Engineering Self-Adaptive Software Systems: From Requirements to Model Predictive Control

KONSTANTINOS ANGELOPOULOS, University of Brighton, UK
ALESSANDRO V. PAPADOPOULOS, Mälardalen University, Sweden
VÍTOR E. SILVA SOUZA, Federal University of Espírito Santo, Brazil
JOHN MYLOPOULOS, University of Ottawa, Canada

Self-adaptive software systems monitor their operation and adapt when their requirements fail due to unexpected phenomena in their environment. This paper examines the case where the environment changes dynamically over time and the chosen adaptation has to take into account such changes. In control theory, this type of adaptation is known as Model Predictive Control and comes with a well-developed theory and myriads of successful applications. The paper focuses on modelling the dynamic relationship between requirements and possible adaptations. It then proposes a controller that exploits this relationship to optimize the satisfaction of requirements relative to a cost-function. This is accomplished through a model-based framework for designing self-adaptive software systems that can guarantee a certain level of requirements satisfaction over time, by dynamically composing adaptation strategies when necessary. The proposed framework is illustrated and evaluated through two simulated systems, namely the Meeting-Scheduling exemplar and an E-Shop.

CCS Concepts:•**Software and its engineering** → **Software design engineering; Requirements analysis;**•**Computing methodologies** → **Modeling methodologies;**

Additional Key Words and Phrases: self-adaptive systems; model predictive control, awareness requirements

1. INTRODUCTION

Self-adaptive systems are expected to operate in highly dynamic environments and fulfil multiple goals. When a failure is detected (i.e. a goal is not achieved) due to external disturbances (e.g. high workload or unexpected user behaviour), a new configuration is adopted. Unfortunately, composing an adaptation plan to overcome changes in the environment is a challenging task. The main obstacle is unsolicited interference from other goals. Therefore, an adaptation strategy A could restore the satisfaction of goal G but fail or worsen goal G'.

Control Theory offers well-developed theoretical and practical frameworks for dealing with systems with multiple parameters (inputs) and assigned with multiple goals (outputs). Some approaches for developing self-adaptive software deal with each goal individually without taking into account interferences [Brun et al. 2009; Filieri et al. 2011; Filieri et al. 2014; Souza et al. 2012]. Some others perform reactive adaptation when the failure has already taken place without any provision for the future [Angelopoulos et al. 2014; Cheng et al. 2006; Filieri et al. 2015; Zoghi et al. 2014; Klein et al. 2014], or attempt to anticipate failure. For instance, when the workload grows and a goal fails, additional resources are disposed to the system to overcome the failure. However, if the workload is continuously increasing, it would be wise to dispose more resources than those required for satisfying the goal, in anticipation of future failures. Other approaches [Gaggero and Caviglione 2015; Ghanbari et al. 2014; Kusic et al. 2009; Roy et al. 2011] apply predictive control in the domain of cloud computing to guarantee non-functional properties. To our knowledge, despite its effectiveness, software engineers have been reluctant to adopt predictive control for other domains, given the lack of tools and methodologies to model, in a control theoretic context, software requirements (functional and non-functional) and parameters.

In [Angelopoulos et al. 2016] we introduced the basic components of Model Predictive Control (MPC) for software systems, how these are related to the requirements of the system-to-be, and also proposed a framework that supports the elicitation of

the analytical models required for MPC. Next, we integrated MPC with previous work on requirements engineering for software adaptation in a framework named *CobRA*.[1] Finally, we evaluated our work with a simulation of the Meeting Scheduler exemplar.

This article extends our previous work by applying CobRA to a system where the goals are not constrained by clear-cut thresholds, but need to be optimized. The absence of thresholds contradicts the principle that a controller continuously performs adaptations in order to ensure that the system's output converges to specific values. Therefore, in this work : a) we extend our requirements monitoring mechanism in order to capture optimization goals of certain attributes of our target system, e.g. maximize revenue; b) we propose a systematic approach for representing such goals in the analytical models of the MPC; c) we enhance our evaluation by applying *CobRA* to the simulation of an E-Shop in order to demonstrate that our approach can also handle optimization goals.

The rest of the paper is structured as follows. Section 2 presents the research baseline for this work. Section 3 describes the basic components of MPC. Section 4 describes the *CobRA* framework. In Section 5 we evaluate *CobRA* using a simulation of the Meeting Scheduler and we compare the results with those of *Zanshin* [Souza et al. 2012], another requirements-based adaptation framework. Finally, in Section 6 we compare related work with our proposal, and Section 7 concludes the paper and discusses future directions of this thread of research..

2. BASELINE

Our proposal adopts concepts from both software and systems engineering. The following provides an overview of the baseline from each of these areas.

2.1. Goal Modeling

Goal-Oriented Requirements Engineering (GORE) models elicited requirements as *goals* and analyzes them accordingly. Each goal is iteratively AND/OR-refined to more detailed ones following *Boolean* semantics, until we reach goals that are detailed enough to be operationalized (usually by *tasks*). For the Meeting-Scheduler exemplar, the root goal Schedule Meeting (see Figure 1) is refined into three sub-goals: Collect Timetables (goal G1), Book Meeting (G2) and Manage Meeting (G3). Goal G1 is fulfilled by either contacting the invited participants by phone (task t1), by email (t2) or let the system collect their timetables automatically from the system's calendar (t3). However, the last option is available only if the *domain assumption* that the participants use the system's calendar to register their appointments holds. Next, for goal G2 to be satisfied, goal G4: Find Room must be satisfied either by letting the meeting organizer select a room from the list (t4) or from those suggested by the system (t5). Finally, the meeting organizer should manage the meeting (G3) by confirming its occurrence or cancellation (t7 and t8 respectively), by sending reminders to the participants (t9) and notifying them about any change concerning the scheduled meeting (t10).

While goals and tasks represent the functional requirements of the system, *softgoals* capture desired non-functional properties. Each of the elicited soft-goals is quantified by a *quality constraint* that allows reasoning about their fulfillment at runtime. For instance, soft-goal Low Cost is satisfied when less than 500 euros is spent weekly for organizing meetings. Good Participation is yet another soft-goal, satisfied when 80% of the invitees show up for a meeting.

Monitoring and evaluating requirements satisfaction is critical for self-adaptive systems. Following the same line of work as in [Angelopoulos et al. 2014] we use *Aware-*

[1]Control-based Requirements-oriented Adaptation

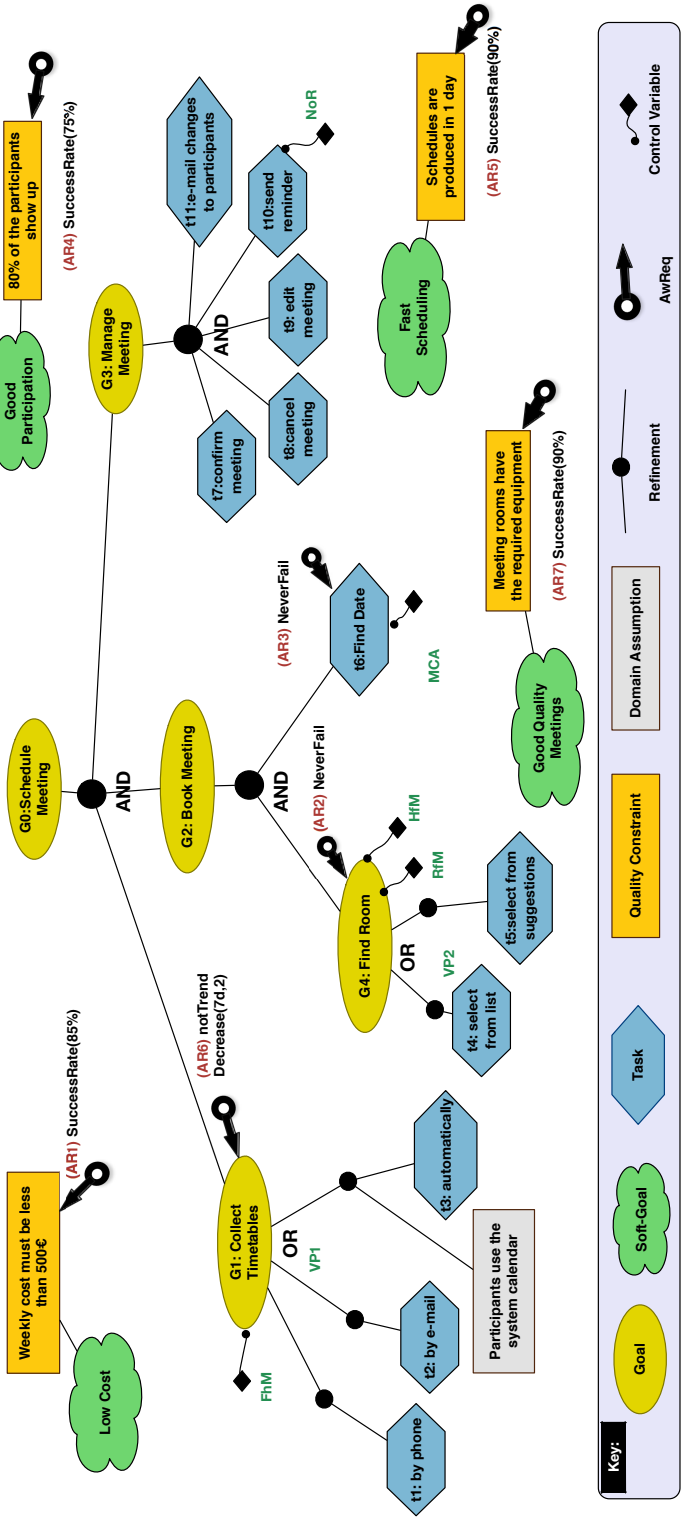

Fig. 1: Meeting Scheduler goal model.

ness Requirements (*AwReqs*) to monitor the success of other requirements. An *AwReq* defines a constraint which triggers adaptation when violated. For example, during an attempt to schedule a meeting, if G4 or t8 fail, a new configuration must be applied ($AR2$). In the same context, if participation is lower than what stakeholders requested more than 25% of the times, a new adaptation is selected ($AR4$).

AwReqs can refer to any element of the goal model. When referring to tasks, they determine if the system has performed a specific set of actions successfully. When monitoring a quality constraint, they determine if the system abides by quality requirements. Pointing to goals/softgoals determines if the system has satisfied a requirement. Finally, *AwReqs* that refer to domain assumptions determine if things that were assumed to be true in the environment indeed are true during system operation. *AwReqs* can also refer to other *AwReqs* (meta-*AwReqs*), creating hierarchical feedback loops for adaptation.

Each *AwReq* is associated with variables named *indicators* that measure the degree of fulfilment for a monitored requirement. Indicator values are influenced by two kinds of parameters: a) *environmental parameters* that are determined by the environment and can't be manipulated by the system and b)*control parameters* that can be adjusted at runtime by the system [Angelopoulos et al. 2015]. Examples of environmental parameters include the number of meeting requests received by the system, also participant availability and punctuality. When the number of meeting requests increases significantly, the value of indicator $I2$ assigned to $AR2$ decreases, since it is harder to find a room. Analogously, when participant punctuality or availability decreases, participation is lower and consequently the value of $I4$ drops. On the other hand, control parameters are tuned by the adaptation mechanism to correct indicator values that are out of range of prescribed thresholds. For instance, the number of participants from whom timetables are collected (FhM) has an impact on how quickly meetings are scheduled ($AR5$) and how good participation is going to be ($AR4$). In the same context, the maximum number of timetable conflicts allowed MCA, the number of reminders sent to the invitees NoR, $VP1$ (collecting timetables by phone, e-mail or automatically), $VP2$ (choosing to find room from a list or use a system suggestion) and the numbers of local and hotel rooms, RfM and HfM respectively, are control parameters the values of which affect the outcome of monitored indicators.

Formal goal modelling approaches such as KAOS [Dardenne et al. 1993] associate goals with objective functions in order to reason about their satisfaction [Letier and van Lamsweerde 2004]. Such objective functions are related to attributes of the system that determine to what level a goal is achieved. In Figure 1 we assume that stakeholders are able to provide clear-cut criteria on whenever a softgoal is satisfied or not. However, it is often the case that stakeholders propose goals that dictate minimization or maximization of certain variables (hereafter optimization goals), e.g. cost, revenue, time etc. We adopt the term quality attribute from [Li et al. 2014] to refer to such variables or equations of them. In Figure 2 the optimization goal 'Maximize Participation' is associated with the quality attribute average participation ($avg\_participation$) in meetings whereas the optimization goal 'Minimize Scheduling Time' is associated to the quality attribute $total\_scheduling\_time = collection\_time + booking\_time$ which captures the sum of required time to collect timetables and book a room. Next, we define a new type of *AwReq*, an Optimization *AwReq* which is defined as Optimize($quality\_attribute$, optimization type [min,max]). The first argument refers to the associated quality attribute and the second defines if this attribute should be minimized or maximized. The value of an indicator of an Optimization *AwReq* is equal to its quality attribute and must continuously be optimized. In the following sections we demonstrate how to use all types of *AwReqs* in order to formulate the requirements satisfaction problem as an optimization problem which is solved with MPC.

(a) Single quality attribute Optimization *AwReq*.

(b) Complex quality attribute Optimization *AwReq*.

Fig. 2: Examples of Optimization *AwReqs*

In contrast to other kinds of *AwReqs*, the thresholds for the satisfaction of optimization *AwReqs* are not explicit. For minimization goals we set as threshold the minimum possible value of the associated quality attribute. For example, the minimum value of cost is set to zero. Similarly, for maximization goals the threshold is set to the maximum possible value of the associated quality attribute. However, if there is no upper limit, the indicator measures the reciprocal value of the quality attribute and the threshold is set to zero. In most cases the system will never reach these thresholds although it will continuously search for a configuration to reach as close as possible to them. This behaviour complies with the self-optimization property according to which self-adaptive systems must *'continually seek opportunities to improve their own performance and efficiency'* [Kephart and Chess 2003].

Another kind of requirement included in our approach is an *Evolution Requirement* (*EvoReq*) [Souza et al. 2013]. These apply when certain conditions hold and replace temporarily or permanently other requirements. These changes are applied through actions named *EvoReq* operations. For example, if $AR1$ constantly fails, probably because the success rate threshold is set too high, it is replaced with a new *AwReq* where the threshold is 75% instead of 85%.

Apart from requirements for the system-to-be, constraints are also imposed on the adaptation process itself, in the form of *Adaptation Requirements* (*AdReqs*) [Angelopoulos et al. 2014]. Examples of such constraints include how much time it should take to restore fulfilment of a failed requirement, or how much is a control parameter allowed to change when its value is modified.

2.2. Dynamic System Modelling

In our previous work [Angelopoulos et al. 2014; Souza et al. 2011], qualitative relations have been used to model the relation between control parameters and indicators. Often, using qualitative adaptation is a necessity, given the lack of quantitative models for software systems. However, in many cases, a sufficiently accurate quantitative dynamic model, can be obtained through system identification techniques [Ljung 1999], and can be used for control design. Letting $u(t) \in \mathbb{R}^m$ be the vector of control parameter values at time t, and $y(t) \in \mathbb{R}^p$ be the vector of indicators, their respective dynamic relation is described as:

$$y_i(t) = \sum_{j=1}^{p} \sum_{k=1}^{n_y} \alpha_{ijk} y_j(t-k) + \sum_{j=1}^{m} \sum_{k=1}^{n_u} \beta_{ijk} u_j(t-k) \quad (1)$$

for all $i = 1, \ldots, p$, and with $\alpha_{ijk} \in \mathbb{R}$, $\beta_{ijk} \in \mathbb{R}$. The quantitative dynamic model (1) relates the values of the indicator y_i at time t with past values of all the indicators

— accounting for possible mutual influences of the indicators — and with past values of control parameters. For example, $I1$ might achieve a high value because of good management of hotel room assignments or because of the constant failure of $I2$. The reason is that if meetings fail to be scheduled, no rooms are reserved and consequently the cost of meetings remains low. Such implicit relationships among indicators can be captured by model (1) to guide the adaptation process. Notice that if some of the mentioned variables are not influencing the value of the indicator $y_i(t)$, then the corresponding parameters are simply zero. An equivalent and more compact representation of this relation is the discrete-time state-space dynamic model:

$$\begin{cases} x(t+1) = Ax(t) + Bu(t) \\ y(t) = Cx(t), \end{cases} \tag{2}$$

where $x(\cdot)$ is a vector named *dynamic state* of the model. While for physical systems, the state $x(\cdot)$ is typically associated with meaningful physical quantities, in general the state can be just an abstract representation of the system, and it is not necessarily measurable. The values of the matrices (A, B, C) fully describe how the inputs dynamically affect the outputs of the system, and these matrices are the outcome of the System Identification process.

The analytical model of Equation (1) shows that the system's output might be related to past outputs and control inputs. Indicators related to aggregate *AwReqs* [Souza et al. 2011] express success rates over time about the satisfaction of an associated goal and, therefore, their current values are naturally bound to their past values and to the values of *AwReqs* that produced them. These are *dynamic systems* in Control Theory. In case no relation with past behaviour of the indicators and of the control inputs is present, A is a matrix with all zero elements, and the system is just mapping inputs to outputs with the *static* relation:

$$y(t) = CBu(t-1).$$

Therefore, the model of Equation (2) accounts for both dynamic and static systems.

Equation (2) can be used to design a control system able to adjust the values of every control parameter, in order to make each indicator converge to the value prescribed by an *AwReq* threshold — under the assumption that the set of chosen control parameters is able to drive the system to the prescribed goals. In contrast to qualitative adaptation, such quantitative models allow one to handle conflicts with precision. For example, an increase of the control parameter MCA results in an increase of $I3$, as it becomes easier to find a commonly agreed timeslot for the meeting, but the participation might drop and consequently $I4$ is decreased. The analytical model can prevent the adaptation mechanism from decreasing $I4$ excessively. Performing such trade-offs on a daily basis while taking into account priorities among indicators based on their business value (higher priority indicators should converge faster than less important ones), and preferences among control parameters (e.g. increasing RfM is preferred to increasing HfM) is a complex process. In the next section we present a control-theoretic approach in order to efficiently implement this process and maintain an equilibrium among conflicting goals.

Notice that software systems are often nonlinear, non-smooth, or even discontinuous [Papadopoulos et al. 2015], and the linear dynamics that are identified are just an approximation of the actual dynamics of the system. Linearizing nonlinear dynamics, is common practice in most control applications [Åström and Murray 2010], and if the control design is carried out correctly, the feedback loop, possibly jointly with some learning strategies, are in charge of handling model inaccuracies, and neglected dynamics in a robust way (see, e.g., [Kothare et al. 1996]). The main challenge when

dealing with computing systems is that obtaining reliable dynamic models is really complicated, since there are no laws of physics, and it is application dependent [Papadopoulos et al. 2015]. This work is part of a long term research line which aim is to automatize the decision making policy design from the elicitation of the system requirements to the actual implementation. The use of model identification techniques ease the construction of the dynamic model, and could be easily included in an automated tool.

3. MODEL PREDICTIVE CONTROL

Based on the dynamic model of Equation (2), different control strategies can be designed. We here extend our previous work [Angelopoulos et al. 2015] and present a *receding horizon* MPC [Camacho and Bordons 2004; Maciejowski 2002] that is able to manage the achievement of multiple conflicting goals by means of multiple control parameters. When the controller is complemented with a Kalman Filter [Ljung 1999], it can learn online how to adapt the controller to the system's behaviour, overcoming inevitable inaccuracies coming from dynamics not captured from model (2) and unknown disturbances acting on the system.

MPC is a control technique that formulates an optimization problem to use a set $u(\cdot)$ of control parameters (actuators) to make a set of indicators $y(\cdot)$ achieve a set of goals $y^{\circ}(\cdot)$ over a prediction horizon H. At every control instant t, the values of the control parameters $u^{\star}$ are obtained by minimizing a cost function J_t, subject to given constraints. The optimization problem includes a prediction of the future behaviour of the system based on the dynamic model (2). An obtained solution is therefore a plan of the future control parameter values $u^{\star} = \left[u_t^{\star}, u_{t+1}^{\star}, \ldots, u_{t+H-1}^{\star}\right]$ over the prediction horizon. This planning is especially needed in the case of delay in the effects of changes of control parameters. For example, increasing the number of hotel rooms requires approval by the administration council that meets only every 2 days. Hence, the adaptation mechanism must be aware of when changes to control parameters impact on the indicators and make look-ahead plans.

According to the receding horizon principle, only the first computed value $u_t^{\star}$ is applied to the system, i.e. $u(t) = u_t^{\star}$. The reason is that for real-world systems, it is impossible to derive perfect models that describe their dynamic behaviour. Therefore, the plan must be corrected at each step and the horizon recedes by one unit. Another reason the plan might fail is a change in the external disturbances (e.g. system workload). In other words, the plan would have been followed as is only if a perfect model were available and no disturbances were present, which in practice is impossible. To tackle this obstacle, at the next control instant, a new plan is computed according to the new measured values of the indicators. This accounts for modelling uncertainties, and possible unpredictable behaviours of the system that are not captured by model (2).

3.1. Formal description

In order to present the underlying rationale of the MPC, it is convenient to rewrite dynamic model (2) in an "augmented velocity form":

$$\overbrace{\begin{bmatrix}\Delta x(t+1)\\ y(t)\end{bmatrix}}^{\tilde{x}(t+1)} = \overbrace{\begin{bmatrix}A & 0_{n\times p}\\ C & I_{p\times p}\end{bmatrix}}^{\tilde{A}} \overbrace{\begin{bmatrix}\Delta x(t)\\ y(t-1)\end{bmatrix}}^{\tilde{x}(t)} + \overbrace{\begin{bmatrix}B\\ 0_{p\times m}\end{bmatrix}}^{\tilde{B}} \Delta u(t)$$

$$y(t) = \overbrace{[C \;\; I_{p\times p}]}^{\tilde{C}} \overbrace{\begin{bmatrix}\Delta x(t)\\ y(t-1)\end{bmatrix}}^{\tilde{x}(t)} \tag{3}$$

Here, $\Delta x(t) = x(t) - x(t-1)$ is the state variation and $\Delta u(t) = u(t) - u(t-1)$ is the control increment. The output of the system $y(t)$ is unchanged, but is now expressed with respect to the state variations $\Delta x(t)$ and not with respect to the state values $x(t)$. The new dynamic model (3) is used as a prediction model over a finite horizon H. This means that the controller will use it to predict what values of the states and of the indicators are going to be after H time steps from the current one. The MPC controller minimizes the cost function

$$J_t = \sum_{i=1}^{H} \left(\left[y^{\circ}_{t+i} - y_{t+i} \right]^T Q_i \left[y^{\circ}_{t+i} - y_{t+i} \right] + \left[\Delta u_{t+i-1} \right]^T P_i \left[\Delta u_{t+i-1} \right] \right),$$

where $Q_i \in \mathbb{R}^{p \times p}$ and $P_i \in \mathbb{R}^{m \times m}$ are symmetric positive semi-definite weighting matrices, that respectively represent the importance of the distance between the goals and the current values and the "inertia" in changing the values of the actuators. In particular, Q_i is a diagonal matrix that contains the values of the set of weights that can be obtained by applying Analytical Hierarchy Process (AHP) [Karlsson and Ryan 1997], in which the stakeholders perform pairwise comparisons to prioritize the elicited goals. This means that when not all the goals are simultaneously feasible (for example because one conflicts with another), the controller will favour the satisfaction of the goals with the higher weights. The matrix P_i captures preferences over control parameters. When a control parameter is requested not to change its value frequently, the assigned weight must be relatively smaller than most of the weights of the other control parameters. In the following we will consider the weight matrix Q as $Q := Q_1 = Q_2 = \ldots = Q_H$, and the weight matrix P as $P := P_1 = P_2 = \ldots = P_H$, i.e., the weight matrices are considered to be constant along the prediction horizon.

The resulting MPC optimization problem can written as follows:

$$\begin{aligned} \text{minimize}_{\Delta u_{t+i-1}} \quad & J_t \\ \text{subject to} \quad & u_{\min} \leq u_{t+i-1} \leq u_{\max}, \\ & \Delta u_{\min} \leq \Delta u_{t+i-1} \leq \Delta u_{\max}, \\ & \tilde{x}_{t+i} = \tilde{A} \cdot \tilde{x}_{t+i-1} + \tilde{B} \cdot \Delta u_{t+i-1}, \\ & y_{t+i-1} = \tilde{C} \cdot \tilde{x}_{t+i-1}, \\ & \quad i = 1, \ldots, H, \\ & x_t = x(t). \end{aligned} \tag{4}$$

This formulation is equivalent to a convex Quadratic Programming (QP) problem [Maciejowski 2002]. The problem has time complexity $\mathcal{O}(H^3 m^3)$ [Wang and Boyd 2010]. A solution to the problem consists of a plan of optimal future $\Delta u^{\star}_{t+i-1}$, $i = 1, \ldots, H$, but only the first one is applied, i.e., $\Delta u(t) = \Delta u^{\star}_t$, as we explained earlier. The new control signal is then:

$$u(t) = u(t-1) + \Delta u(t). \tag{5}$$

The MPC strategy assumes that the state of the system is measurable, but in many cases this is not possible. Indeed, since there is often no correlation with physical quantities, it is impossible to give a meaningful interpretation to $x(t)$, hence it is impossible to measure. However, based on the dynamic model (2), it is possible to estimate its value measuring the values of $y(t)$ and $u(t)$. To accomplish this, we here use a Kalman Filter (KF) that finds an estimate $\hat{x}(t+1)$ of the state $x(t+1)$, measuring the applied control signal $u(t)$ and the output $y(t)$.

$$\begin{aligned} \hat{y}(t) &= C\hat{x}(t) \\ \hat{x}(t+1) &= A\hat{x}(t) + Bu(t) + K\left(y(t) - \hat{y}(t)\right) \end{aligned} \tag{6}$$

Note that the variables of the KF are commonly denoted by a "hat", i.e., $\hat{x}(k)$ and $\hat{y}(k)$, to distinguish them from the variables of the dynamic model (2). Based on the state estimate $\hat{x}(t)$, the KF shown in (6) computes an estimate of the output $\hat{y}(t)$, to measure the difference between the predicted value $\hat{y}(t)$ and the real value $y(t)$. The value of K, called *Kalman gain*, weights the discrepancy between the predicted value $\hat{y}(t)$ and the real value $y(t)$, adjusting the dynamics of the KF [Ljung 1999]. The estimate $\hat{x}(t)$ can be used, in place of $x(t)$, to solve the optimization problem (4).

The adopted KF has a twofold functionality. First, as we just described, based on the dynamic model (2), it computes a state estimate $\hat{x}(t)$ that the MPC uses to compute the next control action. Second, it is adapting the state estimate to the actual behaviour of the system. This is relevant for a number of reasons: the controlled system may change its behaviour over time, there might be unpredictable disturbances acting on the system, or the system is not following the linear dynamics of the dynamic model (2). In all cases, the KF is adapting online the choice of the estimate $\hat{x}(t)$, returning a value that is compatible with the input-output behaviour of the running system, as if it was described exactly by the dynamic model (2) [Ljung 1999].

The block diagram for the resulting control scheme is represented in Figure 3.

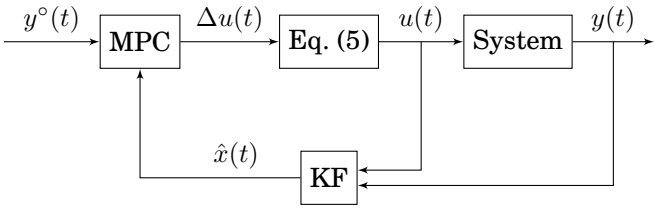

Fig. 3: Reference control scheme for *CobRA*.

3.2. Formal guarantees

Applying Control Theory to software systems provides a set of formal guarantees about the quality of the adaptation process [Filieri et al. 2017]. The MPC adopted in this work belongs to a class of controllers named *optimal controllers*, since the computation of control decisions is based on the solution of an optimization problem. In particular, the MPC accounts for model predictions in order to make optimal adaptation plans with respect to system requirements, and compliance to the requirements about the adaptation process itself [Angelopoulos et al. 2014].

The formal guarantees for the MPC are as follows. First, it is possible to ensure that all the goals are reached, subject to actuators constraints, i.e. there exists a value of the actuators within the given constraints specified in the optimization problem (4) that allow the system to reach its goals. If this is not the case, due to the optimal nature of the controller, the MPC finds a configuration for the actuators that minimizes the distance between the indicators and the goals. Such a distance depends on the chosen weights for each indicator in the cost function of the optimization problem (4).

Furthermore, since the cost function accounts for a time horizon, it is possible to guarantee that the convergence time is minimal. The dynamic model (2) relates control parameters and indicators including the dimension of time. Therefore, the adaptation

mechanism is able to drive the system to the goals as soon as possible, as specified by the cost function of the optimization problem (4). Moreover, the optimization problem (4) can be easily extended in order to account for additional constraints, such as, for example, constraints on the indicators [Camacho and Bordons 2004]. *AwReqs* and *AdReqs* impose such constraints over the elicited goals and the adaptation process respectively which must be taken into consideration when a new adaptation plan is produced.

The MPC formulation is well suited for addressing also real-time issues and have been applied to various domains, such as aircraft control [Qin and Badgwell 2003; Hartley et al. 2014a]. Since the proposed solution requires a solution to an optimization problem at each control instant, it is critical to discuss possible such real-time issues. In many cases, in fact, the time required for computing the value of the next control action might be longer than the time between two subsequent control actions. In order to overcome this challenge, there is significant literature in the control community on how to implement fast solvers [Jerez et al. 2012; Giselsson 2014], especially for embedded systems [Jerez et al. 2014], possibly co-designing also a dedicated hardware for the solution in case of time-critical systems [Hartley et al. 2014b]. An overview on the matter can be found in [Zeilinger et al. 2014].

In many cases, such advanced algorithms are not required when dealing with software components, and for the most time-critical applications some modification to the control problem can help in reducing the complexity. For example, one way to reduce the complexity is to set $\Delta u_{t+1} = \Delta u_{t+2} = \ldots = \Delta u_{t+H-1} = 0$, and let the optimization problem decide only the value for Δu_t, i.e., the one that will be actually applied to the system. This modification reduces the complexity to $\mathcal{O}(m^3)$.

Another way to deal with real-time issues is to exploit simple properties of interior point algorithms. In fact, the solution is obtained in a fixed amount of steps with an iterative method. The current solution is always a suboptimal, yet feasible solution to the optimization problem. This means that if the iterative method did not converge before a new control action is required, it can be forced to stop and return the current sub-optimal solution. This allows the controller to fulfil real-time deadlines.

Finally, another possibility to deal with real-time deadlines is exploiting the proactive nature of the MPC. As we mentioned earlier, the MPC is computing at each iteration step a plan of future actions Δu_{t+i-1}, $i = 1, \ldots, H$, then according to the receding horizon principle, only the first one is applied, i.e., $\Delta u(t) = \Delta u_t^\star$. Assuming that at the next control instant, the solver takes more time to converge and that a new control action is required before the optimal solution is found, one can store the previously computed plan and apply the second control action, i.e., $\Delta u(t+1) = \Delta u_{t+1}^\star$. This is obviously suboptimal, since it neglects the last information about the measured output, but it is able to fulfil the real-time deadlines.

4. APPROACH

Our approach involves two phases: design and runtime. During the first phase all models required for the MPC controller's synthesis and tuning are elicited, whereas during the second phase the controller is deployed in our adaptation framework and adjusts the control parameters of the target system when required.

4.1. Design phase

Our approach starts with the elicitation of all kinds of requirements about the target system. When all goals are refined, *AwReqs* are assigned to those that are considered most critical and prone to failure. An *AwReq* AR_i defines a reference goal $y_i^\circ(\cdot)$ for the system's output. For instance, Table I enlists all the reference goals for the Meeting-Scheduler exemplar.

Table I: Reference goals

AwReq	$y^{\circ}(\cdot)$
$AR1$	$y_1^{\circ}(\cdot) = 85$
$AR2$	$y_2^{\circ}(\cdot) = 100$
$AR3$	$y_3^{\circ}(\cdot) = 100$
$AR4$	$y_4^{\circ}(\cdot) = 75$
$AR5$	$y_5^{\circ}(\cdot) = 2$
$AR6$	$y_6^{\circ}(\cdot) = 90$
$AR7$	$y_7^{\circ}(\cdot) = 90$

As we mentioned earlier, the constraints imposed by *AwReqs* are not always feasible or might become infeasible in the future. For instance, prices of hotel rooms rise every year and consequently $I1$ will fail more often as time passes. In addition, during summer prices are usually higher. Hence, stakeholders could accept a lower success for $I1$ (in other words $y_1^{\circ}(\cdot) < 85$). At this step of the design phase, the domain experts, along with the stakeholders, analyze and evaluate such conditions and specify *EvoReqs* for the system-to-be. The *EvoReqs* operations defined for the *AwReqs* of the Meeting-Scheduler are presented in Table II.

Table II: *EvoReqs* operations

AwReq	*EvoReq* operation
$AR1$	1. Relax($AR1$,$AR1'\_75$) 2. Strengthen($AR1$,$AR1'\_85$)
$AR2$	Relax($AR2$,$AR2'\_90$)
$AR3$	Relax($AR3$,$AR3'\_90$)
$AR4$	1. wait(3 days) 2. Relax($AR4$,$AR4'\_75$)
$AR5$	Replace($AR5$,$AR5'\_3$)
$AR6$	wait(3 days)
$AR7$	wait(2 days)

When summer season begins and hotel prices are higher, the first *EvoReq* operation is triggered, relaxing the reference goal from 85% to 75%. The second *EvoReq* operation is triggered when summer season ends and the threshold is restored to its previous value. Similarly, when $AR2$ and $AR3$ fail for more than 2 days in a row, the reference goals are relaxed for a week[2]. In case of $AR5$, when goal $G1$ tends to fail more than 2 times/week, the constraint is permanently replaced by 3 times/week. Finally, when $AR6$ and $AR7$ fail for more than 2 days the adaptation mechanism ignores them for 3 and 2 days respectively.

Next, by applying AHP, weights are elicited for each indicator to capture their relative importance. As a rule of thumb, indicators assigned to functional requirements have higher priority compared to non-functional ones. These weights are the values of matrix Q of the cost function. The controller, through the optimization function, finds an equilibrium for every goal, putting more effort on fixing the most important ones. As for the control parameters, their weights are empirically elicited assigning lower weights to the control parameters we want to be tuned less often. These weights are the values of matrix P of the cost function. In our exemplar, for instance, increasing

[2]The relaxation duration and the triggering condition are prescribed by the stakeholders.

the number of rooms RfM is preferred over HfM since it is a less costly solution, does not require any authorization and, therefore, takes effect immediately. The elicited priorities for the indicators of Meeting-Scheduler and the weights for control parameters as shown in Table III and Table IV, respectively.

Table III: Indicator Priorities

Indicator	Priority
$I1$	0.15
$I2$	0.3
$I3$	0.3
$I4$	0.06
$I5$	0.2
$I6$	0.05
$I7$	0.04

Table IV: Control Parameter weights

Control Parameter	Weight
FhM	1
MCA	1
RfM	1.2
HfM	0.6
NoR	1.2
$VP1$	0.8
$VP2$	1.4

The last set of requirements to be elicited are the *AdReqs*. These requirements impose constraints to the adaptation process itself. For the particular case of MPC, an *AdReq* specifies the receding horizon of the controller and, consequently, how far in the future the adaptation plan should target. Other *AdReqs* might refer to the magnitude of allowed change of control parameters. For instance, HfM cannot be increased more than 5 units each time.

Finally, a quantitative model such as the one in Equation 2 must be derived. Given the absence of laws of nature over this particular domain, we ran a long simulation of the meeting scheduler system during which the control parameters change often and both control input and output are recorded. With the aid of Matlab and System Identification toolbox[3] we estimate the analytical model of the system. Even if the system-to-be cannot be simulated accurately, the model can be improved later on, when the real system is deployed, by means of a learning mechanism during the runtime phase.

4.2. Runtime phase

When the design phase is completed and the system is implemented, the *CobRA* (Control-based Requirements-oriented Adaptation) framework can be deployed and play the role of the adaptation mechanism. *CobRA*, depicted in Figure 4, has five main components. The monitors and the actuators that integrate *CobRA* with the target system are application specific and must be implemented by the designers of the system.

[3]http://it.mathworks.com/products/sysid/?requestedDomain=www.mathworks.com

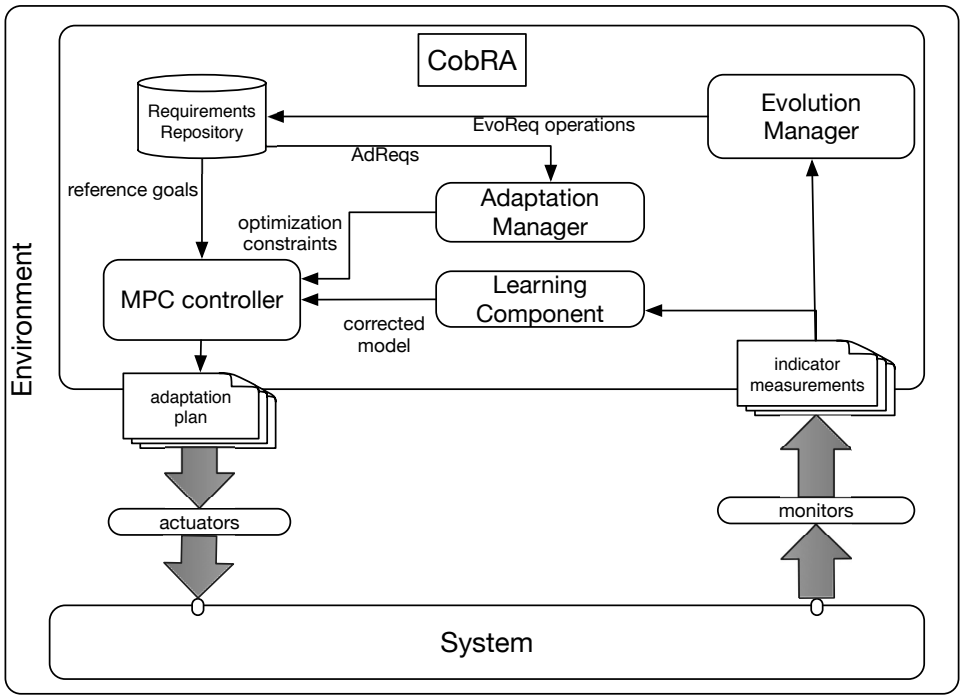

Fig. 4: Scheme of the *CobRA* framework.

Requirements repository. This repository stores all the models produced during the design phase and provides information to the other components of the framework when requested.

Evolution manager. This component analyzes the logs provided by the monitors in order to identify conditions that would trigger *EvoReq* operations. If a requirement is replaced either permanently or temporarily, it updates the requirements repository.

Adaptation manager. This component translates *AdReqs* to constraints for the optimization problem of Equation 4. Such constraints are related to maximum allowed decrease or increase of a control parameter in a single step and the weights of all indicators and control parameters (matrices Q and P).

Learning component. Black-box system identification does not always provide precise models about the system's behaviour. Therefore, we include in our framework a learning component that, based on the applied changes and the outcome values of indicator that occurred as a result of these changes, revises the control law to adapt to changes of the behaviour of the system. More specifically, this component is an implementation of the Kalman Filter as described in the previous section.

MPC controller. The details of this component have been discussed in the previous section. Summarizing its functionalities, the MPC controller requests the requirements repository for the reference goal $y^{\circ}(\cdot)$ of each indicator monitored. It then calculates the distance of each indicator from its respective reference goal and composes an adaptation plan that minimize every distance taking into account the indicator priorities in order to restore equilibrium, subject to the given constraints on the control parameters. The plan includes changes to control parameters in a predefined horizon. For example, the indicators of the Meeting-Scheduler are evaluated daily and the plan

includes values for control parameters so that indicators minimize their distance from $y^{\circ}(\cdot)$ for the next three days. If two days after the plan is applied the result is not what was expected, e.g., because the number of meetings constantly grows, the controller produces a new plan which tries to anticipate future failures in a receding horizon fashion.

The iterative adaptation process with *CobRA* includes the following steps:

— **Step 1:** The monitors collect the measurements of all the indicators of the system;
— **Step 2:** The *Evolution Manager* examines if any event that would trigger an *EvoReq* operation is present and, in that case, updates the evolved requirement in the Requirements Repository;
— **Step 3:** The Adaptation Manager provides the MPC controller with the weights for the indicators and control parameters, as well as constraints for the optimization problem;
— **Step 4:** The Learning Component provides the MPC with a corrected model of the system based on the recent measurements;
— **Step 5:** The MPC controller given the current reference goals provided by the Requirements Repository, and the corrected model produces a revised adaptation plan with the target each indicator value to converge to the reference goal within the prediction horizon;
— **Step 6:** The actuators apply the first step of the plan to the system.

It is important to mention that if an a new requirement is introduced or an older one is removed the design phase must be repeated in order to derive a new analytical model.

5. EVALUATION

In the previous sections, we provided the basic background for the structure and functionalities of an MPC controller. We also presented the *CobRA* framework, which exploits stakeholder goals and uses an MPC controller to compose dynamically adaptation plans when requirements are not met. In this section we evaluate and compare *CobRA* with *Zanshin* [Souza et al. 2012], which also has stakeholder requirements as its baseline for adaptation and adopts concepts from Control Theory.

5.1. Methodology

We have conducted our experiments with a simulation of the Meeting-Scheduler exemplar[4] another for the E-Shop exemplar[5], both implemented in Python and ran on a computer with an Intel i5 processor at 2.5GHz and 16GB of RAM. We stress tested both our simulations for long periods while modifying all control parameters in order to cover all the range of their potential values.The result of this process is a log file for each application with all the values of the inputs and outputs of the system at every step. Then, we executed a Matlab procedure from the Matlab System Identification Toolbox in order to estimate an analytical model that describes each system's behaviour as it is described in Section 2.

After acquiring the system's quantitative model, we stress-tested the simulation by modifying various environmental parameters such as the system's workload. At this phase, we tune the controller by modifying the weights of the outputs and the inputs. If an indicator, especially a not very important one, constantly overshoots, its weight must be reduced. Similarly, there is a control parameter that we do not wish to change often, such as the number of hotel rooms available, its associated weight is increased.

[4]https://gitlab.com/konangelop/it.unitn.disi.konangelop.simulations.meeting_scheduler_v2.git
[5]git@gitlab.com:konangelop/E-Shop-TAAS.git

As a rule of thumb the user must keep the weight values in the same magnitude. Moreover, the order of the modified weights of the indicators must be compliant to the one the stakeholders provided. This process is iterative and requires multiple executions of the simulation. When the MPC controller reaches a desired behaviour, the tuning phase is completed.

Zanshin, as opposed to *CobRA*, does not involve any quantitative models, but only qualitative relations between inputs and outputs based on human expertise. For example, it is known by the domain expert that by increasing MCA the value of $I3$ increases. For our experimentation we used the default adaptation algorithm of *Zanshin* as described in [Souza et al. 2012]. When a failure arrives, *Zanshin* randomly selects a control parameter that will improve this failure and increases or decreases it by a predefined amount. Therefore, we provided such qualitative information to *Zanshin* based on a previous study of the Meeting-Scheduler [Angelopoulos et al. 2014; Souza et al. 2011].

For the evaluation and the comparison of the two frameworks, we put the simulated systems under a stress-test and we compare the behaviour of the outputs in each case. We also compare the values of the the cost-function described in Section 3 through time for both frameworks, comparing which minimizes it most. The selection of *Zanshin* for the purposes of our evaluation is based on two reasons. First, it uses the same requirements-based monitoring mechanism (*AwReqs*) as *CobRA* and, therefore, customization of the adaptation problem was not required. Second, *Zanshin* decides adaptation plans based on qualitative information provided by domain experts, while *CobRA* uses an automatically derived quantitative model that captures the dynamics of the system.

5.2. Meeting Scheduler

The Meeting-Scheduler application receives daily a number of meeting requests. Once the timetables are collected, a date for each meeting must be found. The result of the finding date process is pseudorandom, given that it depends on control and environmental parameters that change based on stochastic processes we have encoded in the simulation. For instance, as the availability of the participants drops, the more often the goal *Find Date* will fail. Similar pseudorandomness has been encoded for other goals such as *Find Room* and *High Participation*. For our experiment, we run the simulation for 60 steps (simulation days), during which the number of meeting requests gradually increases and then decreases along with participants availability. Due to space limitation, we present the results only for indicators $I1$–$I4$.

Figure 5 depicts the values of the indicators at each step of the simulation. In particular, we run the simulation using multiple different values for the horizon of the MPC. As the number of meetings grows the cost for the system increases as well, resulting in a decrease of indicator $I1$. However, *CobRA* manages to recover by preferring local rooms over hotel rooms as can be seen in Figure 6, whereas *Zanshin* fails to recover from the failure. Moreover, *CobRA* adapts faster for higher values of the horizon. This is also captured by Figure 6, where the pace of decreasing hotel rooms and increasing local rooms is faster for higher values of the MPC's horizon. Concerning indicator $I2$, *CobRA* converges almost immediately, whereas *Zanshin* requires considerably more time. The reason of the delay is that *Zanshin* increases its control parameters by a fixed amount rather than basing it on the magnitude of failure as *CobRA* does. In the case of $I3$, the human expertise provided to *Zanshin* matched the identified relation we derived experimentally for *CobRA*, since the two frameworks achieved almost identical values. Both frameworks decreased the value of FhM which results in decreasing the participation if the punctuality of the participants drops. This is what happens the period between the day 30 and 40 which causes *Zanshin* to fail reaching the expected

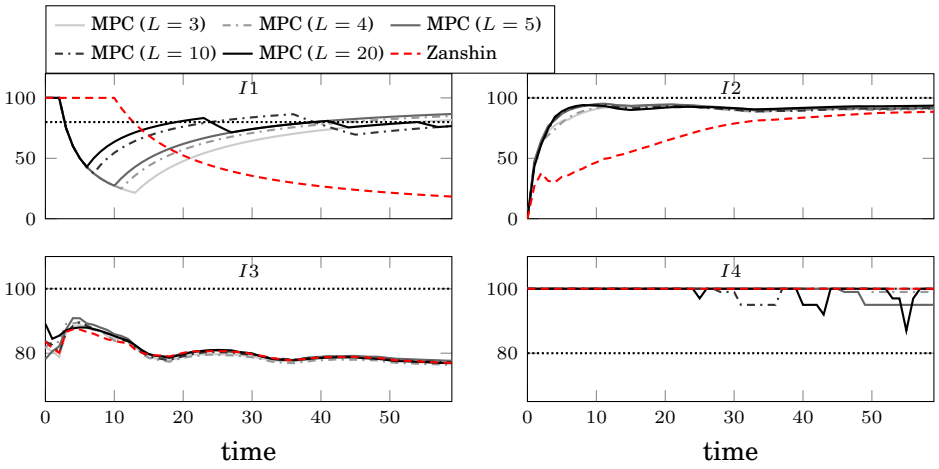

Fig. 5: Indicator measured values for the meeting scheduler exemplar.

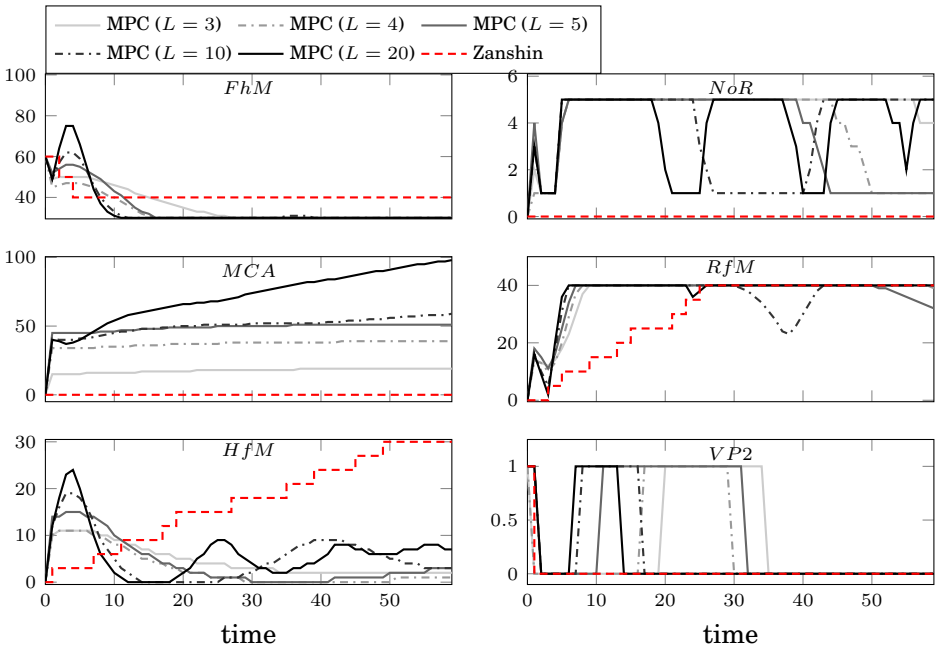

Fig. 6: Control parameter values for the meeting scheduler exemplar.

threshold for indicator $I4$. On the other hand *CobRA* responds to the decrease of punctuality by increasing NoR more than the latter. Finally, for $I2$–$I4$ the impact of the horizon value is not significant since, these indicators are changing at a slower pace than $I1$ and therefore, even with a short prediction horizon the system adapts in the same manner.

CobRA handles the adaptation problem, i.e. continuously search for values for the system's control parameters in order to minimize all control errors with respect to the priority of the associated indicator and the effort required to change these values, as a

multi-objective optimization problem. Therefore, as a metric to evaluate our approach we use the the cost-function $\hat{J}(t) = (y^{\circ}(t) - y(t))^T Q(y^{\circ}(t) - y(t)) + \Delta u(t)^T P \Delta u(t)$. The value of this cost function is calculated from the measured values of the indicators and the changes performed over control parameters. In Figure 7 we present the individual value of the cost function at each step of the simulation, at the top, and the cumulative cost $\sum_t \hat{J}(t)$, at the bottom. $\hat{J}(t)$ captures business value loss because of failing indicators and adaptation costs for changing control parameters. *CobRA* minimizes more at each step of the cost function and by the end of the simulation it produces more stable results. On the other hand, *Zanshin*'s adaptation results in higher losses at most steps, while the accumulated value of the cost-function is growing monotonically. Furthermore, for higher values of the horizon, *CobRA* presents better results compared to short horizons by minimizing more the cost-function. In this example, we can see that different values of the time horizon provide some differences in the obtained system behaviour. In particular, we can observe that for higher values of the horizon, the system exhibit better performance. Recall, however, that the solution of the optimization problem for the MPC formulation scales as L^3, and very high values of the prediction horizon might lead to intractable computation times. When tuning the controller one then needs to find a tradeoff between computation time, and obtained performance, also considering that the cost associated with the minimized cost function monotonically decreases while increasing the prediction horizon. The minimization of the cost over the simulation is highlighted in the cumulative cost showed in the bottom graph of Figure 7.

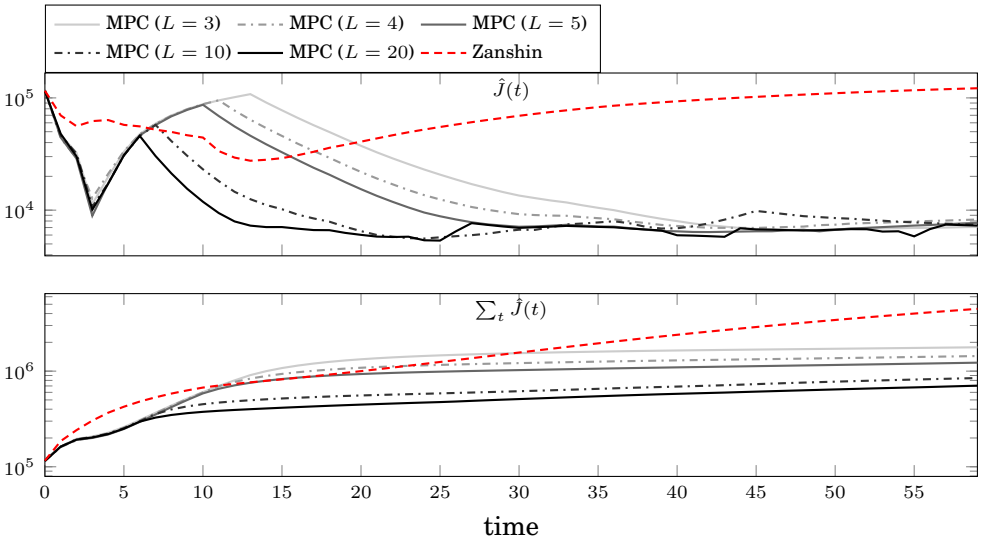

Fig. 7: Adaptation cost for the meeting scheduler exemplar associated with the cost function J for the considered adaptation strategies. The top graph shows the instantaneous cost, while the bottom graph shows the cumulative cost.

5.3. E-Shop

The E-Shop application consists of three simulated components: a) a Load Balancer; b) a server pool; and c) a traffic generator. The traffic generator produces a trace of arrivals per minute for the entire duration of the simulation, which is decided by the user.

Every minute the Load Balancer receives new arrivals and distributes them equally to the active servers of the pool. The purpose of an E-Shop is to sell and advertise products (G1 and G2 respectively), as depicted in Figure 8. Satisfying G1 requires to handle requests (G3) for loading the E-Shop's page, provide a product search functionality (t4) that allows customers to better navigate through the catalogue and display the selected products (G5) either including multimedia information, e.g. videos (t2), or showing only textual description (t3). Next, the customer orders must be handled (G4) by receiving the payment (t5), shipping the order (G6) and printing an invoice (t8). For shipping an order, a courier company must be selected (t5), provide tracking to the customer (t6) and eventually deliver the order (t7). Finally, the E-Shop also displays ads (t9) and sends advertisement e-mails to its customers (t10).

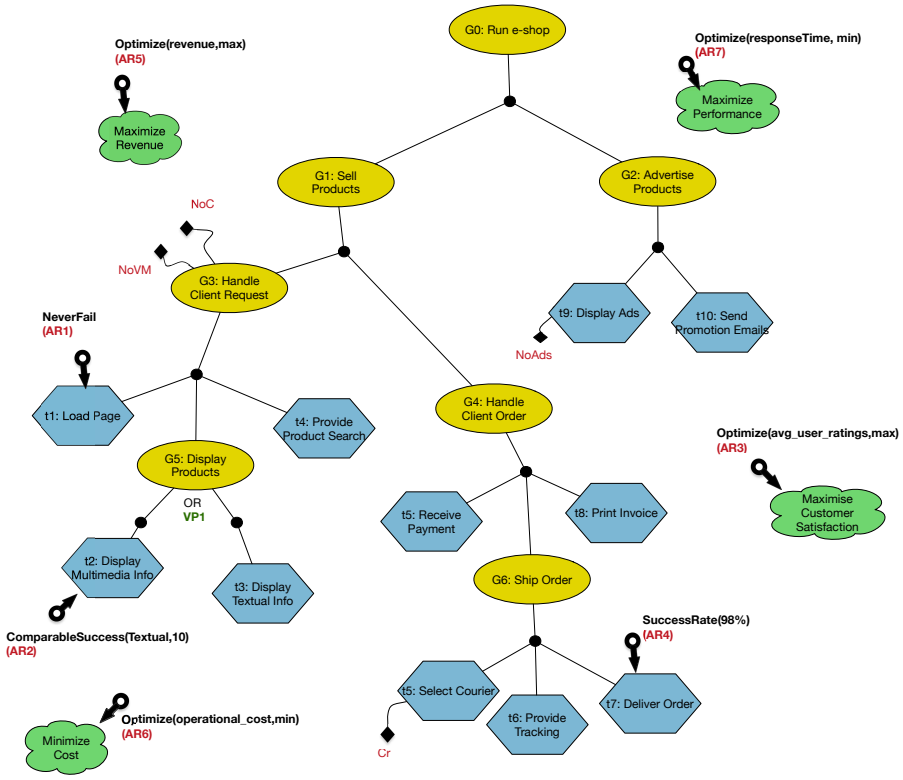

Fig. 8: E-Shop goal model.

We have identified seven *AwReqs* for the E-Shop application. First, the servers must never fail to load the websites page (AR1). Next, the system must display products with multimedia information ten times more than in textual description only form (AR2) and the order delivery must be successful 98% of the times (AR4). Finally, there are three Optimization *AwReqs*: minimize the system's response time (AR7); maximize the number of sales (AR5); and minimize the operational cost (AR6). The thresholds of the reference goals related to these *AwReqs* are presented in Table V whereas the priorities of their associated indicators are presented in Table VI.

To control the indicators of our system we elicited five control parameters. The number of virtual servers (NoS) deployed which host the content of the E-Shop website and

Table V: Reference goals

AwReq	$y^{\circ}(\cdot)$
$AR1$	$y_1^{\circ}(\cdot) = 100$
$AR2$	$y_2^{\circ}(\cdot) = 10$
$AR3$	$y_3^{\circ}(\cdot) = 0$
$AR4$	$y_4^{\circ}(\cdot) = 98$
$AR5$	$y_5^{\circ}(\cdot) = 0$
$AR6$	$y_6^{\circ}(\cdot) = 0$
$AR7$	$y_7^{\circ}(\cdot) = 0$

Table VI: Indicator Priorities

Indicator	Priority
$I1$	0.1
$I2$	0.05
$I3$	0.15
$I4$	0.1
$I5$	0.05
$I6$	0.15
$I7$	0.45

the number of cores (NoC) assigned to each of them are two of these parameters. The response time, the operational cost and the probability of failing to load the website's page are highly influenced by NoS and NoC. The response time is also influenced by the number of ads ($NoAds$) that are displayed on the webpage, since they pose a performance overhead. Another control parameter is the major courier company (Cr) the E-Shop chooses for assigning the 75% of its orders whereas the remaining 25% is equally distributed to other companies. In our application we have three courier companies, each of them having a different level of reliability as it concerns the success of delivery and different cost (the higher the cost, the more reliable the courier is). Our last control parameter is $VP1$, i.e., the choice between showing multimedia content or only textual information for the product description. Using continuously multimedia content might guarantee the satisfaction of AR2 but when the website is under stress it has negative impact on the response time. Table VII presents the weights of the control parameters, derived during the tuning phase of the controller. Finally, we elicited to ***EvoReqs***, presented in Table VIII for relaxing the threshold of AR4, during holiday periods, when all courier companies are dealing with high workload and strengthening it back to its initial value when these periods end.

Table VII: Control Parameter weights.

Control Parameter	Weight
NoS	1
NoC	0.8
$NoAds$	1.2
Cr	0.6
$VP1$	0.8

For our evaluation we run twice our the E-Shop for 800 simulation minutes with simulated traffic as presented in Figure 9. The first time we applied *CobRA* and the

Table VIII: *EvoReqs* operations.

AwReq	*EvoReq* operation
$AR4$	1. Relax($AR4$,$AR4'\_90$) 2. Strengthen($AR1$,$AR1'\_98$)

second *Zanshin*. We monitored AR6 and AR7 that are related to the optimization goals Minimize Cost and Maximize Performance respectively. The control parameters used for optimizing these goals are NoS, NoC, $NoAds$ and $VP1$ which takes the value zero if textual mode is selected or the value one if multimedia content is served. In addition, the virtual servers require two to three minutes for deploying. There are two *AdReqs* that constrain our problem a) the $NoS \geqslant 4$ and b) $NoC \geqslant 8$. In Figure 10 are presented the output results of both frameworks for each goal and in Figure 11 the corresponding inputs values. The indicator of AR7 is measured through the system's response time (milliseconds) whereas the one of AR6 through the operational cost (dollars/minute). In Table VI it is shown that the system's performance is more important than the cost and therefore, *CobRA* when the traffic increased, reacted immediately by adding more servers and cores, while removing ads and switching to textual mode. On the other hand, *Zanshin* failed to deal with with the increased traffic since it tried interchangeably to minimize cost and response time and as a result adaptation actions at each step cancelled the effect of adaptations in the previous one. Our observations are also confirmed by Figure 12 where the cost function has considerably lower value in the case of MPC compared to *Zanshin*.

During our trials we did not succeed in identifying an accurate model that would include all the inputs and outputs of our system. The reason is that some goals and consequently the related output, such the success rate of order deliveries (AR4) and the maximization of customer satisfaction (AR3) vary much more slowly than response time and cost. In our future work we plan to deal with this problem by grouping the goals based on how fast their values change and construct separate controllers for each group. Each controller of course needs to be aware of the adaptation actions produced by the others. This type of control is known as Hierarchical Control [Findeisen et al. 1980].

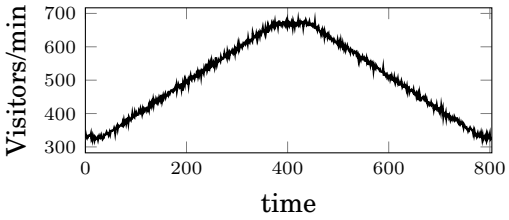

Fig. 9: Active visitors per minute for the performed experiments.

5.4. Discussion

From the experimental results we can safely claim that *CobRA* can produce adaptation plans that allow the system to recover faster from failures while maintaining an equilibrium among conflicting requirements. Moreover, our framework outperforms the qualitative adaptation of *Zanshin* in most cases and confirms that Control Theory can be applied to generic software systems such as the Meeting-Scheduler exemplar.

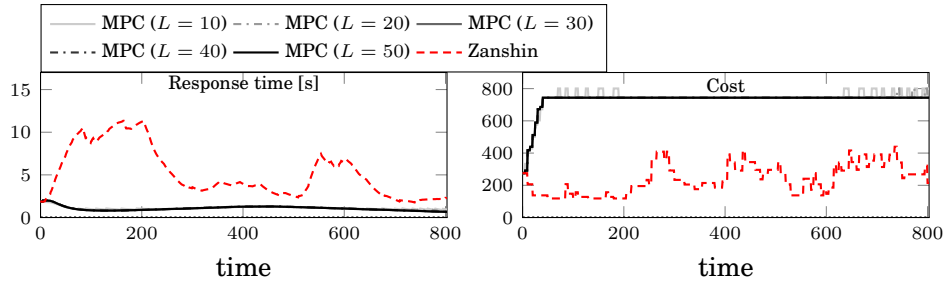

Fig. 10: Indicator measured values for the E-Shop exemplar.

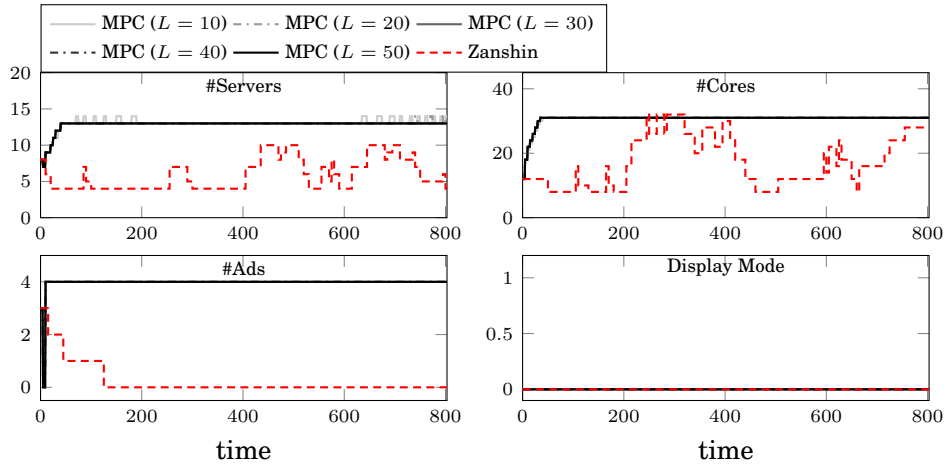

Fig. 11: Control parameter values for the E-Shop exemplar.

Another notable feature of *CobRA* is that it managed to adapt even if the underlying system has nonlinear behaviours. The simulators we used, in fact, include also nonlinear relationships between inputs and outputs, to obtain more realistic behaviours. In practice, most systems have input-output relations that are nonlinear and, therefore, it is important for an adaptation mechanism to handle model imperfections efficiently. In particular, the Kalman Filter contributes to correcting the model as the system runs, allowing MPC to make more accurate predictions.

For both our simulations, a linear model was sufficient for predicting the system's behaviour. However, this might not be always the case. For systems with nonlinear dynamics, either tailored models can be used [Papadopoulos et al. 2015], or more advanced system identification techniques are available [Ljung 2010; 1999], and nonlinear MPC formulations can be adopted [Allgöwer and Zheng 2000]. As future work, we intend to evaluate further our approach using more complex systems, identify their particularities and apply variations of MPC to deal with them.

The experiments conducted that fed the identification procedure, were designed similarly to [Filieri et al. 2014], i.e., they required to span the actuator values over their range. Due to the combinatorial nature, the experiment may take quite long in order to test all the possible combinations. This issue can be mitigated, since it is sufficient to select a subset of the values of the control parameters' domain. In practice, such an approach provides a viable way to perform experiments in a reasonable amount of

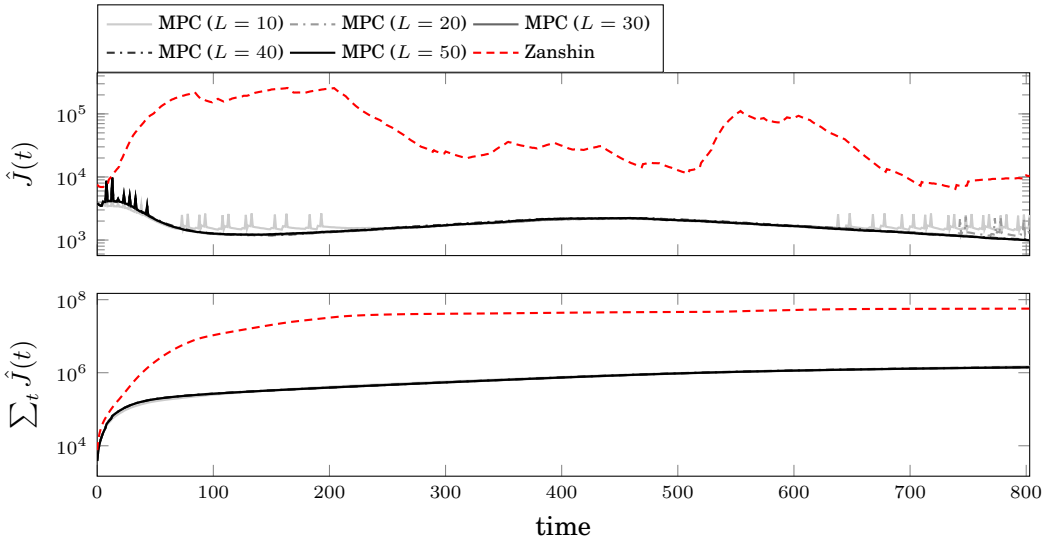

Fig. 12: Adaptation cost for the E-Shop exemplar associated with the cost function J for the considered adaptation strategies. The top graph shows the instantaneous cost, while the bottom graph shows the cumulative cost.

time. Notice that this issue appears only during the *offline* identification phase, and not when the decision has to be taken at runtime. The same issue appears in other adaptation strategies, but at runtime [Moreno et al. 2015; Moreno et al. 2017], where the presence of discrete variable makes the formulation of the optimization problem generally NP-hard.

A main drawback of *CobRA* is that it requires an accurate simulation or historical data of the system in order to derive the analytical model it needs to operate. This is not always possible since for software systems there are no methodologies yet, as for physical systems, to guide system designers as they simulate a model that can produce data sufficiently similar to those of the real system. It is out of the scope of this paper to evaluate the accuracy of a simulated system, metrics that will allow designers to understand how realistic is their simulation are crucial for the applicability of our framework. Therefore, in our future work we plan to investigate new methodologies for producing realistic simulations for software systems.

Another challenge of our approach is tuning the Kalman filter noise covariances and the MPC cost weights which is an empirical process. In the literature of Control Theory, this is an open problem as studied by [Tran et al. 2014; Ionescu et al. 2016; Liu and Wang 2000]. In addition, the automatic tuning of the controller often depends on the physical laws that describe the model of the system. However, in our work, the model is identified from data produced by a simulation, thus just an approximation of the actual dynamics, which is the reason that we apply a Kalman filter. The Kalman filter covariance matrix is selected just as an initial guess, but it is adapted online on the basis of the actual behavior of the system. Therefore, the selection of such parameters affects only the initial transient. As it concerns the weights of the optimization problem, we use AHP to elicit initial values and then, while performing trials, we refine them in order to trade off performance and robustness of the controlled system.

Another open challenge for our approach is the automation of the design and implementation of controllers [Tran et al. 2014; Ionescu et al. 2016; Liu and Wang 2000].

However, developing methodologies for software engineers, as well as establishing guidelines for tuning the MPC parameters and facilitating the controller design, are part of our future research agenda. Another limitation of *CobRA* is that when not all goals change at the same pace an accurate analytical model cannot be identified easily and Hierarchical Control is required.

Finally, it is worth noting that some control parameters of our exemplar, such the number of rooms, are discrete, but according to Equation 1 control parameters are continuous variables. In Control Theory this problem is known as the actuation design problem. There are two different approaches to handle it: a) using rounding of the continuous variable computed by the MPC; and b) adopting a Pulse Width Modulation-like policy [Maggio et al. 2012]. The second approach, during the period between two adaptations, applies interchangeably the rounded-up and the rounded-down value of the continuous variable, which makes it not suitable for our exemplar and therefore we adopted the first one.

6. RELATED WORK

In the field of self-adaptive systems, a variety of approaches uses optimization techniques in order to accommodate conflicting requirements during the adaptation process. Rainbow [Cheng et al. 2006] uses Utility Theory: the adaptation strategies are defined at design time based on human expertise and the optimal one is selected at runtime to maximize the over utility of the system. In the same line of work, [Cámara et al. 2015] propose the use of Probabilistic Model Checking in order to compose strategies dynamically. Our approach differs to theirs as they require precise knowledge of how each control parameter of the system influences its output, such as adding one server improve response time by one second. In systems with dynamic environments such relations might change over time and are not always linear. *CobRA*'s MPC uses the derived analytical model to reason about the impact of changing a control parameter instead of human experience and overcomes non-linearities by applying a Kalman Filter.

[Zoghi et al. 2014] propose the use of Search Based Optimization. This approach, similarly to ours, provides a set of control parameters elicited using a goal model. Then, control parameters that affect non-functional requirements in the same manner are ranked by the designers of the system and the controller uses a search algorithm to find a solution that maximizes the total obtained utility. Human expertise can contribute to improving software adaptation by making optimal selections. However, the use of analytical models as the one *CobRA* uses allow better precision and performance of the adaptation process.

[Sykes et al. 2010] assign utility properties to all components of the system. Then, based on the component availability, the satisfaction of non-functional requirements and the component dependencies, the adaptation mechanism selects an architectural configuration. Compared to our work, the dependencies among control parameters can be modelled as constraints in the optimization problem that has to be solved by the MPC controller each time the system must adapt.

Another Requirements-based and control theoretic approach is presented in [Chen et al. 2014]. In this work, the authors propose the use of a PID controller that finds a different configuration over a goal model that captures the system requirements. A SAT-solver is used to find the best configuration based on goal preferences. When soft-goals are not met, the controller tunes the values of the assigned preferences so the SAT-solver can find a better configuration. Even though the approach offers an interesting way of controlling the goal selection process, the output evaluation is limited to a range of values *Satisfied-Denied*. In our approach we monitor the success of

our requirements using *AwReqs* and indicators that provide measurements of higher granularity and allow more precise adaptation.

Recently, an automated solution to introduce the use of control for software systems in a seamless way was proposed in [Filieri et al. 2014]. This solution treats Single Input and Single Output (SISO) systems by varying a single input and measuring the output. The solution builds on a simple and qualitative dynamic model which is identified online. More precise, yet complicated models can be used at the cost of a higher overhead at runtime [Aleti et al. 2013]. The solution in [Filieri et al. 2014] works only for SISO systems, while the case of Multiple Inputs and Multiple Outputs (MIMO) cannot be addressed within that framework. A possible way to deal with MIMO systems is treated in [Filieri et al. 2015] where the MIMO control is obtained as an automated synthesis by composing SISO controllers in a hierarchical way. The approach presented in [Filieri et al. 2015] is a more modular approach with respect to the one proposed in this paper. However, it has the limitation that the influence of different control parameters on the indicators is not included in the model and it is treated only coupling a single control parameter to a specific indicator. The approach presented in this paper includes all the mutual influences in the single model used for the control design. This can be exploited when deciding the values of the control parameters in order to obtain a better adaptation plan.

Additionally, the authors in [Shevtsov and Weyns 2016] formulate the conversion of the continuous references into discrete settings as a linear optimization problem. The proposed approach avoids partitioning actuators into disjoint sets and allows actuation to minimize a cost function (e.g., considering the priority of different actuators), but it does not provide an explicit means for handling conflicting goals when the satisfaction of one makes others infeasible.

Finally, in the domain of Cloud Computing, variations of MPC have been applied extensively. In [Gaggero and Caviglione 2015; Kusic et al. 2009; Lakew et al. 2017] the authors apply look-ahead control to improve the energy consumption and the performance of the cloud. Similarly, in [Ghanbari et al. 2014] MPC is applied to improve the replica placement mechanism and deal with multiple Service Level Objectives. These approaches offer significant improvements to their respective applications, although are highly customized to the specific problem they are solving. On the other hand, our approach is more generic and therefore easier for software engineers that have no expertise on Control Theory to use it. Moreover, in our work we integrate control design and Requirements Engineering in order to provide a guideline on how to integrate MPC with the development of self-adaptive software.

7. CONCLUSIONS

The main contribution of this paper is to adopt the concept of MPC for the design of self-adaptive software systems. To accomplish this, we propose a framework, named *CobRA*, that integrates MPC components with previous work on software engineering for self-adaptive systems. We also provide guidelines on how to tune the variables of the MPC controller for better results during the adaptation process.

The distinct feature of *CobRA* compared to other approaches is the use of an analytical model to capture the relationship between the control parameters and the output of the system. This model can accurately predict the system's behaviour and allows *CobRA* to react to environmental changes and dynamically compose adaptation plans. The analytical model is the product of an automated system identification process, capturing relations that human experts might not be aware of. We evaluated *CobRA* using two simulations one for the Meeting-Scheduler exemplar and one for an E-Shop to compare our the *Zanshin* framework. The results of our evaluation show that control-theoretic concepts can be very effective in producing adaptation plans for software

systems and most of the times provides better results than human experience-based approaches.

Our approach needs to be further evaluated with more and larger case-studies and compared with more adaptation frameworks other than *Zanshin*. Moreover, we plan to apply Hierarchical Control in order to overcome difficulties of controlling goals that their level of satisfaction changes at a different rate.

8. ACKNOWLEDGMENTS

This work has been supported by the ERC advanced grant 267856 Lucretius: "Foundations for Software Evolution" (April 2011 - March 2016, http://www.lucretius.eu). Alessandro Vittorio Papadopoulos is funded by the Swedish Foundation for Strategic Research under the project "Future factories in the cloud (FiC)" with grant number GMT14-0032. Vítor E. Silva Souza is currently funded by Brazilian research agencies FAPES (# 0969/2015) and CNPq (# 485368/2013-7, # 461777/2014-2).

References

Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. 2013. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Trans. Softw. Eng.* 39, 5 (2013), 658–683. DOI: http://dx.doi.org/10.1109/TSE.2012.64

Frank Allgöwer and Alex Zheng. 2000. *Nonlinear model predictive control*. Vol. 26. Birkhäuser Basel. DOI: http://dx.doi.org/10.1007/978-3-0348-8407-5

Konstantinos Angelopoulos, Alessandro Vittorio Papadopoulos, and John Mylopoulos. 2015. Adaptive Predictive Control for Software Systems. In *Proceedings of the 1st International Workshop on Control Theory for Software Engineering (CTSE 2015)*. ACM, New York, NY, USA, 17–21. DOI: http://dx.doi.org/10.1145/2804337.2804340

Konstantinos Angelopoulos, Alessandro Vittorio Papadopoulos, Vítor E. Silva Souza, and John Mylopoulos. 2016. Model Predictive Control for Software Systems with CobRA. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '16)*. ACM, New York, NY, USA, 35–46. DOI: http://dx.doi.org/10.1145/2897053.2897054

Konstantinos Angelopoulos, Vítor E. Silva Souza, and John Mylopoulos. 2014. Dealing with multiple failures in zanshin: a control-theoretic approach. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Proceedings (SEAMS 2014)*. 165–174. DOI: http://dx.doi.org/10.1145/2593929.2593936

Konstantinos Angelopoulos, Vítor E. Silva Souza, and John Mylopoulos. 2015. Capturing Variability in Adaptation Spaces: A Three-Peaks Approach. In *Conceptual Modeling - 34th International Conference, ER 2015, Proceedings*. 384–398. DOI: http://dx.doi.org/10.1007/978-3-319-25264-3_28

K.J. Åström and R.M. Murray. 2010. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press.

Yuriy Brun, Giovanna Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. 2009. Software Engineering for Self-Adaptive Systems. Springer-Verlag, Berlin, Heidelberg, Chapter Engineering Self-Adaptive Systems Through Feedback Loops, 48–70. DOI: http://dx.doi.org/10.1007/978-3-642-02161-9_3

Eduardo F. Camacho and Carlos Bordons. 2004. *Model Predictive Control*. Springer London.

Javier Cámara, David Garlan, Bradley R. Schmerl, and Ashutosh Pandey. 2015. Optimal planning for architecture-based self-adaptation via model checking of stochastic games. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. 428–435. DOI: http://dx.doi.org/10.1145/2695664.2695680

Bihuan Chen, Xin Peng, Yijun Yu, and Wenyun Zhao. 2014. Uncertainty Handling in Goal-driven Self-optimization - Limiting the Negative Effect on Adaptation. *J. Syst. Softw.* 90 (April 2014), 114–127. DOI: http://dx.doi.org/10.1016/j.jss.2013.12.033

Shang-Wen Cheng, David Garlan, and Bradley R. Schmerl. 2006. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, SEAMS 2006*. 2–8. DOI: http://dx.doi.org/10.1145/1137677.1137679

Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. 1993. Goal-directed requirements acquisition. *Science of Computer Programming* 20, 1 (1993), 3 – 50. DOI: http://dx.doi.org/10.1016/0167-6423(93)90021-G

Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. 2011. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 283–292. DOI:http://dx.doi.org/10.1109/ASE.2011.6100064

Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2014. Automated design of self-adaptive software with control-theoretical formal guarantees. In *36th International Conference on Software Engineering, ICSE '14*. 299–310. DOI:http://dx.doi.org/10.1145/2568225.2568272

Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2015. Automated Multi-objective Control for Self-adaptive Software Design. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 13–24. DOI:http://dx.doi.org/10.1145/2786805.2786833

Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro Vittorio Papadopoulos, Suprio Ray, Amir M. Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. 2017. Control Strategies for Self-Adaptive Software Systems. 11, 4, Article 24 (Feb. 2017), 31 pages. DOI:http://dx.doi.org/10.1145/3024188

Wladyslaw Findeisen, Frederic N Bailey, M Brdys, Krzysztof Malinowski, Piotr Tatjewski, and A Wozniak. 1980. Control and coordination in hierarchical systems. (1980).

Mauro Gaggero and Luca Caviglione. 2015. Predictive Control for Energy-Aware Consolidation in Cloud Datacenters. *Control Systems Technology, IEEE Transactions on* (2015), 1–14. DOI:http://dx.doi.org/10.1109/TCST.2015.2457874

Hamoun Ghanbari, Marin Litoiu, Przemyslaw. Pawluk, and Cornel Barna. 2014. Replica Placement in Cloud through Simple Stochastic Model Predictive Control. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. 80–87. DOI:http://dx.doi.org/10.1109/CLOUD.2014.21

Pontus Giselsson. 2014. Improved Fast Dual Gradient Methods for Embedded Model Predictive Control. In *IFAC World Congress*, Vol. 19. 2303–2309. DOI:http://dx.doi.org/10.3182/20140824-6-ZA-1003.00295

Edward Nicholas Hartley, Juan Luis Jerez, Andrea Suardi, Jan M Maciejowski, Eric C Kerrigan, and George A Constantinides. 2014a. Predictive control using an FPGA with application to aircraft control. *Control Systems Technology, IEEE Transactions on* 22, 3 (2014), 1006–1017.

Edward N. Hartley, Juan L. Jerez, Andrea Suardi, Jan M. Maciejowski, Eric C. Kerrigan, and George A. Constantinides. 2014b. Predictive Control Using an FPGA With Application to Aircraft Control. *Control Systems Technology, IEEE Transactions on* 22, 3 (May 2014), 1006–1017. DOI:http://dx.doi.org/10.1109/TCST.2013.2271791

C. M. Ionescu, D. Copot, A. Maxim, E. Dulf, R. Both, and R. De Keyser. 2016. Robust autotuning MPC for a class of process control applications. In *2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*. 1–6. DOI:http://dx.doi.org/10.1109/AQTR.2016.7501356

Juan L. Jerez, Paul J. Goulart, Stefan Richter, George A. Constantinides, Eric C. Kerrigan, and Manfred Morari. 2014. Embedded Online Optimization for Model Predictive Control at Megahertz Rates. *Automatic Control, IEEE Transactions on* 59, 12 (Dec 2014), 3238–3251. DOI:http://dx.doi.org/10.1109/TAC.2014.2351991

Juan L. Jerez, Eric C. Kerrigan, and George A. Constantinides. 2012. A sparse and condensed QP formulation for predictive control of LTI systems. *Automatica* 48, 5 (2012), 999–1002. DOI:http://dx.doi.org/10.1016/j.automatica.2012.03.010

Joachim Karlsson and Kevin Ryan. 1997. A Cost-Value Approach for Prioritizing Requirements. *IEEE Softw.* 14, 5 (Sept. 1997), 67–74. DOI:http://dx.doi.org/10.1109/52.605933

Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *Computer* 36, 1 (Jan. 2003), 41–50. DOI:http://dx.doi.org/10.1109/MC.2003.1160055

Cristian Klein, Alessandro Vittorio Papadopoulos, Manfred Dellkrantz, Jonas Dürango, Martina Maggio, Karl-Erik Årzén, Francisco Hernández-Rodriguez, and Erik Elmroth. 2014. Improving Cloud Service Resilience using Brownout-Aware Load-Balancing. In *Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on*. 31–40. DOI:http://dx.doi.org/10.1109/SRDS.2014.14

Mayuresh V. Kothare, Venkataramanan Balakrishnan, and Manfred Morari. 1996. Robust constrained model predictive control using linear matrix inequalities. *Automatica* 32, 10 (1996), 1361–1379. DOI:http://dx.doi.org/10.1016/0005-1098(96)00063-5

Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. 2009. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing* 12, 1 (2009), 1–15.

Ewnetu Bayuh Lakew, Alessandro Vittorio Papadopoulos, Martina Maggio, Cristian Klein, and Erik Elmroth. 2017. KPI-agnostic Control for Fine-Grained Vertical Elasticity. In *17th IEEE/ACM*

International Symposium on Cluster, Cloud and Grid Computing (CCGrid '17). 589–598. DOI:http://dx.doi.org/10.1109/CCGRID.2017.71

Emmanuel Letier and Axel van Lamsweerde. 2004. Reasoning About Partial Goal Satisfaction for Requirements and Design Engineering. *SIGSOFT Softw. Eng. Notes* 29, 6 (Oct. 2004), 53–62. DOI:http://dx.doi.org/10.1145/1041685.1029905

Feng-Lin Li, Jennifer Horkoff, John Mylopoulos, Renata S. S. Guizzardi, Giancarlo Guizzardi, Alex Borgida, and Lin Liu. 2014. Non-functional requirements as qualities, with a spice of ontology. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*. 293–302. DOI:http://dx.doi.org/10.1109/RE.2014.6912271

Wei Liu and George Wang. 2000. Auto-tuning procedure for model-based predictive controller. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, Vol. 5. 3421–3426 vol.5. DOI:http://dx.doi.org/10.1109/ICSMC.2000.886537

Lennart Ljung. 1999. *System Identification: Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Lennart Ljung. 2010. Approaches to identification of nonlinear systems. In *Control Conference (CCC), 2010 29th Chinese*. 1–5.

Jan M. Maciejowski. 2002. *Predictive Control: With Constraints*. Prentice Hall.

Martina Maggio, Henry Hoffmann, Alessandro Vittorio Papadopoulos, Jacopo Panerati, Marco Domenico Santambrogio, Anant Agarwal, and Alberto Leva. 2012. Comparison of Decision Making Strategies for Self-Optimization in Autonomic Computing Systems. *ACM Transactions on Autonomous and Adaptive Systems* 7, 4, Article 36 (2012), 32 pages. DOI:http://dx.doi.org/10.1145/2382570.2382572

Gabriel Moreno, Alessandro Vittorio Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley Schmerl. 2017. Comparing Model-Based Predictive Approaches to Self-Adaptation: CobRA and PLA. In *12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 42–53. DOI:http://dx.doi.org/10.1109/SEAMS.2017.2

Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive Self-adaptation Under Uncertainty: A Probabilistic Model Checking Approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 1–12. DOI:http://dx.doi.org/10.1145/2786805.2786853

Alessandro Vittorio Papadopoulos, Martina Maggio, Federico Terraneo, and Alberto Leva. 2015. A Dynamic Modelling Framework for Control-based Computing System Design. *Mathematical and Computer Modelling of Dynamical Systems* 21, 3 (2015), 251–271. DOI:http://dx.doi.org/10.1080/13873954.2014.942785

S. Joe Qin and Thomas A. Badgwell. 2003. A survey of industrial model predictive control technology. *Control engineering practice* 11, 7 (2003), 733–764.

Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 500–507. DOI:http://dx.doi.org/10.1109/CLOUD.2011.42

Stepan Shevtsov and Danny Weyns. 2016. 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 (FSE 2016)*. ACM, New York, NY, USA, 229–241. DOI:http://dx.doi.org/10.1145/2950290.2950301

Vítor E. Silva Souza, Alexei Lapouchnian, Konstantinos Angelopoulos, and John Mylopoulos. 2013. Requirements-driven software evolution. *Computer Science - R&D* 28, 4 (2013), 311–329. DOI:http://dx.doi.org/10.1007/s00450-012-0232-2

Vítor E. Silva Souza, Alexei Lapouchnian, and John Mylopoulos. 2011. System Identification for Adaptive Software Systems: A Requirements Engineering Perspective. In *Conceptual Modeling - ER 2011, 30th International Conference, ER. Proceedings*. 346–361. DOI:http://dx.doi.org/10.1007/978-3-642-24606-7_26

Vítor E. Silva Souza, Alexei Lapouchnian, and John Mylopoulos. 2012. Requirements-Driven Qualitative Adaptation. In *On the Move to Meaningful Internet Systems: OTM 2012*, Robert Meersman, Hervé Panetto, Tharam Dillon, Stefanie Rinderle-Ma, Peter Dadam, Xiaofang Zhou, Siani Pearson, Alois Ferscha, Sonia Bergamaschi, and IsabelF. Cruz (Eds.). Lecture Notes in Computer Science, Vol. 7565. Springer Berlin Heidelberg, 342–361. DOI:http://dx.doi.org/10.1007/978-3-642-33606-5_21

Vítor E. Silva Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. 2011. Awareness requirements for adaptive systems. In *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*. 60–69. DOI:http://dx.doi.org/10.1145/1988008.1988018

Daniel Sykes, William Heaven, Jeff Magee, and Jeff Kramer. 2010. Exploiting Non-functional Preferences in Architectural Adaptation for Self-managed Systems. In *Proceedings of the 2010*

ACM Symposium on Applied Computing (SAC '10). ACM, New York, NY, USA, 431–438. DOI:http://dx.doi.org/10.1145/1774088.1774180

Quang N Tran, Joni Scholten, Leyla Ozkan, and A.C.P.M. Backx. 2014. A model-free approach for auto-tuning of model predictive control. *IFAC Proceedings Volumes* 47, 3 (2014), 2189–2194. DOI:http://dx.doi.org/10.3182/20140824-6-ZA-1003.01494 19th IFAC World Congress.

Yang Wang and S. Boyd. 2010. Fast Model Predictive Control Using Online Optimization. *Control Systems Technology, IEEE Transactions on* 18, 2 (March 2010), 267–278. DOI:http://dx.doi.org/10.1109/TCST.2009.2017934

Melanie N. Zeilinger, Davide M. Raimondo, Alexander Domahidi, Manfred Morari, and Colin N. Jones. 2014. On real-time robust model predictive control. *Automatica* 50, 3 (2014), 683–694. DOI:http://dx.doi.org/10.1016/j.automatica.2013.11.019

Parisa Zoghi, Mark Shtern, and Marin Litoiu. 2014. Designing search based adaptive systems: a quantitative approach. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings*. 7–16. DOI:http://dx.doi.org/10.1145/2593929.2593935