Technical Briefing: Control Theory for Software Engineering

Antonio Filieri
Imperial College London
London, UK
a.filieri@imperial.ac.uk

Martina Maggio
Lund University
Lund, Sweden
martina@control.lth.se

ABSTRACT

Pervasiveness and complexity of modern software are challenging engineers to design applications able to guarantee the desired quality of service despite unpredictable runtime variations in their execution environment. A variety of techniques have been proposed in the last year for the design of self-adaptive applications; however, most of them is tailored to specific applications or can provide limited guarantees of effectiveness and dependability.

Control theory has, on the other hand, developed a wide set of mathematically grounded methods for many engineering domains that interact with the physical world. However, applying these methods to software systems is not straightforward. Software is rarely designed to be controllable and its behavior is hard to model in formalisms amenable to control.

In this technical briefing we will recall a set of foundational concepts of control theory, explain how they can be transposed in the software engineering domain, and discuss some insights into the design of controllable software.

1. INTRODUCTION

The pervasiveness of software in every context of life is placing new challenges to Software Engineering. Highly dynamic environments, rapidly changing requirements, unpredictable and uncertain operating conditions are pushing new paradigms for software design, leveraging runtime adaptation mechanisms to overcome the lack of knowledge at design time and design more robust software [1, 2, 3]. Self-adaptive software systems are getting growing importance for their ability of continuously assuring their requirements in spite of dynamic, unpredictable, and uncertain execution environments. However, many of the current approaches lack the formal grounding needed to guarantee the effectiveness, robustness, and dependability of the adaptation mechanisms [4].

Modern software should be already *designed* to deal with change. Runtime variations are not sporadic: a software that is written to be executed on one platform is then executed on another, with different resources that may be shared among many applications at the same time; users are unpredictable and may change their behavior due to external constraints; architectures are faulty and may not provide the desired performance. Dealing with change is a complex problem. Some of the changes that occur in a modern software are under the control of the developer; others are under the control of the owner. In general, the complexity is induced by the presence of multiple stakeholders, with conflicting goals and by the vast number of actuators that these stakeholders can control in the system: clock speed, computing resources, idle time, caches, memory allocation, disk consumption.

All the existing systems to deal with change fall short on a few important points. These systems are also often tailored to a specific architecture and fail at portability. They are often developed ad hoc, studying the system behavior in certain conditions. Whenever the running conditions are modified, the provided solutions might misbehave. Finally, these systems usually do not provide theoretical and formal guarantees.

In the past decades, Control Theory developed a broad set of mathematically grounded techniques for adapting physical plants. Despite the parallel within the two adaptation problems is self-evident, defining control theoretical techniques for software adaptation places unprecedented challenges for both disciplines [5, 6, 7, 8]. Software engineers are required to abstract system behaviors into mathematical models suitable for control and to devise broadly applicable development processes taking "controllability" as a first class concern already from early stages of design [9, 10]. Control Theory, used to deal mostly with unchangeable physical laws and limited measurable quantities, has to readjust to a new playground where the "physics" of virtual environments is often a design choice, calling for a paradigm shift from classic control. Most of the attempts to apply "off-the-shelf" control theory to software applications have failed. It is indeed challenging to model software systems as *dynamical system* — i.e. by means of differential equations — because of the intrinsic non-linearity, the variety of usage profiles, and the interconnection of heterogeneous components.

Recently, several control theoretical adaptation techniques appeared in literature [11], some focusing on specific parts of the feedback control loop (e.g., learning the system model), while other going through the entire feedback loop, including the automatic synthesis of the system model and the controller. In this Technical Briefing we will overview some broadly applicable state of the art techniques and demonstrate how they can be used to support the adaptivity of a variety of systems.

2. TOPICS

The main topics of the technical briefing will include: foundational definitions concerning Control Theory and self-adaptive systems, software behavior modeling from the two perspectives, system identification, PID and optimal control, and multi-objective control. Recommended course material will include an extended

ICSE '16 May 14-22, 2016, Austin, TX, USA

ACM ISBN 978-1-4503-4205-6/16/05.

DOI: http://dx.doi.org/10.1145/2889160.2891058

version of the paper Control Theory meets Software Engineering presented at SEAMS 2015 [12], plus additional material related to the specific topics.

The control design process will start with the definition of the goals of the system that will be used by the controller as feedback signals and to measure and quantify when the adaptive system is fulfilling its objectives. The process continues with the identification of the quantities that can be changed at runtime to realize the adaptation features, usually called “knobs" or “actuators". We will cover how to define a model for the system, based on these identified notions. In this section of the briefing we will describe the concept of control-theoretical software models. In fact, many of the control methods that will be described require a model in terms of a differential or difference equations which software engineers would not normally define. To model software systems mathematically, the numerous methods from *system identification* can be used. Broadly, these methods construct models using measurements. For software, building these models might be easier than for physical systems as running software is much cheaper and can be done faster. This means large amounts of data for identification purposes are easily available.

Given the model of the system, the next step in the briefing will be to synthesize a controller with the most appropriate method among the many different ones that control theory offers. Such a scheme will be complemented by auto-tuning, continuous learning and adaptive control.

A section of the briefing will be devoted to highlighting what kind of formal guarantees can be offered and how to formally derive the mathematical proofs of setpoint tracking, behavior during the transient phase, rejection of disturbances and robustness to inaccurate or delayed measurements. These theoretical guarantees will be mapped into the corresponding control properties of stability, absence of overshooting, settling time and robustness.

3. METHOD

The envisioned teaching method will alternate the presentation of the topics and their practical demonstration on a few small cases. The participants will get access to a code repository containing ready to use examples, useful both to see how the presented tools look like on code and to bring initial code snippets at home for their own projects.

4. RELEVANCE AND TIMELINESS

The growing number of publications in the last 5 years incorporating control theoretical results for self-adaptive software, both in specialized venues as SEAMS, ICAC, and SASO, and in broader ones as ICSE, ESEC/FSE, ASE, and ASPLOS, but also the growing number of publications focusing on computing system appearing on large conferences in control, e.g., CDC, suggest a current and large interest of the two communities on the topic.

5. ORGANIZERS

Antonio Filieri

Antonio Filieri is a Lecturer (Assistant Professor) at Imperial College London. His main research interests are in the application of mathematical methods for software engineering, in particular Probability, Statistics, Logic, and Control theory. The main topics of his recent publications include control-theoretical software adaptation, exact and approximate analysis methods for probabilistic software analysis, quantitative software modeling and verification at runtime, and incremental verification. More info at: www.antonio.filieri.name .

Martina Maggio

Martina Maggio is currently an assistant professor at the Lund University, where she is one of the actors in the Cloud Control project, an ambitious project that brings control theory into cloud computing. Her research is mainly focused on the application of control theory to computing systems and resulted in important international publications in different computer science subfields, from hardware to software. More info at: www.martinamaggio.com .

References

[1] R. Laddaga. “Guest Editor’s Introduction: Creating Robust Software through Self-Adaptation”. In: *IEEE Intelligent Systems* 14 (3 1999), pp. 26–29. DOI: 10.1109/MIS.1999.769879.

[2] J. O. Kephart and D. M. Chess. “The Vision of Autonomic Computing”. In: *IEEE Computer* 36.1 (2003), pp. 41–50. DOI: 10.1109/MC.2003.1160055.

[3] M. Salehie and L. Tahvildari. “Self-adaptive Software: Landscape and Research Challenges”. In: *ACM Transactions on Autonomous and Adaptive Systems* 4.2 (May 2009), 14:1–14:42. DOI: 10.1145/1516533.1516538.

[4] R. de Lemos et al. “Software Engineering for Self-Adaptive Systems: A second Research Roadmap”. In: Dagstuhl Seminar Proceedings 10431. Schloss Dagstuhl, 2011.

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

[6] Y. Diao et al. “Self-Managing Systems: A Control Theory Foundation”. In: *Proceedings of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*. ECBS. IEEE CS, 2005, pp. 441–448. DOI: 10.1109/ECBS.2005.60.

[7] A. Filieri et al. “Software Engineering Meets Control Theory”. In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’15. IEEE Press, 2015, pp. 71–82.

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

[9] A. Filieri, H. Hoffmann, and M. Maggio. “Automated Design of Self-adaptive Software with Control-theoretical Formal Guarantees”. In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE ’14. ACM, 2014, pp. 299–310. DOI: 10.1145/2568225.2568272.

[10] 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. ACM, 2015, pp. 13–24. DOI: 10.1145/2786805.2786833.

[11] T. Patikirikorala, A. Colman, J. Han, and L. Wang. “A Systematic Survey on the Design of Self-adaptive Software Systems Using Control Engineering Approaches”. In: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS ’12. IEEE Press, 2012, pp. 33–42. DOI: 10.1109/SEAMS.2012.6224389.

[12] *CTSE 2015: Proceedings of the 1st International Workshop on Control Theory for Software Engineering, co-located with ESEC/FSE 2015*. ACM, 2015.