Towards Bridging the Gap between Control and Self-Adaptive System Properties

Javier Cámara
University of York, UK
javier.camaramoreno@york.ac.uk

Alessandro V. Papadopoulos
Mälardalen University, Sweden
alessandro.papadopoulos@mdh.se

Thomas Vogel
Humboldt University Berlin, Germany
thomas.vogel@cs.hu-berlin.de

Danny Weyns
KU Leuven, Belgium; Linnaeus, Sweden
danny.weyns@kuleuven.be

David Garlan
Carnegie Mellon University, USA
garlan@cs.cmu.edu

Shihong Huang
Florida Atlantic University, USA
shihong@fau.edu

Kenji Tei
Waseda University, Japan
ktei@aoni.waseda.jp

ABSTRACT

Two of the main paradigms used to build adaptive software employ different types of properties to capture relevant aspects of the system’s run-time behavior. On the one hand, control systems consider properties that concern static aspects like stability, as well as dynamic properties that capture the transient evolution of variables such as settling time. On the other hand, self-adaptive systems consider mostly non-functional properties that concern concerns such as performance, reliability, and cost. In general, it is not easy to reconcile these two types of properties or identify under which conditions they constitute a good fit to provide run-time guarantees. There is a need of identifying the key properties in the areas of control and self-adaptation, as well as of characterizing and mapping them to better understand how they relate and possibly complement each other. In this paper, we take a first step to tackle this problem by: (1) identifying a set of key properties in control theory, (2) illustrating the formalization of some of these properties employing temporal logic languages commonly used to engineer self-adaptive software systems, and (3) illustrating how to map key properties that characterize self-adaptive software systems into control properties, leveraging their formalization in temporal logics. We illustrate the different steps of the mapping on an exemplar case in the cloud computing domain and conclude with identifying open challenges in the area.

CCS CONCEPTS

• Software and its engineering → Extra-functional properties, Formal software verification.

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

SEAMS ’20, October 7–8, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7962-5/20/05... $15.00
https://doi.org/10.1145/3387939.3391568

1 INTRODUCTION

Two of the main paradigms used to build adaptive software employ different types of properties to capture relevant aspects of the system’s run-time behavior. On the one hand, control systems consider properties that concern static aspects like stability, as well as dynamic properties that capture transient aspects such as settling time. On the other hand, self-adaptive systems consider mostly non-functional properties that include concerns such as performance, cost, and reliability.

Self-adaptive software can clearly benefit from the potential that control theory provides in terms of enabling better analyzability and enforcement of constraints on run-time system behavior. Being able to formally reason about the non-functional concerns of a system (e.g., security, energy, performance) in terms of control properties in the presence of an unpredictable environment can optimize operation and improve the level of assurances that engineers can provide about the systems they build.

However, applying control theory to software systems poses a set of challenges that do not exist in other domains [10, 21]. One of the main challenges is that control-based solutions demand the availability of precise mathematical models that capture both the dynamics of the system under control, as well as the properties that engineers want to impose and reason about. When control is applied to physical plants, the laws that govern the system are captured by accurate mathematical models that are well-understood, and relevant properties like stability or performance are formally characterized by definitions that are precise and standard in the control community [4].
While obtaining accurate models of non-functional aspects of software behavior can to some extent be achieved using different methods like system identification [24], the self-adaptive software systems community still lacks a standard repertoire of runtime properties formally characterized in a way that makes them amenable to formal analysis using techniques applied by software engineers in self-adaptive systems (e.g., run-time verification, model checking). Having such a repertoire would not only help individual system designers express and check certain common fundamental properties, but also help promote norms for system assurance across the community of adaptive system developers.

Solving in software the kind of problems that control theory solves in other domains entails understanding how control properties relate to software requirements and formally characterizing such properties in a way that facilitates their instantiation and automated analysis using standard tools.

To advance the understanding of how self-adaptive system requirements relate to control properties, in this paper we: (1) identify a set of key properties in control theory, (2) illustrate the formalization of some of these properties employing temporal logic languages commonly used to engineer self-adaptive software systems, and (3) indicate how to map key properties that characterize self-adaptive software systems into control properties, leveraging their formalization in temporal logics. We illustrate the different steps of the mapping on an exemplary case in the cloud computing domain and conclude with identifying open challenges in the area.

2 BACKGROUND

In this section, we first present a basic set of concepts in control systems, followed by a description of a general class of discrete models that can be employed as building blocks to reason about properties that capture quantitative aspects of system behavior over time.

2.1 Control Terminology

In this paper, we focus mainly on continuous-time signals and systems, but equivalent definitions are present in the case of discrete-time [4].

For every output \( y(t) \) of the plant, one defines a desired behavior for it, which in control terms is called a setpoint or reference signal, and is represented by \( y^*(t) \in \mathbb{R}^p \).

The difference between the desired behavior and the actual behavior of the plant is called error, and is represented as \( e(t) \in \mathbb{R}^p \):

\[
e(t) = y^*(t) - y(t).
\]

The controller is a decision-making mechanism that given the error, decides what is the value of the control signal \( u(t) \in \mathbb{R}^m \) in order to make the error converge to zero. In principle, the control signal and the plant input should be the same, i.e., \( u(t) = u(t) \), but in practice, there might be a load disturbance \( l(t) \in \mathbb{R}^m \), that affects the controller decision. Therefore, it holds that \( u(t) = \hat{u}(t) + l(t) \).

The load disturbance is one of the main disturbances that affect the performance of control systems.

In addition, there might be a disturbance that is acting directly on the output of the plant, which is called output disturbance, and it is represented as \( d(t) \in \mathbb{R}^p \). Finally, there is noise \( n(t) \in \mathbb{R}^p \) that affects the measurements that one takes of the output. These two last sources of disturbances are typically “high-frequency” disturbances, and can be counteracted by a suitable filtering at design time of the controller.

As a main reference to these concepts, the interested reader can refer to the publicly available book by Åström and Murray [4].

2.2 Discrete Models

We consider the self-adaptive system as a black-box on which a set of output variables can be monitored over time. Concretely, we model the non-functional run-time behavior of a self-adaptive system as a transition system that captures the evolution over time of a set of relevant variables (i.e., state is characterized by a collection of \( n \) real-valued random variables \( Y = \{y_1, \ldots, y_n\} \)). These variables can be considered to be analogous to the output \( y(t) \) in a control system. Sampling these variables in space and time results in their quantization and time discretization.

Let \( [\alpha_i, \beta_i] \) be the range of \( y_i \), with \( \alpha_i, \beta_i \in \mathbb{R} \), and \( \eta_i \in \mathbb{R}^+ \) be its quantization parameter. Then, \( y_i \) takes its values in the set:

\[
[\mathbb{R}]y_i = \{ r \in [\mathbb{R}] \mid r = k \eta_i, k \in \mathbb{Z}, \alpha_i \leq r \leq \beta_i \}.
\]

Hence, given an observed value of \( y_i \) at time \( t \) (denoted as \( y_i(t) \)), the corresponding quantized value is obtained as:

\[
\text{quant}(y_i(t)) = \min(\arg \min_{r \in [\mathbb{R}]y_i} (|y_i(t) - r|)).
\]

Variables in \( Y \) define a state-space \( [\mathbb{R}^n]_Y = [\mathbb{R}]y_1 \times \ldots \times [\mathbb{R}]y_n \).

Furthermore, we assume a time discretization parameter \( \tau \in \mathbb{R}^+ \) associated with the sampling period established for the observation of variables, determining the transition time.

Figure 2 compares an arbitrary continuous system output \( y(t) \) with its quantized counterpart \( y_q(t) \) in the discrete timeline. \( y_q(t) \) takes values only in multiples of \( \eta_q \), and is represented in the figure as constant for intervals of duration \( \tau \).

Discrete models can be enriched with rewards and costs that help capture quantitative aspects of system behavior (e.g., elapsed time, energy consumption, cost) in a precise manner. These rewards can be employed as building blocks to reason about properties that capture quantitative aspects of system behavior over time.

\( ^1\)For convenience, we write in the following \( y_q(t) \) instead of \( \text{quant}(y(t)) \).
control the proportion of responses that include optional content. The goals of the target system are summarized in two functional and three non-functional requirements (Table 1).

| Requirement | Description |
|-------------|-------------|
| F1          | The target system shall respond to every request for serving its content. |
| F2          | The target system shall serve optional content to the connected clients. |
| NFR1        | The target system shall operate under low cost. Subject to NFR1 and NFR2, the cost (i.e., the number of servers) should be minimized. |
| NFR2        | The target system shall provide high availability of the optional content. Subject to NFR1, the percentage of requests with optional content (i.e., the dimmer value) should be maximized. |

There is a strict preference order among the non-functional requirements that deal with optimization, so trade-offs among different dimensions to be optimized are not possible (i.e., no solution should compromise maximizing the percentage of requests with optional content to reduce cost). The imposition of a preference order is aimed at better capturing real scenarios and is not a limitation imposed by any of the compared approaches, which are also able to capture non-strict preference orders among requirements.

4 CHARACTERIZING CONTROL PROPERTIES

Control systems are usually concerned about four main objectives [10], namely: (a) setpoint tracking, which is related to achieving the specified setpoint whenever it is reachable, (b) transient behavior, concerned about how setpoints are reached, in particular in the presence of abrupt changes, (c) robustness to inaccurate or delayed measurements, related to the ability of a controller to behave correctly even when transient errors or delayed data is provided to it, and (d) disturbance rejection, related to the ability of avoiding any effect of external interferences on system goals. These high level objectives can be mapped in control theory into the satisfaction by design of properties like stability, guaranteed settling time, integrated squared error, that relate to the achievable runtime performance of the control system. In this section, we describe these properties, going from their mathematical formulation into their characterization in temporal logics commonly used in formal verification like LTL [17], CTL [7], and PCTL [11]. Other properties exist in control theory, but having a complete catalogue here is beyond the scope of this paper, and it is left as future work.

4.1 Stability

The concept of stability in control theory differs from the notion of stability used in self-adaptive software. A control system is stable even if the error \( e(t) \) is not converging to zero, but it is bounded. More specifically, in control terms, if the initial value of system output \( y(0) \) is “close” to the equilibrium value \( y^0 \), then the evolution over time of the output \( y(t) \) will be bounded (and not diverge) from \( y^0 \). More formally:

\[
\text{stby} \equiv \forall \epsilon > 0 \exists \delta(e) \left[ \| y(0) - y^0 \| < \delta(e) \Rightarrow \| y(t) - y^0 \| < \epsilon, \forall t > 0 \right]
\]

A system is asymptotically stable, if it is stable (as per the previous definition), and also if the evolution over time of the system output...
will eventually converge to $y^\infty$. More formally:

$$\text{as}_{stby} \equiv \text{stby} \land \lim_{t \to \infty} \|y(t) - y^\infty\| = 0 \tag{2}$$

![Figure 4: Example of system output stabilization.](image)

Characterization in Temporal Logic. Characterizing stability in temporal logic requires capturing the constraints imposed by the definition of stability given in Expression 1. Such characterization can be given on a quantized version of the variables and constants required to define stability:

$$\left[s_{by}\right] \equiv \|y_q - y^\infty_q\| < \delta_q \Rightarrow \Box(\|y_q - y^\infty_q\| < \epsilon_q) \tag{3}$$

In Expression 3, the subscript $q$ indicates that the constant or variable on which it appears is the quantized version of its continuous counterpart (i.e., $y_q(t) \equiv \text{quant}(y(t))$, cf. Section 2.2). It is worth noticing that variables in a software system are quantized by definition, and requiring a notion of asymptotic stability may be too restrictive. The current definition captures the same concept with $\epsilon_q$ being the resolution of the quantization or a tolerance parameter. Moreover, the absence of explicit time indexes is consistent with the implicit notion of time introduced by the temporal operators. For instance, when $y_q$ is not within the scope of any temporal operator (like in the antecedent of the implication given in the formula), the expression refers to the value of the variable in the first state of the trace (i.e., $y_q \equiv y_q(0)$). However, if the same term is within the scope of a temporal operator as it happens with $\Box$ on the right-hand side of the expression, then the same $y_q$ refers to the value of $y_q(t)$ in all subsequent states of the discrete temporal line (i.e., $y_q(t)$ when $t = 0, r, 2r, \ldots$). The non-probabilistic version of this property is directly expressible in LTL and CTL (as $A[s_{by}]$), whereas its probabilistic version can employ the probability quantifier of PCTL (e.g., $P_{\geq}[s_{by}], P_{eq}[s_{by}]$).

Instantiation in RUBiS. Expression 4 instantiates $[s_{by}]$, in a straightforward manner for the response time variable $r$, assuming a setpoint equivalent to the threshold $T$. It states that when the error becomes smaller than $\delta_q$ it will stay within the band $[T - \epsilon_q, T + \epsilon_q]$:

$$\|r_q - T\| < \delta_q \Rightarrow \Box(\|r_q - T\| < \epsilon_q) \tag{4}$$

4.2 Settling Time

One of the key indicators of how the system reaches its goals is settling time $t_s$, which is the time needed by the system to reach a new steady-state equilibrium. For an arbitrary $\epsilon \in \mathbb{R}^+$, the $\epsilon$-settling time is defined by:

$$t_{s,\epsilon} \equiv \inf \left\{ \delta \text{ s.t. } \|y(t) - y^\infty\| < \epsilon, \forall t \in [\delta, \infty] \right\} \tag{5}$$

In Expression 5, the settling time is captured as the infimum of the set of time values in the continuous timeline for which the error is bounded by $\epsilon$ in the following. Note that the infimum is the greatest lower bound that always exists, meaning that it takes the value $\infty$ if the stability condition is never satisfied.

Characterization in Temporal Logic. In contrast with stability, which is a boolean property that is either satisfied by the system or not (Expression 3), settling time is a quantitative property and therefore we characterize it as a temporal logic expression that employs a reward quantifier. Since in this case the reward captures time, we assume the existence of a transition reward function $[\text{time}] \equiv (\text{true}, \tau)$ that accrues the time quantum employed for time in the discrete model whenever a transition in the discrete timeline is taken:

$$[t_{s,\epsilon}] \equiv R_{\geq}[\text{time}] \left[ \Box \Box[\|y_q - y^\infty_q\| < \epsilon_q] \right] \tag{6}$$

Expression 6 characterizes the settling time as the time reward accrued until the system reaches a state from which the error is bounded by $\epsilon_q$. There are two aspects of this characterization that are important to highlight. First, the reachability formula accrues reward until it reaches a state that satisfies the reachability predicate, but the reward in the latter state is not included. Second, when the reachability predicate is not satisfied, the semantics of the reward quantifier assign an infinite reward as the value that is obtained when the expression is quantified (e.g., in PCTL, co-safe LTL with rewards). These two aspects make this characterization consistent with the definition given in Expression 5, which defines the settling time as the time instant immediately prior to the one in which the error is already bound by $\epsilon$, and becomes infinite if the error is not always bound by $\epsilon$, starting at some arbitrary point in the timeline. Note that, due to the nesting of temporal operators including $\Box$, this property is not (currently) directly expressible in temporal logics with Markovian rewards as implemented in probabilistic model checkers like PRISM [15] and Storm [8]. However, assuming finite traces in our discrete models, we can perform a preprocessing step on the traces, labeling explicitly states from which $\Box[\|y_q - y^\infty_q\| < \epsilon_q]$ as $p$, and then model check the property as:

$$R_{\geq}[\text{time}] [\Diamond \Box[\|r_q - T\| < \epsilon_q]] \tag{7}$$

Instantiation in RUBiS. Expression 8 instantiates $[t_{s,\epsilon}]$ with similar assumptions to those adopted for Expression 4:

$$R_{\geq}[\text{time}] \left[ \Diamond \Box[\|r_q - T\| < \epsilon_q] \right] \tag{8}$$

4.3 Integrated Squared Error

Relevant quantitative measures of a system’s performance are also often based on the behavior of the error $e(t)$. We consider here as a representative index the integrated squared of the error (ISE):

$$\text{ISE} \equiv \int_0^T e^2(t)dt \tag{9}$$

The ISE integrates the square of the error over time (see Figure 5), penalizing large errors more than smaller ones (the square of a large error will be much bigger). Control systems specified to minimize ISE of the tracking error $e(t)$, e.g., MPC or LQG [5], tend to eliminate large errors quickly, but tolerate small ones persisting for a long period of time. This often leads to fast responses, but with considerably low-amplitude oscillation.
Towards Bridging the Gap between Control and Self-Adaptive System Properties

Characterization in Temporal Logic. Similar to the settling time, the ISE is a quantitative property and therefore we characterize it making use of a reward quantifier. Since in this case the reward has to capture accrued error over time, we assume the existence of a transition reward function \([\text{error}] \equiv (true, (\|y_q - y_q^s\|)^2)\) that accrues the square of the instantaneous error whenever a transition in the discrete temporal line is taken.

Then, we can write an expression that accrues the error reward over the discrete timeline before stability is achieved:

\[
[\text{ISE}] \equiv R_{\leq t}^\text{error}[(\square\Box)(y_q - y_q^s) < \epsilon_q] \quad (10)
\]

Due to the nesting of \(\Diamond\), this property is not directly expressible in PCTL/Co-safe LTL with rewards. However, under the same assumptions described for the settling time property, a similar model preprocessing step can enable its practical verification through a simpler probabilistic reachability property (cf. Expression 7).

Instatiation in RUBiS. We assume that RUBiS is working on a steady state, but suddenly receives a spike on request arrival rate, causing the average response time \(r\) to go above threshold \(T\) (Figure 6). After violating the threshold, the system adds a server to drive down the response time below \(T\). Before the system stabilizes, its response time may experience some oscillations that make \(r\) go above and below \(T\) several times. For simplicity, we assume \(y^0 = T\).

To obtain an indication of how well the system is adapting, we can employ a modified version of the \([\text{ISE}]\) property (Expression 10). In this case, we are only interested in accruing a penalty whenever the output of the system is above the threshold \(T\), therefore we adapt the reward structure for the error, constraining it to accrue reward only whenever \(r > T\), i.e., \([\text{penalty}] \equiv (r > T, (r - T)^2)\):

\[
R_{\leq t}^\text{penalty}[(\square\Box)(r - T) < \epsilon_q] \quad (11)
\]

We can observe that the accrued error corresponds to the colored areas enclosed by \(T\) and \(r(t)\) in Figure 6. Since negative error (i.e., when \(r < T\)) does not constitute a violation of the response time threshold, we do not accrue it, in contrast with the more general property described in Expressions 9 and 10.

5 FORMALIZING NON-FUNCTIONAL REQUIREMENTS

The non-functional run-time behavior of self-adaptive systems can be captured by an external observer as a set of quantitative indicators that represent attributes of different concerns such as performance, cost, or availability. In this section, we employ the characterization of control properties in temporal logic introduced in the previous section as building blocks to formalize non-functional requirements in RUBiS.

NFR1. The target system shall demonstrate high performance. The average response time \(r\) should not exceed \(T\). This requirement can be captured by combining temporal logic properties of: (i) stability as described by Expression 4, (ii) settling time as captured by Expression 8, and (iii) an integrated error property analogous to Expression 11 using the penalty, \([\text{penalty}] \equiv (r > T, r - T)\) which should be guaranteed to be always less or equal to zero, i.e.:

\[
R_{\leq t}^\text{penalty}[(\square\Box)(r - y_q^s) < \epsilon_q] \leq 0 \quad (12)
\]

Note that in the expression above, the error term \((r - y_q^s)\) does not make the simplifying assumption included in Expression 11, and incorporates an arbitrary setpoint different from \(T\). This makes sense in a realistic setting because, if \(y_q^s \neq T\), oscillations around the setpoint during transients would always result in response time threshold violations. This is also applicable to properties (i) and (ii) for this requirement.

NFR2. The target system shall provide high availability of the optional content. Subject to NFR1, the percentage of requests with optional content (i.e., the dimmer value \(d\)) should be maximized. Capturing this requirement requires instantiating the integrated error property on variable \(d\), which should be always as close as possible to 1 (maximum optional content):

\[
R_{\leq t}^\text{optional}[(\square\Box)(1 - d_q) < \epsilon_d] \quad (13)
\]

where \([\text{optional}] \equiv (true, d)\). Note that in this case, the magnitude of the error is always below 1, so minimizing the non-squared error is a more sensible choice.

NFR3. The target operating system shall operate under low cost. Subject to NFR1 and NFR2, the cost (i.e., the number of servers \(s\)) should be minimized. The formalization of this requirement can be captured using the following properties defined over the response time variable \(r\): (i) stability as described by Expression 4, (ii) settling time as captured by Expression 8. Finally, we can capture the penalty of using extra servers during the transient by employing an integral error property which should minimize the use of servers according to \([\text{penalty}] \equiv (r > T, s^2)\):

\[
R_{\leq t}^\text{penalty}[(\square\Box)(r - T) < \epsilon_q] \quad (14)
\]

Note that in this case, stability and settling time properties are defined over response time \(r\), whereas penalty is defined over the number of servers employed \(s\), making an interesting case in which formalizing a single requirement involves combining different control properties across variables.

All variables might present similar patterns in terms of control properties, but the composition of the self-adaptive properties is...
We have grouped related work in four parts: control applied to computing systems, automatically generated control solutions, verification of control properties, and evaluation of quality properties.

**Control Applied to Computing System.** In 2004, Hellerstein et al. wrote a pioneering book on applying control theory to computing systems [12]. Over the years, control-based approaches have been applied extensively to computing systems, mostly focusing on controlling lower-level resources. Abdelzaher et al. apply different types of controller models (e.g., PI and PID) to deal with performance requirements of servers [2]. Wang et al. present DEUCON that allocates local controllers to computing units that only coordinate with neighbors [26]. Stability analysis is based on the location of poles of the composite system’s transfer function. Imes et al. present CoPPer, a control-theoretic approach that applies adaptive control to meet soft performance goals by manipulating hardware power limits [13]. In contrast, our work targets a mapping between classic control properties and typical software qualities.

**Automatically Generated Control Solutions.** To deal with the complexity of control theory, researchers have started investigating automatic generation of control solutions to adapt software [10, 23, 27]. Filieri et al. introduce the push-button methodology (PBM) that automatically constructs a linear model of a software system for a PI controller to adapt the system for one setpoint goal [9]. Shevtsov et al. propose a solution to control a software system for multiple goals, including an optimization goal [22]. Maggio et al. apply model-predictive control (MPC) to software adaptation [16], while Anagelopoulos et al. apply a requirements-driven approach with MPC [3]. These approaches highlight properties that are important from a control-theoretic viewpoint, but this accounts for only one side of the problem we target in this paper, namely, a rigorous specification and verification of classic control properties.

**Verification of Control Properties.** Some work exists on the formalisation and verification of properties of control properties. We highlight two representative examples. Preuse and Hanisch apply model checking to verify safety, liveness and deadlock properties of manufacturing control systems that are specified in temporal logic [18]. Yan et al. use approximate bisimulation for comparing the similarity between a complex (continuous) cyber-physical system and a (discretized) higher level model of it [29]. The authors illustrate the approach for a safety property. Our work complements these approaches by focusing on typical software quality properties and the formal mapping of these with control properties.

**Evaluation of Quality Properties.** A number of approaches zoom in on the evaluation of quality properties in self-adaptive systems. Weyns and Ahmad [28] performed a systematic literature review identifying the main property considerations in self-adaptation: efficiency/performance of the system (55% of the studies), reliability (41%), and flexibility (28%). Reincke et al. [20] propose a payoff metric to measure the “success” of adaptation. This metric is a user-defined function aggregating QoS metrics observed on the running system similarly to a utility function. Villegas et al. [25] present a framework to evaluate adaptation properties, i.e., stability, accuracy, settling time, overshoot, robustness, termination of adaptation, consistency, scalability, and security. The properties are informally defined and mapped to software qualities based on examples from literature. Raibulet et al. [19] focus on quality attributes to evaluate the utility of a self-adaptive system, and software metrics to evaluate the quality of the adaptation at runtime, whereas Cámara and de Lemos [6] evaluate resilience properties formalized in PCTL. Each of these approaches contributes to a better understanding of quality properties and their evaluation from a software engineering point of view. However, this only accounts for one side of the mapping problem we target in this paper, i.e., a traditional software engineering perspective.

**Conclusion.** While control theory and self-adaptive systems contribute knowledge about properties in their domain, there is little understanding on the mapping between the two types of properties, which is precisely the target of the research presented in this paper.

**7 CONCLUSIONS AND FUTURE WORK**

In this paper, we have taken the first step in bridging the gap between control and self-adaptive system properties. We have (1) identified key properties in control theory (stability, settling time, and integral error), (2) formalized these properties in temporal logic languages, which are typically used to specify properties (requirements) of software systems, and (3) illustrated how non-functional properties of self-adaptive systems (performance, availability, and costs) can be mapped into these control properties by using this formalization and the RUBiS exemplar. To achieve the formalization and mapping, we have discussed the abstraction of transition systems describing discrete state spaces on which self-adaptive system attributes are measured and how this abstraction is able to represent continuous system dynamics in which control properties are typically characterized. Models of such transition systems and control properties formalized in a temporal logic can serve as input for off-the-shelf run-time verification tools and model checkers.

This approach advances the understanding of how non-functional requirements relate to control properties (e.g., which requirements can be characterized by which control properties) and paves the way for an improved operation and assurance of self-adaptive systems via formal reasoning (e.g., by run-time verification) based on control. Our approach is currently limited by the set of control properties that we have formalized, requirements that we have mapped into control properties (cf. previous paragraph), and the expressiveness of temporal logics, which might not be able to fully capture the nuances of some control properties (cf. Section 4.2).

Our long-term goal is to understand whether control theory can be used as a formal foundation for specifying and analyzing self-adaptive systems, and if so, under which conditions. Towards that goal, work is needed to identify further corresponding and complementing properties between self-adaptive and control systems (e.g., whether real-time or security requirements can be mapped into control properties), and to leverage the formalization of properties for a formal assessment of controllers in self-adaptive systems (e.g., to provide guarantees for the correctness of controllers).
