2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems

Model Predictive Control for Software Systems with CobRA

Konstantinos Angelopoulos
University of Trento
Trento, Italy
k.angelopoulos@unitn.it

Alessandro V. Papadopoulos
Lund University
Lund, Sweden
alessandro@control.lth.se

Vítor E. Silva Souza
Federal University of Espírito Santo
Vítoria, Brazil
vitorsouza@inf.ufes.br

John Mylopoulos
University of Trento
Trento, Italy
jm@disi.unitn.it

ABSTRACT

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 a simulation of the Meeting-Scheduling System exemplar.

CCS Concepts

•**Software and its engineering** → **Software creation and management; Software verification and validation;** *Operational analysis;*

Keywords

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 of various parameters from the configuration space with multiple goals. Therefore, an adaptation strategy A could restore the satisfaction of goal G but fail or worsen goal G'.

Control Theory has provided prominent theoretical and practical frameworks for dealing with systems with multiple parameters (inputs) and assigned with multiple goals (outputs). Current approaches for developing self-adaptive software either deal with each goal individually without taking into account interferences [6, 11, 12, 32] or perform reactive adaptation when the failure has already taken place without any provision for the future [4,10,13,23,39] or effort for failure anticipation. 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 [15,16,24,31] 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.

The main purpose of this paper is to introduce 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 propose a framework that supports the elicitation of the analytical models required for MPC. We integrate MPC with previous work on requirements engineering for software adaptation in a framework named *CobRA* [1]. Then, we apply our framework to the Meeting-Scheduler exemplar to illustrate and evaluate our proposal.

The rest of the paper is structured as follows. Section 2 presents the research baseline for this work. Section 3 de-

[1]Control-based Requirements-oriented Adaptation

SEAMS'16, May 16-17, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897053.2897054

scribes 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*. Finally, in Section 6 we compare the related work with our proposal, and Section 7 concludes the paper.

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 subgoals: 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, *soft-goals* 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 [4] we use *Awareness 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 G5 fail, a new configuration must be applied. In the same context, if the participation is lower than what stakeholders requested more than 25% of the times, a new adaptation is selected. Each *AwReq* is associated with variables named *indicators* which measure the degree of success a monitored requirement.

Indicator values are influenced by two kinds of parameters: a) *environmental parameters* that cannot be manipulated and b) *control parameters* that can be adjusted at runtime [5]. 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 is increased 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, the 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, from how many participants timetables are collected FhM has an impact on how fast the meetings are scheduled ($AR5$) and how good the participation is going to be ($AR4$). In the same context, the maximum 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 select a system's suggestion) and the numbers of local and hotel rooms, RfM and HfM respectively, are control parameters the values of which affect the outcome of the monitored indicators.

Another kind of requirement included in our approach is an *Evolution Requirement* (*EvoReq*) [33]. 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*) [4]. 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 [4,34], 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 [25], 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-

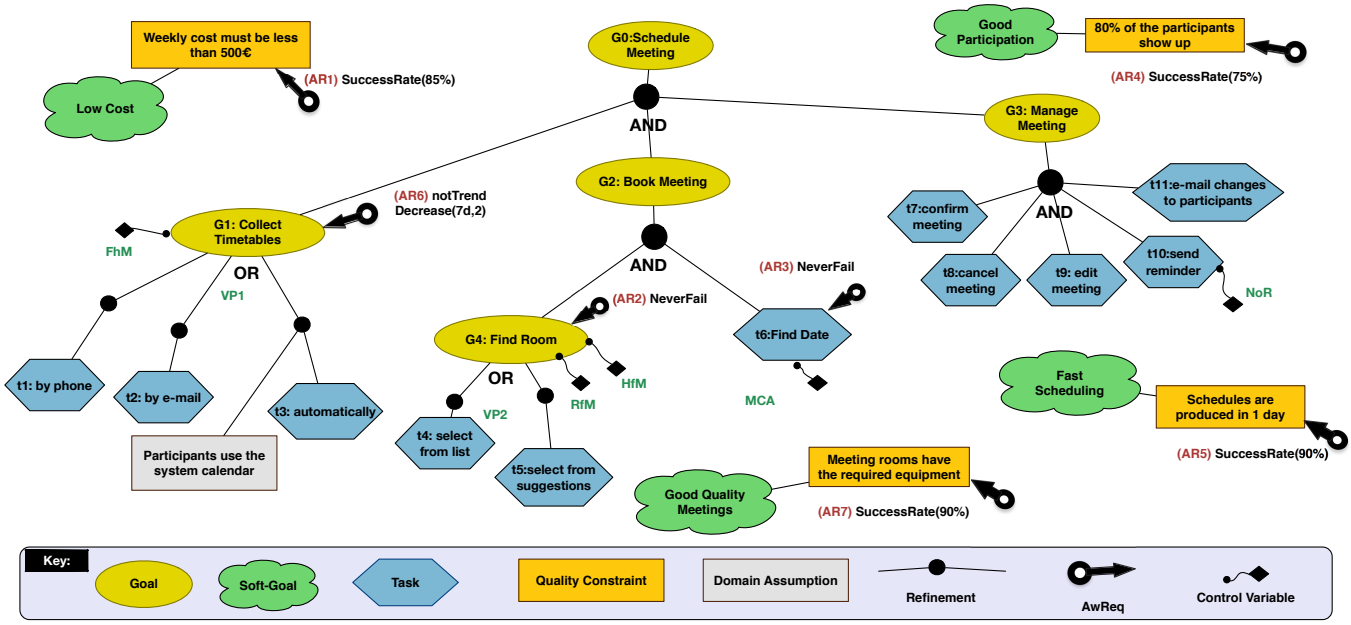

Figure 1: Meeting Scheduler goal model

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* [35] 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.

3. MODEL PREDICTIVE CONTROL

Based on the dynamic model of Equation (2), different control strategies can be designed. We here extend our previous work [3] and present a *receding horizon* MPC [7, 27] 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 (KF) [25], 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 $u^{\star} = [u_t^{\star}, u_{t+1}^{\star}, \ldots, u_{t+H-1}^{\star}]$ 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":

$$\begin{aligned}\overbrace{\begin{bmatrix}\Delta x(t+1)\\ y(t)\end{bmatrix}}^{\tilde{x}(t+1)} &= \overbrace{\begin{bmatrix}A & 0_{n\times p}\\ C & I_{p\times p}\end{bmatrix}}^{\tilde{A}}\overbrace{\begin{bmatrix}\Delta x(t)\\ y(t-1)\end{bmatrix}}^{\tilde{x}(t)} + \overbrace{\begin{bmatrix}B\\ 0_{p\times m}\end{bmatrix}}^{\tilde{B}}\Delta u(t)\\ y(t) &= \overbrace{\begin{bmatrix}C & I_{p\times p}\end{bmatrix}}^{\tilde{C}}\overbrace{\begin{bmatrix}\Delta x(t)\\ y(t-1)\end{bmatrix}}^{\tilde{x}(t)}\end{aligned} \tag{3}$$

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

$$\begin{aligned}J_t = \sum_{i=1}^{H} &\left[y_{t+i}^\circ - y_{t+i}\right]^T Q_i \left[y_{t+i}^\circ - y_{t+i}\right]\\ &+ \left[\Delta u_{t+i-1}\right]^T P_i \left[\Delta u_{t+i-1}\right],\end{aligned}$$

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) [22], 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 preferences over control parameters. When a control parameter is requested not to change frequently its value 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:

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

This formulation is equivalent to a convex Quadratic Programming (QP) problem [27]. The problem has time complexity $\mathcal{O}(H^3 m^3)$ [37]. A solution to the problem consists of a plan of optimal future $\Delta u_{t+i-1}^\star$, $i = 1,\ldots,H$, but only the first one is applied, i.e., $\Delta u(t) = \Delta u_t^\star$, 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 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 [25]. 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 the 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) [25].

The block diagram for the resulting control scheme is represented in Figure 2.

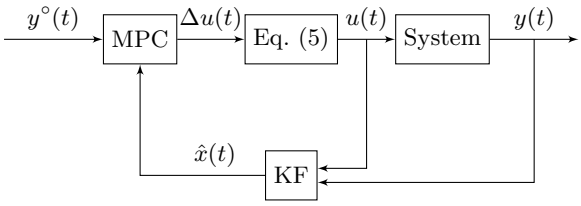

Figure 2: Control scheme.

3.2 Formal guarantees

Applying Control Theory to software systems provides a set of formal guarantees about the quality of the adaptation process [14]. 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 [4].

The formal guarantees for the MPC have 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 minimum. 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 ones on the indicators. *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 [19, 30]. Since the proposed solution requires a solution to an optimization problem at each control instant, it is critical to discuss possible such 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 [17,21], especially for embedded systems [20], possibly co-designing also a dedicated hardware for the solution in case of critical systems [18]. An overview on the matter can be found in [38].

In many cases such kind of advanced algorithms are not required when dealing with software components, and for the most 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: a) design time phase and b) runtime phase. During the first phase all the 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 important and prone to failure. An *AwReq* AR_i defines a reference goal $y_i^\circ(\cdot)$ for the controller's output. Table 1 enlists all the reference goals for the Meeting-Scheduler exemplar.

Table 1: 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, the prices of hotel rooms rise every year and consequently $I1$ will fail more often as time passes. Alternatively, during summer prices are usually higher. Hence, stakeholders could accept a lower success for

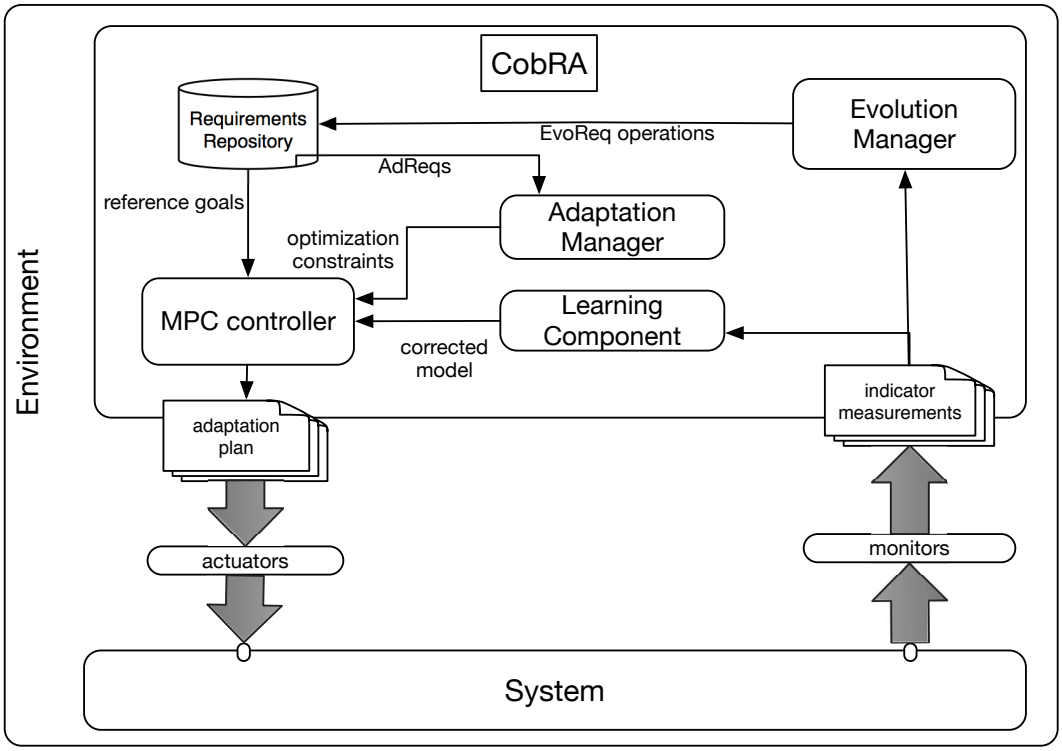

Figure 3: CobRA framework

$I1$ (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 2.

Table 2: *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 compare 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 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 3 and Table 4, respectively.

Table 3: 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

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

Table 4: 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 increase more than 5 units each time.

Finally, a quantitative model such as that in Equation 2 must be derived. Given the absence of laws of nature 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 3, 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.

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 it is 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 anticipated future failures in a receding horizon fashion.

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

1. **Step 1:** The monitors collect the measurements of all the indicators of the system.

2. **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.

3. **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.

4. **Step 4:** The Learning Component provides the MPC with a corrected model of the system based on the recent measurements.

5. **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.

6. **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 that 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* that also has as its baseline for adaptation stakeholder requirements and adopts concepts from Control Theory.

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

5.1 Methodology

We have conducted our experiments with a simulation of the Meeting-Scheduler exemplar[4] implemented in Python and ran on a computer with an Intel i5 processor at 2.5GHz and 16GB of RAM. We ran the simulation for 10.000 steps while automatically modifying all the control parameters which must cover all the range of their potential values. The result of this process is a log file with all the values of the inputs and outputs of the system at every step. Then, we executed once a Matlab procedure from the Matlab System Identification Toolbox in order to estimate an analytical model that describe the 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 user's availability, punctuality and the number of meeting requests that must be scheduled every day. 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 reduced. Similarly, if a control parameter that we do not wish to change often, such as the number of hotel rooms available, the associated weight must be increased. 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. When the MPC controller reaches a desired behaviour, the tuning phase is completed.

Zanshin as opposed to *CobRA* doesn't 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 [32]. 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 studies of Meeting-Scheduler [4, 34].

For the evaluation and the comparison of the two frameworks, we put the simulated system 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 using *AwReqs* as *CobRA* and therefore, customization of the adaptation problem was 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.

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 pseudo-randomness has been encoded for other goals such as *Find Room* and *High Participation*. For the purpose of 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 the participants availability. Due to space limitation, we present the results only for indicators $I1$-$I4$.

[4]https://gitlab.com/konangelop/it.unitn.disi.konangelop.simulations.meeting_scheduler_v2.git

5.2 Experimental Results

Figure 4 depicts the values of the indicators at each step of the simulation. As the number of meetings grows the cost for the system increases as well, resulting in the decrease of the indicator $I1$. However, *CobRA* though manages to recover by preferring local rooms over hotel rooms as it can be seen in Figure 5, whereas *Zanshin* fails to restore the failure. As it concerns 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. Finally, for indicator $I4$ *CobRA* outperforms *Zanshin*, by increasing NoR more than the latter.

The last metric we use for our evaluation is 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 the cost function is calculated from the measured values of the indicators and the changes performed over the control parameters. In Figure 6 we present the individual value of the cost function at each step the simulation, on the top and the cumulative cost $\sum_t \hat{J}(t)$ on the bottom. $\hat{J}(t)$ captures the magnitude business value loss because of failing indicators and adaptation costs for changing control parameters. *CobRA* minimizes more at each step 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. The minimization of the cost over the simulation is highlighted in the cumulative cost showed in the bottom graph of Figure 6.

5.3 Discussion

From the experimental results we can safely assume 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 outperformed the qualitative adaptation of *Zanshin* in most cases and proved that Control Theory can be applied to generic software systems such as the Meeting-Scheduler exemplar.

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

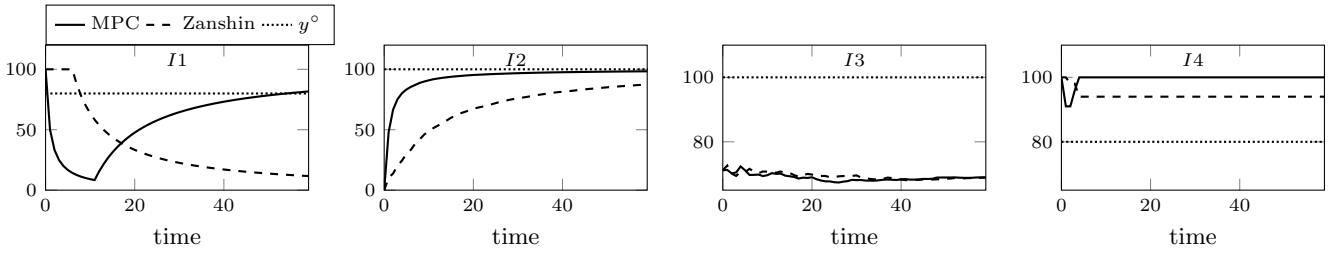

Figure 4: Indicator measured values

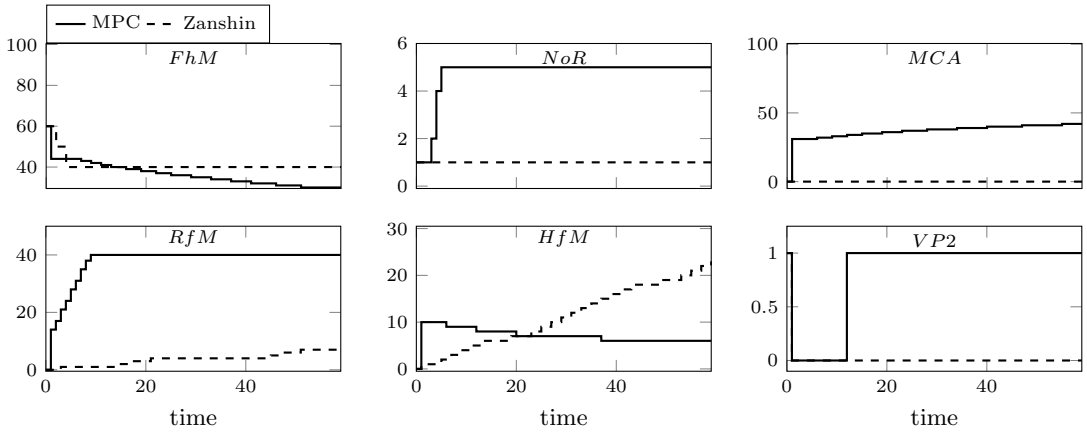

Figure 5: Control parameter values

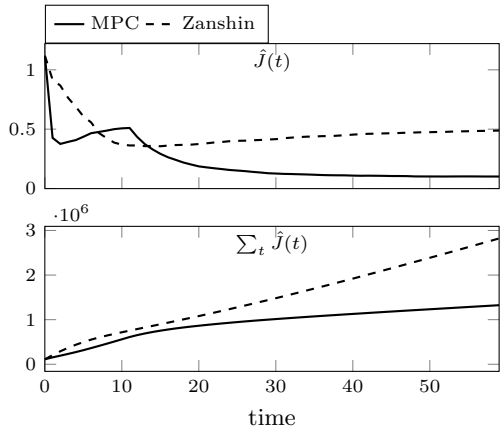

Figure 6: Adaptation cost

accurate predictions.

For the Meeting-Scheduler exemplar a linear model was sufficient for predicting the system's behaviour. However, this might not be always the case. For systems with non-linear dynamics, either tailored models can be used [29], or more advanced system identification techniques are available [25, 26], and nonlinear MPC formulations can be adopted [2]. 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.

A main drawback of *CobRA* is that it requires a 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 the system designers simulate a model that can produce data sufficiently similar to those of the real system. Developing such methodologies for software engineers, as well as establishing guidelines for tuning the MPC parameters, are part of our future research agenda.

Finally, it is worth noting that some control parameters of our exemplar, such the number of rooms, are discrete, but according to the equation 1 control parameters are continuous variables. In Control Theory this problem is known as the actuation design problem. There are two different approaches a) using rounding of the continuous variable computed by the MPC and b) by adopting a Pulse Width Modulation-like policy [28].

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. First, Rainbow [10], with the use of 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, the authors of [8] propose the use of Probabilistic Model Checking in order to compose strategies dynamically. Our approach differs to theirs as it requires precise knowledge of how each control parameter of the system influences the output of the system, 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.

In [39] Zoghi et al. 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. in their work [36] 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 [9]. 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 in order the SAT-solver to 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 control in a seamless way was proposed in [12]. 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 [1]. The solution in [12] 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 [13] where the MIMO control is obtained as an automated synthesis by composing SISO controllers in a hierarchical way. The approach presented in [13] 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.

Finally, in the domain of Cloud Computing variations of MPC have been applied extensively. In [15, 24] the authors apply look-ahead control to improve the energy consumption and the performance of the cloud. Similarly, in [16] 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 about to how to integrate MPC with the development of self-adaptive software.

7. CONCLUSIONS

The main contribution of this paper is adopt the concept of an MPC to 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 compose dynamically 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 our framework using an implementation of the Meeting-Scheduler exemplar and compared the result to those of *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*.

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 currently funded by the Swedish Research Council under contract number C0590801 (2012-5908), and by the LCCC Linnaeus Center. Vítor is currently funded by Brazilian research agencies Fapes (# 0969/2015), CNPq (# 461777/2014-2), and by Ufes' FAP (# 6166/2015).

9. REFERENCES

[1] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Trans. Softw. Eng.*, 39(5):658–683, 2013.

[2] F. Allgöwer and A. Zheng. *Nonlinear model predictive control*, volume 26. Birkhäuser Basel, 2000.

[3] K. Angelopoulos, A. V. Papadopoulos, and J. Mylopoulos. Adaptive predictive control for software systems. In *Proceedings of the 1st*

International Workshop on Control Theory for Software Engineering, CTSE 2015, pages 17–21, New York, NY, USA, 2015. ACM.

[4] K. Angelopoulos, V. E. S. Souza, and J. Mylopoulos. 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, pages 165–174, 2014.

[5] K. Angelopoulos, V. E. S. Souza, and J. Mylopoulos. Capturing variability in adaptation spaces: A three-peaks approach. In *Conceptual Modeling - 34th International Conference, ER 2015, Proceedings*, pages 384–398, 2015.

[6] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Software engineering for self-adaptive systems. chapter Engineering Self-Adaptive Systems Through Feedback Loops, pages 48–70. Springer-Verlag, Berlin, Heidelberg, 2009.

[7] E. Camacho and C. Bordons. *Model Predictive Control*. Springer London, 2004.

[8] J. Cámara, D. Garlan, B. R. Schmerl, and A. Pandey. Optimal planning for architecture-based self-adaptation via model checking of stochastic games. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 428–435, 2015.

[9] B. Chen, X. Peng, Y. Yu, and W. Zhao. Uncertainty handling in goal-driven self-optimization - limiting the negative effect on adaptation. *J. Syst. Softw.*, 90:114–127, Apr. 2014.

[10] S. Cheng, D. Garlan, and B. R. Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, SEAMS 2006*, pages 2–8, 2006.

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

[12] A. Filieri, H. Hoffmann, and M. Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *36th International Conference on Software Engineering, ICSE '14*, pages 299–310, 2014.

[13] A. Filieri, H. Hoffmann, and M. Maggio. 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, pages 13–24, New York, NY, USA, 2015. ACM.

[14] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel. Software engineering meets control theory. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2015 IEEE/ACM 10th International Symposium on*, pages 71–82, May 2015.

[15] M. Gaggero and L. Caviglione. Predictive control for energy-aware consolidation in cloud datacenters. *Control Systems Technology, IEEE Transactions on*, pages 1–14, 2015.

[16] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna. Replica placement in cloud through simple stochastic model predictive control. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 80–87, June 2014.

[17] P. Giselsson. Improved fast dual gradient methods for embedded model predictive control. In *IFAC World Congress*, volume 19, pages 2303–2309, 2014.

[18] E. Hartley, J. Jerez, A. Suardi, J. Maciejowski, E. Kerrigan, and G. Constantinides. Predictive control using an fpga with application to aircraft control. *Control Systems Technology, IEEE Transactions on*, 22(3):1006–1017, May 2014.

[19] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides. Predictive control using an fpga with application to aircraft control. *Control Systems Technology, IEEE Transactions on*, 22(3):1006–1017, 2014.

[20] J. Jerez, P. Goulart, S. Richter, G. Constantinides, E. Kerrigan, and M. Morari. Embedded online optimization for model predictive control at megahertz rates. *Automatic Control, IEEE Transactions on*, 59(12):3238–3251, Dec 2014.

[21] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides. A sparse and condensed QP formulation for predictive control of LTI systems. *Automatica*, 48(5):999–1002, 2012.

[22] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Softw.*, 14(5):67–74, Sept. 1997.

[23] C. Klein, A. V. Papadopoulos, M. Dellkrantz, J. Dürango, M. Maggio, K.-E. Årzén, F. Hernández-Rodriguez, and E. Elmroth. Improving cloud service resilience using brownout-aware load-balancing. In *Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on*, pages 31–40, 2014.

[24] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.

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

[26] L. Ljung. Approaches to identification of nonlinear systems. In *Control Conference (CCC), 2010 29th Chinese*, pages 1–5, July 2010.

[27] J. Maciejowski. *Predictive Control: With Constraints*. Prentice Hall, 2002.

[28] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva. Comparison of decision making strategies for self-optimization in autonomic computing systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(4):36:1–36:32, 2012.

[29] A. V. Papadopoulos, M. Maggio, F. Terraneo, and A. Leva. A dynamic modelling framework for control-based computing system design. *Mathematical*

and Computer Modelling of Dynamical Systems, 21(3):251–271, 2015.

[30] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764, 2003.

[31] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 500–507, July 2011.

[32] V. Souza, A. Lapouchnian, and J. Mylopoulos. Requirements-driven qualitative adaptation. In R. Meersman, H. Panetto, T. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I. Cruz, editors, *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565 of *Lecture Notes in Computer Science*, pages 342–361. Springer Berlin Heidelberg, 2012.

[33] V. E. S. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos. Requirements-driven software evolution. *Computer Science - R&D*, 28(4):311–329, 2013.

[34] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos. System identification for adaptive software systems: A requirements engineering perspective. In *Conceptual Modeling - ER 2011, 30th International Conference, ER. Proceedings*, pages 346–361, 2011.

[35] V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. Awareness requirements for adaptive systems. In *2011 ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, pages 60–69, 2011.

[36] D. Sykes, W. Heaven, J. Magee, and J. Kramer. Exploiting non-functional preferences in architectural adaptation for self-managed systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 431–438, New York, NY, USA, 2010. ACM.

[37] Y. Wang and S. Boyd. Fast model predictive control using online optimization. *Control Systems Technology, IEEE Transactions on*, 18(2):267–278, March 2010.

[38] M. N. Zeilinger, D. M. Raimondo, A. Domahidi, M. Morari, and C. N. Jones. On real-time robust model predictive control. *Automatica*, 50(3):683–694, 2014.

[39] P. Zoghi, M. Shtern, and M. Litoiu. Designing search based adaptive systems: a quantitative approach. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings*, pages 7–16, 2014.