SimCA vs ActivFORMS: Comparing Control- and Architecture-Based Adaptation on the TAS Exemplar

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

M. Usman Iftikhar
Linnaeus University
Växjö, Sweden
usman.iftikhar@lnu.se

Danny Weyns
Linnaeus University
Växjö, Sweden
danny.weyns@lnu.se

ABSTRACT

Today customers require software systems to provide particular levels of qualities, while operating under dynamically changing conditions. These requirements can be met with different self-adaptation approaches. Recently, we developed two approaches that are different in nature — control theory-based SimCA and architecture-based ActivFORMS — to endow software systems with self-adaptation, providing guarantees on desired behavior. However, it is unclear which of the two approaches should be used in different adaptation scenarios and how effective they are in comparison to each other. In this paper, we apply SimCA and ActivFORMS to the Tele Assistance System (TAS) exemplar and compare obtained results, demonstrating the difference in achieved qualities and formal guarantees.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; I.2.8 [**Computing Methodologies**]: Problem Solving, Control Methods, and Search—*Control theory*

Keywords

Adaptation, self-, adaptive system, software, MAPE, control theory, controller, architecture, feedback, SimCA, activforms

1. INTRODUCTION

The burden on software developers has drastically increased in recent years as customers expect software to cope with continuous change. They expect the software to run seamlessly on different platforms, deal with varying resources, and adapt to changes in system goals. Often, these runtime changes are difficult to predict, requiring software engineers to design the software with incomplete knowledge.

Self-adaptation is widely encouraged to handle software design with incomplete knowledge [4, 7]. A classic approach to realize self-adaptation is architecture-based adaptation [15, 13], where a system maintains an explicit architectural model of itself, reasons about this model, and adapts itself to particular adaptation goals when relevant changes occur. However, achieving guarantees of the system behaviour with architecture-based adaptation is hard [3]. For this reason adaptation mechanisms based on control theory [10, 9] attracted the attention of self-adaptive systems community[1].

Recently, we developed two approaches for runtime adaptation, one from each of the mentioned fields: *ActivFORMS* (Active Formal Models for Self-adaptation) [11] an architecture-based approach, and *SimCA* (Simplex Control Adaptation) [16] that is based on principles from control theory and linear optimization. In this paper, we present a comparative evaluation of SimCA and ActivFORMS on a set of scenarios. To the best of our knowledge, no systematic comparison between approaches for runtime adaptation based on control theory and architecture-based adaptation has been performed so far. We compare obtained results of runtime adaptation (the system output), and we analyze the provided guarantees and system behavior in the presence of disturbances.

The evaluation is conducted using the TAS exemplar [17]. TAS provides remote health support to patients by composing a number of services. Each service can be implemented by multiple providers. These implementations have different reliability, performance and cost, which affect the overall quality of the application. Choosing a concrete service provider to ensure the required quality of service (QoS) at runtime is a key factor to adaptation of service-based systems as TAS.

The remainder of the paper is structured as follows. An adaptation scenario with the TAS exemplar is introduced in Section 2. Section 3 summarises the two studied adaptation approaches: SimCA and ActivFORMS. In Section 4 these two approaches are applied to TAS and compared in multiple scenarios. Section 5 provides discussion of received results. Finally, conclusions and directions for future research are presented in Section 6.

2. ADAPTATION SCENARIO: TAS

In this section, we introduce a scenario of TAS used for evaluating SimCA and ActivFORMS. 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 1.

Each incoming request is first processed by the Medical Service. This service receives messages from patients with their

[1]In this paper by self-adaptive system we mean a system equipped with any kind of adaptation/control mechanism which may or may not be adaptive itself.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

CTSE'15, August 31, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3814-1/15/08...$15.00
http://dx.doi.org/10.1145/2804337.2804338

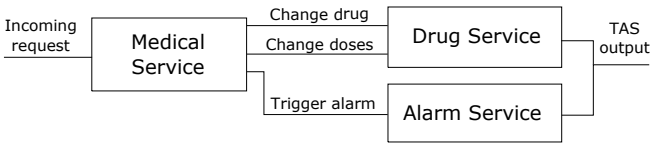

Figure 1: TAS workflow.

vital parameters, analyses the data, and replies with instructions to (1) change the drug or (2) change the drug doses, or (3) invoke an alarm at the First-Aid squad in case of an emergency. When invoked, the Drug Service notifies a local pharmacy to deliver new medication to the patient or change his/her dose of medication. When the Alarm Service is invoked, it dispatches an ambulance to the patient.

For service-based systems such as TAS, the functionality of each service can be implemented by a number of 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 services offered by the providers. The services are considered to be part of the environment because they are not under control of TAS. For example, the failure profile of a concrete service implementation may change at runtime, due to the changing workloads at the provider side or unexpected network failures.

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 1 shows example properties of available services based on data from [2].

Table 1: Properties of all services used in TAS.

Service	Name	Fail.rate, %	Resp.time, ms	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 (Goal)	**0.03**	**26**	**min**

The system requirements are the following:

R1. The average failure rate should not exceed 0.03 %*

R2. The average response time should not exceed 26 ms

R3. Subject to R1 and R2, the cost should be minimized.

The requirements R1-R3 as well as the properties of the services may change at runtime and the system should adapt accordingly. The adaptation task is to decide, for each incoming messages with a patient's vital parameters, which combination of services to select in order to continuously satisfy the three requirements.

*The system design assumes that in case of a failure the request is not dropped and can be send to the same or another service provider for re-execution. Hence, the goal failure rate is lower than the rates of individual implementations.

2.1 The Adaptation Problem

Generalizing from the concrete TAS scenario, the adaptation problem we are aiming to solve is the following:

> *Maintain a desired level of quality for 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.*

Defining and developing such an adaptive solution introduces several key challenges. First, the appropriate sensors (measured variables) and actuators (knobs that can influence the software behavior) must be carefully chosen. Second, the software system must be properly modeled. Finally, the appropriate adaptation mechanism that achieves the goals, rejecting external disturbances and optimising the solution according to additional goal, must be developed. The following section describes two approaches that tackle these challenges.

3. STUDIED APPROACHES

3.1 SimCA

The adaptation logic of SimCA consists of multiple SISO controllers that independently compute control signals for each of the goals, and a simplex block that receives the control signals as input and produces an output that is used for adapting the software system. Figure 2 schematically shows the primary building blocks of SimCA.

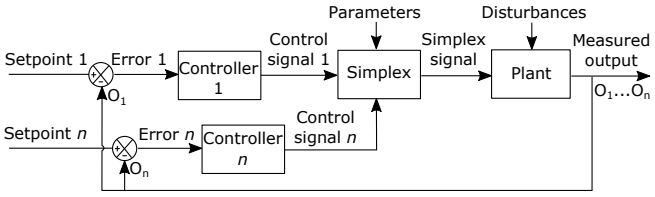

Figure 2: A self-adaptive software with SimCA.

A detailed explanation of SimCA is available in [16]. In short, each system goal (reliability, performance) except cost is represented as a setpoint $s_i(k)$. On every adaptation step k the system outputs $O_i(k)$ are measured. Based on the error $e_i(k) = s_i(k) - O_i(k)$ and a linear model $\mathcal{M}_i$ described below, each controller C_i produces a signal $u_i(k)$ which represent the value of each goal that should be reached by the system. The simplex block receives $u_i(k)$ and additional environment/plant parameters (e.g., the invocation cost of external services) as inputs and produces a simplex signal $u_{sx}(k)$ as output. $u_{sx}(k)$ contains the values of system knobs which affect the plant behaviour. For TAS, $u_{sx}(k)$ is a vector containing the probabilities to select each of the available service providers. Disturbances affecting $O_i(k)$ are handled by controller C_i via adjusting the control signal $u_i(k)$.

SimCA works in three phases: identification, control, and optimization.

In the *identification* phase, n linear models of the controlled system are built. Each model $\mathcal{M}_i$, $i \in [1, n]$, is responsible for one goal s_i. The identification phase starts by feeding all possible control signal values to the plant. The goal of identification is to determine the influence of control signal u_i on the corresponding system output O_i at every time step

k. The dependency between u_i and O_i is captured by the coefficient α_i which is further used to build controller i. α_i is calculated during identification based on linear regression using the APRE tool [14]. As a result, the following set of linear models is obtained:

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

In $\mathcal{M}_i$ the control signal is a value from the interval $[min_i, max_i]$, where max_i and min_i are the maximal and minimal values that can be achieved by TAS for the i-th goal:

$$u_i(k) = (max_i - min_i) \times \eta_i(k) + min_i \qquad (1)$$

η_i is the control coefficient. During the identification phase, η_i changes from 0.0 to 1.0 using an increment of 0.05 on every step k. As a consequence, most feasible values of goal i are produced as u_i. This allows us to measure all the possible values of the outputs $O_i(k)$, calculate α_i, and build $\mathcal{M}_i$.

The model $\mathcal{M}_i$ generally describes the system behavior but does not take into account small disturbances or sudden failures that typically occur in software systems. To deal with inaccuracies in $\mathcal{M}_i$, SimCA uses a Kalman filter to adapt the model at runtime, and a change point detection mechanism that allows reacting to critical changes in the system.

An important note concerning the control methodology of SimCA is that the simplex method does not change the value of control signal i. Instead, simplex is responsible for seamless translation of control signals into a proper actuation signal. For example in TAS, if the control signal equals to failure rate = 0.03, simplex finds a combination of services (S1-S9) that assures this failure rate is not exceeded. Hence, simplex is not considered when building a system model and synthesizing controllers that manage this model. Instead, controllers are assumed to affect the plant output via control signals. Every goal is controlled separately during the first two phases of SimCA. This means that during identification and control phases SimCA works in parallel with multiple Single-Input-Single-Output (SISO) controllers, and then combines control outputs with the help of simplex during optimization phase.

In the *control* phase, a set of n controllers is synthesized. Each controller C_i is responsible for the i-th goal. C_i calculates the control coefficient η_i at the current time step k depending on the previous value of control coefficient $\eta_i(k-1)$, adjustment coefficient α_i, controller pole p_i and the error e_i:

$$\eta_i(k) = \eta_i(k-1) + \frac{1-p_i}{\alpha_i} \times e_i(k) \qquad (2)$$

The controller pole p_i belongs to the open interval $(0, 1)$ to maintain stability and avoid oscillations. The pole also allows to trade-off robustness to external disturbances with the convergence speed (known as settling time): higher values of p_i lead to slower convergence [8].

In the *optimization* phase, SimCA combines the signals $u_i(k)$ from multiple controllers using the simplex method to optimally drive the measured output of the system towards its desired behavior. Simplex solves the following problem:

$$\text{Minimize Cost: } min\, C = \sum_{j=1}^{p} c_j \cdot x_j$$

subject to:

$$\begin{cases} \sum_{i=1}^{n} \sum_{j=1}^{p} a_{ij} \cdot x_j = u_i \\ x_j \geq 0,\ u_i \geq 0,\ with\ i = 1 \ldots n,\ j = 1 \ldots p \end{cases} \qquad (3)$$

where:

- p: a number of variables — each variable represents one service provider;
- n: a number of equations — each equation represents a goal to be satisfied;
- x_j are the values of system knobs, i.e. probabilities to select each of the service providers;
- u_i are the control signals;
- a_{ij} and c_j are the monitored parameters: the failure rate, response time and cost of external services.

SimCA finds the variables of the problem x_j and passes them to the plant in the form of a simplex signal. A detailed explanation on how simplex solves the system of equations (3) can be found in [16], and additional background in [5].

3.2 ActivFORMS

ActivFORMS is an architecture-based approach for self-adaptation that uses an integrated formal model of the adaptation components of the feedback loop and the knowledge they share [11]. ActivFORMS distinguishes itself from existing architecture-based approaches in three ways:

- The formally verified model of the feedback loop is directly executed by the virtual machine, hence called the active model. This allows to guarantee at runtime the system properties verified at design time. As the active model is directly executed, the approach does not require coding.
- ActivFORMS supports dynamic changes of the active model. A new feedback loop model can be deployed at runtime to meet new or changing goals.
- ActivFORMS supports goal management and model verification at runtime.

ActivFORMS follows a three-layered reference model proposed by Kramer and Magee [13], see Figure 3. The bottom layer comprises the managed system[2] that implements the domain-specific functionality. The active model monitors and adapts the managed system through probes and effectors respectively[3]. The goal management layer monitors the active model and environment, and deals with adaptation issues that cannot be handled by the current active model; e.g., it dynamically changes the active model to deal with changing system goals.

The active model realizes a MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) feedback loop [12] that monitors the managed system and adapts it according to the system goals. ActivFORMS supports feedback loops modeled using networks of timed automata. A timed automaton is a finite-state machine that models a behavior, extended with clock variables.

Recalling the TAS scenario, we used a utility function as a part of the planner component of a MAPE-K formal model to provide the required adaptation, see Figure 4. When triggered by a signal from the analysis component, the planner calculates the utility function coefficients and sends a signal to the execution component to update the managed system.

[2]Managed system is called plant in control theory.

[3]Probe corresponds to sensor in control theory terminology, effector corresponds to actuator.

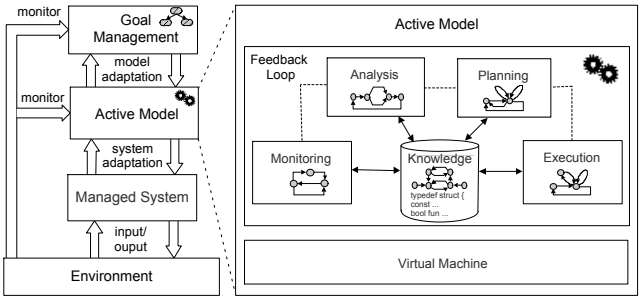

Figure 3: The ActivFORMS approach

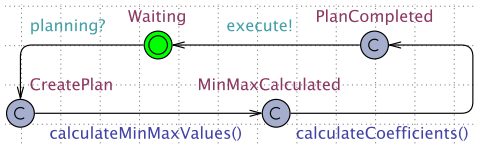

Figure 4: Planning automaton

The utility function works as follows: all system goals (reliability, performance) and optimization goal (cost) are represented as G_i. During the first adaptation period the cost goal is set to a minimum achievable value: min_{cost}. On every adaptation step k the system outputs $M_i(k)$ are measured. Knowing the error $e_i(k) = G_i - M_i(k)$, we calculate how big it is in relation to the actual values of the goal i:

$$\lambda_i(k) = \frac{e_i(k)}{max_i - min_i} \tag{4}$$

For example, if the measured response time $M_{RT}(k)$ is 29.04 on a first adaptation step and other service parameters are equal to the values from Table 1, we get:

$$\lambda_{RT}(1) = \frac{29.04 - 26}{35.4 - 21.68} = 0.22$$

Then we calculate coefficients $cf_i(k)$ for the utility function. $cf_i(k)$ represents the selection probability for a QoS strategy in the next adaptation period:

$$cf_i(k) = \frac{\lambda_i(k)}{\sum_{j=1}^{n} \lambda_j(k)} \tag{5}$$

Using the same example, we get:

$$cf_{RT}(1) = \frac{0.22}{0.30 + 0.22 + 0} = 0.43$$

In the provided example, a coefficient for performance of $cf_{RT} = 0.43$ means that in the next adaptation period the service provider with the lowest difference between G_{RT} and service response time will be chosen 43% of the times.

For offline model-checking we use Uppaal [1], a tool that supports modeling of behaviors and verification of properties of networks of timed automata. For the specification of properties, we use Timed Computation Tree Logic (TCTL). TCTL allows checking individual states of the system state space as well as traces over the state space. The latter allows to verify reachability, safety, and liveness properties.

The ActivFORMS virtual machine can perform the following functions: execute the formal model according to the semantics of timed automata, interact with the managed system and environment through probes and effectors, support online verification of the active model, and update the active model when requested. ActivFORMS provides a set of formal templates to design the MAPE-K elements [6], and abstract Java classes to implement probes and effectors. Probes track the managed system and the environment and transfer data to the Monitor automata of the feedback loop, while Effectors transfer actions generated by the Execution automata to the managed system.

Goal Management comprises a tree-based goal model where nodes have associated MAPE-K models to realize adaptations. Goal management monitors goals via the virtual machine. When a goal violation is detected, the models associated with an alternative goal that matches the changing conditions are used to update the deployed models via the virtual machine. Goal models can be updated at runtime. Here, we do not focus on the goal layer, we refer the interested reader to [11].

4. COMPARATIVE EVALUATION

We now evaluate the approaches. Section 4.1 describes the experimental setting. Section 4.2 shows the adaptation behavior of SimCA and ActivFORMS on a basic TAS scenario and in response to different runtime changes. Finally, Section 4.3 discusses guarantees provided by both approaches.

4.1 Experimental Setting

We use the TAS case described in Section 2 to compare the adaptation approaches. The TAS exemplar [17] is realized as a Java application and extended with SimCA and ActivFORMS classes. The starting parameters of the services and system requirements are specified in Table 1.

Adaptation is performed once per 100 invocations of TAS (k = 100 inv.). At each adaptation step the application calculates the average measured value of the i-th goal (e.g., measured failure rate) during the past 100 inv. Then it calculates error e_i as the difference between i-th setpoint (e.g., target failure rate) and measured value of the i-th goal. The application also monitors the cost of serving the incoming requests.

The task of both SimCA and ActivFORMS is to keep the goals at their setpoints and minimize the cost. SimCA achieves this task by calculating the value of the *simplex signal*, which represents the probability of selecting the services in the list $\{S1, S2, ..., S9\}$. ActiveFORMS achieves this task by *prioritizing goals* with the help of utility function and updating cost setpoint in case of goal violations. Due to high runtime fluctuations in the values of service parameters, the controller pole p in SimCA is set to 0.98 which allows to reject errors of high magnitude. For the same reason, the cost increment Δ_C in ActivFORMS is set to a low value of 0.1¢ [4].

The application collects the data of the system and the service implementations to build performance graphs, which are used to compare the adaptation approaches in the following section. The x-axis of the graphs are time instants t, each instant corresponds to a series of Δt inv. of TAS. Thus, the y-axis shows the average values of the measured feature per Δt inv. of TAS. Δt can be changed in the TAS interface.

4.2 Adaptation Results

The graphs in Figure 5 show adaptation results of SimCA and ActivFORMS on TAS configured according to Table 1.

[4]We use ¢ symbol to represent cost throughout the paper.

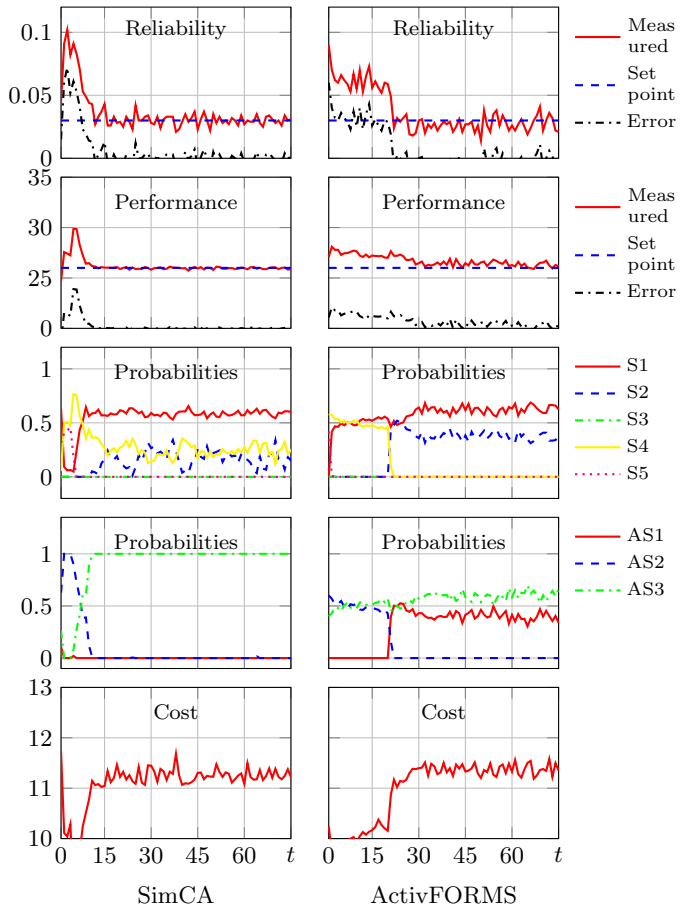

Figure 5: SimCA vs activFORMS on a TAS scenario (Δt = 1000 inv.)

SimCA starts with an identification phase ($t < 7$). The control phase, which is immediately followed by an optimization phase, starts after the relationship between control signals, simplex signal and system output is identified (from t equal 7 onwards). This phase is stable, all minor spikes on the SimCA graphs are caused by the random nature of failures in TAS, e.g., a failure rate of 3% does not always lead to 3 fails per 100 inv.

In the same scenario ActivFORMS starts with a cost setpoint of 8.5¢ which is the average minimal invocation cost of the TAS workflow. In this scenario we assume no prior offline verification of properties so the optimal cost is considered unknown. As a result, both the reliability and performance goals are violated ($t < 20$ on the right graphs). The adaptation slowly increases the cost setpoint which leads to a zero reliability error and a small performance error after $t = 30$.

Comparing TAS reliability achieved by SimCA and ActivFORMS, it is notable that the latter approach requires three times more time to reach a stable state with no goal violation. Comparing performance, ActivFORMS slightly violates the goal even after $t = 75$, but the most notable error can be observed when $t < 30$. So in this case SimCA needs 4 times less time in order to reach a setpoint. As for the picked services, the strategies differ. Though both approaches use S1 approximately equal amount of times ($\approx 60\%$), SimCA prefers a combination of S2, S4 and AS3, while ActivFORMS uses S2, AS1 and AS3.

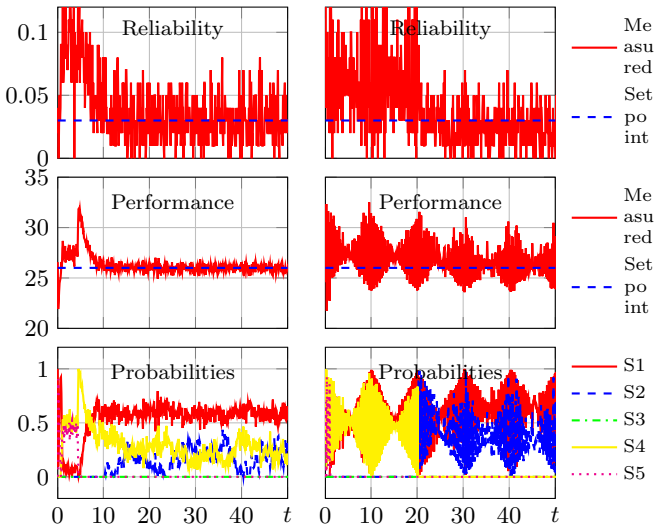

Figure 6: SimCA vs ActivFORMS on a TAS scenario (Δt = 100 inv.)

The Cost graphs (Figure 5) seem to be similar, but closer examination shows that the average cost measured during stable state (t between 30 and 75) is 11.26¢ for SimCA vs 11.37¢ for ActivFORMS. This means a saving of 110¢ per 1000 workflow invocations for SimCA compared to ActivFORMS.

To further compare the approaches, we decrease the measurements interval Δt from 1000 to 100 workflow invocations, see Figure 6. The measured reliability spikes are almost equal on the top plots, however performance and probabilities to select services in ActivFORMS have noticeably higher oscillation amplitude than in SimCA. Such effect is caused by the nature of the algorithm that selects utility function coefficients: the closer the measured value of a goal i gets to the setpoint i, the lower is the priority of goal i.

It is worth mentioning that simplex operating under disturbances has the property of distributing probabilities for selecting services equally among all available services [16]. We do not observe this behaviour in the solution with ActivFORMS. With ActivFORMS, the probabilities of selecting some of the services can suddenly change from 0 to 100% in 1000 workflow inv. (e.g., see the probability to select S2 in $30 < t < 40$). Hence, with SimCA, the load on service providers will be relatively smooth over time, while the load with the ActivFORMS solution can change abruptly, requiring the service providers to keep 100% of their resources available all the time.

Both adaptation approaches have a mechanism that allows to trade-off settling time for other properties, so we study how such trade-off will affect the system output, see Figure 7.

Lowering pole p in SimCA from 0.98 to 0.7 drastically increases the oscillation amplitude of all outputs. After $t = 40$ the system even triggers re-identification as the measured system output values are too far from their setpoints. This is caused by a combination of low settling time (i.e., the system immediately follows any change in the output) and small adaptation period that leads to high measurement errors.

Raising the cost increment Δ_C in ActivFORMS from 0.1 to 1.0 also increases oscillations of all system outputs. However, their amplitude is much lower, see the 'Reliability' and 'Cost' plots. The system is also more stable than in SimCA. This experiment allows to conclude that ActivFORMS is a better

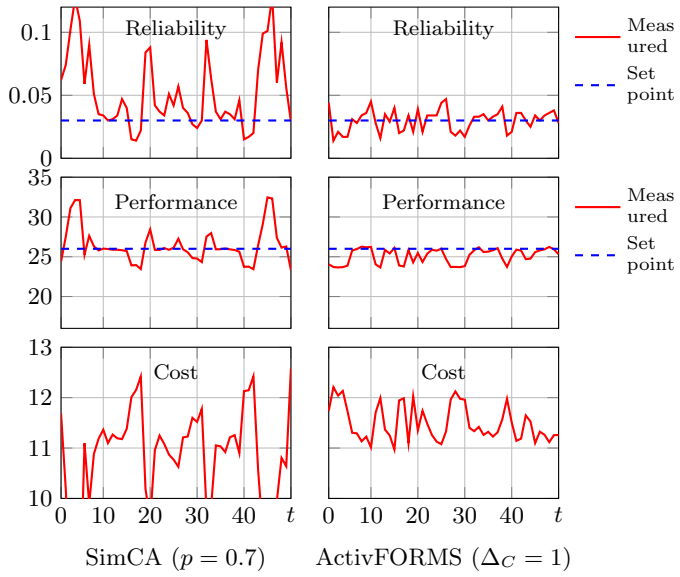

Figure 7: SimCA vs ActivFORMS, the settling time trade-off (Δt = 1000 inv.)

adaptation solution for a system with settling time priority.

Finally, we show how both approaches react to runtime changes, see Figure 8. First of all, having the same initial conditions as in the previous experiments, we set ActivFORMS' cost setpoint to 11.0¢ instead of 8.5¢ because we already know the approximate optimal cost value. This adjustment results almost immediately in convergence to the desired setpoint values. Hence, by doing a system pre-run or offline validation, the ActivFORMS adaptation time can be greatly decreased.

Coming back to Figure 8, there are three major changes happening in the system at runtime:

- *Goal change.* Both goals are changed at t=22: failure rate from 0.03 to 0.05%, resp.time from 26 to 28 ms;
- *Abrupt change.* Medical Service 2 breaks at t=33;
- *Parameter change.* Response time of Medical Service 1 increases from 22 to 52 ms at t=55.

The plots show that both adaptation approaches deal with all types of change: the measured values of both goals follow their setpoints. As in the previous experiments, ActivFORMS requires more time to adapt to the new setpoints.

More importantly, the cost of TAS workflow execution at t=22 decreases to 10.2¢ with SimCA vs 10.8¢ with ActivFORMS. When Service 2 shuts down at $t = 33$, it is already barely used by SimCA because there is another combination of services that can produce the same result with lower cost. However, the ActivFORMS solution even increases the utilization of Service 2 when t is between 22 and 33. When the response time of Service 1 increases at $t = 55$, the difference between the workflow invocation cost is around 1¢ per invocation. This scenario shows that SimCA better solves optimization tasks.

4.3 Guarantees

By using the simplex method, SimCA guarantees optimal cost in TAS without violating the failure rate and response time goals. Simplex has proven to be a practical and fast algorithm for solving this kind of optimization problems [5].

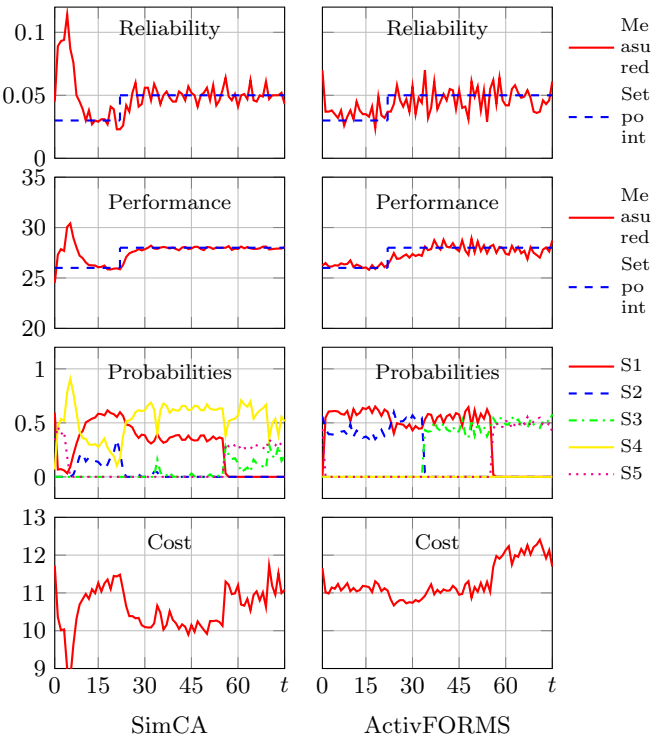

Figure 8: SimCA vs ActivFORMS during runtime changes (Δt = 1000 inv.)

The control-theoretical guarantees provided by SimCA are divided into four main categories: asymptotic stability, steady-state error, settling time and overshoot. If a closed-loop system is asymptotically stable, it reaches the proximity of the setpoint s_i. If the system has zero steady-state error, its setpoint s_i is reached after a certain time and $O_i(k) = s_i(k), k \geq \bar{K}$. The amount of time $\bar{K}$ after this happens is called settling time. If the controlled value exceeded the setpoint before reaching a stable area, this is called overshoot and should be avoided.

The control-theoretical guarantees provided by SimCA are confirmed by the data shown on the Figures:

- The control system is asymptotically stable and converges without overshooting, since it is designed to have only a single pole p which belongs to the open interval $(0, 1)$;
- According to the system output equation of SimCA [16], the output O_i during steady-state equals s_i which leads to a zero steady-state error: $\Delta e = O_i - s_i = 0$. The absence of a steady-state error can be observed on the performance plot of Figure 5 when $t > 15$;
- The settling time $\bar{K}$ of a unit step of every controller 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 [10, p.85], the commonly used value of Δs is 0.02 (2%). Hence $\bar{K} = \frac{\ln 0.02}{\ln 0.98} = 193.6$. This means that changing the response time from 26 to 28 (step of amplitude 0.2) would take around $193.6 * 0.2 \approx 40$ adaptation steps. This guarantee can be observed on the performance plot of Figure 8 when t is between 20 and 24*;

*As adaptation is performed every 100 workflow invocations

Due to the nature of the simplex method, SimCA will only work when the setpoint of every goal lays between minimal and maximal values of that goal: $min_i \leq s_i \leq max_i$. Therefore, when having an infeasible goal, a system with SimCA will not converge to the closest feasible value for that goal. Instead, the user will be notified that the task is not solvable and the change point detection mechanism will cause an infinitely loop of identification phases, see Figure 9. On the contrary, ActivFORMS will adapt the system output so that it converges to the closest feasible goal value.

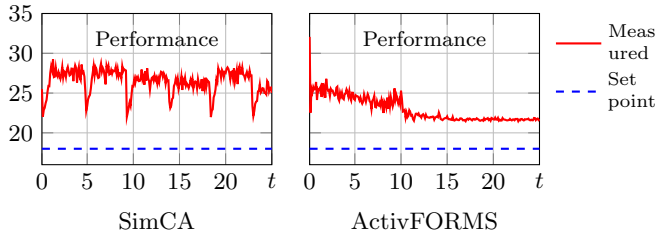

Figure 9: SimCA vs ActivFORMS with infeasible goal (Δt = 1000 inv.)

The ActivFORMS approach provides offline and online guarantees on the system behaviour. The offline guarantees come from the verification of the formal model through TCTL properties at the design time with the Uppaal model checker; verified properties are: reachability, safety, liveness, and deadlock freeness [1].

To achieve the guaranteed cost optimality required by TAS, we first implemented the utility function algorithm as a part of the planner component of a MAPE-K loop. Then, we determined the lowest value of cost that does not violate the goal failure rate and response time with the help of formal offline verification of the following safety property:

```
A[] (gCost == MIN_VALUE) imply
  (measuredFR <= gFR AND measuredRT <= gRT)
```

The system parameters and goals for verification were taken from Table 1. The MAPE-K model is also verified to guarantee the correctness of the algorithm's functionality. E.g., to guarantee that the system model is deadlock free, i.e. it does not have erroneous states and the system does not get stuck in any particular state, Uppaal provides a special formula:

```
A[] not deadlock
```

The following liveness property guarantees that whenever the planning behavior is invoked to create a plan, then eventually a plan is executed.

```
Planning.CreatePlan --> Execution.PlanExecuted
```

The runtime guarantees of ActivFORMS are provided in two ways. First, the model that was formally verified offline is directly executed by the virtual machine at runtime so these guarantees are preserved at runtime. Second, as offline verification is limited to the input provided by the verification models, ActivFORMS allows to continue verification of properties at runtime and inform the user about violations. We refer the interested reader to [11] for details.

and Δt = 1000 invocations, 4 time steps on the graph equals to 40 adaptation steps

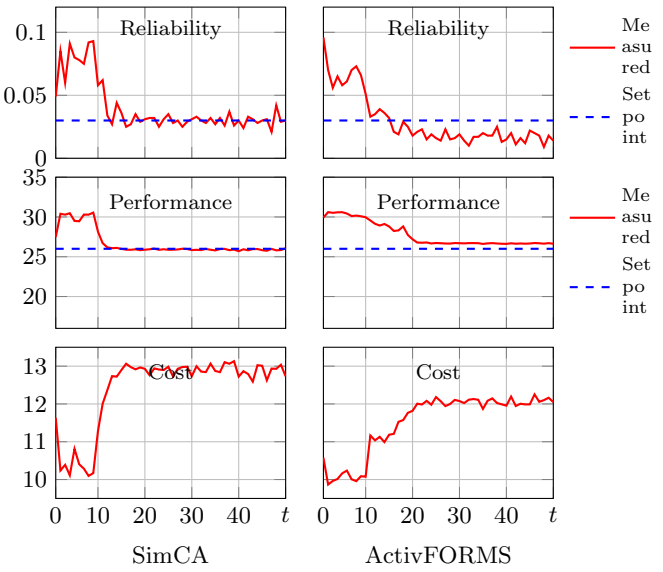

Figure 10: SimCA vs ActivFORMS with a constant disturbance (Δt = 1000 inv.)

In the studied TAS scenarios we used both types of runtime guarantees provided by ActivFORMS. The direct execution of the verified model allowed to guarantee the correctness of the adaptation mechanism implementation; the runtime verification mechanism of ActivFORMS guaranteed that the minimal cost, previously determined offline, does not lead to violation of the failure rate or response time goals at runtime. To insure the latter, we added the following runtime goals:

```
G1: A[] (measuredFR <= gFR AND measuredRT <= gRT)
G2: A[] (measuredFR > 0.9*gFR AND measuredRT > 0.9*gRT)
```

Goal $G1$ assures that the failure rate and response time of TAS do not exceed their goals. Goal $G2$ assures that the values of both goals will not drop lower than 90% of their goals, i.e. the system will not waste resources. In case of a runtime goal violation, the cost setpoint $gCost$ is adapted accordingly: for $G1$ violation $gCost$ is increased by the cost increment Δ_C, for $G2$ violation $gCost$ is decreased by Δ_C.

Changing Δ_C allows trading-off the speed of the system reaction to runtime changes (settling time) for the amplitude of system outputs oscillations around the goal value (accuracy).

4.4 Disturbance Handling

Throughout the experiments discussed in Section 4.2, both approaches handled randomly distributed measurement disturbances caused by the changes of service parameters at runtime. Different to ActivFORMS, SimCA allows to formally evaluate the amount of disturbance the system can withstand.

According to the PBM approach [8] that lays at the core of SimCA, the amount of disturbance the system can withstand $\Delta(d)$ by using a PBM controller can be estimated as follows: $0 < \Delta(d) < \frac{2}{1-p}$. This means that the value of the pole p defines how SimCA will react to disturbances. For $p = 0.98$ that was used in most experiments, the measurement can be inaccurate by a factor of 100, and the controller of SimCA will still adapt the system to follow the goals.

However, as mentioned in Section 4.3, the simplex method will not be able to reach an infeasible goal. Hence, the measurement inaccuracy on values of goal i that SimCA can with-

stand is: $min_i \leq \Delta(d) \leq max_i$. This property of SimCA can be observed on Figure 7 after $t = 40$: the reliability output reaches a value that is higher than maximal achievable value for a TAS workflow which causes a re-identification phase.

When it comes to constant disturbances, both approaches successfully perform adaptation. In the scenario shown on Figure 10, there is a constant response time measurement disturbance of +3 ms (e.g., the sensor shows 29 ms instead of 26 ms). As in the previous experiments with the same pole and cost increment, SimCA converges to the setpoint faster. It is notable that ActivFORMS achieves better cost by slightly violating the performance goal.

5. DISCUSSION

Both SimCA and ActivFORMS successfully solved the adaptation problem. The choice between these approaches depends on the particular scenario. When it comes to formal guarantees, both approaches provide the main guarantee required from adaptation, i.e. to minimize the cost without violating performance and reliability goals. SimCA guarantees this by using the simplex method and ActivFORMS verifies the solution optimality offline. However, at runtime SimCA performs better because offline system validation of ActivFORMS relies on particular system properties that may not hold online.

Other formal guarantees provided by the two approaches are different. With SimCA it is possible to formally prove a number of system properties, such as stability, settling time, amount of disturbance the system withstands, etc., which do not depend on the particular system parameters. On a contrary, with ActivFORMS these properties are inherent to the underlying algorithm, such as a utility function that was used in our study, and do not come from the design of the adaptation mechanism. With ActivFORMS it is possible to formally verify these properties of the algorithm on a particular set of system parameters by exploring the whole state space. However, trying to verify the TAS system with all possible combinations of service parameters will lead to an explosion of the state space. This is compensated by runtime verification.

On the other hand, with ActivFORMS the correctness of the implementation of adaptation mechanism can be formally proven, in particular the absence of erroneous states and correct interaction between adaptation components.

6. CONCLUSIONS

In this paper we presented a comparative evaluation of the two different approaches for self-adaptation: SimCA which is based on control theory and linear optimization, and architecture-based ActivFORMS equipped with a utility function. The study was performed using the TAS exemplar. The analysis of adaptation results have shown that both approaches can deal with multiple goals and provide guaranteed optimality with respect to an additional goal. However, SimCA achieves better results in the presence of runtime changes as it does not rely on data verified at design time. At the same time, ActivFORMS is a good choice for systems that require low settling time.

Except optimality, the two adaptation approaches offer different guarantees. The design of SimCA adaptation mechanism, such as the pole value or the system output equation, allows to formally prove the properties of underlying system and guarantee that they will hold at runtime independent of system parameters. ActivFORMS allows formal verification of system properties based on the particular input data. Runtime verification allows complementary verification when the system parameters change. An important feature of ActivFORMS is that it allows to formally guarantee the functional correctness of the implementation of adaptation algorithm.

This work is a first step towards understanding the effectiveness and formal power of different adaptation mechanisms. As a part of future efforts, we want to compare other adaptation approaches, such as QoSMOS [2] to SimCA and ActivFORMS. We are also planning to test the adaptation mechanisms on different types of systems and scenarios.

7. REFERENCES

[1] G. Behrmann, R. David, and K. G. Larsen. A tutorial on UPPAAL. pages 200–236. Springer, 2004.

[2] R. Calinescu et al. Dynamic qos management and optimization in service-based systems. *IEEE TSE*, 37(3):387–409, May 2011.

[3] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9), 2012.

[4] B. H. Cheng et al. Software engineering for self-adaptive systems: A research roadmap. In *SEfSAS*, 2009.

[5] G. B. Dantzig and M. Thapa. *Linear Programming 2: Theory and Extensions*. Springer, 2003.

[6] D. G. de la Iglesia and D. Weyns. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM TAAS*, 2015.

[7] R. de Lemos et al. Software engineering for self-adaptive systems: A second research roadmap. In *SefSAS II*, 2012.

[8] A. Filieri, H. Hoffmann, and M. Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *ICSE*, 2014.

[9] A. Filieri, M. Maggio, et al. Software Engineering Meets Control Theory. In *SEAMS*, 2015.

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

[11] M. U. Iftikhar and D. Weyns. ActivFORMS: Active formal models for self-adaptation. In *SEAMS*, 2014.

[12] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1), 2003.

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

[14] M. Maggio and H. Hoffmann. ARPE: A tool to build equation models of computing systems. In *Feedback Computing*, 2013.

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

[16] S. Shevtsov, D. Weyns, and M. Maggio. Keep it SIMPLEX: Handling multiple concerns in control-based self-adaptive systems. *Technical Report, 2015. Online: http://homepage.lnu.se/staff/daweaa/papers/SimCA-report.pdf.*

[17] D. Weyns and R. Calinescu. Tele assistance: A self-adaptive service-based system exemplar. In *SEAMS*, 2015.