Keep it SIMPLEX: Satisfying Multiple Goals with Guarantees in Control-Based Self-Adaptive Systems

Stepan Shevtsov
Linnaeus University
Växjö, Sweden
stepan.shevtsov@lnu.se

Danny Weyns
KU Leuven, Belgium
Linnaeus University, Sweden
danny.weyns@kuleuven.be

ABSTRACT

An increasingly important concern of software engineers is handling uncertainties at design time, such as environment dynamics that may be difficult to predict or requirements that may change during operation. The idea of self-adaptation is to handle such uncertainties at runtime, when the knowledge becomes available. As more systems with strict requirements require self-adaptation, providing guarantees for adaptation has become a high-priority. Providing such guarantees with traditional architecture-based approaches has shown to be challenging. In response, researchers have studied the application of control theory to realize self-adaptation. However, existing control-theoretic approaches applied to adapt software systems have primarily focused on satisfying only a single adaptation goal at a time, which is often too restrictive for real applications. In this paper, we present Simplex Control Adaptation, *SimCA*, a new approach to self-adaptation that satisfies multiple goals, while being optimal with respect to an additional goal. SimCA offers robustness to measurement inaccuracy and environmental disturbances, and provides guarantees. We evaluate SimCA for two systems with strict requirements that have to deal with uncertainties: an underwater vehicle system used for oceanic surveillance, and a tele-assistance system for health care support.

CCS Concepts

• **Software and its engineering** → *Designing software;*
• **Computing methodologies** → *Computational control theory;*

Keywords

Self-adaptive system, control theory, simplex, multiple goals

1. INTRODUCTION

The ever growing demand on software has drastically increased the burden on software engineers. Customers expect software to cope with continuously changing conditions. They expect the software to deal seamlessly with varying resources, mask sudden failures, and adapt to changes in system goals. Often, these changing conditions are difficult to predict at design time and handling these uncertainties has become an important concern of software engineers.

Self-adaptation is widely encouraged to address such uncertainties [1, 2]. Self-adaptation handles uncertainties at runtime, when the knowledge becomes available. To that end, the system is equipped with a feedback loop that monitors the system and environment and adapts the system to meet the requirements under changing conditions. As more systems with strict requirements require self-adaptation, providing guarantees for adaptation has become a high-priority concern [3, 4, 5, 6]. Architecture-based approaches for self-adaptation [7, 8, 9], where feedback loops consist of components that realize monitor-analyze-plan-execute (MAPE) functions, have been widely used to ensure system goals under uncertainty. However, recent research has pointed out that providing assurances for such systems is very challenging [10, 11], calling for new perspectives on engineering self-adaptive systems.

More then a decade ago, Hellerstein et al. [12] argued for using principles from control theory as a solution for runtime adaptation with formal guarantees. This viewpoint has recently gained increasing attention, e.g., [13, 14]. In this approach, a software system is treated as a plant to be controlled[1], and a control feedback loop empowers the software with self-adaptation capabilities, providing formal guarantees, regardless of uncertain operating conditions [15, 16].

Recently, a strategy for applying control theory to computing systems in a general way has been proposed in the form of the Push-Button Methodology (*PBM*) [17]. PBM can automatically build a controller of an adaptive software system that rejects environmental disturbances, while providing control-theoretical guarantees for key properties. As the approach is automated, PBM can be used by practitioners with little control-theoretical background. However, PBM deals only with one quantifiable goal at a time, which is often too restrictive for real applications. For example, consider an e-commerce website that should guarantee particular response times for different categories of customers, using available resources, while maximizing profit from advertisements. Another example is a video streaming service that should provide a particular video quality for each class of costumers, employing the available computation facilities, while minimizing congestions along the streaming paths to consumers.

[1]In control theory terminology, "plant" usually refers to a physical system that is adapted. It is often called *the managed system* by software engineers.

In our research, we focus at one relevant adaptation problem that requires satisfying multiple goals while optimizing the solution according to an additional goal, such as the examples given above. A well-known approach to handle such problems is the simplex method [18] and its variations. However, simplex cannot be applied "as is" to realistic software problems as it can not handle the variety of uncertainties and disturbances that are inherent to software systems. Simplex has no mechanism for rejecting disturbances, transient noise, measurements inaccuracies, etc., nor does it guarantee system stability or absence of errors in the system output. Recent work has explored a control-based approach to handle multiple objectives [19], and pointed to its relevance for practice. However, that approach has restrictions regarding the guarantees it can provide and the engineering support it offers. We further elaborate on this in Section 5.

In this paper we present a new approach called *Simplex Control Adaptation* (SimCA) that aims at solving the problem of adaptation for multiple objectives with guarantees. SimCA builds upon PBM and the simplex method, combining strengths of both approaches. SimCA is able to find 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 controllers that operate on these models, and combines controller outputs using the simplex method to adapt the system. The controllers of SimCA use Kalman filters to dynamically adapt the linear model in order to cope with disturbances and non-linearities.

The evaluation of SimCA is conducted in two steps. First, we evaluate the control theoretical guarantees provided by the approach, including system stability, settling time, absence of overshoot and zero steady-state error, solution optimality, robustness, and detection of infeasible solutions. Not achieving these guarantees may violate certain software qualities. For example, lack of robustness guarantees may lead to instability under disturbances, violating reliability requirements. A more detailed mapping between guarantees and software qualities is given in Section 4. Second, the effectiveness and generality of SimCA is demonstrated on two cases: a UUV (unmanned underwater vehicle) system performing surveillance missions, and a service-based system for health care. These systems are from different domains, but both have strict requirements and need to operate under disturbances. Self-adaptation must guarantee that the requirements of these systems are achieved at runtime, regardless of the disturbances. In addition, we provide a qualitative comparison of SimCA with the approach presented in [19].

The remainder of the paper is structured as follows. A motivating scenario for SimCA is introduced in Section 2. Section 3 presents SimCA and explains how to build self-adaptive systems with the approach. The formal evaluation of guarantees provided by SimCA is given in Section 4. In Section 5, SimCA is empirically evaluated using two cases. Section 6 discusses related work. Finally, conclusions and directions for future research are presented in Section 7.

2. MOTIVATING SCENARIO: UUV SYSTEM

We describe a UUV system (based on [20]) that we use as one of the cases to evaluate SimCA in Section 5 and to illustrate the technical description of SimCA in the next section. UUVs are increasingly used for a wide range of tasks. Here we look at UUVs used for oceanic surveillance, e.g., to monitor pollution of an area. UUVs have to operate in an environment that is subject to restrictions and disturbances: correct sensing may be difficult to achieve, communication may be noisy, etc., requiring a UUV system to be self-adaptive.

Furthermore, there is a need for *guarantees* as UUVs have strict requirements, i.e., the system should not impact the ocean area, and since vehicles are expensive equipment that should not be lost during missions.

The self-adaptive UUV system in our study that is used to carry out a surveillance and data gathering mission is equipped with 5 on-board sensors that can measure the same attribute of the ocean environment (e.g., water current or salinity). Each sensor performs scans with a certain speed and accuracy, while consuming a certain amount of energy (see Table 1). A scan is performed every second.

Table 1: Parameters of sensors of the UUV.

UUV on-board sensor	Energy cons., J/s	Scan Speed, m/s	Accuracy, %
Sensor1	170	2.6	97
Sensor2	135	3.6	89
Sensor3	118	2.6	83
Sensor4	100	3.0	74
Sensor5	78	3.6	49

The UUV system has to satisfy the following requirements:

R1: A segment of surface over a distance of S (100 km) should be examined by the UUV within a given time t (10 hours in the scenario);

R2: To perform the mission, a given amount of energy E is available (5.4 MJ in the scenario);

R3: Subject to R1 and R2, the accuracy of measurements should be maximized.

To realize the requirements, sensors can be dynamically turned on and off during a mission. We assume that only one sensor is active at a time, however, we use a combination of sensors during each adaptation period. For example, to perform a mission with energy consumption of 135 J/s we may either use Sensor2 100% of the time, or use Sensor1 50% of the time (using 170*0.5=85 J/s) and Sensor4 50% of the time (using 100*0.5=50 J/s).

The requirements R1 and R2 are critical to the success of the surveillance mission, but they may change at runtime due to unpredictable events in the environment. In addition, the adaptation task is not trivial because the system is affected by different disturbances such as:

- Fluctuations in the expected behavior of the UUV (actual scanning speed or energy consumption differs from the specification) up to ±10% of the expected values;

- Inaccurate measurements: e.g., the monitoring mechanism reports a scanning speed of 2.6 m/s instead of the actual value of 2.2 m/s;

- Constant deviations of the sensor output due to a sensor problem, e.g., a sensor starts consuming 50% more energy than stated in the specification;

- Sensor failures;

- Gaussian or Random noise in the communication channel, which may cause errors of the communicated data.

In summary, to realize its mission, the UUV needs to self-

adapt to changes in requirements and rejects different types of disturbances. The achievement of goals must be guaranteed.

Problem definition: The general adaptation problem we aim to solve is the following:

> *To guarantee the satisfaction of multiple goals and optimize the solution according to another goal, regardless of possible fluctuations in the system parameters, measurement accuracies, requirement changes, and dynamics in the environment that are difficult to predict.*

The UUV scenario offers one concrete instance of this general problem. Defining and developing an adaptive solution for this general problem introduces several key challenges. First, the appropriate adaptation sensors (measured variables) and actuators (knobs that can influence the software behavior) must be carefully chosen. Second, the software system must be modeled. Third, the appropriate adaptation mechanism that controls the model and satisfies multiple goals, while rejecting external disturbances, must be developed. Fourth, the system must incorporate an optimization approach to optimize the solution according to additional goal. The following section proposes SimCA that aims to address these challenges.

3. SIMPLEX CONTROL ADAPTATION

To build an adaptive system with SimCA, the approach requires four elements from a software engineer:

1. A working prototype of the software (plant).
2. A set of quantifiable goals to be controlled plus one optimization goal. For requirements with time-dependent constraints (e.g. a constraint on the available energy to be used in time window), we transfer the constraints to setpoints that satisfy the constraints over time.
3. Tunable parameters (actuators) that can be used to adapt the running system to address the goals.
4. Adaptation sensors[2] to measure the effect of the adaptation on the system.

With these four elements SimCA is able to build a self-adaptive system that solves the adaptation problem formulated in Section 2. In order to use SimCA, engineers do not need to construct software models. Instead, the approach works in three runtime phases:

- First, in the *Identification* phase, SimCA synthesizes models that capture the dependency between the adaptation parameters and the measured system outputs.
- Second, in the *Controller Synthesis* phase, SimCA constructs an appropriate set of controllers for the synthesized models.
- Third, in the *Operation* phase, the controllers carry out control and the outcome of multiple controllers is combined using the simplex method to optimally drive the outputs of the system towards the set goals.

The three phases of SimCA are performed during system operation. We describe the phases in detail in the following subsections.

[2]Not to be confused with UUV sensors in the motivating case.

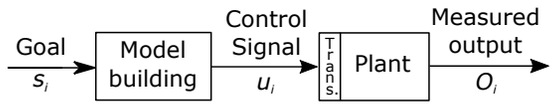

Figure 1: Identification phase of SimCA.

UUV scenario

Before that, we illustrate the four elements required from a software engineer to apply SimCA to the UUV system:

1. A working prototype is the UUV system itself.
2. The quantifiable goals are the scanning speed and energy consumption, the optimization goal is the measurement accuracy. We transform requirements R1 and R2 into quantifiable goals as follows: we keep the average scanning speed on a particular level such that the target area of surface is examined in the given amount of time; similarly, we keep the average energy consumption on a particular level such that the mission is performed with the available energy.
3. The actuator of the UUV system is the combination of sensors that are used for performing the mission.
4. The sensor is the monitoring mechanism that measures the scanning speed and energy consumption of the UUV system during the mission.

3.1 Identification Phase

During the first phase, a set of n linear models of the controlled system is automatically built, where n is the total number of goals excluding the optimization goal. Each model $\mathcal{M}_i$, $i \in [1, n]$, is responsible for one goal s_i.

Similar to basic PBM, identification starts by systematically feeding sampled values of the goal s_i in the form of a control signal u_i to the plant and measuring their effect on the system output O_i (see Figure 1). The vector of control signals u_i used for identification looks as follows:[3]

$$u_i = [min_i,\ min_i + \delta,\ min_i + 2\delta,\ min_i + 3\delta,\ \ldots,\ max_i]$$

Where min_i and max_i are the minimal and maximum achievable values for the i-th goal, δ is the sampling rate. δ is a tunable parameter chosen by the system engineer; by default $\delta = (max_i - min_i) * 0.05$. A higher sampling rate will provide a more accurate model, but increase the identification time; whether this is required depends on the domain.

During the Identification phase (and the Control Synthesis phase, see below), the control signal $u_i(k)$ is automatically translated (marked *Trans.* on Figures 1 and 2) to an actuation signal before feeding it to the plant. A control signal may for example be translated to the change of a parameter setting of the system or the selection of a component or a service. This translation is performed by the simplex method that serves as a straightforward translator of control signals to an actuation signal during the Identification and Control Synthesis phases. Such simplified translation works because at this stage we need an approximate model and not an optimal solution.

After recording all combinations of control signals and resulting system outputs, the dependency between the control signal $u_i(k-1)$ and its effect on the measured output $O_i(k)$ is captured by the coefficient α_i which is further used to build

[3]u_i is an array of elements, with $u_i(k)$ being the k-th element of that array, where k equals to one adaptation period

controller C_i. Coefficient α_i is calculated based on linear regression using the APRE tool [21]. As a result, a set of first order linear models is obtained, representing the reaction of the system to control signals for the different goals:

$$O_i(k) = \alpha_i \times u_i(k-1) \qquad (\mathcal{M}_i)$$

The model $\mathcal{M}_i$ describes the system behavior, but does not take into account small disturbances or sudden failures that typically can occur in practical software systems. For example, this model may not be able to deal with a particular component failure at runtime.

Earlier work has show that $\mathcal{M}_i$ is a linear model that is practical and works for a variety of applications [17], and this is confirmed by the two studies presented in Section 5. The different cases have shown that to be effective, the model does not need to capture the precise (usually non-linear) relationship between the control signal and the system output. In addition, the synthesized controller has mechanisms (see the following section) that allow to use $\mathcal{M}_i$ even for non-linear systems working under disturbances.

Exploring the effect of a range of values of a goal on the system output during Identification may affect the realization of a temporal constraint associated with that goal. This effect was not taken into account in the original PBM [17]. To ensure that the constraint is not violated, the Model building module measures the time Δt_i and resources ΔR_i spent for Identification, subtracts this amount from the available time t_i and resources R_i respectively and automatically adjusts the quantifiable goal s_i accordingly:

$$s_i = \frac{R_i - \Delta R_i}{t_i - \Delta t_i} \qquad (1)$$

UUV scenario

We illustrate the Identification phase for the energy consumption goal of the UUV system. According to Table 2, the minimal available energy consumption is $min_E = 78J/s$, while the maximum is $max_E = 170J/s$. Then, by default, $\delta = (max_E - min_E) * 0.05 = 4.6$. The model identification starts with sending $u_i(0) = min_E = 78J/s$ to the plant that is automatically translated by simplex to *use Sensor 5 all of the time*, because Sensor 5 consumes exactly that amount of energy according to specification. The goal of this procedure is to measure and record the actual output energy consumption of the vehicle $O_E(1)$. After that, the plant receives $u_i(1) = min_E + \delta = 82.6J/s$, simplex translates it to a corresponding combination of sensors to be used, and the output $O_E(2)$ is measured again. When the values of O_i are measured for all k, coefficient α_i is calculated with the APRE tool, resulting in a system model for the energy consumption goal of the UUV system. We observed a typical value of α_i for this model in the range $0.9 \ldots 1.1$. After the Identification, the goal s_E is adjusted according to the amount of consumed energy and time (see eq. 1), for the UUV case: $s_E = (5.4 * 10^6 - 0.2 * 10^6)/(10 * 3600 - 0.5 * 3600) = 152J/s$

3.2 Controller Synthesis Phase

The second phase of SimCA, labeled *controller synthesis*, consists of two sub-phases: controller building and controller re-building, see Figure 2. Controllers are built once, when the system starts, and may be rebuilt during system operation.

Controller Building: during this first sub-phase, a set of n controllers is built using the set of models $\mathcal{M}_i$, $i \in [1 \ldots n]$, each controller managing one goal.

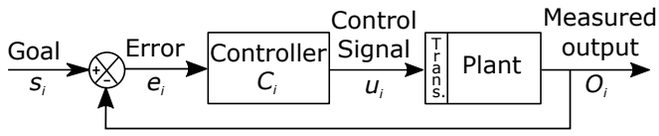

Figure 2: Control Synthesis phase of SimCA.

A controller C_i has one tunable parameter, called *pole* denoted with p_i. To maintain stability and avoid oscillations, the pole value should belong to the open interval $(0, 1)$. The pole is chosen by the controller designer and allows to trade-off certain system properties (see discussion in Section 4).

As shown in [17], the system output equation, representing the measured output $O_i(k)$ in response to a unit step[4] setpoint s_i is defined as follows:[5]

$$O_i(k) = s_i \times (1 - p_i^k) \qquad (2)$$

By using Z-transform — a frequency domain representation of a discrete time control signal — on $(\mathcal{M}_i)$ and (2), and by analyzing the system input-output relationships, the following controller equation can be obtained:

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

The synthesized controller C_i, $i \in [1, n]$, 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 adjustment coefficient α_i, controller pole p_i and the error $e_i(k-1)$, with $e_i = s_i - O_i$.

Controller Rebuilding: during the second sub-phase, the controllers handle inaccuracies in $\mathcal{M}_i$. To that end, the controllers of SimCA incorporate two additional mechanisms introduced by PBM:

1. Each controller uses a Kalman filter to constantly update the value of α, adapting the linear model at runtime. This mechanism allows to cope with small perturbations that could not be tracked by non-adaptive $\mathcal{M}_i$ and assures robustness for non-linear behaving systems.

2. Each controller is equipped with a change point detection mechanism, which allows to react to unexpected critical changes in the system. The mechanism updates the system parameters or, in some scenarios, re-initiates the Identification phase and rebuilds $\mathcal{M}_i$. An example of a critical change may be a software component that suddenly becomes unavailable. Although requiring extra computations, the mechanism is quite simple and makes the controller extremely robust.

UUV scenario

We illustrate Control Synthesis with examples. Assume that the identification phase has produced a model for the energy consumption goal with $\alpha_E = 1$. If the engineer has set the pole for the controller to $p_E = 0.9$, then the Controller Building phase will synthesize the following controller:

$$u_E(k) = u_1(k-1) + 0.1 \times e_E(k-1) \qquad (3)$$

[4]Step in the setpoint of magnitude one – for example, when scanning speed is required to change from 2 to 3 m/s.
[5]p_i^k is p_i to the power k.

If, during system operation the UUV slows down due to unexpected underwater streams in some area, the Kalman filter will change α_E accordingly and trigger controller re-building. If, during operation the change point detection mechanism detects a critical change, e.g. some of the UUV sensors fail, a re-identification will be triggered resulting in a new value of α_E which will be updated in the controller equation.

3.3 Operation Phase

The third phase of SimCA, labeled *operation* also consists of two sub-phases: control and optimization, see Figure 3.

Control: in the first sub-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 $u_i(k)$ that is fed to simplex (see Optimization below). The α_i value of the controller can be updated on the fly by the embedded Kalman filter to handle non-linear system behavior (see Controller Rebuilding). The change point detection mechanism can interrupt the controller to deal with invasive changes of the system. SimCA will then restart Identification, followed by Controller Building.

Optimization: during the second sub-phase, SimCA collects all control signals $u_i(k)$ and the system parameters $\mathcal{P}(k)$,[6] and passes these data to the simplex block. Simplex calculates the actuation signal u_{sx} that drives the system towards an output that satisfies all adaptation goals.

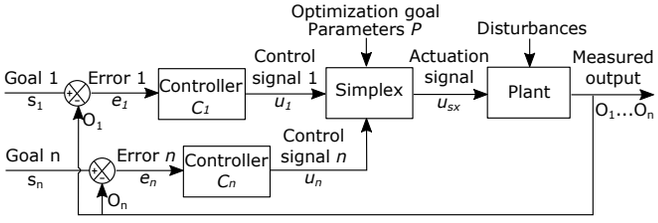

Figure 3: Operation phase of SimCA.

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

$$\max\{c^{\mathrm{T}}x \mid Ax \leq b; x \geq 0\} \tag{4}$$

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 [22].

SimCA uses a simplex variant with equalities ($Ax = b$) because we do not want simplex to change the effect of control signals on the output signals. Instead, simplex is responsible for seamless translation of control signals to actuation signals.

In SimCA each equation, except the last one, represents a goal 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.

For details on how simplex solves the system of equations (4) we refer to the linear programming literature [22, 23, 24].

[6] $\mathcal{P}(k)$ contains relevant parameters of system components that can be measured.

UUV scenario

Assume that the energy consumption goal is set to $s_E = 152$ J/s. We illustrate how the controller calculate the control signal at time $k = 200$, assuming that the control signal at the previous adaptation step $u_E(199) = 149$ and the amount of energy consumed by the UUV at the previous adaptation period $O_E(199) = 150$J/s. By substituting the according values in (3), we get the control signal value:

$$u_E(200) = 149 + 0.1 \times (152 - 150) = 149.2$$

The controller will send this value to the simplex block.

To illustrate the optimization sub-phase, we rewrite (4) as a system of equations using the UUV scenario:

Maximize *Accuracy*:

$$max[Acc_1 \times x_1 + Acc_2 \times x_2 + \cdots + Acc_5 \times x_5]$$

Subject to:

$$\begin{cases} E_1 \times x_1 + E_2 \times x_2 + \cdots + E_5 \times x_5 = u_1 \\ V_1 \times x_1 + V_2 \times x_2 + \cdots + V_5 \times x_5 = u_2 \\ x_1 + x_2 + \cdots + x_5 = 1 \end{cases} \tag{5}$$

Where: x_j (with $j \in [1;5]$) is the portion of time (in decimals) the sensor j should be used during system operation; Acc_j is the accuracy of sensor j; E_j is the energy consumed by sensor j; V_j is the scanning speed of sensor j (for the concrete values of Acc_j, E_j, V_j, see Table 1); and u_1 and u_2 are control signals received from energy consumption controller and scanning speed controller respectively.

As it can be observed from the comparison of (4) and (5), the monitored parameters $\mathcal{P}(k)$ of the system are the sensor energy consumption E_j and the scanning speed V_j with the active sensor j. Vector c^{T} is replaced with accuracies Acc_j of sensors. The last equation of (5) ensures that at each time instance during the mission one sensor is working. The vector x represents the portion of time each sensor should be used during system operation.

4. EVALUATION OF GUARANTEES

We start the evaluation of SimCA by formally analyzing the adaptation guarantees provided by the approach.

4.1 Guaranteed Goal Achievement

The achievement of system goals (except the optimization goal) is guaranteed by the controllers used in SimCA. Specifically, by using controllers we can formally prove the following four system properties: stability, steady-state error, settling time and overshoot. Stability relates to most software qualities that are subject of adaptation and shows the ability of an adaptation mechanism to achieve goal s_i. For example, lack of stability for a security goal implies periods with high vulnerability of the system. If the system has zero steady-state error, its goal s_i is reached after a certain time $\bar{K}$ and $O_i(k) = s_i(k), k \geq \bar{K}$. $\bar{K}$ is called settling time, and shows the time it takes for an adaptation mechanism to bring measured quality properties close to their goals. Settling time is computed for a step in the setpoint of magnitude one – e.g., demanding the scanning speed to vary from 2 to 3 m/s. Settling time and steady-state error are also related to most software qualities that are subject of adaptation. For example, fast achievement of an energy consumption goal (with low settling time) means spending less resources in a transient state. Avoiding overshoot, that is, the controlled signal does

not exceed the goal before reaching its stable area, avoids penalties on the respective software quality. E.g., an overshoot of system response time may violate a service level agreement. Figure 4 illustrates these system properties.

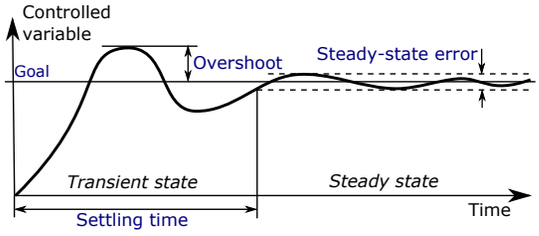

Figure 4: Properties guaranteed by the controllers

The control system used in SimCA is designed to be *stable* and *avoid overshoots*, since it has only a single pole and its value p_i belongs to the open interval $(0, 1)$.

To evaluate the *steady-state error* (Δe) and unit-step *settling time* ($\bar{K}$) we recall the output equation (2). First, we calculate the system output during steady-state, i.e. when $k \to \infty$. As $p \in (0, 1)$, in this case $p^k \to 0$. From (2):

$$O_i(k \to \infty) = s_i \times (1 - p^k) = s_i \tag{6}$$

Based on (6), the steady-state error equals: $\Delta e = s_i - O_i = 0$.

Theoretically, it will take infinite time for O_i to converge to the exact value of goal s_i, i.e. to make Δe zero, we need $k \to \infty$. However, the settling time is formally defined as the time $\bar{K}$ in which the measured variable reaches a value very close to the goal (usually it has reached a certain percentage of the goal value – we denote this value with s_i^*). Based on this, O_i can be replaced with $(1 - \Delta s_i) \times s_i$, where Δs_i is the difference between s_i and s_i^* in percents. From (2) we get:

$$(1 - \Delta s_i) \times s_i = s_i \times (1 - p_i^k) \Rightarrow k = \frac{\ln \Delta s_i}{\ln |p_i|} \tag{7}$$

From this equation it can be concluded that the settling time $\bar{K}$ of every controller C_i depends on the pole p_i: higher values of p_i lead to slower output convergence to the goal value. Δs_i is a constant chosen by the system engineer. According to [12, p.85], the common value of Δs is 0.02 (2%).

As we are using an instance of simplex method with equalities (see Section 3.3), it will not change the effect of control signal u_i on the output signal O_i. Hence, simplex will not alter the mentioned above guarantees provided by controllers.

4.2 Guaranteed Optimality and Scalability

Simplex guarantees the optimization goal of the obtained solution. The simplex method was proven to always find an optimal solution (if it exists) to a linear problem [22, 23], such as the one formulated in Section 3.3.

The scalability of SimCA is also inherited from simplex. To understand the scalability of simplex, an interested reader may ask about the number of iterations required to solve a problem using this algorithm. Examples shown in [25] require $(2m-1)$ iterations worst case, with m the number of equations. Such cases would require too much computation. For practical problems, the method usually finds a solution in just a few iterations [18]. The mismatch between theory and practice is not formulated yet, although a number of efforts have been conducted, incl. the use of probabilistic models to synthesize and solve linear programs to calculate the number of required iterations. Additional details are provided in Section 6.

4.3 Guaranteed Robustness

By *robustness* we mean the amount of perturbation the system can withstand while remaining in stable state or the amount of inaccurate estimate in the model the system can tolerate. Robustness directly influences system reliability. In line with the formal assessment of basic PBM [17], conclusions about the system robustness can be derived for SimCA in a similar fashion: the value of the pole p_i allows to trade robustness for settling time $\bar{K}$.

Formally, the amount of disturbance the system can withstand $\Delta(d)$ by using a controller presented in Section 3.2 can be estimated as follows: $0 < \Delta(d) < \frac{2}{1-p_i}$*. This means that the value of the pole p_i defines how SimCA will react to disturbances. For $p_i = 0.9$, which is used in most of our experiments, the measurement can be inaccurate by a factor of 20, and the controller of SimCA will still adapt the system to follow the goals. In general, higher values of p_i lead to better robustness while lower p_i decreases the settling time.

4.4 Detection of Infeasible Solution

The simplex method brings an additional guarantee for the adaptation strategy: it detects infeasible solutions. According to the principles of linear programming, every linear program (including those solved by SimCA) is subject to one of the following [22, 26]: (1) has an optimal solution; (2) has no feasible solution (e.g., setting the scanning speed of a UUV to 5 m/s which is unreachable with any of the sensors); (3) has an unbounded optimal solution, i.e. the objective function value seeks ∞ (or -∞), which occurs if variable values can grow indefinitely without violating any constraint.

As SimCA uses only equalities, it cannot produce an unbounded solution. However, when the goal is infeasible, SimCA will converge to the nearest achievable value of the according goal and alert the user that the goal is not reachable. Such clear detection of an infeasible solution offer an advantage with respect to the basic PBM approach, for which it is unclear if a non-zero error appears due to disturbances or due to an unfeasible goal being set for the system.

4.5 Boundaries of Guarantees

First of all, the *guarantees are achieved on the model*; if the system is not capable to identify a sufficiently good model then the controller will not be able to achieve its goals and guarantees. The importance of successful identification is one of the main reasons to perform it at runtime in real operating conditions. However, as practice shows, even with poor testing of corner cases or transient behavior during identification, the model is representative enough to provide the guarantees.

Second, the guarantees on achieving time-dependent requirements depend on correct measuring the time and resources spent during identification and computing the adjustment of the corresponding goal.

Third, the guarantees are provided after controllers are built, meaning that control-theoretical guarantees do not apply during the Identification and Controller Synthesis phases.

Fourth, in the current realization, SimCA cannot provide guarantees when goals are added/removed at runtime or when the system behavior/architecture is invasively changed.

*Details on how to obtain this formula can be found in [17].

5. EXPERIMENTAL EVALUATION

We empirically evaluate SimCA with two cases. Section 5.1 describes the experimental setting of the UUV case. Section 5.2 shows the software adaptation performed by SimCA when the goals of the system are changed and in response to variations in the sensor behavior at runtime. In Section 5.3 we show the guarantees provided by SimCA with the case study. The scalability of our approach is tested by adding a panel of sensors to the UUV in Section 5.4. The second case with Tele Assistance System is described and evaluated in Section 5.5. In addition, we provide a qualitative comparison of SimCA with the approach presented in [19] in Section 5.6. Finally, Section 5.7 discusses threats to validity. The experiments are performed on a Dell Notebook with 2.7 GHz Core i7 processor, and 16 GB 1600MHz DD3 RAM. All evaluation material is available at the project website.[7]

5.1 Experimental Setting: UUV case

We use the UUV system described in Section 2 as a primary case to evaluate SimCA. The system is implemented in a Java simulation environment that allows to model and study the behavior of software systems. The initial parameters of the sensors are specified in Table 1. The actual data that is used by the adaptation mechanism at runtime is subject to a randomly distributed disturbance up to ±10% of the expected values, simulating fluctuations of actual parameters of sensors (compared to their specification).

Adaptation is performed every 100 surface measurements of the UUV system: 1 $k = 100$ measurements, and a measurement is performed each second. At each adaptation step the application calculates the average measured value of the i-th goal (e.g., energy consumption) during the past 100 measurements. Then it calculates the error e_i as the difference between i-th setpoint (e.g., target energy consumption) and the measured value of the i-th goal. The application also monitors the accuracy of surface measurements.

The task of SimCA is to maximize the measurement accuracy by exploiting the available energy and set the scanning speed to examine the required surface in the given time frame. SimCA achieves this task by calculating the value of the *actuation signal*, which represents the portion of time each sensor $\{S1, \ldots, S5\}$ is used during every adaptation period. As an indication of the complexity of the data used in the evaluation: the total number of sensor configurations that can be selected in the UUV scenario is 5.5×10^6.

Due to high dynamics and the unpredictable nature of the environment, the controller pole p_i in SimCA is set to 0.9 which allows to reject errors/disturbances of high magnitude. δ is kept at a default value: $\delta = (max_i - min_i) * 0.05$.

The application collects the UUV data to build performance graphs, which are used to evaluate SimCA in the following sections. The x-axis of the graphs are time instants k. Thus, the y-axis shows the average values of the measured feature per 100 surface measurements of the UUV system.

5.2 Adaptation Results

Figure 5 shows the adaptation results of SimCA on the UUV system configured according to Table 1 and requirements set according to UUV scenario (Section 2). Adaptation starts with the Identification phase that is clearly visible when k is between 0 and 20. At time k=20 the energy consumption setpoint slightly increases based on the energy consumed during identification (see Section 3.1). The Control Synthesis phase, followed by the Optimization phase, starts after the relationship between control signals $u_i(k)$ and system outputs $O_i(k)$ is identified (from k equals 21 onwards). The two upper plots in Figure 5 show that during Operation the system is stable, i.e., the measured energy consumption and scanning speed follow their goals. At $k = 100$ we change the available energy change from 5.4 to 5.0 MJ, at $k = 160$ we change the distance to be scanned from 10 to 10.5 km. The plots show that these changes in requirements lead to corresponding changes in goals and adaptation of the system output.

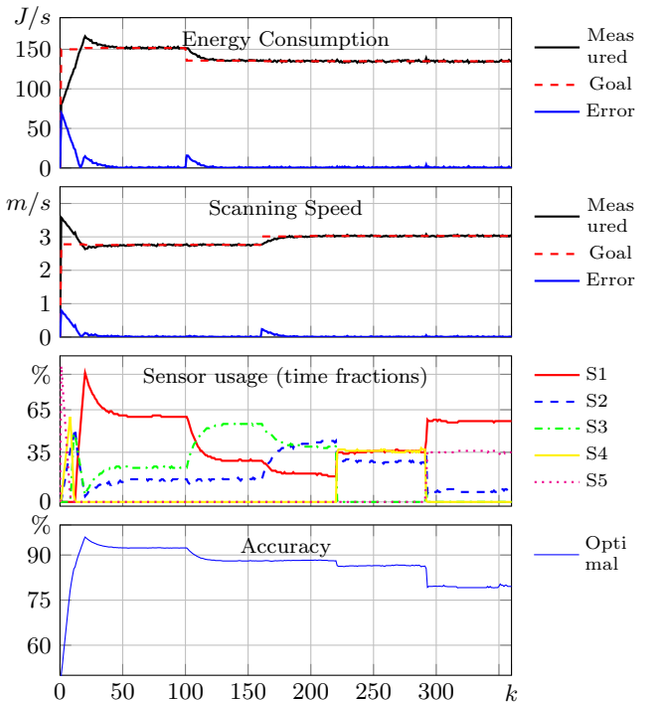

Figure 5: Adaptation of UUV according to different runtime changes.

Figure 5 also shows how SimCA reacts to changes in sensor parameters and sensor failures. At $k = 220$,the measurement accuracy of sensor $S3$ drastically decreases from 83% to 43%. With such a low accuracy, $S3$ is not a part of the optimal solution anymore and the system selects a better sensor $S4$ at $k = 221$, see the "Sensor usage" plot. At $k = 290$, $S4$ stops working, which again leads to switching the sensors to the optimal solution, while the measured energy consumption and scanning speed of the UUV remain on the required level. At this point the measurement accuracy decreases from 87% to 77%. It happens because without $S4$, to satisfy all goals, the system is forced to use $S5$, which has lower accuracy.

The experiment ends at $k = 360$, i.e. after 10 hours of time. The total distance scanned is 10.5 km, the amount of consumed energy is 5 MJ. Over a series of 50 experiments, we measured an error of less than 0.01% on these values.

5.3 Adaptation Guarantees

We now confirm the guarantees formally evaluated in Section 4 with the UUV case study.

Guaranteed Goal Achievement. SimCA's guarantees for achieving the are confirmed by the data shown on Figure 5:

[7] http://homepage.lnu.se/staff/daweaa/simplex.htm

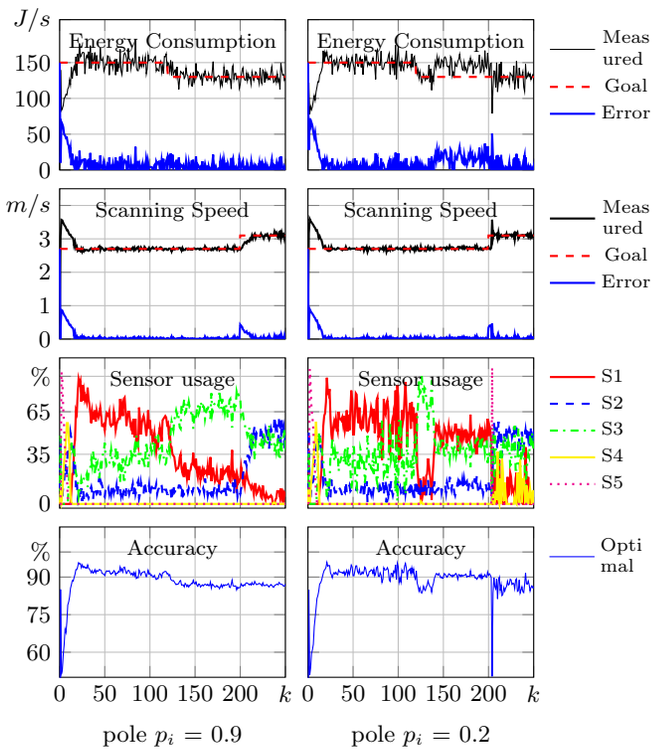

Figure 6: The influence of controller pole on SimCA.

- The system is stable and converges without overshooting, since it is designed to have only a single pole p_i which belongs to the open interval $(0, 1)$;
- According to the system output equation 2, the output O_i during steady-state equals s_i which leads to a zero steady-state error: $\Delta e = s_i - O_i = 0$. The absence of a steady-state error can be observed, for example, on the "Scanning Speed" plot when $k > 25$;
- The settling time $\bar{K}$ of every controller C_i depends on the pole p_i and a constant Δs_i chosen by the system engineer: $\bar{K} = \frac{\ln \Delta s_i}{\ln p_i}$. According to [12, p.85], the commonly used value of Δs is 0.02 (2%). Hence $\bar{K} = \frac{\ln 0.02}{\ln 0.9} = 40$ adaptation steps. This means that changing the scanning speed from 2.7 to 3.1 (step of amplitude 0.4) would take around $\bar{K} = 40 * 0.4 = 16$ adaptation steps. This guarantee can be observed on the "Scanning Speed" plot of Figure 5 when k is between 160 and 176;

Guaranteed Robustness. The next experiment shows the effects of the controller pole p_i on the tradeoff between system robustness and settling time. For this, we add a random disturbance of amplitude up to ±25% of the expected values to the energy consumption output signal. Figure 6 compares the performance of controllers with $p_i = 0.9$ and $p_i = 0.2$ in such conditions.

As described in Section 4, adaptation with SimCA is influenced by the values of the pole. First, a smaller pole leads to a shorter settling time. This effect can be observed when the distance requirement is changed at $k = 200$. The system with a smaller pole (right plots) converges to a new operational goal almost immediately, while a system with a higher pole (left plots) needs 16 adaptation steps to converge. Experimentally

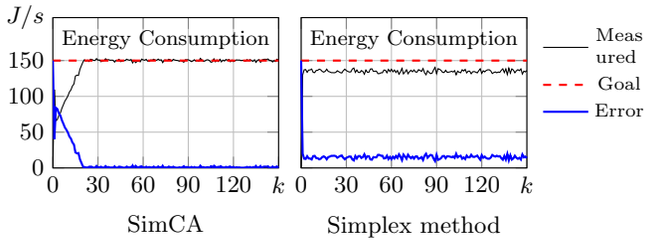

Figure 7: SimCA vs Simplex: constant disturbance.

we determined that due to fast convergence, the total average accuracy of measurements is 0.1% higher for controller with lower pole.

Second, despite the decrease of the settling time, lowering p_i leads to weaker disturbance rejection. This property of adaptation mechanism of SimCA can be observed after $k = 120$. The system with $p_i = 0.2$ unsuccessfully tries to find an optimal solution until $k = 200$. The system with $p_i = 0.9$ continues working as expected. Hence, the system with lower pole is not reliable under disturbances of high magnitude.

Another benefit of a high p_i value is less oscillation of sensor usage and a smoother accuracy curve (compare according plots on Figure 6). This means that a higher pole value leads to a system that is less responsive to variations in parameters but at the same switches less between solutions.

In addition to rejecting noise and measurement inaccuracy, SimCA can reject constant disturbances. E.g., due to an error, a monitor that measures the vehicle energy consumption can constantly decrease the measured value by $15 J/s$. The plot on the left side of Figure 7 shows the behavior in such a scenario. Although monitoring is not working properly, SimCA still adapts the system by defining proper relationship between control signal $u_i(k)$ and system output $O_i(k)$.

Unlike SimCA, the pure simplex method fails at guaranteeing the control-theoretical properties such as disturbance rejection. Hence, the simplex method produces an incorrect output, see the right plot of Figure 7.

Detection of Infeasible Solution. Figure 8 shows the detection of an infeasible solution. The energy consumption remains at the required level during the entire experiment.

Initially, we set the goal distance to be examined to 10 km. After Identification (k>20), the system functions normally. At $k = 150$ the total distance to be examined changes to 13 km, hence the output scanning speed grows until reaching its

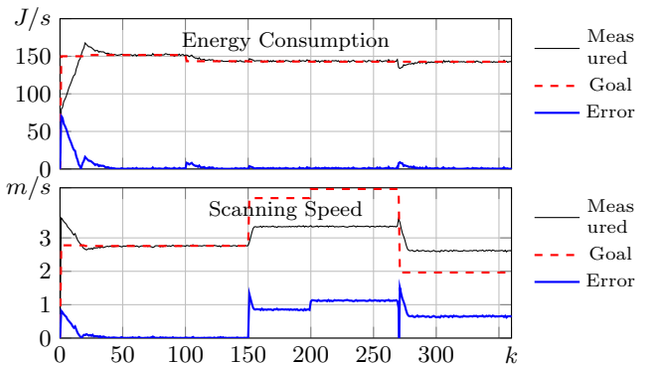

Figure 8: Detection of Infeasible Solution in SimCA.

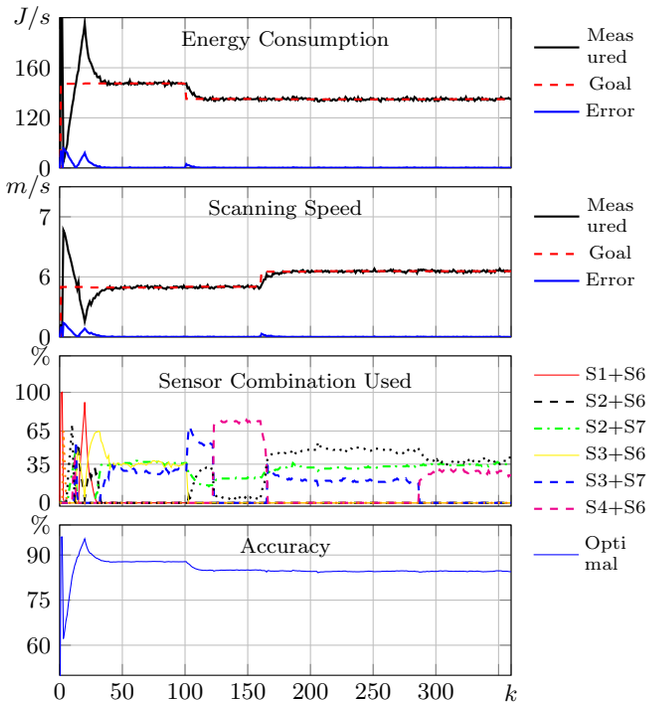

Figure 9: SimCA with 25 sensor combinations.

maximum feasible value of $3.2m/s$ at $k = 155$; and the user is notified about the infeasible solution.

At $k = 270$, we change the distance requirement to an unreachable value of 9 km and the scans start with the minimum possible speed among those sustaining energy consumption at the goal, and the user is notified of the infeasible goal.

It is worth mentioning that getting an infeasible solution does not necessarily mean that the concrete goal is entirely unreachable. For example, the scanning speed of $3.6m/s$ can be achieved by using $S2$ and $S5$. However, the energy consumption goal of $150J/s$ will be violated in such scenario as both mentioned sensors has lower energy consumption. Hence, in case of an infeasible solution the system may inform the user about the contradictory goals set for the system.

5.4 Scalability of SimCA

To demonstrate the scalability of SimCA we extend the UUV case by significantly increasing the number of possible actuation options (combinations of sensors). In particular, we consider now an UUV equipped with two sensor panels, one on the left side and one on the right side. Each panel is provided with 5 on-board sensors that monitor a surface equal to the surface monitored by the single panel in the original case. The panels simultaneously monitor the respective surface, hence, a combination of two sensors (one from each panel) is used at the same time. The sensors have characteristics similar to those in Table 1. Due to space constraints, we refer to the project website for a detailed overview of the parameters of sensor combinations (energy consumption, scanning speed, and accuracy). The task of SimCA is to choose among 25 sensor combinations in order to satisfy the following goals:

R1: The underwater vehicle must examine $S = 210\ km$ of surface within a period of $t = 10$ hours (i.e., the scanning speed $= S/t = 5.83\ m/s$).

R2: The amount of available energy E is limited to 5.3 MJ (i.e., mission energy consumption $= E/S = 147\ J/s$).

Note that the scanning speed specified for a combination of two sensors is double the value of the vehicle speed as both panels scan surface in parallel.

Figure 9 shows results of a scalability scenario with 2 sensor panels working in parallel. The sensor data, as in the previous experiments, is subject to random disturbances of small amplitude. In general, the system shows the same adaptation behavior (convergence to the goal value, adaptation to sensor parameters change, etc.) as in the case of a single sensor panel, e.g. the change of goals at $k = 100$ and 160 switches the sensor combination of the optimal solution. The 'Sensor Combination Used" plot shows that during operation only 6 of the 25 sensor combinations are used for this scenario. However, note that during Identification (k = 0 to 20) other sensor combinations are tested as well.

As SimCA has the scalability properties of simplex, we can conclude that increasing the number of on-board sensors will not change the adaptation outcomes.

5.5 Evaluation Scenario 2: TAS

To show the generality of SimCA, we evaluate the approach with a second case: the TAS exemplar [27]. TAS is a service-oriented application that provides remote health support to patients. The main goal of TAS is to track a patient's vital parameters in order to adapt the drug or drug doses when needed, and take appropriate actions in case of emergency. To satisfy this goal, TAS combines three types of services in a workflow, shown on Figure 10.

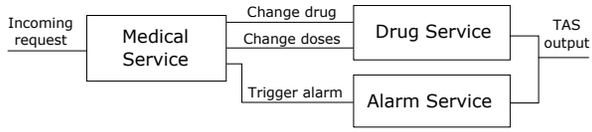

Figure 10: TAS workflow.

For service-based systems such as TAS, the functionality of each service can be implemented by multiple providers that offer services with different quality properties: reliability, performance, and cost. The system design assumes that these properties can be quantified and measured. E.g., reliability is measured as a percentage of service failures, while performance is measured as the service response time. At runtime, it is possible to pick any of the provided services.

We consider that five service providers offer the Medical

Table 2: Properties of all services used in TAS.

Service	Name	Fail.rate, %	Resp.time, time units	Cost, ¢
S1	Medical Service 1	0.06	22	9.8
S2	Medical Service 2	0.1	27	8.9
S3	Medical Service 3	0.15	31	9.3
S4	Medical Service 4	0.25	29	7.3
S5	Medical Service 5	0.05	20	11.9
AS1	Alarm Service 1	0.3	11	4.1
AS2	Alarm Service 2	0.4	9	2.5
AS3	Alarm Service 3	0.08	3	6.8
D	Drug Service	0.12	1	0.1
	Requirements	**min**	**30**	**9**

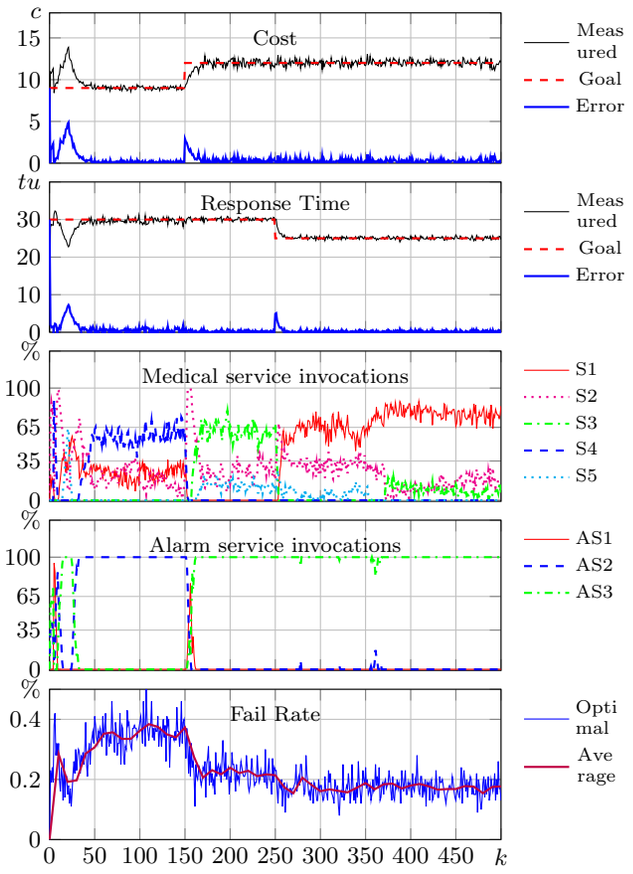

Figure 11: SimCA on TAS scenario.

Service, three providers offer the Alarm Service and only one provider offers the Drug Service. Table 2 shows example properties of available services based on data from [28].

The properties of the whole TAS system depend on the choice of concrete service providers that process user requests. For example, invoking $S1$ and $AS1$ will lead to the failure rate $TAS_{FR} = S1_{FR} + AS1_{FR} = 0.36\%$, while invoking $MS2$ and D will lead to the failure rate $TAS_{FR} = S2_{FR} + D_{FR} = 0.22\%$ The system requirements are the following:

R1. The average cost for invoking TAS service is set to 9¢

R2. The expected average response time is 30 time units

R3. Subject to R1 and R2, the failure rate of TAS should be minimized.

Unlike the UUV case, the TAS is expected to run continuously. The requirements R1-R3 and the properties of the services may change at runtime and the system should adapt accordingly. The adaptation task is to decide, for each request with a patient's vital parameters, which combination of services to select such that the requirements are satisfied.

The TAS case is realized based on the TAS exemplar [27]. The results of SimCA applied to a TAS scenario are shown in Figure 11. The adaptation works as intended: system outputs follow the goal changes at $k = 150$ and 250, the optimal solution is changed when $S5$ stops responding at $k = 370$. As services in TAS fail randomly, the optimal value of fail rate oscillates. However, on average (see the purple line on the "Fail Rate" plot) it decreases from ≈0.37% to ≈0.22% when more resources are available to the application at $k = 150$. Note that SimCA manages to keep the failure rate low with a more strict demand of response time from $k = 250$ onwards.

The TAS case confirms the results obtained with the UUV study. It supports the generality of the approach by showing that SimCA is effective in adapting software systems independent of concrete goals or software components that take part in the adaptation.

5.6 Comparison of SimCA with AMOCS

Recently, [19] proposed an interesting approach for Automated Multi-Objective Control of Self-adaptive software design (AMOCS in short). AMOCS automatically constructs a system of cascaded controllers to deal with multiple goals. Unfortunately, no replication package was available to quantitatively compare AMOCS with SimCA. Therefore, we perform a qualitative comparison based on the reported results.

Compared to SimCA, AMOCS has the advantage that it does not require the extra optimization step with simplex. Furthermore, the approach supports two schemes for ordering goals: user-defined prioritization of goals, and automatic ordering where the controller automatically ranks goals based on available actuators to achieve as many goals as possible.

However, SimCA's formal guarantees go significantly beyond these of AMOCS reported in [19]:

- SimCA provides guarantees for robustness and steady-state error; AMOCS' "robustness analysis is system-dependent and unsuitable for an automated control strategy" [19], while steady-state error is not analyzed.
- Simplex provides optimal solutions, while AMOCS "uses systematic or randomized exploration of solution subspace which introduces an approximation of the optimal solution" [19].
- AMOCS cannot provide guarantees for time-dependent goals (such as the energy consumption goal in the UUV case) because it does not take into account resources spent on learning.
- "Short overshoots are expected in AMOCS" [19]; SimCA can avoid overshoots.

SimCA also provides important engineering support not covered by AMOCS:

- SimCA supports trading settling time with robustness (pole placement). In AMOCS, settling time can just be set based on domain characteristics.
- Sampling and model learning is automated in SimCA. AMOCS requires specialized knowledge and extra efforts for this (e.g., quasi-Montecarlo and grid sampling [19]).
- AMOCS "requires the number of knobs to be greater than or equal to the number of goals" [19]. Finding enough knobs may not be trivial or even be artificial (consider for example the TAS case).

In conclusion, SimCA contributes with a novel control-based approach for satisfying multiple goals that significantly improves over AMOCS, both in terms of the guarantees it provides and the engineering support it offers.

5.7 Threats to Validity

SimCA can handle one class of adaptation problems (satisfying multiple goals, while optimizing one additional goal), but this class of problems applies to a significant number of

systems, as illustrated with the cases used in this paper and for example also those used in [19]. Supporting other types of adaptation goals is subject of future work.

We used standard controller guarantees. In Section 4.1, we provide an initial mapping of the controller guarantees to software quality guarantees. However, additional research is required both to refine and extend this mapping and to understand the coverage of the guarantees that can be provided with the standard controller properties. We did not test the impact of δ on the model quality/guarantees in different operating environments; this could be a part of future work.

Regarding the scope of applicability of SimCA. First, the approach is not applicable to systems undergoing drastic changes in their behavior at runtime as continuous re-identification is very costly. Second, SimCA requires that goals can be quantified as a setpoint, which may not be easy for all properties; an example is security. Third, in the experimental setting we have used only some types of disturbances (e.g., sensor failures and noise). Understanding the impact of other disturbance on the adaptation properties of SimCA requires additional evaluation. We want to highlight that in the current state of the research in control-based software adaptation, it is difficult to outline precise criteria that delineate which systems can/cannot be supported by SimCA (and other approaches such as AMOCS). The study and empirical evaluation of new approaches can contribute to build up this knowledge.

We evaluated SimCA in two domains, focusing on adaption for a typical set of stakeholder requirements (resource usage, performance, reliability, cost). While these systems can be considered as representative instances of a significant family of contemporary software systems, additional evaluation is required to validate SimCA for other types of systems.

Finally, we evaluated SimCA for simulated systems. This is inline with the evaluation conducted by others such as [29, 28, 30]. However, further evaluation of SimCA is required to confirm the evaluation results in real deployed systems.

6. RELATED WORK

The problem of handling multiple goals in self-adaptation is obviously not new. Most of the existing research, including those in *architecture-based* adaptation and *linear programming*, solve this problem by introducing an optimization task that trades off the conflicting qualities looking for an optimal solution. These solution uses many different techniques such as preemption [31] to give preference to more time-critical adaptation requirements, utility functions [32, 33, 34] to optimize component/service selection based on weights of QoS criteria, estimates of performance models [35] to select services with optimal response time, linear programming [36] to deal with different operating environments and conflicting QoS requirements, or combine linear programming with local search [37] to find configurations with minimum cost, and hybrid approaches [38] that first decompose end-to-end QoS constraints into local QoS constraints and then perform local selections. Most of these approaches do not provide the broad set of formal guarantees provided by SimCA. Furthermore, the computational costs of most of the proposed solutions grow exponentially with the size of the problem. SimCA can rely on the scalability of simplex as shown in the evaluation.

An advanced example of an architecture-based solution is the QoSMOS framework [27], which also uses the TAS exemplar for evaluation. QoSMOS employs runtime quantitative verification to provide formal guarantees for satisfying multiple QoS goals, while optimizing cost. The control-theoretical guarantees provided by SimCA are out of focus in [28]. However, the main difference is that QoSMOS requires a set of tools that need to be glued together to realize the feedback loop, while with SimCA, the feedback loop is relatively straightforward and derived automatically.

Besides [19], another approach that trades-off different qualities and provides adaptation guarantees is presented in [16] casting a discrete time Markov model for reliability requirements to a dynamic system. The synthesized controller trades reliability for cost by solving an optimization problem. In [39], compared an initial version of SimCA with ActivFORMS [40], a formally founded architecture-based approach for self-adaptation. The evaluation underlines the pros and cons of both approaches in terms of robustness to disturbances and types the guarantees that can be provided.

Simplex is a proven and practical optimization method [22]. Several variants have been developed for specific classes of problems, e.g. [25]. Simplex and its variants do not require an exponentially growing number of iterations when the problem space increases and do not depend on the structure of equations. Today, *Simplex* remains a very popular optimization method that is used in a wide variety of domains; recent examples are [34] where the method was used to support exploring optimal controller parameters for complex industrial systems, and [41] where simplex was used to support the exploration of the large design space of a cyber-physical system architecture. Another widespread method to solve optimization tasks is called the interior point method. The choice of simplex over the interior point method in SimCA was based on the scope of the problem: the interior point method is faster but only for specific (usually very large) problems [42].

[43] compared control-theoretical and optimization approaches, showing that continuous controller feedback offers higher potential to meet system goals under constantly changing loads, and provides better settling time and less overshooting. Contrary to using either a control-theoretical or an optimization approach, SimCA integrates the simplex optimization method with a control-theoretic method (enhanced version of PBM) to endow software systems with the self-adaptive capabilities, exploiting the best of both worlds.

7. CONCLUSIONS

In this paper we presented SimCA: a new approach that allows building self-adaptive software systems that satisfy multiple goals, while reaching optimality with respect to an extra goal. In addition, SimCA achieves robustness to environmental disturbances and measurement inaccuracy, and provides guarantees for the adaptation results. The effectiveness of SimCA was formally evaluated and demonstrated on two cases with strict requirements.

SimCA contributes towards the application of formal techniques to adapt the behavior of software systems, which is one key approach for providing guarantees. At the same time, by automatically building a control mechanism that adapts the software, SimCA does not require a strong mathematical background from a designer, which is a key aspect to pave the way for software engineers to use the approach in practice.

In future research, we plan to study the impact of δ on the model and extend SimCA to handle on the fly adding and removing goals. Our long term goal is to study and develop reusable control-based adaptation solutions that provide assurances for different types of goals.

8. REFERENCES

[1] B. H. Cheng *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*. LNCS vol. 5525, Springer, 2009.

[2] R. de Lemos *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*. LNCS, Springer, 2012.

[3] G. Tamura, N. Villegas, H. Muller, J. P. Sousa, B. Becker, G. Karsai, S. Mankovskii, M. Pezze, W. Schaefer, L. Tahvildari, and K. Wong, "Towards practical runtime verification and validation of self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*, ser. Lecture Notes in Computer Science. Springer, 2013, vol. 7475.

[4] J. Camara, R. de Lemos, C. Ghezzi, and A. Lopes, *Assurances for Self-Adaptive Systems: Principles, Models, and Techniques*, ser. Lecture Notes in Computer Science, Vol. 7740. Springer, 2013.

[5] B. H. Cheng, K. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. Muller, P. Pelliccione, A. Perini, N. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. Villegas, "Using models at runtime to address assurance for self-adaptive systems," in *Models@run.time*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8378, pp. 101–136.

[6] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad, "A survey of formal methods in self-adaptive systems," in *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, ser. C3S2E '12. New York, NY, USA: ACM, 2012, pp. 67–79. [Online]. Available: http://doi.acm.org/10.1145/2347583.2347592

[7] P. Oreizy, N. Medvidovic, and R. Taylor, "Architecture-based runtime software evolution," in *ICSE*, 1998.

[8] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," *FOSE*, 2007.

[9] D. Weyns, S. Malek, and J. Andersson, "Forms: Unifying reference model for formal specification of distributed self-adaptive systems," *ACM Trans. Auton. Adapt. Syst.*, 2012.

[10] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, Sep. 2012. [Online]. Available: http://doi.acm.org/10.1145/2330667.2330686

[11] D. Weyns, N. Bencomo, R. Calinescu, J. Cámara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli, "Perpetual Assurances in Self-Adaptive Systems," *Software Engineering for Self-Adaptive Systems: Assurances (Dagstuhl Seminar 13511)*, 2014. [Online]. Available: http://homepage.lnu.se/staff/daweaa/papers/2015Dagstuhl.pdf

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

[13] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, V. Papadopoulos, Alessandro, S. Ray, M. Sharifloo, Amir, S. Shevtsov, M. Ujma, and T. Vogel, "Software Engineering Meets Control Theory," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Firenze, Italy, May 2015. [Online]. Available: https://hal.inria.fr/hal-01119461

[14] R. de Lemos, D. Garlan, and H. Giese, "Software Engineering for Self-Adaptive Systems: Assurances, Dagstuhl Seminar 13511," 2013.

[15] Y. Brun *et al.*, "Engineering self-adaptive systems through feedback loops," ser. Lecture Notes in Computer Science, vol. 5525, 2009.

[16] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, Nov 2011, pp. 283–292.

[17] A. Filieri, H. Hoffmann, and M. Maggio, "Automated design of self-adaptive software with control-theoretical formal guarantees," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 299–310. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568272

[18] G. B. Dantzig, *Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation*. New York: Wiley, 1951, ch. XXI.

[19] A. Filieri, H. Hoffmann, and M. Maggio, "Automated multi-objective control for self-adaptive software design," in *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2015.

[20] M. Seto, L. Paull, and S. Saeedi, "Introduction to autonomy for marine robots," in *Marine Robot Autonomy*, M. L. Seto, Ed. Springer New York, 2013, pp. 1–46.

[21] M. Maggio and H. Hoffmann, "Arpe: A tool to build equation models of computing systems," in *Presented as part of the 8th International Workshop on Feedback Computing*. San Jose, CA: USENIX, 2013.

[22] G. B. Dantzig and M. N. Thapa, *Linear Programming 1: Introduction*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997.

[23] G. B. Dantzig and M. Thapa, *Linear Programming 2: Theory and extensions*, ser. Springer series in operations research. New York: Springer, 2003.

[24] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press, 1988.

[25] V. Klee and G. J. Minty, "How good is the simplex algorithm," 1972.

[26] T. S. Ferguson, "Linear Programming: A Concise Introduction."

[27] D. Weyns and R. Calinescu, "Tele assistance: A self-adaptive service-based system exemplar," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Firenze, Italy, 2015. [Online]. Available:

homepage.lnu.se/staff/daweaa/papers/2015SEAMS.pdf

[28] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 387–409, May 2011.

[29] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *International Conference on Software Engineering*, 2009.

[30] R. Calinescu, S. Gerasimou, and A. Banks, *Self-adaptive Software with Decentralised Control Loops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 235–251. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-46675-9_16

[31] R. Raheja, S.-W. Cheng, D. Garlan, and B. Schmerl, "Improving architecture-based self-adaptation using preemption," in *Proceedings of the First International Conference on Self-organizing Architectures*, ser. SOAR'09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 21–37. [Online]. Available: http://dl.acm.org/citation.cfm?id=1880569.1880572

[32] S.-W. Cheng, D. Garlan, and B. Schmerl, "Architecture-based self-adaptation in the presence of multiple objectives," in *Proceedings of the 2006 International Workshop on Self-adaptation and Self-managing Systems*, ser. SEAMS '06. New York, NY, USA: ACM, 2006, pp. 2–8. [Online]. Available: http://doi.acm.org/10.1145/1137677.1137679

[33] Q. Liang, X. Wu, and H. C. Lau, "Optimizing service systems based on application-level qos," *Services Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 108–121, April 2009.

[34] H. Chen and J. Xu, "Exploring optimal controller parameters for complex industrial systems," in *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015 IEEE International Conference on*, 2015.

[35] C. Ghezzi, V. Panzica La Manna, A. Motta, and G. Tamburrelli, "Performance-driven dynamic service selection," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 3, 2015.

[36] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "Qos-driven runtime adaptation of service oriented architectures," in *European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2009.

[37] D. Ardagna, G. Gibilisco, M. Ciavotta, and A. Lavrentev, "A multi-model optimization framework for the model driven design of cloud applications," in *Search-Based Software Engineering*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8636.

[38] S. X. Sun and J. Zhao, "A decomposition-based approach for service composition with global qos guarantees," *Information Sciences*, vol. 199, 2012.

[39] S. Shevtsov, M. U. Iftikhar, and D. Weyns, "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*, ser. CTSE 2015. New York, NY, USA: ACM, 2015, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/2804337.2804338

[40] M. U. Iftikhar and D. Weyns, "Activforms: Active formal models for self-adaptation," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS 2014. New York, NY, USA: ACM, 2014, pp. 125–134. [Online]. Available: http://doi.acm.org/10.1145/2593929.2593944

[41] J. Finn, P. Nuzzo, and A. Sangiovanni-Vincentelli, "A mixed discrete-continuous optimization scheme for cyber-physical system architecture exploration," in *IEEE/ACM International Conference on Computer-Aided Design*, 2015.

[42] M. H. Wright, "The interior-point revolution in optimization: history, recent developments, and lasting consequences," *Bull. Amer. Math. Soc. (N.S*, vol. 42, pp. 39–56, 2005.

[43] Y. Diao, C. W. Wu, J. Hellerstein, A. Storm, M. Surenda, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chu, and J. Colaco, "Comparative studies of load balancing with control and optimization techniques," in *American Control Conference, 2005. Proceedings of the 2005*, June 2005, pp. 1484–1490 vol. 2.