Control-theory and models at runtime

Pierre-Alain Muller [1], Olivier Barais [2]

[1] Université de Haute-Alsace
Mulhouse, France
pierre-alain.muller@uha.fr

[2] IRISA / INRIA Rennes
Rennes, France
Olivier.barrais@irisa.fr

Abstract. Models at runtime are considered a key enabling technology for systems that control themselves as they operate. The automatic-control community has developed extensive theories and experiences in qualifying the properties of controller and systems; including stability, observability and commandability. In this position paper we propose to use control-theory for describing self-adaptive model-driven systems

1 Introduction

We are witnessing significant interest in the model-driven community for using models at runtime, beyond the now traditional construction phases of software applications. Behind models at runtime, there is the idea that automatic or even self-adaptability of the running system may be achieved by taking decisions based on monitoring information captured by runtime models. In the summary of last year's edition of the models@runtine workshop [1], workshop organizers illustrated this by saying that they foresaw that models could help moving a system from a consistent architecture to another one.

While this trend is new in the modeling community, models at runtime have been around for decades in the automatic-control community; where they have been applied to many different domains, including: avionics, automotive, robotics, finance and biology. There are certainly major differences behind what these two communities recognize as models; however we have the feeling that there are also significant commonalities, such as dealing with the behavior of dynamic systems, or being an interdisciplinary branch of engineering and mathematics; and that the model-driven community could benefit from part of the experience and knowledge gathered by the automatic-control community.

In this short position paper we suggest that control-theory may be used for describing model-driven self-adaptive systems.

2 Quick overview of control-theory

Control-theory is about controlling the behavior of dynamic systems. Analytical models are used by controllers to drive systems. Those models are typically built as mathematical expressions (for instance differential equations) and reflect the behavior of the system under study. They can be queried in place of the system, mainly to answer such questions as: "what happens if the system is excited in some given way?", or "how much margin do we have in some dimension before the system gets unstable?".

A system (often also called a process in the automatic-control community) is seen as a box with input and output variables. The goal of a controller is to act on input signals, in such a way that outputs of the systems follows (reflects at any time) a given reference signal.

The following bullets summarize basic notions of control-theory:

- Identification. Identification is related to the definition of an analytical model which translates the physical reality of the system.
- Control-law. A control law is part of a controller, and calculates the control input (of the process) based on the values of other inputs (variables, states). Types of control laws depend on types of models found during identification, and on response requirements for the automatically controlled system.
- Open-loop controller. The control-law of the controller takes the reference and calculates the inputs to the system under control. Open-loop controllers work best if the model at hand is a faithful translation of reality.
- Closed-loop controller. The controller takes the difference (known as the error) between the reference and the actual outputs to change the inputs to the system under control. At this point, it is enough to say that closed-loop controllers outperform open-loop controllers in terms of reference tracking, sensitivity to parameter variations, disturbance rejection and stabilization of unstable processes.
- Stability. Stability ensures that a system will not be endangered by out of range (theoretically infinite) values. For any bounded input over any amount of time, the output will also be bounded.

- Controllability. Controllability is related to the possibility of forcing the system into a particular state by using an appropriate control signal. If a state is not controllable, then no signal will ever be able to stabilize the system.

- Observability. Observability is related to the possibility of observing, through output measurements, the state of a system. If a state is not observable, the controller will never be able to correct the closed-loop behavior if such a state is not desirable.

- Determinism. A system that will always produce the same output for a given input is said to be deterministic. A system that will produce different outputs for a given input is said to be stochastic.

A typical control loop is represented in Figure 1. Control-theory splits the world into a controller and a plant. The controller is responsible for sending signals to the plant, according to a control law, so that the output of the plant follows a reference (the expected ideal output). Actually, as the system is working here in closed-loop, the controller is driven by the difference (known as the error) between the ideal reference and the actual output of the system.

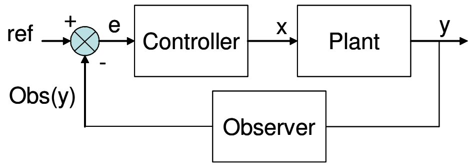

Figure 1: typical control loop for dynamic systems

3 Applicability to model-driven development

The applicability of control-theory to self-organizing software has already been suggested by Kokar et al. in [2]. In this position paper, we build on their view, while focusing on models, especially models at runtime.

A model-driven software application is parameterized by input models. These input models may be either type or token models [3] (this dichotomy includes metamodels and conforming models). These two types of models correspond to different ways of parameterizing the system, thus of adapting the system. Type models contain general set-like descriptions which apply to collections of items. Token models contain specific items, in a one-to-one correspondence with the system's items that they represent. Typical examples of applications' generators following this dual scheme include EMF [4] and Netsilon [5]. Adaptation occurs when the generated software application (or part of it) is regenerated, in order to cope with some evolution request.

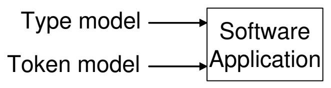

Figure 2: Model-driven parameterization of a running software application

In the framework of control-theory, the overall development of adaptive model-driven software application may be represented as a closed-loop system. The plant is the running software application, and the reference is the goal of the system (made of functional and non-functional requirements). The controller is the development approach in place, by which the software application is built and deployed, and evolved over time. The observer monitors relevant runtime information; it is a place where runtime models are consumed, manipulated and transformed.

Model-driven approaches take models as input and produce software development artifacts as output. The actual behavior of the generated software applications is supposed to be inline with the expected behavior described in the models. Feedback is performed when developers operate changes (perform control) either on the input models, or on the transformation process, in order to evolve the software application. The challenge of self-adaptive model-driven application is to embed automatic controllers in running systems; ultimately to reengineer the application automatically.

There are many places where models may come into play at runtime in such self-adaptive software system. As we have seen earlier, type and token models may be used to parameterize software system. This includes runtime parameterization of the plant, but also of the controller and the observer. The feedback loop is another place where models at runtime may play an important role. Monitoring models may capture information about the running conditions of the system, and the automatic controller may then change the inputs of the plant (remember that these may be models as well) based on the observation of these monitoring models.

Obviously, control-theory provides a framework for organizing the discussion about models at runtime. By using this conceptual framework, adaptive model-driven software applications should benefit from the theoretical results and practical experiences from the automatic control community. One issue is to be sure (to prove) that such self-adaptive systems keep running within acceptable boundaries. We expect support from control-theory for ensuring stability of the behavioral semantics of adaptive model-driven applications.

4 Research directions for applying control-theory to model-driven development

Applying control-theory to model-driven development implies re-defining basic notions of control-theory (as exposed earlier in section 2) in the context of software engineering. At least, we need two define the following three items.

Defining stability criteria for software

A system is said to be stable when little disturbances applied to the system have negligible effects on its behavior. In terms of model-driven software systems, this means that small changes in the input models do not radically change the behavior of the system. Most important is the notion of margin, that is the amount of changes that can be applied to inputs without disturbing too much the system, and make it become unstable.

Defining controllability criteria for software

A system is controllable if it can be driven (say by a model) in a desired direction. Controllability is about proving that systems will perform according to the specifications when inputs change. For instance, this might be true as long as token models conform to type models; the system is then likely to react as expected. General criteria for controllability of software have to be defined (in the same way that control-theory created tools for linear systems). The database field may be of interest as this community has gathered extensive experience about schema migration (that is type model migration while preserving conforming token models data).

Defining observability criteria for software

For a system to be controllable, with a closed-loop controller, it must be possible get precise information about any state of the system at any time. The point here is to be able to build representations (models) which cover all relevant aspects of a system, as far as control is concerned. This is the place for analytical models which gather information about systems, including at runtime, such as trace models.

5 Conclusion

In this position paper we have shown how control-theory may be used to represent the development of self-adaptive model-driven applications. We have suggested that control-theory might be an important element for the study of models at runtime, especially in the context of self-adaptation. We have suggested research directions to reformulate basic notions of control-theory applied to model-driven development, in the context of self-adaptive systems.

References

[1] N. Bencomo, G. Blair, et R. France, Summary of the Workshop Models@run.time at MoDELS 2006, *Electronic Source:* http://www.comp.lancs.ac.uk/~bencomo/MART06/

[2] M. M. Kokar, K. Baclawski, et Y. A. Eracar. "Control Theory-Based Foundations of Self-Controlling Software", *IEEE Intelligent Software*, vol., 1999.

[3] T. Kühne. "Matters of (meta-) modeling", *SoSyM*, vol. 5 (4), 2006.

[4] Eclipse, Eclipse Modeling Framework (EMF), *Electronic Source:* http://www.eclipse.org/emf/

[5] P.-A. Muller, P. Studer, F. Fondement, et J. Bézivin. "Platform Independent Web Application Modeling and Development with Netsilon", *Software and Systems Modeling*, vol. Vol 4 (4), pp. 424-442, 2005.