Adaptive Predictive Control for Software Systems

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

Alessandro Vittorio Papadopoulos
Department of Automatic Control, Lund University
Lund, Sweden
alessandro@control.lth.se

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

ABSTRACT

Self-adaptive software systems are designed to support a number of alternative solutions for fulfilling their requirements. These define an adaptation space. During operation, a self-adaptive system monitors its performance and when it finds that its requirements are not fulfilled, searches its adaptation space to select a best adaptation. Two major problems need to be addressed during the selection process: (a) Handling environmental uncertainty in determining the impact of an adaptation; (b) maintain an optimal equilibrium among conflicting requirements. This position paper investigates the application of Adaptive Model Predictive Control ideas from Control Theory to design self-adaptive software that makes decisions by predicting its future performance for alternative adaptations and selects ones that minimize the cost of requirement failures using quantitative information. The technical details of our proposal are illustrated through the meeting-scheduler exemplar.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*methodologies*

Keywords

Control theory, self-adaptive systems, software requirements, predictive control

1. INTRODUCTION

Self-adaptive systems are designed to maintain the fulfillment of their requirements in dynamic environments. When a failing requirement is encountered the system adapts by switching to an alternative configuration. The set of the available configurations constitute the system's adaptation space. Unfortunately, configuration selection is not an easy task as requirements are often conflicting and an adaptation that restores a failed requirement may break another one. For example, restoring a failed performance requirement by adding a server to a system may fail an operating costs requirement. In addition, stakeholders often over-constrain the system-to-be, or propose unrealistic unfeasible requirements [14]. Such conflicts can be accommodated by setting realistic thresholds, making a satisfactory equilibrium attainable. However, for many software systems setting accurate thresholds is at best guesswork, since there are no physical laws that account for the relationship between control parameters and requirements for an adaptive system.

Current approaches [5, 8, 21] deal with conflicting requirements and adaptation costs by making predictions about the system's environment and anticipate failures by making reconfiguration plans that optimize the utility output over time using a control theoretic technique, named Model Predictive Control (MPC), and variations of it [12]. However, these approaches are specific to resource provisioning and therefore architectural configurations, ignoring the dimensions of requirements and behavior of the adaptation space [2]. Moreover, the lack of a software engineering methodology which relates the elements of MPC and those of a self-adaptive software system prevents designers from applying this technique to domains other than service-based applications, where the current approaches focus on.

In this position paper we describe how a combination of concepts from Software Engineering (SE) and Control Theory (CT), in the same line of work as in [4], can tackle in a systematic way the problem of conflicting requirements and overestimation of system capabilities while applying adaptation strategies that maximize the system's outcome over time with minimum adaptation effort. Towards this direction we propose a combined use of MPC [12] and an online learning mechanism [10], similarly to [13, 15], to predict the future behavior of the controlled system within a specified horizon and dynamically compose adaptation strategies that will minimize the divergence of each requirement from the specified threshold prescribed by the stakeholders. Furthermore, we examine how the synthesis of a controller can be part of a SE process for designing self-adaptive systems. The adoption of MPC guarantees the avoidance of overshooting, management of constraints, and optimal tradeoff among conflicting requirements across time using prioritization techniques. In particular, we use Analytic Hierarchy Process (AHP) [1, 7].

The rest of the paper is organized as follows. Section 2 introduces the baseline of our proposal. Section 3 investigates how MPC can be applied for quantitative adaptation. Finally, Section 4 concludes the paper.

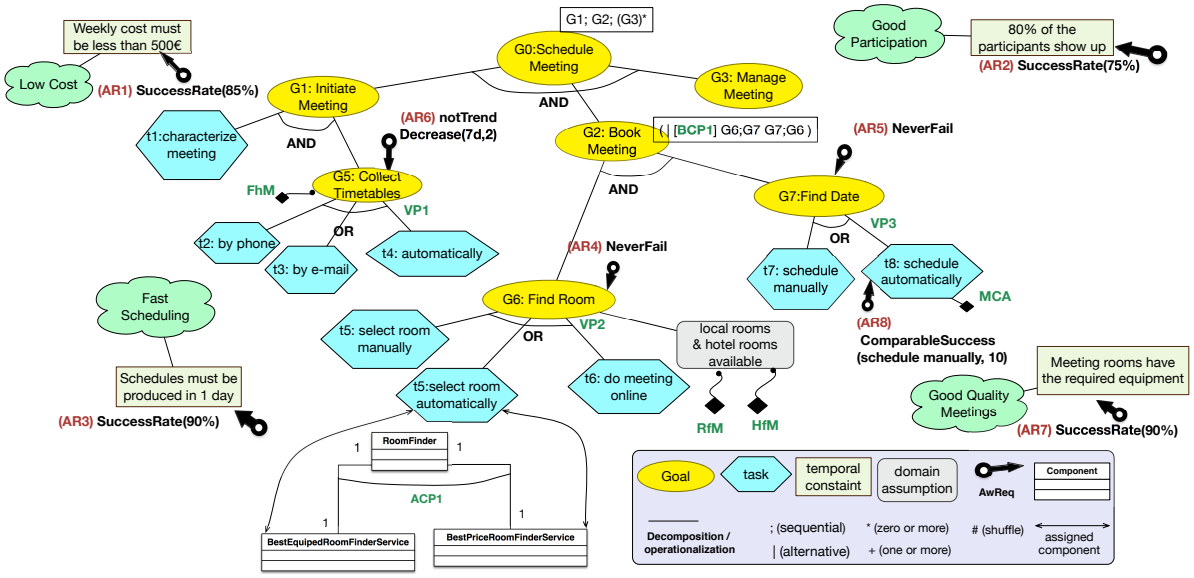

Figure 1: Three-peaks model for the Meeting Scheduler case study.

2. PRELIMINARIES AND MOTIVATING EXAMPLES

Our proposal adopts concepts from Goal-Oriented Requirements Engineering (GORE) such as goals for modeling stakeholder requirements, *softgoals* for modeling quality requirements, and AND/OR refinements that refine goal G into simpler goals whose satisfaction (all/at least one) implies the satisfaction of G, following traditional boolean semantics. Tasks are actions that a component (hardware /software components, or an external actor) can implement to fulfill or operationalize a goal. For example, in Fig. 1, *Schedule meeting* is a top-level goal, while *Good participation* is a softgoal. The goal *Schedule meeting* is AND refined into subgoals *Initiate Meeting*, *Book Meeting* and *Manage Meeting*.

In our previous work [1] we proposed a qualitative adaptation process inspired by feedback loop control. The design of this process includes three basic concepts: Awareness Requirements, Evolution Requirements and System Identification.

Awareness requirements (*AwReqs*) impose constraints on the failure of other requirements and correspond to the set-points of the adaptation mechanism. *AwReqs* are associated with variables named indicators that measure their success degree. For example, $AR4$ dictates that the goal *Find Room* must never fail, whereas $AR1$ prescribes that 85% of time the weekly cost of meetings must be less than 500 Euros. Hence, the associated indicators are $I_4 = 100\%$ and $I_6 \geq 85\%$. Every indicator is constantly monitored and when its value diverges from the one prescribed by the stakeholders, the associated *AwReq* fails and adaptation is triggered. In control-theoretical terms, indicators correspond to the system's *outputs*.

Evolution requirements (*EvoReqs*) [18] describe when and how other requirements should change at runtime. For example, an *EvoReq* may be "If requirement R fails three times in a row, replace it with requirement R^-", where R^- is a weaker (i.e., easier to fulfill) requirement. Such requirements are useful to evolve unfeasible requirements that were initially elicited from the stakeholders.

System Identification. Indicators are controlled by control parameters (*CPs*) set by the adaptation mechanism. In our recent work [2] we proposed an iterative process, named *three-peaks* to guide the software designers elicit a larger adaptation space that includes control parameters from the three dimensions of a software system, requirements, behavior and architecture. *Requirement control parameters* (*ReqCPs*) are derived either from OR-refinements or physical, information resources required by the system, in order to operate. For instance, $VP1$ in Fig. 1 is a *ReqCP* that gets its values from the ordered set $\{t2 \longrightarrow t3 \longrightarrow t4\}$. On the other hand, RfM and HfM are integer variables that represent how many local rooms and hotel rooms respectively are provided for meetings. MCA is yet another *ReqCP* that represents how many conflicts for the timeslot chosen and the participant timetables, while FhM represents *from how many* participants the system should collect time tables in order to satisfy the goal $G5$. *Behavioral control parameters* (*BCPs*) stem from system behavior, modeled with flow expressions (see Fig. 1), that capture allowed sequences of fulfillment of subgoals in order to fulfill a parent goal. For example, the goal *Book Meeting* can be fulfilled either by finding room first and then a date $(G6; G7)$ or find a date first and a meeting room afterwards $(G7; G6)$, see Fig. 1. These two potential sequences constitute the set of values for $BCP1$. Finally, Architectural Control Parameters (*ACPs*) capture variability in cases where more than one component are assigned with the fulfillment of the same goal or task, such as the task *select room automatically* which is carried out either by a component that finds the best equipped room or the another one which finds the cheapest room available. $ACP1$ gets as value the selected component's name.

Apart from *CPs*, the system's indicators are influenced by parameters found in the system's environment that cannot be controlled, named Environmental Parameters (*EPs*). Ex-

amples of such parameters include the price of hotel rooms, the response time of invited participants to timetable collection requests and their punctuality in attending the meetings after confirming their presence. *EPs* capture environmental uncertainty, since they are changing in a non-deterministic manner, adding disturbances to the system. For instance, the hotel room prices in certain periods rise, sometimes decreasing the indicator I_1 since the costs of meetings exceeds the allocated budget. Therefore, the system should respond to such changes of the environment and if possible anticipate them.

The qualitative positive or negative influence of *CPs* on indicators is captured by differential relations. For example, the differential relation $\Delta(I_2/MCA) < 0$ means that by increasing MCA by one unit I_2 will decrease, while $\Delta(I_5/MCA) > 0$ means that by increasing MCA I_5 will also increase. Similarly, the differential relations $\Delta(I_1/ACP1)$ $[BestEqip\ Room\ Service \longrightarrow BestPriceRoomService] > 0$ (the arrows indicate growing enumeration values), means switching to the component that finds the available room with the best price increases the success rate of I_1. The differential relations are symmetric and are provided by domain experts, which makes them prone to human errors and inaccuracies.

Indicators related to the same CP with conflicting influence, such as I_5 and I_2 from the previous example, are called conflicting indicators. When multiple failures are detected the adaptation mechanism must perform trade-offs, fixing the most important requirements first. Towards this direction, we set priorities over indicators using AHP. Due to lack of quantitative information on the impact of *CPs* on indicators, we define alternative conservative and optimistic adaptation policies to guide the trade-off process. A conservative adaptation policy forbids the adaptation mechanism from fixing a failing indicator if the value of a non-failing indicator is about to decrease. The absence of quantitative information limits the precision of the adaptation process, given that there might exist values of *CPs* that fix low priority indicator while all the other affected indicators still remain above their thresholds. On the other hand, optimistic adaptation policies allow tuning parameters that decrease indicators of higher priority requirements hoping they remain above their threshold. Again, the lack of quantitative relations and planning in the adaptation process can result in leading to failure important requirements in order to fix other, less significant ones.

3. A CONTROL-BASED APPROACH

Defining an adaptation strategy able to satisfy the most important requirements under the presence of uncertainty is not an easy task without adopting quantitative approaches. This section sketches a general control-based design procedure that can be used to accommodate this task.

A design process. As in various types of systems, some of the indicators might depend not only on the chosen value of the control parameter, but also on its past and on the values of other indicators [9, 16]. For instance, if the participation to the meetings drops and the value indicator I_2 is 60% instead of 75%, decreasing MCA, in order to fix this failure will not have immediate impact, but gradually I_2 will increase until it reaches the desired value. Such systems are called dynamic systems.

The first step is to better understand how the control parameters affect the indicators. The differential relations

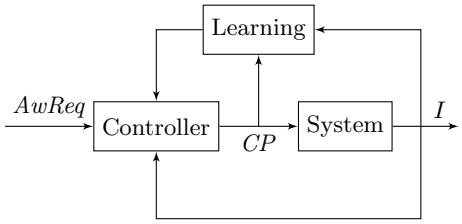

Figure 2: Reference control scheme.

presented in the previous section, provide only qualitative information, and cannot be easily exploited with control techniques. A more expressive way to capture these relations is to consider the relation between control parameters $CP(\cdot) \in \mathbb{R}^m$, and indicators $I(\cdot) \in \mathbb{R}^p$, as a discrete-time linear dynamic system

$$\begin{cases} x(t+1) = A \cdot x(t) + B \cdot CP(t) \\ I(t) = C \cdot x(t) \end{cases} \tag{1}$$

where $x(\cdot) \in \mathbb{R}^n$ is the *state* of the system—notice that the state might not have a meaningful interpretation, but it is functional to defining in a more compact form the relation between $CP(\cdot)$ and $I(\cdot)$. Since the system has m inputs, and p outputs, it is a Multiple-Input and Multiple-Output (MIMO) system.

The matrices (A, B, C) describe the dynamics of the system, and can be identified from experimental data through system identification techniques for MIMO systems, e.g., subspace identification methods see [10, 19, 20]. Note that in case A is a matrix of all zeros, the system is characterized as static. However, our MPC is also applicable to static MIMO systems. This analytical model can be used to predict the future behavior of the system over a finite time horizon, and as such is also referred as *prediction model*. Having identified the model of the system (1), the control scheme of Fig. 2 can be set up.

The next step is to design a control mechanism, therefore, decide what type of "Controller" to use [4, 13, 15]. MPC is a natural choice for the problem-at-hand for various reasons [12]. First of all, it is naturally formulated for MIMO systems. There are generally many control parameters, and indicators that need to be controlled. Moreover, both indicators and control parameters have upper and lower bounds that the decision-making strategy should take into account. MPC allows one to formulate the problem as the minimization of a functional subject to given constraints as follows:

$$\begin{aligned} \text{minimize}_{CP_{t+k}} \quad & \sum_{k=0}^{N-1} J(AwReq_{t+k}, I_{t+k}, CP_{t+k}) \\ \text{subject to} \quad & I_{\min} \leq I_{t+k} \leq I_{\max} \\ & CP_{\min} \leq CP_{t+k} \leq CP_{\max} \\ & x_{t+k+1} = A \cdot x_{t+k} + B \cdot CP_{t+k} \\ & I_{t+k} = C \cdot x_{t+k} \\ & x_t = x(t), \quad k = 0, \ldots, N-1. \end{aligned} \tag{2}$$

The optimization problem (2) is solved over a finite horizon of N steps ahead, thanks to the prediction model (1). The solution of the optimal control problem is a plan of future control parameter values $CP^\star_t, \ldots, CP^\star_{t+N-1}$. Only the first

element of this plan is applied, i.e., $CP(t) = CP^{\star}_{t}$. At time $t+1$, a new optimization problem is solved analogously.

It is important to notice that the cost function has to be designed according to the specific domain. A common choice is:

$$J(AwReq_t, I_t, CP_t) = \sum_i q_i \left(AwReq_{t,i} - I_{t,i}\right)^2 + \sum_j r_j CP^2_{t,j}$$

where $q_i \geq 0$ and $r_j > 0$. The values q_i are weighting the error between the setpoint and the output, using the requirement prioritization result of AHP. On the other hand, the values r_j represent penalties of using one control parameter over others, and it can be chosen according to the specific domain [3]. Finally, minimizing the cost function is interpreted as an effort by the adaptation mechanism to anticipate requirement failures using analytical prediction models, giving priority to those of higher importance, while minimize the aggregate penalties of the planned adaptation strategy. Whenever, it is unfeasible to eliminate completely within the time horizon all the failures, *EvoReqs* can be used to weaken the failure-insisting requirements, lowering their thresholds.

Apparently, the quality of the obtained solution depends on the accuracy of the extracted prediction model. It is possible that the dynamics relating the inputs and the outputs of the system are changing over time, or that the linear model (1) is not able to capture more complex dynamics of the system. Therefore, the "Learning" block, in the control scheme in Fig. 2 is in charge to monitor the input-output relation of the system, and update the model that is used in the control mechanism according to the actual behavior of the system. The learning mechanism can be based on many different algorithms, ranging from recursive least squares to recursive subspace identification [6, 11]. Independently of the learning mechanism, the "Learning" block is in charge of updating online the model adopted by the MPC algorithm to predict the future behavior of the system.

The proposed solution is able to cope with environmental uncertainty, while overcomes the limitation of having qualitative information coming from domain experts. In principle, designers can collect data on the input-output behavior of the system and therefore to identify a reasonably accurate linear model describing the dynamics of the system. The control algorithm exploits such a model for planning a suitable adaptation strategy, named plan of control signals in control-theoretical terms. Finally, the learning mechanism is in charge of updating the prediction model whenever it behaves differently than expected.

It is worth noticing that the proposed control scheme is generic in that the "Controller" and "Learning" blocks can be realized in terms of any controller or learning algorithm respectively. For our case, tools that combine optimization and satisfiability modulo theories, e.g. [17], can handle also boolean constraints to the MPC optimization problem. Such constraints are "If timetables are collected automatically, then the schedule is made automatically as well" (see Fig. 1).

The main limitation of the approach comes whenever new requirements are added to the problem. In that case, there are two possibilities. First, new indicators (i.e., new outputs) are introduced to the system and second new control parameters (i.e., inputs) are added. In both cases, the whole design procedure needs to be repeated, since no control-based technique is able to manage structural changes in the systems.

4. CONCLUSIONS AND FUTURE WORK

In this position paper we investigated the problem of designing an adaptation mechanism for MIMO software systems that operate within environmental uncertainty. We used meeting-scheduler as an exemplar to demonstrate the weaknesses of qualitative adaptation and the need of analytical models for better adaptation.

We address this problem by proposing the use of MPC. This type of control uses analytical models that describe the system's behavior allowing to forecast future failures in our requirements and anticipate them in an optimal way with respect to their priorities.

Finally, we plan to implement an MPC controller and experiment with simulations of the meeting-scheduler and other case studies to further evaluate our proposal.

5. 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). This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control", and through the LCCC Linnaeus and ELLIIT Excellence Centers.

6. REFERENCES

[1] K. Angelopoulos, V. E. S. Souza, and J. Mylopoulos. Dealing with multiple failures in zanshin: a control-theoretic approach. In *SEAMS 14*, pages 165–174. ACM, 2014.

[2] K. Angelopoulos, V. E. S. Souza, and J. Mylopoulos. Capturing variability in adaptation spaces: A three-peaks approach. In *Conceptual Modeling – ER 2015*. Paul Johannesson and Mong Li Lee and Stephen W. Liddle and Oscar Pastor, 2015 (to appear).

[3] A. E. Bryson and Y.-C. Ho. *Applied Optimal Control: Optimization, Estimation and Control*. Taylor & Francis, 1975.

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

[5] 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.

[6] I. Houtzager, J.-W. Wingerden, van, and M. Verhaegen. Fast-array recursive closed-loop subspace model identification. In *15th IFAC Symposium on System Identification*, volume 15, pages 96–101, 2009.

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

[8] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 3–12, June 2008.

[9] A. Leva, M. Maggio, A. V. Papadopoulos, and F. Terraneo. *Control-based operating system design*. Control Engineering Series. IET, 2013.

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

[11] M. Lovera, T. Gustafsson, and M. Verhaegen. Recursive subspace identification of linear and non-linear wiener state-space models. *Automatica*, 36(11):1639–1650, 2000.

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

[13] 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.

[14] B. Nuseibeh. Conflicting requirements: When the customer is not always right. *Requirements Engineering*, 1(1):70–71, 1996.

[15] A. V. Papadopoulos, M. Maggio, S. Negro, and A. Leva. General control-theoretical framework for online resource allocation in computing systems. *IET Control Theory & Applications*, 6(11):1594–1602, 2012.

[16] 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.

[17] R. Sebastiani and P. Trentin. Optimathsat: A tool for optimization modulo theories. In *In proc. Int. Conf. on Computer Aided verification, CAV'15*, 2015 (to appear).

[18] V. E. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos. Requirements-driven software evolution. *Comput. Sci.*, 28(4):311–329, Nov. 2013.

[19] G. van der Veen, J.-W. van Wingerden, M. Bergamasco, M. Lovera, and M. Verhaegen. Closed-loop subspace identification methods: an overview. *Control Theory Applications, IET*, 7(10):1339–1358, July 2013.

[20] M. Verhaegen and V. Verdult. *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, New York, NY, USA, 2012.

[21] Q. Zhang, Q. Zhu, M. Zhani, and R. Boutaba. Dynamic service placement in geographically distributed clouds. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 526–535, June 2012.