Handling New and Changing Requirements with Guarantees in Self-Adaptive Systems using SimCA*

Stepan Shevtsov
Department of Computer Science,
Linnaeus University, Sweden
Email: stepan.shevtsov@lnu.se

Danny Weyns
Department of Computer Science,
KU Leuven, Belgium
Linnaeus University, Sweden
Email: danny.weyns@kuleuven.be

Martina Maggio
Department of Automatic Control
Lund University
Email: martina@control.lth.se

Abstract—**Self-adaptation provides a principled way to deal with change during operation. As more systems with strict goals require self-adaptation, the need for guarantees in self-adaptive systems is becoming a high-priority concern. Designing adaptive software using principles from control theory has been identified as one of the approaches to provide guarantees. However, current solutions can only handle pre-specified requirements either in the form of setpoint values (S-reqs) or values to be optimized (O-reqs). This paper presents SimCA* that makes two contributions to control-based self-adaptation: (a) it allows the user to specify a third type of requirement that keeps a value above/below a threshold (T-reqs); and (b) it can deal with requirement sets that change at runtime (i.e., requirements can be adjusted, activated, and deactivated on the fly). SimCA* offers robustness to disturbances and provides adaptation guarantees. We evaluate SimCA* for two systems with strict goals from different domains: an underwater vehicle system used for oceanic surveillance, and a tele-assistance system for health care support. The test results demonstrate that SimCA* can deal with the three types of requirements (STO-reqs) operating under various types of dynamics and the set of requirements can be changed on the fly.**

I. Introduction

Software applications need, more than ever, being able to deal with change [1], [2]. The need for continuous availability of software requires developers to consider change as part of the development process. Software is expected to deal seamlessly with different types of change, such as varying resources, sudden failures, and changes in the operating environment. Often, these changing conditions are difficult to predict at design time, requiring software to execute with incomplete knowledge and face changing requirements during operation [3], [4]. Consequently, software engineers are developing new techniques to handle change at runtime without incurring penalties and downtime, which is commonly referred to as self-adaptation [5], [6], [7].

Today, many of the software systems need to comply with strict goals, hence requiring guarantees for adaptation, such as robustness to disturbances, system stability and others [8], [9], [10]. Control theory has been identified as a promising approach to design adaptation solutions with formal guarantees [11], [12], [13], [14]. However, most of the approaches using control theory to design self-adaptive systems were developed to solve specific problems for a particular domain. We, on the other hand, are interested in creating a reusable approach that can satisfy different stakeholder requirements and provide adaptation guarantees under changing operating conditions.

A number of reusable approaches have already been proposed, but they are limited to satisfying certain types of stakeholders requirements. For example, the approach described in [15] can satisfy only one requirement at a time, while other approaches such as [16], [17] can satisfy multiple requirements either in the form of setpoint values (S-reqs) or values to be optimized (O-reqs). However, many software systems today need to address a third type of requirement: a threshold requirement that keeps a value above/below a threshold (T-reqs). A typical example is limiting the response time of a web server. Approaches such as described in [18], [19], [20] solve this problem either by optimizing the response time (O-req) or by defining a setpoint for response time that the controller should guarantee (S-req), when the actual requirement is to keep response time lower than a certain threshold. The idea of T-reqs is similar to the recently explored notion of "constraint" from control theory, see for example [21].

Besides a lack of first-class support for T-reqs, existing approaches also provide limited support for changing the set of requirements during operation, which requires on the fly adjusting, activation and deactivation of requirements. Changing requirements are important in practice, e.g., to deal with drastic changes in the system or its environment that may require the system to change from one set of requirements to another.

In this paper, we use control theory to simultaneously deal with S-reqs, T-reqs, and O-reqs (we refer to a combination of these requirements as *STO-reqs*) and enable the system to change the set of requirements by adjusting/activating/deactivating requirements at runtime. In particular, we solve a typical adaptation problem inherent to systems with strict goals, that is: *to satisfy multiple stakeholder requirements (STO-reqs) that may change at runtime, in the presence of environmental disturbances and inaccurate measurements, and provide formal guarantees on the adaptation results.*

To deal with this adaptation problem, we developed SimCA*(Simplex Control Adaptation[1]). SimCA* leverages

[1]The "*" symbol refers to the ability of the approach to handle different types of requirements that can dynamically change.

upon an earlier version that we developed, SimCA [17], that can satisfy S-/O-reqs, but is not able to solve the adaptation problem discussed above. Hence, SimCA* is an automated control-theoretic approach to build self-adaptive software systems that satisfy multiple, possibly conflicting STO-reqs, achieves robustness to environmental disturbances and measurement inaccuracy, and provides a broad set of control-theoretical adaptation guarantees.

The evaluation of SimCA* is conducted with two cases from different domains: an Unmanned Underwater Vehicle (UUV) system performing surveillance missions, and a service-based system for health care. Both systems have to operate under disturbances and must self-adapt to guarantee the satisfaction of STO-reqs at runtime, as well as to deal with changes in the requirements.

The remainder of the paper is structured as follows. Section II positions SimCA* in the state-of-the-art control-theoretical approaches for building self-adaptive systems that satisfy multiple requirements. Section III elaborates on the adaptation problem that we address in this paper and illustrates it with an experimental scenario. Section IV presents SimCA*. The formal guarantees provided by SimCA* are evaluated in Section V. In Section VI, SimCA* is empirically evaluated using multiple scenarios of two cases. Finally, conclusions and directions for future research are presented in Section VII.

II. State of the art overview

There is a vast body of research available that applies principles from control theory to adapt computing systems, see e.g. [22], [23]. However, most of the suggested approaches tend to solve specific problems within a certain domain. Creating a generally applicable approach to build self-adaptive software that satisfies different stakeholder requirements and provides adaptation guarantees has been a topic of research for a couple of years [13]. One of the first attempts to create such an approach is the Push-Button Methodology (PBM) [15]. PBM automatically creates a linear model of software and a controller that adapts the software to meat a non-functional requirements specified by stakeholder. The main limitation of basic PBM is that it can only satisfy one requirement at a time. In recent work [16], the authors of PBM proposed a new approach for Automated Multi-Objective Control of Self-adaptive software (AMOCS in short). AMOCS automatically constructs a system of cascaded controllers to deal with multiple S-reqs and an O-req. The approach supports goal prioritization. Despite the advantages, AMOCS has difficulties with addressing O-reqs as the approach may produce suboptimal solutions [16] and it is lacking some of the guarantees, e.g. the absence of overshooting. Finally, in [17] we introduced an initial version of SimCA, an approach that builds self-adaptive software able to satisfy multiple S-reqs, while being optimal according to a single O-req. The approach makes the system robust to disturbances and provides a broad set of adaptation guarantees.

However, none of the existing automated approaches can simultaneously deal with a typical set of stakeholder requirements (STO-reqs). The main reason is that control theoretic solutions usually work with goals specified as setpoints (S-reqs). Furthermore, existing approaches cannot handle activation and deactivation of requirements during system operation. This may be too restrictive for practical software system that are subject to continuously change. SimCA* on the other hand can (besides S/O-reqs) satisfy T-reqs that are not typical for control theoretical solutions and deal with adjusting/activation/deactivation of requirements during operation.

III. Problem Definition

Based on the analysis of the state-of-the-art, we identified the following problem definition:

> *To guarantee the satisfaction of multiple STO-reqs and deal with adjustment/activation/deactivation of requirements at runtime, regardless of fluctuations in the system parameters, measurement accuracies, and environmental dynamics that are difficult to predict.*

Compared to state of the art approaches, two key challenges must be addressed to deal with this problem. First, the solution must incorporate a mechanism to guarantee the satisfaction of T-reqs. This is not trivial as T-reqs are not a typical type of requirement applied in control theory. Second, the solution needs a mechanism to adapt the adaptation logic on the fly in order to address changing requirements, i.e., adjusting/activation/deactivation of requirements. SimCA* described in Section IV addresses these challenges.

Problem example: UUV System

We describe a UUV system (based on [24]) that we use as one of the cases to evaluate SimCA* in Section VI and to illustrate the adaptation problem we aim to solve. 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 a maritim 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 goals, i.e., vehicles are expensive equipment that should work accurately and productively, and they should not impact the ocean area or get 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 I). A scan is performed every second.

In normal operating mode, the UUV system has the following initial requirements:

- $R1$: A segment of surface over a distance of $S \geq 100$ km should be examined within time t = 10 hours;
- $R2$: To perform the mission, a given amount of energy E = 5.4 MJ is available;

TABLE I
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

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

$R1$ is a T-req, $R2$ is a S-req, while $R3$ is an O-req. We use $M1$ to refer to normal operating mode, i.e. the UUV must satisfy the three requirements simultaneously: $M1 = \{R1, R2, R3\}$. In other words, in normal operation mode, the UUV should examine as much surface as possible using all the available energy, while ensuring maximum accuracy. To grasp the difference between T-reqs and S-reqs, if the task of the UUV would be to scan exactly S km in t hours ($R1$) using as little energy as possible of the available E = 5.4 MJ ($R2$), while ensuring maximum accuracy ($R3$), $R1$ would become an S-req and $R2$ a T-req. 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.

In addition to normal operating model, the adaptation should also deal with two additional operating modes. First, when a UUV experiences a sudden energy leak, it must switch to a new operating mode that minimizes the vehicle energy consumption instead of maximizing the measurement accuracy. In this case the UUV will switch to a mode $M2 = \{R1, R2^*\}$, where $R2$ (S-req) changes to $R2^*$ (O-req) defined as:

- $R2^*$: The energy consumption should be minimized;

Second, the vehicle may enter a deep water zone, where sensors fail to produce accurate measurements with a certain rate. The UUV should then switch to a mode $M3 = \{R1, R2, R3, R4\}$, where a requirement $R4$ needs to be activated at runtime, defined as:

- $R4$: The average failure rate should be $F \leq 0.02\%$;

Switching modes and changing requirements is critical to the success of the surveillance mission. The adaptation task is not trivial because the system is affected by different disturbances such as fluctuations in the expected behavior of the UUV sensors, inaccurate measurements, sensor problems and failures, and noise in the communication channel.

In summary, to realize its mission, the UUV needs to self-adapt in order to continuously address STO-reqs, cope with different operating modes by activating/deactivating requirements at runtime, and reject different types of disturbances. These requirements needs to be guaranteed.

IV. SIMPLEX CONTROL ADAPTATION* - SIMCA*

In this section we provide an overview of SimCA*. In particular, Section IV-A lists the element required for SimCA* to build a self-adaptive system, Sections IV-B - IV-E describe different phases of SimCA*, and Section IV-F shows how SimCA* deals with changing the set of requirements.

A. Required Elements to Apply SimCA*

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

1) A working prototype of the software.
2) A set of STO-reqs to be satisfied.
3) Tunable parameters (actuators) that can be used to adapt the running system to address the requirements.
4) Adaptation sensors to measure the effect of the adaptation on the system.
5) Monitoring mechanisms that notify the system about changing conditions that lead to requirement changes.

As for the second element, it should be possible to quantify the requirements, i.e. transform the system requirements (STO-reqs) into corresponding quantifiable goals (STO-goals). For some requirements this transformation is straightforward (e.g., a requirement of keeping average response time at 3 ms is transformed to an S-goal = 3 ms), while for other requirements with time-dependent constraints the quantifiable goals needs to satisfy these requirements over time (e.g., a requirement of using 120 units of a resource in 60 sec is transformed into an S-goal = 2 units/sec).

With the elements listed above SimCA* is able to build a self-adaptive system that solves the adaptation problem formulated in Section III.[2] SimCA* works in four phases that are performed during system operation, see Figure 1.

- First, in the *Identification* phase, SimCA uses systematically sampled values of S- and T-goals to synthesize models that capture the dependency between different actuator values (in form of control signals that effect software) and the measured system outputs for these goals.
- Second, in the *Controller Synthesis* phase, SimCA constructs an appropriate set of controllers for the synthesized models, where each controller is responsible for one S- or T-goal.
- Third, in the *Goal Transformation* phase, the T-goals are transformed into controller goals (C-goals) using simplex. For a T-goal that needs to keep a value below a threshold the C-goal represents the lowest possible value that satisfies all other goals, for a T-goal that needs to keep a value above a threshold the C-goal represents the highest possible value that satisfies other goals.
- Fourth, in the *Operation* phase, the controllers carry out control using S- and C-goals as the values to be achieved by the system. Then, the outcome of multiple controllers is combined using the simplex method that takes into account the O-goals and optimally drives the outputs of the system towards the set goals.

Compared to the initial version of SimCA [17] , SimCA* includes a new Goal Transformation phase and the necessary mechanisms to support changing system requirements by

[2]We discuss the scope of applicability of SimCA* in Section VI-G.

activating/deactivating goals. We start with explaining how SimCA* deals with STO-reqs. Then we explain how the approach deals with changing requirements.

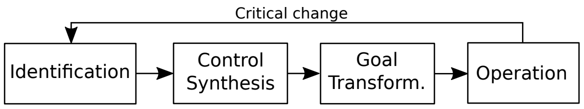

Fig. 1. Phases of SimCA*

B. Phase I. Identification

During the Identification Phase, SimCA* synthesizes a set of linear models that capture the dependency between different actuator values (in form of control signals that effect software) and the measured system outputs [17]. Each model $\mathcal{M}_i$ is responsible for one S- or T-goal, referred to as s_i. As optimization tasks are not solved in the first phases of SimCA*, we take into account only the threshold values of T-goals (and not the values above/below the threshold) during Identification and Control Synthesis. Model $\mathcal{M}_i$ is built by systematically feeding sampled values of goal s_i in the form of a control signal u_i to the system and measuring its effect on the output O_i:

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

Coefficient α_i captures the dependency between the control signal u_i at the previous time instance $k-1$ and its effect on the measured output O_i of S- or T-goal s_i at the current time k. The time instances between measurements during identification can be chosen by the system engineer, hence influencing the model quality [17].

Model $\mathcal{M}_i$ describes the system behavior ignoring small disturbances and sudden system changes. As small disturbances are difficult to predict at design time and therefore cannot be factored in the model construction, they will be dealt with by using feedback from the running system. Sudden changes will re-trigger the identification phase when necessary.

As exploring different actuator values during Identification may violate requirements with time-dependent constraints, the system measures the time Δt_i and resources ΔR_i (e.g., energy) spent for Identification, subtracts this amount from the available time t_i and resources R_i respectively and automatically adjusts the goal s_i accordingly:

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

A similar update of s_i is applied after any goal is changed (where Δt_i and ΔR_i represent time and resources spent before the goal change) as the system under control either over/under-consumes resources during the transition to a new goal.

C. Phase II. Controller Synthesis

In the Controller Synthesis Phase, SimCA* constructs a set of controllers for the synthesized models; each controller C_i is responsible for one S- or T-goal (s_i). A controller C_i has one tunable parameter, called *pole* denoted with p_i. The pole is chosen by the system designer and allows to trade-off controller responsiveness to change and the amount of disturbance it can withstand [17].

In SimCA*, we use the following controller:

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

The synthesized controller C_i calculates the control signal $u_i(k)$ at the current time step k depending on the previous value of control signal $u_i(k-1)$, model adjustment coefficient α_i, controller pole p_i and error $e_i(k-1)$, with e_i being the difference between S- or T-goal s_i and the measured output O_i.

The controller C_i also handles inaccuracies in the model $\mathcal{M}_i$. To that end, each controller incorporates: (1) a Kalman filter adapting the linear model at runtime; (2) a change point detection mechanism, which allows to react to unexpected critical changes in the system by triggering re-Identification [17].

D. Phase III. Goal Transformation

The Goal Transformation Phase transforms all T-goals into Controller goals (C-goals), see Figure 2; a C-goal represents a particular value of a corresponding T-goal. For example, a T-goal that should keep a value below a threshold will be transformed into a C-goal with a value that is equal to the lowest possible value of the goal below that threshold that satisfies all other requirements. As the values of the C-goals depend on other requirements and system parameters, we use simplex during Goal Transformation. This phase is skipped if there are no T-goals in the system.

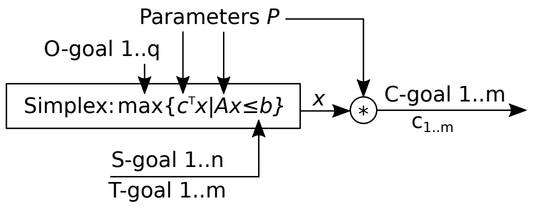

Fig. 2. Goal Transformation phase of SimCA*.

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

$$\max\{c^{\mathrm{T}}x \mid Ax \leq b; x \geq 0\} \qquad (2)$$

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 [25].

In the Goal Transformation phase of SimCA* each equation, except the last one, represents an S-goal or T-goal to be satisfied. Equalities are used for S-goals, while inequalities are used for T-goals. The last equation ensures that the system selects a valid solution by constraining the values that can be taken by elements of the vector x, e.g. $x \geq 0$. The values of S-/T-goals to be achieved replace constants b, whereas matrix A and vector c^{T} are substituted with the monitored parameters $\mathcal{P}(k)$ of the system (i.e., relevant parameters of system components that can be measured[3]). Note that vector c^{T} is replaced with parameters of the O-goals. The goal of simplex is to find a proper combination of variables (vector x) that satisfies all STO-goals. For details on how simplex solves the system of equations (2) we refer to the linear programming literature [25], [26], [27].

Knowing the vector x, each T-goal is transformed into C-goal c_i as follows: $c_i = \mathcal{P}_i(k) * x$. Note that controllers are not involved during the Goal Transformation phase and as such simplex will not change the control signals $u_i(k)$.

E. Phase IV. Operation

In the Operation Phase, the set of controllers effectively perform 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, see Figure 3. A simplex is dealing with the O-goals, only C-goals obtained during Goal Transformation and original S-goals are used in the Operation Phase.

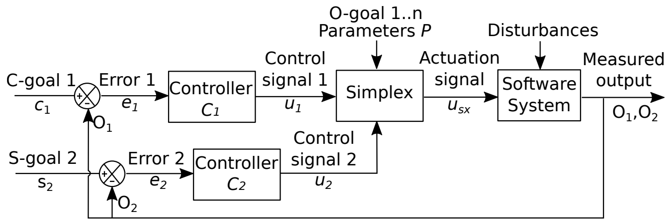

Fig. 3. Operation phase of SimCA* (illustrated for one S- and C-goal).

In particular, SimCA* collects all control signals $u_i(k)$ and the system parameters $\mathcal{P}(k)$ and passes these data to the simplex block. Similarly to the Goal Transformation phase, SimCA* solves the system of equations (2) in order to find a solution (actuation signal u_{sx}) that drives the system towards an output that satisfies all STO-goals. However, the system of equations (2) has now a slightly different structure. First of all, each equation, except the last one, now represents an S-goal or a C-goal to be satisfied. Then, only equalities are used to assure a seamless translation of control signals $u_i(k)$ to an actuation signal u_{sx}, which allows to sustain all the guarantees provided by controllers. Finally, the constants b in (2) are replaced by control signals $u_i(k)$ obtained from C_i, which allows to use all the advantages provided by controllers.

*F. Dealing with New Requirements in SimCA**

We now describe how SimCA* adapts the system when requirements are changed during system operation. To that end, we extended the initial SimCA* workflow shown in Figure 1 with additional components, see Figure 4.

Any change of system requirements during operation is monitored by the Requirement Monitor that triggers the corresponding adaptation components. Subsequently, we look at the activation of requirements, deactivation, and changing

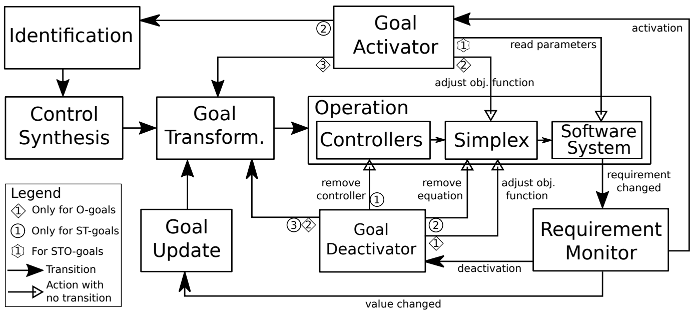

Fig. 4. Dealing with requirements changes in SimCA*. Numbers in circles/diamonds show the sequence of actions.

requirement types. Updates of requirements (goal updates) are already explained in the Identification Phase in Section IV-B.

To deal with *requirement activation*, the Goal Activator first transforms the requirement into a quantifiable goal (see Section IV-A) and reads the relevant parameters $\mathcal{P}$ related to that goal.[4] The next actions depend on the type of the requirement that is activated. If an O-req is activated, the Goal Activator inserts $\mathcal{P}$ into the objective function c^{T} of simplex, performs a Goal Transformation (Section IV-C) and proceeds to standard Operation. If an S-/T-req is activated, the Goal Activator performs an Identification for the new S-/T-goal. An advantage of SimCA* is that it does not require a complete re-identification of all goals when a requirement is activated, because each corresponding goal is managed by a separate model-controller pair. After Identification, SimCA* builds a controller for the new goal using Controller Synthesis, followed by a Goal Transformation, after which the system returns to standard operation.

For a *requirement deactivation*, the Goal Deactivator removes the according elements of the adaptation mechanism. Namely, when an S- or T-req is deactivated, the corresponding controller is removed together with the equation responsible for the goal being deactivated. When an O-req is deactivated, the corresponding variables are removed from the objective function c^{T} of simplex. Finally, the Goal Deactivator always triggers a Goal Transformation adapting the configuration of the control system to the new set of requirements, after which the system returns to standard operation.

SimCA* also supports changes of *requirement types* at runtime. To that end, SimCA* performs the following: (i) if an S-req is changed to a T-req (or vice versa), the corresponding equality is changed to inequality in the system of equations (2), followed by a Goal Transformation; (ii) if an S-/T-req is changed to an O-req, the parameters $\mathcal{P}$ relevant to this goal are copied from the corresponding equation into the objective function c^{T} of simplex. After that, the S-/T-req is deactivated according to the standard requirement deactivation procedure (see above); (iii) if an O-req is changed to an S-/T-req, the O-req is deactivated according to the standard requirement

[3]E.g. in the UUV scenario $\mathcal{P}(k)$ are the sensor parameters from Table I

[4]For example, if the fail rate goal of a UUV ($R4$) is activated, the Goal Activator reads failure rates of all UUV sensors from the specification.

deactivation procedure, while the new S-/T-req is activated according to the requirement activation procedure (see above).

V. Formal Evaluation of Guarantees

As in the original SimCA, a feature of SimCA* is that it provides a broad set of adaptation guarantees. The guarantees provided by controllers include, see Figure 5:

- Stability: the ability of an adaptation mechanism to converge to S- or C-goals (s_i/c_i);
- Absence of overshoot: the measured quality property does not exceed the goal s_i/c_i before reaching its stable area;
- Zero steady-state error: the measured quality property does not oscillate around goal s_i/c_i during steady state;
- Tuneable settling time: time it takes to bring a measured quality property close to its goal s_i/c_i;
- Tuneable robustness: the amount of perturbation the system can withstand while remaining in stable state.

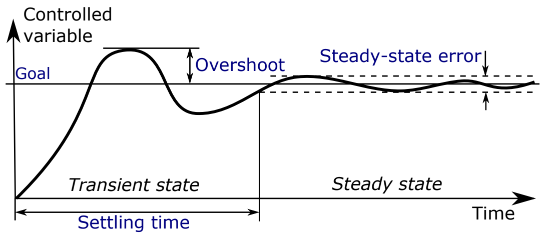

Fig. 5. Properties guaranteed by the controllers in SimCA*

Since simplex does not introduce any additional dynamics and works as a straight-forward translator of control signals into an actuation signal, we can formally analyze the following guarantees. 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), we recall the output equation of control system used in SimCA* [17]:

$$O_i(k) = s_i \cdot (1 - p_i^k) \tag{3}$$

During steady-state the time $k \to \infty$. As $p \in (0, 1)$, in this case $p^k \to 0$. Then, the steady-state error Δe is:

$$O_i(k \to \infty) = s_i \cdot (1 - p^k) = s_i; \quad \Delta e = s_i - O_i = 0$$

As for the settling time ($\bar{K}$) and robustness $\Delta(d)$, in [17] we show that by analyzing the control system, we get:

$$\bar{K} = \frac{\ln \Delta s_i}{\ln |p_i|} \qquad 0 < \Delta(d) < \frac{2}{1 - p_i} \tag{4}$$

In other words, lowering p_i leads to weaker disturbance rejection but faster responsiveness to change. Note that in the equations above s_i can be replaced with c_i without any effect on the guarantees as C-goals represent particular values (setpoints) to be achieved by the system similar as S-goals.

As inequalities are used only to transform T-goals to C-goals, while during operation we are only using the simplex method with equalities (see Section IV-E), it will not change the effect of control signal u_i on the output signal O_i. Hence, simplex will not introduce additional system dynamics and will not alter the guarantees mentioned above provided by the controllers. As for the change of requirements, we assume that it will not lead to an unfeasible solution. Under this assumption the guarantees will hold, because changing the number of controllers will not alter the structure of the control system.

The guarantees provided by controllers relate to the quality properties that are subject of adaptation. For example, overshooting on the energy consumption goal leads to an overconsumption of energy (more details are available in [17]).

Simplex provides the following guarantees:

- Optimality: achievement of O-goals without violating any of the S- or C-goals. Simplex was proven to always find an optimal solution to systems of equations used by SimCA*, such as the one presented in Section IV [25], [26].
- Scalability: small amount of extra time and effort required to solve problems of growing scale. For practical problems, simplex usually finds a solution in just a few iterations [28]. This also ensures that the overhead is low for requirement changes as only one extra simplex iteration is required.
- Detection of infeasible solution: ability to detect that the goal s_i/c_i is unreachable. When s_i/c_i is infeasible, SimCA* will converge to the nearest achievable value of s_i/c_i and alert the user.
- Detection of unbounded solution: the ability to detect that the objective function value seeks ∞ (or -∞). Unbounded solution occurs if values of u_{sx} in simplex can grow indefinitely without violating any constraint, i.e., when the system has contradicting requirements. SimCA* will alert the user about unbounded solutions.

Boundaries of Guarantees. First, 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 usually representative enough to provide the guarantees. Second, the guarantees are achieved under certain assumptions, e.g. the activation of requirement should not lead to an unfeasible solution (see discussion above). Third, the guarantees on time-dependent requirements depend on correct measuring the time and resources spent during identification and computing the adjustment of the corresponding goal. Fourth, the guarantees are provided after the controllers are built, i.e., control-theoretical guarantees apply only during the Operation phase. Fifth, SimCA* guarantees the STO-reqs regardless of possible dependencies between the goals, to the extent that the goals are feasible (otherwise, SimCA* will alert the user). Finally, in the current realization, SimCA* cannot provide guarantees when the system behavior/architecture is invasively changed.

VI. Experimental Evaluation

We empirically evaluate SimCA* with two cases. Section VI-A describes the experimental setting of the UUV case. Section VI-B shows the software adaptation performed by SimCA* with STO-reqs in different operating conditions. In Section VI-C we experimentally verify the guarantees and quality trade-offs provided by SimCA*. Section VI-D describes adaptation in an energy leak scenario when the amount and types of requirements change at runtime. In Section VI-E we perform experiments with a UUV when a new requirement is activated at runtime. The scalability of our approach is tested by adding a panel of sensors to the UUV in Section VI-F. The second case with Tele Assistance System is described and evaluated in the Appendix. Finally, Section VI-G discusses threats to validity. The experiments are performed on a Dell machine with a 2.7 GHz Core i7 processor and 16 GB 1600MHz DD3 RAM. All evaluation material is available at the project website.[5]

A. Experimental Setting UUV Case

We use the UUV system described in Section III 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 I. The actual data that is used by the adaptation mechanism at runtime is subject to a randomly distributed disturbance up to $\pm 10\%$ 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 and changes of system requirements.

The task of SimCA* is to exploit the available energy and set an appropriate scanning speed in order to examine as much surface as possible in the given time frame with maximum measurement accuracy. SimCA* achieves this task by calculating the value of the *actuation signal*, which represents the portion of time each sensor $\{S1, \dots, S5\}$ 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 case is $5.5 \cdot 10^6$.

The controller pole p_i is set to values between 0.6 and 0.9 which allows to reject errors/disturbances of high magnitude; the choice of concrete pole values is discussed in Section VI-C. 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]https://people.cs.kuleuven.be/danny.weyns/software/simplex/index.htm

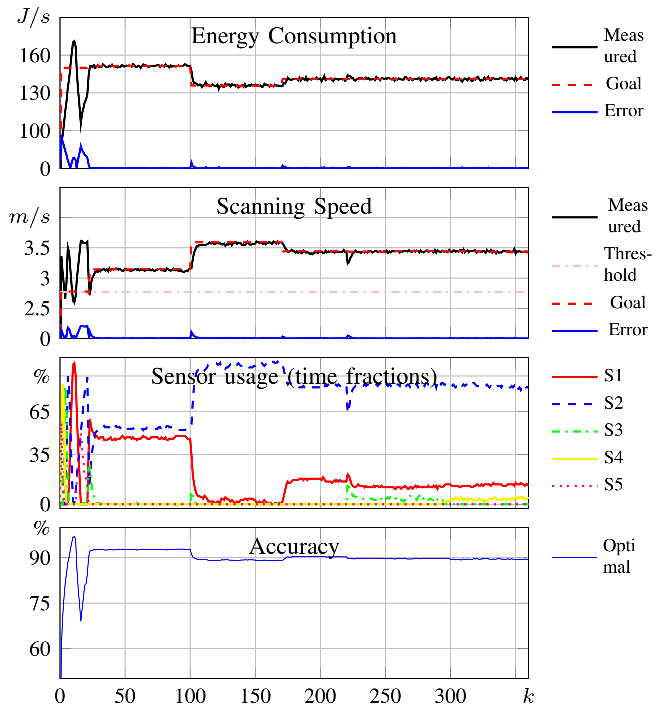

Fig. 6. UUV adaptation with STO-reqs, $p = 0.6$.

B. Adaptation with STO-reqs

Figure 6 shows the adaptation results of SimCA* on the UUV system configured according to Table I and requirements set according to UUV scenario (Section III); the controller pole p is set to 0.6. Adaptation starts with the Identification phase that is clearly visible when k is between 0 and 21. The Control Synthesis phase, immediately followed by the Goal Transformation phase, starts after the relationship between control signals $u_i(k)$ and system outputs $O_i(k)$ is identified (k = 22). For comparison, the "Scanning Speed" plot contains an additional line (see "Threshold" in Figure 6) depicting requirement $R1$ as if it was an S-req, i.e., it shows the scanning speed required to monitor exactly 100 km of surface within 10 hours using the available energy pool. As in our case $R1$ is a T-req, i.e., the UUV must scan $S \geq$ 100 km, during the Goal Transformation phase SimCA* finds a combination of sensors that allows to scan more surface using the same energy without loosing accuracy and updates the scanning speed goal from 2.7 to 3.2 m/s.

After the goal is updated, the Operation phase starts (from k equals 22 onwards). The two upper plots in Figure 6 show that during Operation the system is stable, i.e., the measured energy consumption and scanning speed follow their goals. To demonstrate how SimCA* deals with requirement changes, we adjust the available energy twice: at k = 100 from 5.4 to 5.0 MJ and at $k = 170$ from 5.0 to 5.1 MJ. Both adjustments trigger

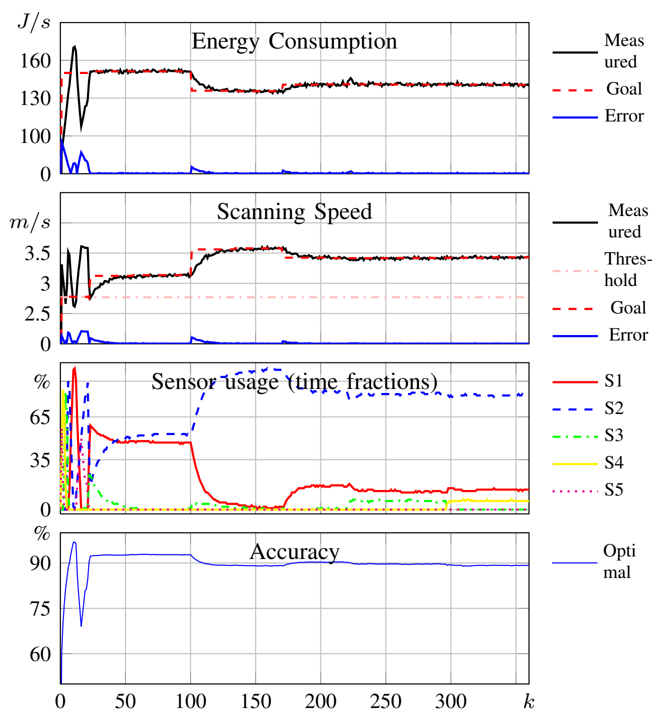

Fig. 7. UUV adaptation with STO-reqs, $p = 0.9$.

the Goal Transformation phase where the scanning speed is updated according to new conditions. Note that lowering the amount of available energy at $k = 100$ increases the scanned distance (speed changes from 3.2 to 3.55 m/s) but decreases the measurement accuracy.

Figure 6 also shows how SimCA* reacts to changes in sensor parameters and sensor failures. At $k = 220$, the energy consumption of sensor $S1$ increases from 170 to 190 J/s. To compensate for this overconsumption, a portion of time $S1$ was used is given to a less consuming sensor $S3$, see the "Sensor usage" plot. However, at $k = 290$, $S3$ stops working and is replaced by sensor $S4$, while the measured energy consumption and scanning speed of the UUV remain on the required level.

The experiment ends at $k = 360$, i.e. after 10 hours of time. Over a series of 50 experiments, we measured the following outcomes: the total distance scanned is 121.3 $\pm$ 0.32 km, the amount of consumed energy is 5.1 MJ $\pm$ 135 J, the measurement accuracy is 89.94 $\pm$ 0.04%.

C. Adaptation Guarantees and Trade-offs

In order to experimentally verify the guarantees and different property trade-offs provided by SimCA*, we perform the same experiment with STO-reqs using controllers with pole p = 0.9, see Figure 7. After 50 runs we got the following results: total distance scanned is 121 $\pm$ 0.28 km, the amount of consumed energy is 5.1 MJ $\pm$ 170 J, the measurement accuracy is 89.94 $\pm$ 0.04%.

From the Figures, it can be observed that in both scenarios the systems are stable, have a zero steady-state error and converge to their goals without overshooting. The results confirm that the system requirements are satisfied.

As described in Section V, adaptation with SimCA* is influenced by the values of the pole p. First of all, a smaller pole leads to a shorter settling time. In particular, 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 [29, p.85], the commonly used value of Δs is 0.02 (2%). Hence:

$$\bar{K}_{0.6} = \frac{\ln|0.02|}{\ln|0.6|} = 7.66 \quad \bar{K}_{0.9} = \frac{\ln|0.02|}{\ln|0.9|} = 37.3$$

These values show the amount of adaptation steps required to obtain a change of amplitude 1 in the measured value of a goal. For example, this guarantee can be observed at k = 100 on the "Scanning Speed" plot of both figures where the speed is required to change from 3.14 to 3.58 m/s (change of amplitude 0.44). Then, $\bar{K}_{0.6} = 7.66 * 0.44 = 3.4$ steps and $\bar{K}_{0.9} = 37.3 * 0.44 = 16.4$ steps. These values explain why the measured scanning speed makes almost a vertical jump at k = 100 in Figure 6, while in Figure 7 it takes 17 adaptation steps to converges to a target value.

By comparing the experiment outcomes obtained throughout 50 runs, it can be concluded that a smaller pole leads both to a bigger scanned distance and a smaller error in the energy consumption with the same scanning accuracy. This property of SimCA* can be explained by the fact that a higher settling time makes the system waste more resources in a transient phase.

However, lowering the pole is not always a better option as it leads to weaker disturbance rejection. Due to a small noise amplitude, in the tested scenario both controller successfully rejected disturbances. This may not be the case in real operating conditions, where a UUV is influenced by underwater streams, pressure, etc. Besides, a smaller pole makes the adaptation mechanism react faster not only to goal changes but to disturbances as well. This property can be observed, for example, by comparing the usage curve of sensor $S2$. In Figure 7 it is smoother and has a much lower spike at k = 220 than in Figure 7. In this case slower reaction may be a benefit as it allows to switch less between different sensor combinations.

D. Requirement Change Scenario: Energy Leak

Figure 8 presents the adaptation results of SimCA* during an energy leak at runtime. First, the system is configured and starts working according to UUV scenario (Sections III and VI-B) in a normal operating mode $M1$; the pole p is set to 0.75. However, at some point in time during operation (k = 60 in this case) the system detects an unexpected drastic loss of energy. As UUV is a very expensive equipment, the priority of the system becomes not running out of energy during operation. As a result, the accuracy requirement $R3$ is deactivated as it cannot be addressed anymore. From the other hand, the mission could still be completed, so requirement $R1$ remains in the system and the UUV enters the mode

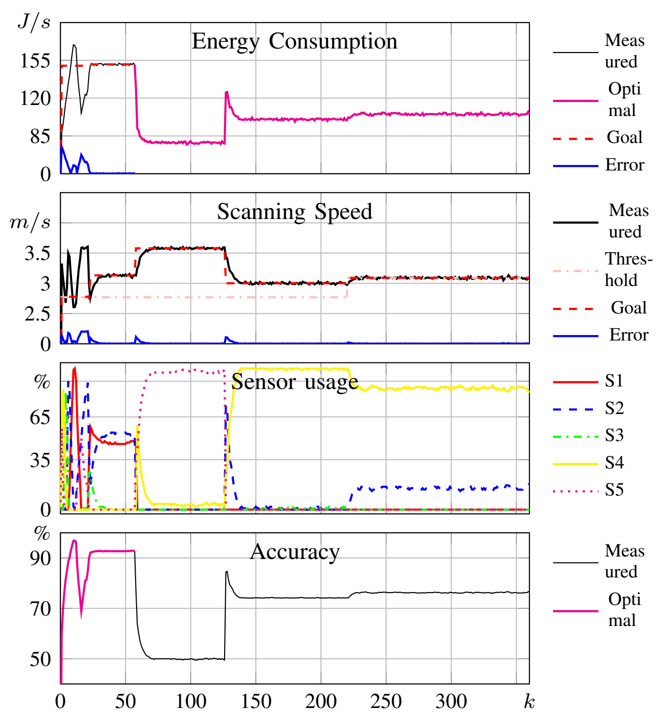

Fig. 8. UUV adaptation during energy leak, $p = 0.75$.

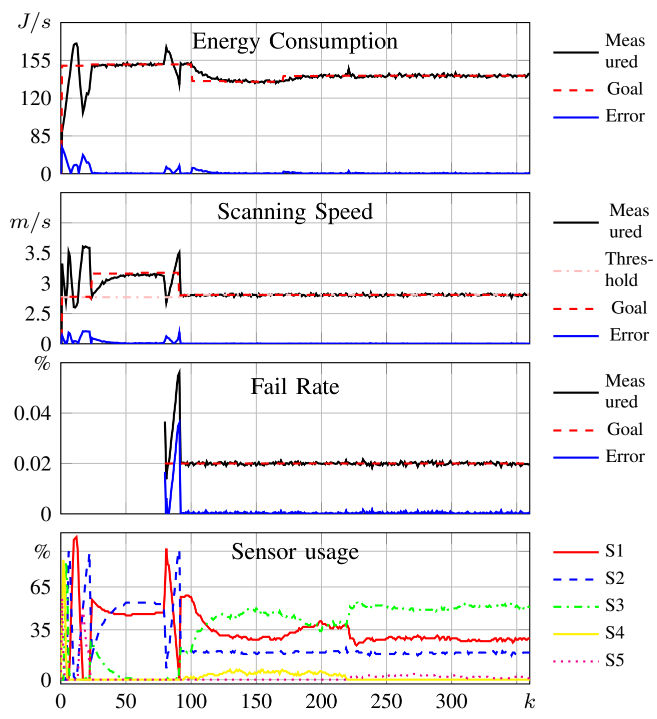

Fig. 9. New requirement activated at runtime, $p = 0.9$.

$M2 = R1, R2^*$. Summarizing, after the energy leak, the system requirements become: (i) $R1$: a segment of surface over a distance of bigger or equal to S (100 km) should be examined by the UUV within a given time t (10 hours); (ii) $R2^*$: subject to $R1$, the energy consumption of the vehicle should be minimized.

It is evident from figure 8 that after the energy leak (k = 60 onwards) SimCA* adapts the system to use the least energy consuming sensor $S5$, which also allows to keep the vehicle at a high speed, thus addressing $R1$. However, due to a high utilization, $S5$ breaks at k = 130. At this point, the UUV starts using $S4$ that is the least energy consuming sensor after $S5$. $S4$ has lower scanning speed than $S5$ (see how scanning speed decreases at k = 130), but it is still enough to address $R1$. At k = 220 the requirement $R1$ changes to $S \geq 114$ km. To cope with the updated goal, the UUV is forced to use sensor $S2$ leading to an increase in energy consumption. Note that after k = 220 the "Goal" and "Threshold" lines coincide on the Scanning Speed plot. It means that the vehicle will be able to scan exactly 114 km of surface and not more, while minimizing the consumption of remaining energy.

As in the previous case, the experiment ended after 10 hours of time. In total, the UUV consumed 3.79 MJ of energy and scanned 113.97 km of surface with 73% accuracy. After a series of experiments, we can conclude that SimCA* copes with the energy leak independent of point in time it happens. We also note that when a sensor failure occurs, the transition to a new goal value can be not as smooth as during the experiment presented in Figure 8. The reason for such behavior is that sensor failures occur in between adaptation actions (recall that adaptation period is 100 surface measurements). Thus, until next adaptation action, a random sensor is working instead of the broken one, making the system behave not according to the goals set by SimCA*.

E. New Requirement Scenario: Fail Rate

In this scenario, a new T-req is activated during system operation, see adaptation results in Figure 9. As previously, the system starts working according to the UUV scenario in mode $M1$; the pole p is set to 0.9. At k = 75 the UUV enters a deep water zone where sensors may fail to provide measurement. According to specification, each sensor has a certain fail rate in a deep water, namely: $S1$ = 0.01%, $S2$ = 0.06%, $S3$ = 0.01%, $S4$ = 0.02% , $S5$ = 0.04%. As high fail rate influences the mission outcomes, an extra requirement $R4$ is activated and the UUV enters the mode $M3 = R1, R2, R3, R4$, with $R4$: The average failure rate should be $F \leq 0.02\%$.

Figure 9 shows that a new Identification phase is started as soon as T-req is activated (k between 75 and 86). The aim of this phase is to create a model for the Fail Rate goal. After that a Fail Rate controller is added to the system and a new inequality is added to Simplex, which leads to a usual Operation phase (k = 87 onwards). As in the previous experiments, we change requirement $R1$ at k = 100 and 170; we also shut down $S4$ at $k = 215$ and increases the energy consumption of $S1$ at $k = 220$. In response to all of these changes SimCA* works as expected and selects an appropriate combination of sensors to be used, see Figure 9. For example, when $S4$ stops working, the UUV is forced to use sensor $S5$ and more of $S3$.

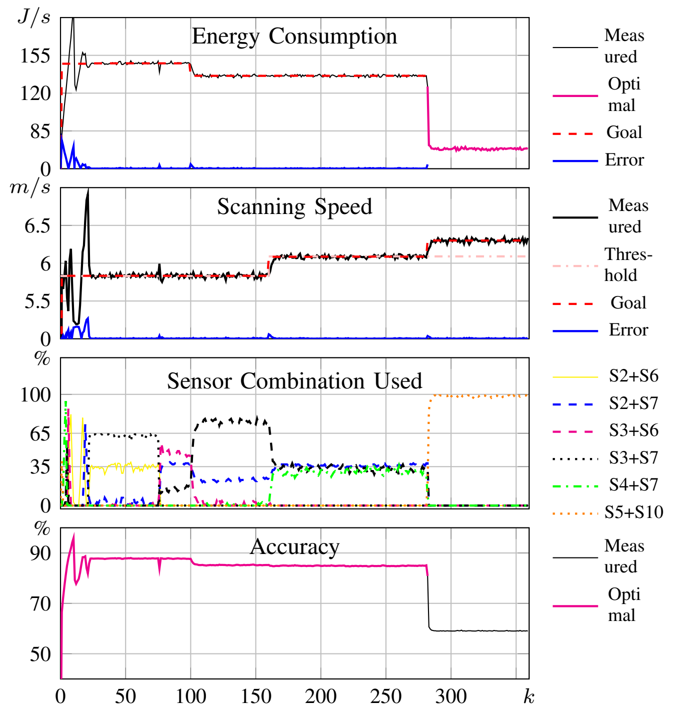

Fig. 10. UUV adaptation with 2 sensor panels, $p = 0.6$.

*F. Scalability of SimCA**

To demonstrate the scalability of SimCA* we significantly increase the number of actuation options (combinations of sensors) by equipping the UUV with two sensor panels. 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 I. 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: (i) R1: The underwater vehicle must examine $S \geq 210$ km of surface within a period of t = 10 hours (i.e., the scanning speed = $S/t \geq 5.83$ m/s); (ii) R2: The amount of available energy E is limited to 5.3 MJ (i.e., mission energy consumption = E/S = 147 J/s).

Figure 10 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. Note that the "Sensor Combination Used" plot shows only the 6 most used combinations for this scenario.

In general, the system shows the same adaptation behavior as in the case of a single sensor panel (converging to a goal value, adaptation to changes. etc.). In particular:

1) At $k = 75$ the sensor combination $S3$+$S7$ stops working leading to a switch of sensors being used;
2) The change of goals at $k = 100$ and 160 switches the sensor combination of the optimal solution;
3) At $k = 280$ we repeat the energy leak scenario leading to the use of combination $S5$+$S10$ that consumes the least amount of energy.

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.

G. Threats to Validity

SimCA* can handle one class of adaptation problems (satisfying multiple STO-reqs and adapting to requirement changes), but this class of problems apply to a significant number of software systems. At the same time, the approach should not be used to systems undergoing drastic changes in their behavior at runtime as continuous re-identification is very costly. SimCA* works with STO-reqs that can be transformed into quantifiable goals, which may not be easy for all properties; an example is security. SimCA* cannot handle conflicting requirements that lead to unfeasible solutions (e.g., to satisfy R1, the system is forced to ignore R2). However, when requirement are interrelated (e.g., increase in R1 leads to decrease in R2), the solution will be found if feasible.

We evaluated SimCA* in two domains, focusing on adaptation 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, further evaluation is required to validate SimCA* for other types of systems. In the experimental setting we have used only some types of disturbances (e.g., sensor failures and noise) and considered particular scenarios with changing requirements. Understanding the impact of other types of disturbances and other adaptation scenarios on SimCA* requires additional evaluation. We also used simulated systems for evaluation, which is inline with the evaluation conducted by others such as [30], [31], [32]. However, the deployment of SimCA* in a real-world setting is required to confirm the obtained results in practice.

VII. Conclusions

In this paper we presented SimCA* an approach that allows building self-adaptive software systems that satisfy multiple STO-reqs, can handle changes of requirements at runtime, and achieve robustness to environmental disturbances and measurement inaccuracy. SimCA* provides guarantees for the adaptation results. The effectiveness of SimCA* was formally evaluated and demonstrated on two cases with strict goals.

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 extend SimCA* to handle architecture reconfigurations at runtime and to apply the approach in real-world scenarios.

Appendix: Evaluation Scenario 2: TAS

To show the generality of SimCA*, we evaluate the approach with a second case: the TAS exemplar [33]. 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 in Figure 11.

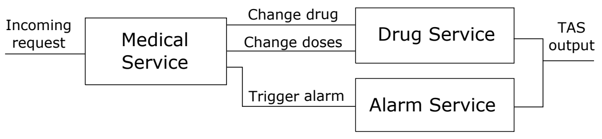

Fig. 11. TAS workflow.

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 Service, three providers offer the Alarm Service and only one provider offers the Drug Service. Table II shows example properties of available services based on data from [31].

The properties of TAS 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 the TAS service is set to 12¢;
- R2. The average response time should be below 35 tu;
- R3. Subject to R1 and R2, the failure rate of TAS should be minimized.

All requirements must be satisfied during normal system operation mode $M1 = \{R1, R2, R3\}$. Unlike the UUV case,

TABLE II
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**

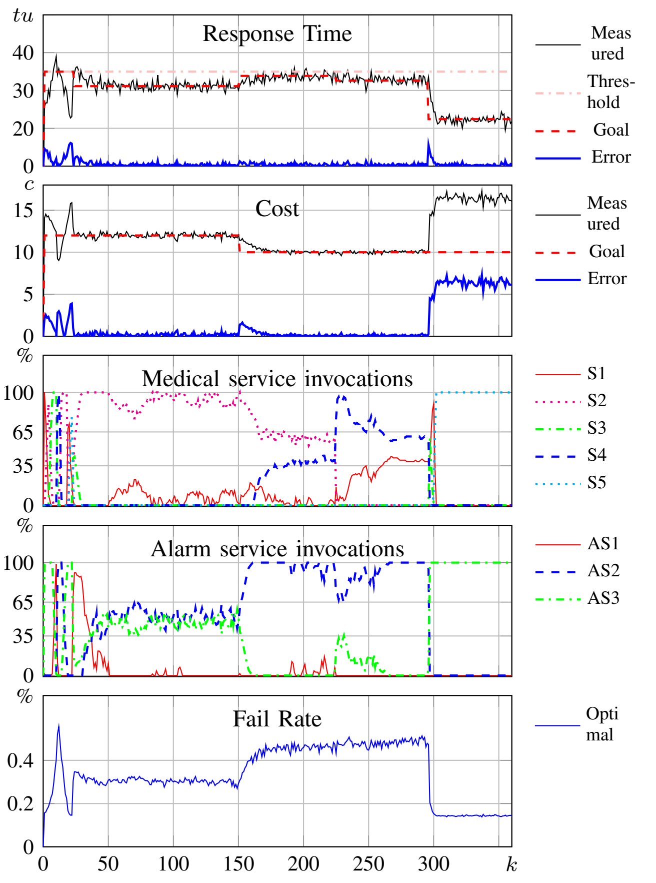

Fig. 12. SimCA* on TAS scenario.

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 [33]. The results of SimCA* applied to a TAS scenario are shown in Figure 12. The adaptation works as intended: the T-req is addressed by changing the Response time goal to 31 tu (at $k = 21$), system outputs follow the goal change at $k = 150$, the optimal solution is changed when $S2$ stops responding at $k = 225$. The cost goal is deactivated at $k = 295$ and the system enters mode $M2 = \{R2, R3\}$, where TAS uses services $S5$ and $AS3$, because they have the lowest response time and failure rate.

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.

References

[1] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Runtime software adaptation: Framework, approaches, and styles," in *Companion of the 30th International Conference on Software Engineering*, ser. ICSE Companion '08. New York, NY, USA: ACM, 2008, pp. 899–910. [Online]. Available: http://doi.acm.org/10.1145/1370175.1370181

[2] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, 2003.

[3] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, "(requirement) evolution requirements for adaptive systems," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 155–164. [Online]. Available: http://dl.acm.org/citation.cfm?id=2666795.2666820

[4] D. Weyns, N. Bencomo, R. Calinescu, J. Camara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jezequel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli, "Perpetual assurances for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems IV: Assurances, Lecture Notes in Computer Science*. Springer, 2016.

[5] 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.

[6] R. de Lemos *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II, LNCS vol. 7475*. Springer, 2013.

[7] D. Weyns, "Software Engineering of Self-Adaptive Systems: An Organised Tour and Future Challenges," *Chapter in Handbook of Software Engineering*, 2017, Springer, (forthcoming; https://people.cs.kuleuven.be/danny.weyns/papers/2017HSE.pdf).

[8] 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.

[9] 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.

[10] 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

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

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

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

[14] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin, "What does control theory bring to systems research?" *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, pp. 62–69, Jan. 2009. [Online]. Available: http://doi.acm.org/10.1145/1496909.1496922

[15] 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

[16] ——, "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.

[17] S. Shevtsov and D. Weyns, "Keep it SIMPLEX: Satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Pro-*

[17] S. Shevtsov and D. Weyns, "Keep it SIMPLEX: Satisfying multiple goals with guarantees in control-based self-adaptive systems," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 229–241.

[18] M. Kihl, A. Robertsson, and B. Wittenmark, "Performance modelling and control of server systems using non-linear control theory," in *Providing Quality of Service in Heterogeneous EnvironmentsProceedings of the 18th International Teletraffic Congress - ITC-18*, ser. Teletraffic Science and Engineering, R. L. J. Charzinski and P. Tran-Gia, Eds. Elsevier, 2003, vol. 5, pp. 1151 – 1160. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1388343703802640

[19] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: A probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786853

[20] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014, pp. 700–711.

[21] K. Angelopoulos, A. V. Papadopoulos, V. E. Silva Souza, and J. Mylopoulos, "Model predictive control for software systems with CobRA," in *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '16. New York, NY, USA: ACM, 2016, pp. 35–46. [Online]. Available: http://doi.acm.org/10.1145/2897053.2897054

[22] 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 *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, June 2012, pp. 33–42.

[23] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. H. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 1014–1027, Sept 2006.

[24] 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.

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

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

[27] 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.

[28] 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.

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

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

[31] 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.

[32] R. Calinescu, S. Gerasimou, and A. Banks, *Self-adaptive Software with Decentralised Control Loops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 235–251.

[33] 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