed to decide the control actions, the controller undergoes an initial *learning phase* where the configuration space is explored to fit a parametric model of the system behavior. The two problems to undertake are thus two: 1) what could be a suitable parametric model to fit and 2) how to sample the configuration space since its exhaustive exploration may be unpractical.

For the first problem, by reasoning on the structure of the system we can identify a simple polynomial parametric family of models to describe the possible behaviors of the system: $p_1 \cdot q_1 + (1-p_1) \cdot (p_2 \cdot q_2 + (1-p_2) \cdot q_3)$. The values of p_1 and p_2 are set by the controller and therefore known, while q_i is a placeholder for the estimators of the values d_j (j in {reliability, performance, cost}) when the service operates at level l_i. Fitting this parametric model requires only 15 values to be estimated for each service (three dimensions times five service levels), for a total of 45 estimates.

The exploration of the configuration space is performed in two phases. In the first phase the baseline configuration is thoroughly assessed. In the second phase a systematic exploration of the configuration space is performed through a quasi-Montecarlo sampling based on the Halton sequence, a low-discrepancy sequence with demonstrated effectiveness for multidimensional spaces [54] (the exploration of the extreme values for each knob's domain are manually added to the sample set to span the learning over the full control domain). The use of the Halton sequence may increase the convergence of the estimators for the qualities q_i up to be linear in the number of samples. In practice, with only 1000 samples reasonably good initial estimates for $d_j(\vec{c})$ have been achieved.

To overcome possible inaccuracy of the initial estimation and to deal with changes of the services behavior during runtime, the initial estimates are kept updated during the control phase; *i.e.,* at each time step a new sample is pushed to the estimators and used to continuously refine the estimate. The estimator we use here has been proposed in [65] and allows for the incremental estimation of both the mean and the variance of each parameter q_i. Each estimator requires only three floating point values to be stored and a few arithmetic operations to update the estimate after each new sample. Furthermore, given the variance of the parameters is estimated too, a simple change point detection mechanism can be implemented based on the frequency of outliers in the new samples (*e.g.,* those lying further than n times the standard deviation away from the mean). When such frequency gets too high, it is possible that the system undergone an abrupt change which invalidates the current model and requires a new learning phase [7, 59].

The results of this experiment are reported in Fig. 4. The setting for these experiment is: $r_1 = .9$, $t_1 = 2$, $c_1 = 15$, $r_2 = .65$, $t_2 = 10$, $c_2 = 10$, $r_3 = .45$, $t_3 = 20$, and $c_3 = 5$. On the three plots the setpoint for the reliability (leading dimension) and the performance (subsidiary) are represented by a dashed black line, while the obtained quality is in a red continuous line. The initial learning phase is marked with a striped pattern. The control starts from time 100 and achieves both the goals, though it takes some effort to keep performance to the setpoint given the required reliability. At time 300 the goal for reliability is reduced; both the goals are still feasible but the performance can be achieved more smoothly. At time 420 the setpoint for reliability is changed to an infeasible goal; the controller therefore approaches the goal, being as close as possible; performance is instead achievable. At time 520 the required performance is stressed more; the goal is still achievable, though at an higher cost (*i.e.,* higher service levels are required). Finally, at time 650 the goal for reliability is raised to a higher value; both reliability and performance are achievable, though the cost is much higher than before.

6. CONCLUSION AND FUTURE WORK

In this paper we proposed an automated control strategy to achieve multiple objectives using the many knobs available in production software systems. To take advantage of the formal guarantees that the proposed methodology offers, users need only identify a set of knobs and the maximum timescale of all the knobs that belong to the set (*e.g.,* the time it takes to change the processor frequency). With this information, our methodology automatically devises a control strategy, composed of multiple connected loops, which guarantees, whenever feasible, all the chosen dimensions, and optimizes the last free one. We have shown three case studies, introducing the features of our system in order of increasing complexity. The fist case study assumes perfect knowledge of the actions that can be taken on the system and all the involved dimensions, while more uncertainty is introduced in the following two. In the last case study, we have shown how the system deals with infeasible goals. This work advances the state-of-the-art on automated control strategies for software systems, moving existing efforts much closer to what is necessary in a real system.

7. ACKNOWLEDGMENTS

This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control" and "Power and temperature control for large-scale computing infrastructures", and through the LCCC Linnaeus and ELLIIT Excellence Centers. Henry Hoffmann is funded by the U.S. Government under the DARPA PERFECT program, by the Dept. of Energy under DOE DE-AC02-06CH11357, and by the NSF under CCF 1439156.

References

[1] J. O. Kephart and D. M. Chess. "The Vision of Autonomic Computing". In: *Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055.

[2] R. Laddaga. "Guest Editor's Introduction: Creating Robust Software through Self-Adaptation". In: *IEEE Intelligent Systems* 14 (3 1999), pp. 26–29. DOI: 10.1109/MIS.1999.769879.

[3] M. Salehie and L. Tahvildari. "Self-adaptive Software: Landscape and Research Challenges". In: *ACM Trans. Auton. Adapt. Syst.* 4.2 (2009), 14:1–14:42. DOI: 10.1145/1516533.1516538.

[4] W. Levine. *The control handbook*. CRC Press, 2005.

[5] 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: SEAMS. 2012, pp. 33–42. DOI: 10.1109/SEAMS.2012.6224389.

[6] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel. "Software Engineering Meets Control Theory". In: *SEAMS*. IEEE, 2015.

[7] A. Filieri, H. Hoffmann, and M. Maggio. "Automated Design of Self-adaptive Software with Control-theoretical Formal Guarantees". In: ICSE. ACM, 2014, pp. 299–310. DOI: 10.1145/2568225.2568272.

[8] H. Hoffmann. "CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems". In: ECRTS. IEEE CS, 2014. DOI: 10.1109/ECRTS.2014.32.

[9] C. Imes and H. Hoffmann. "Minimizing energy under performance constraints on embedded platforms: resource allocation heuristics for homogeneous and single-ISA heterogeneous multi-cores". In: *SIGBED Review* 11.4 (2014), pp. 49–54. DOI: 10.1145/2724942.2724950.

[10] J. Kramer and J. Magee. "Self-Managed Systems: An Architectural Challenge". In: FOSE. IEEE CS, 2007, pp. 259–268. DOI: 10.1109/FOSE.2007.19.

[11] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. MÃijller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. "Software Engineering for Self-Adaptive Systems: A Research Roadmap". In: *Software Engineering for Self-Adaptive Systems*. Vol. 5525. LNCS. Springer, 2009, pp. 1–26. DOI: 10.1007/978-3-642-02161-9_1.

[12] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N. M. Amato, and L. Rauchwerger. "A Framework for Adaptive Algorithm Selection in STAPL". In: PPoPP. ACM, 2005, pp. 277–288. DOI: 10.1145/1065944.1065981.

[13] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. "PetaBricks: A Language and Compiler for Algorithmic Choice". In: PLDI. ACM, 2009, pp. 38–49. DOI: 10.1145/1542476.1542481.

[14] T. Karcher and V. Pankratius. "Run-time Automatic Performance Tuning for Multicore Applications". In: Euro-Par. Springer-Verlag, 2011, pp. 3–14. DOI: 10.1007/978-3-642-23400-2_2.

[15] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. "Dynamic Knobs for Responsive Power-aware Computing". In: ASPLOS. ACM, 2011, pp. 199–212. DOI: 10.1145/1950365.1950390.

[16] W. Baek and T. M. Chilimbi. "Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation". In: PLDI. ACM, 2010, pp. 198–209. DOI: 10.1145/1806596.1806620.

[17] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. "Eon: A Language and Runtime System for Perpetual Systems". In: SenSys. ACM, 2007, pp. 161–174. DOI: 10.1145/1322263.1322279.

[18] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. "Accelerating Critical Section Execution with Asymmetric Multi-core Architectures". In: ASPLOS. ACM, 2009, pp. 253–264. DOI: 10.1145/1508244.1508274.

[19] R. Bitirgen, E. Ipek, and J. F. Martinez. "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach". In: MICRO. IEEE CS, 2008, pp. 318–329. DOI: 10.1109/MICRO.2008.4771801.

[20] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez. "Core Fusion: Accommodating Software Diversity in Chip Multiprocessors". In: ISCA. ACM, 2007, pp. 186–197. DOI: 10.1145/1250662.1250686.

[21] Y. Dotsenko, S. S. Baghsorkhi, B. Lloyd, and N. K. Govindaraju. "Auto-tuning of Fast Fourier Transform on Graphics Processors". In: PPoPP. ACM, 2011, pp. 257–266. DOI: 10.1145/1941553.1941589.

[22] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (2008), pp. 107–113. DOI: 10.1145/1327452.1327492.

[23] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya. "On Using Pattern Matching Algorithms in MapReduce Applications". In: ISPA. IEEE CS, 2011, pp. 75–80. DOI: 10.1109/ISPA.2011.24.

[24] S. Babu. "Towards Automatic Optimization of MapReduce Programs". In: SoCC. ACM, 2010, pp. 137–142. DOI: 10.1145/1807128.1807150.

[25] IBM Inc. *IBM Autonomic Computing website*. http://www.research.ibm.com/autonomic/. 2009.

[26] O. Krieger, M. Auslander, B. Rosenburg, R. W. Wisniewski, J. Xenidis, D. Da Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. Uhlig. "K42: Building a Complete Operating System". In: EuroSys. ACM, 2006, pp. 133–145. DOI: 10.1145/1217935.1217949.

[27] Oracle Corp. *Automatic Workload Repository (AWR) in Oracle Database 10g*. http://www.oracle-base.com/articles/10g/AutomaticWorkloadRepository10g.php.

[28] Intel Inc. *Reliability, Availability, and Serviceability for the Always-on Enterprise*. www.intel.com/assets/pdf/whitepaper/ras.pdf. 2005.

[29] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad. "A Survey of Formal Methods in Self-adaptive Systems". In: C3S2E. 2012, pp. 67–79. DOI: 10.1145/2347583.2347592.

[30] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. "Self-Managing Systems: A Control Theory Foundation". In: ECBS. IEEE CS, 2005, pp. 441–448. DOI: 10.1109/ECBS.2005.60.

[31] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[32] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. "Self-adaptive Software Meets Control Theory: A Preliminary Approach Supporting Reliability Requirements". In: ASE. IEEE CS, 2011, pp. 283–292. DOI: 10.1109/ASE.2011.6100064.

[33] E. Yuan, N. Esfahani, and S. Malek. "A Systematic Survey of Self-Protecting Software Systems". In: *ACM Trans. Auton. Adapt. Syst.* 8.4 (2014), 17:1–17:41. DOI: 10.1145/2555611.

[34] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. "Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers". In: *IEEE Trans. Parallel Distrib. Syst.* 17.9 (2006), pp. 1014–1027. DOI: 10.1109/TPDS.2006.123.

[35] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck. "From Data Center Resource Allocation to Control Theory and Back". In: CLOUD. IEEE CS, 2010, pp. 410–417. DOI: 10.1109/CLOUD.2010.55.

[36] D. Kusic and N. Kandasamy. "Risk-aware Limited Lookahead Control for Dynamic Resource Provisioning in Enterprise Computing Systems". In: *Cluster Computing* 10.4 (2007), pp. 395–408. DOI: 10.1007/s10586-007-0022-y.

[37] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal. "A Generalized Software Framework for Accurate and Efficient Management of Performance Goals". In: EMSOFT. IEEE Press, 2013, 19:1–19:10. DOI: 10.1109/EMSOFT.2013.6658597.

[38] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. "POET: a portable approach to minimizing energy under soft real-time constraints". In: *RTAS*. 2015.

[39] S. Oberthür, C. Böke, and B. Griese. "Dynamic Online Reconfiguration for Customizable and Self-optimizing Operating Systems". In: EMSOFT. ACM, 2005, pp. 335–338. DOI: 10.1145/1086228.1086288.

[40] C. Karamanolis, M. Karlsson, and X. Zhu. "Designing Controllable Computer Systems". In: HOTOS. USENIX Association, 2005, pp. 9–15.

[41] M. Maggio, H. Hoffmann, M. Santambrogio, A Agarwal, and A Leva. "Controlling software applications via resource allocation within the heartbeats framework". In: CDC. IEEE, 2010, pp. 3736–3741. DOI: 10.1109/CDC.2010.5717893.

[42] M. Maggio, H. Hoffmann, M. Santambrogio, A. Agarwal, and A. Leva. "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation". In: *IEEE Trans. Control Syst. Technol.* 21.1 (2013), pp. 239–246. DOI: 10.1109/TCST.2011.2177499.

[43] H. Hoffmann, J. Holt, G. Kurian, E. Lau, M. Maggio, J. E. Miller, S. M. Neuman, M. Sinangil, Y. Sinangil, A. Agarwal, A. P. Chandrakasan, and S. Devadas. "Self-aware Computing in the Angstrom Processor". In: DAC. ACM, 2012, pp. 259–264. DOI: 10.1145/2228360.2228409.

[44] M. Harman, Y. Jia, W. B. Langdon, J. Petke, I. H. Moghadam, S. Yoo, and F. Wu. "Genetic Improvement for Adaptive Software Engineering (Keynote)". In: SEAMS. ACM, 2014, pp. 1–4. DOI: 10.1145/2593929.2600116.

[45] R. C. Dorf. *Modern Control Systems*. 7th. Addison-Wesley Longman Publishing Co., Inc., 1995.

[46] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin. "What Does Control Theory Bring to Systems Research?" In: *SIGOPS Oper. Syst. Rev.* 43.1 (2009), pp. 62–69. DOI: 10.1145/1496909.1496922.

[47] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. "Software Architecture Optimization Methods: A Systematic Literature Review". In: *IEEE Trans. Softw. Eng.* 39.5 (2013), pp. 658–683. DOI: 10.1109/TSE.2012.64.

[48] A. Filieri, C. Ghezzi, and G. Tamburrelli. "Run-time Efficient Probabilistic Model Checking". In: ICSE. ACM, 2011, pp. 341–350. DOI: 10.1145/1985793.1985840.

[49] S. Bradley, A. Hax, and T. Magnanti. *Applied mathematical programming*. Addison-Wesley Pub. Co., 1977.

[50] D. H. Kim and H. Hoffmann. *Racing and Pacing to Idle: Minimizing Energy Under Performance Constraints*. Tech. rep. TR-2014-10. University of Chicago, 2014.

[51] T. Zheng, M. Woodside, and M. Litoiu. "Performance Model Estimation and Tracking Using Optimal Filters". In: *IEEE Trans. Softw. Eng.* 34.3 (2008), pp. 391–406. DOI: 10.1109/TSE.2008.30.

[52] A. Filieri, C. Ghezzi, and G. Tamburrelli. "A formal approach to adaptive software: continuous assurance of non-functional requirements". In: *Formal Aspects of Computing* 24.2 (2012), pp. 163–186. DOI: 10.1007/s00165-011-0207-2.

[53] R. Calinescu, Y. Rafiq, K. Johnson, and M. E. Bakir. "Adaptive Model Learning for Continual Verification of Non-functional Properties". In: ICPE. ACM, 2014, pp. 87–98. DOI: 10.1145/2568088.2568094.

[54] C. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer-Verlag, 2010.

[55] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010.

[56] S. Marsland. *Machine Learning: An Algorithmic Perspective*. Taylor & Francis, 2011.

[57] M. Zakerifar, W. Biles, and G. Evans. "Kriging metamodeling in multi-objective simulation optimization". In: WSC. IEEE, 2009, pp. 2115–2122. DOI: 10.1109/WSC.2009.5429645.

[58] A Filieri and C. Ghezzi. "Further steps towards efficient runtime verification: Handling probabilistic cost models". In: FormSERA. 2012, pp. 2–8. DOI: 10.1109/FormSERA.2012.6229785.

[59] A. Filieri, L. Grunske, and A. Leva. "Lightweight Adaptive Filtering for Efficient Learning and Updating of Probabilistic Models". In: *ICSE*. IEEE, 2015.

[60] F. P. Miller, A. F. Vandome, and J. McBrewster. *Advanced Encryption Standard*. Alpha Press, 2009.

[61] *OpenAES*. http://code.google.com/p/openaes.

[62] H. Hoffmann, A. Agarwal, and S. Devadas. "Selecting Spatiotemporal Patterns for Development of Parallel Applications". In: *IEEE Trans. Parallel Distrib. Syst.* 23.10 (2012), pp. 1970–1982. DOI: 10.1109/TPDS.2011.298.

[63] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. "A journey to highly dynamic, self-adaptive service-based applications". In: *Automated Software Engineering* 15.3-4 (2008), pp. 313–341. DOI: 10.1007/s10515-008-0032-x.

[64] A Filieri, C. Ghezzi, A Leva, and M. Maggio. "Reliability-driven dynamic binding via feedback control". In: *SEAMS*. 2012, pp. 43–52. DOI: 10.1109/SEAMS.2012.6224390.

[65] D. Knuth. *The Art of Computer Programming: Seminumerical algorithms*. v. 2. Addison-Wesley, 1997.