Uniform and Model-Driven Engineering of Feedback Control Systems

Filip Křikava, Philippe Collet, Mireille Blay-Fornarino

Uniform and Model-Driven Engineering of Feedback Control Systems

Filip Křikava
Université Nice Sophia Antipolis, I3S - CNRS UMR 6070, Sophia Antipolis, France
filip.krikava@i3s.unice.fr

Philippe Collet
Université Nice Sophia Antipolis, I3S - CNRS UMR 6070, Sophia Antipolis, France
philippe.collet@unice.fr

Mireille Blay-Fornarino
Université Nice Sophia Antipolis, I3S - CNRS UMR 6070, Sophia Antipolis, France
blay@polytech.unice.fr

ABSTRACT

Engineering and reusing feedback control systems face challenging issues, such as structuring control loops to allow for fine-grained reasoning about their architecture. We propose a model-driven approach in which all major parts of the feedback control are uniformly designed as first-class adaptive elements. Expected properties of the approach are discussed and illustrated on a real scenario of overload control in a grid middleware.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.11 [**Software Engineering**]: Software Architectures

Keywords

Autonomic Computing, Model-Driven Engineering, Software Architecture, Grid Computing

General Terms

Design, Reliability

1. INTRODUCTION

The 24/7 deployment of distributed systems is dramatically increasing the complexity and maintenance costs of software systems. Autonomic Computing aims at providing computing systems and applications that can dynamically adjust themselves to accommodate changing environments and user needs with minimal or no human intervention [6]. However, reliable and cost-effective engineering of such systems is challenging [3, 8]. Despite significant work on architectures, such as the MAPE-K decomposition [6], and frameworks [4], some major difficulties still rest in finding the appropriate model for the controlling part [5], structuring and coordinating several resulting control loops, making loops and their elements explicit and reusable [3], as well as having a flexible system level support with an appropriate level of abstraction to allow for rapid prototyping [1].

Following the general principle that control loops should be made explicit [3], we propose a model-driven approach to build externally defined control mechanisms using feedback control loops. All elements of control loops are elevated to be first-class entities at design time, so that precise control can be modeled and manipulated on the control loop elements themselves.

2. APPROACH

We focus on externalized control in which the adaptation concerns are separated from the target system functionalities. In this way the self-adaptive capabilities can be more easily modified or removed. The architecture of the adaptation is designed in a technologically agnostic model that allows for supporting different degrees of separation in respect to the target system: from running in a separate runtime to being integrated inside the system using aspect-oriented techniques or direct source code generation.

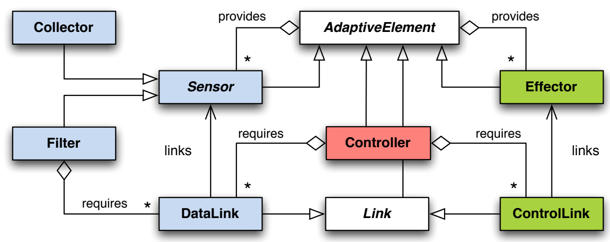

Figure 1: Structural Model (*excerpt*)

Fig. 1 presents an excerpt of the structural model that represents the architecture of a feedback control system. The main elements represent an observing (*sensor*), an adapting (*controller*) and an actuating part (*effector*), as well as a binding in between them (*link*).

One originality of the approach is in making all these elements adaptive themselves by inheriting from an *adaptive element*, which itself can provide its own sensors and effectors. This feature allows them to be observed (meta-data, state) and modified in the very same way as the target controlled system is. One can hierarchically compose not only control on the top of controllers, but also on the top of sensors, effectors and links. This way the added adaptation becomes self-adaptable as well, and in an uniform way. This notably distinguishes the model from other model-driven proposals [2].

In the model, the acyclic graph of connected sensors that forms the monitoring part of the system is responsible for providing all inputs necessary for controllers to infer the state of the system and their environment. The leaf nodes in the structure are *collectors*. They encapsulate data obtained from external entities like operating system resources, service calls, etc. An *active collector* is responsible for updating itself in cases where the update is based on some external notifications like a file change, new socket connection, etc. A *passive collector* on the other hand, waits until it is explicitly requested by an associated link to provide data. The other sensors (*filters*) are used to aggregate or in some other way process data that are coming from their children nodes. The data communication between the nodes can be configured at runtime in either an *observing* or a *notifying* mode, corresponding to having a parent node triggering the observation of its children or children posting a notification to its parents. The model can be checked to

ICAC'11, June 14–18, 2011, Karlsruhe, Germany.
ACM 978-1-4503-0607-2/11/06.

ensure that the dataflow toward controllers is well typed and not interrupted.

The adaptation part is represented by controllers and effectors. Controllers orchestrate the bounded effectors (via *control link*) that carry out the actual system modification. There are many different kinds of controllers that can be used depending on the intended purpose [7, 5]. However, in our approach, we are not directly concerned by the actual design of the adaptation behavior itself, we focus on supporting the surrounding architecture as a whole.

The links themselves support adaptation so that their transmission properties, such as the notification or observing periods, or in case of remote links connection timeouts, can be advertised and used by other loops to make the system more robust.

3. CASE STUDY IN GRID MIDDLEWARE

Among others, our work is motivated by several use cases that aim at improving parts of the gLite[1] grid middleware operation by introducing self-adaptive capabilities. One of them is the overload control scenario of the Workload Management System (WMS), which is responsible for distributing and managing jobs across grid resources.

Fig. 2 exemplifies our proposal with a simplified version of the corresponding autonomic architecture. The main controller (1) objective is the optimization of the number of tasks (N) in the WMS Task Queue by adjusting the job submission requests arrival rate (λ), based on the resource usage observed from the system sensors (R). The adjustment can be either by simply refusing the requests or, for aware clients, by imposing a delay before next submission. The second controller (2) coordinates the first one by setting the reference value (R^*) for the resource usage based on the system administrator preference ($\overline{R}$) and the system availability (S). Shall these two controllers fail (incorrect model, system in an unanticipated state, etc.), the system may get overloaded and so to prevent this, the controller (3) is employed, running at lower pace, restarting the process when resource consumption is over a limit. The controller (4) is used to try to start the WMS in cases it is not running. In order not to conflict with the controller (3), it double checks the heartbeat with a delay.

The primary concern in this overload control scenario is to protect every unbounded resource usage, regardless whether it is a system resource or a service usage. This has to be done recursively since protecting one resource might in consequence jeopardize another and thus transforming one problem into a different one that could be potentially much worse. Following a *"at least do no harm"* principle, one can prevent the system being flooded with restart or start requests, for example by using an exponential back-off to control the minimum time between the requests. The controllers (2) and (4) are examples of controllers running at different time scales [7].

Besides the controllers, like (1) and (2) in our case, which usually have to be always designed specifically for each particular scenario, the other parts of the architecture are likely to be reused in other scenarios. This is an undergoing experimentation.

4. CONCLUSION & ONGOING WORK

We have presented an approach to support the architectural design of control loop elements and their connection links. Each of them being an adaptive element enables architects to uniformly explicit control patterns, from coordination between loops to adaptive monitoring. The presented model is complemented by a deployment model, not shown here, which deals with element locations and instantiation issues.

[1] `http://glite.web.cern.ch/glite/`

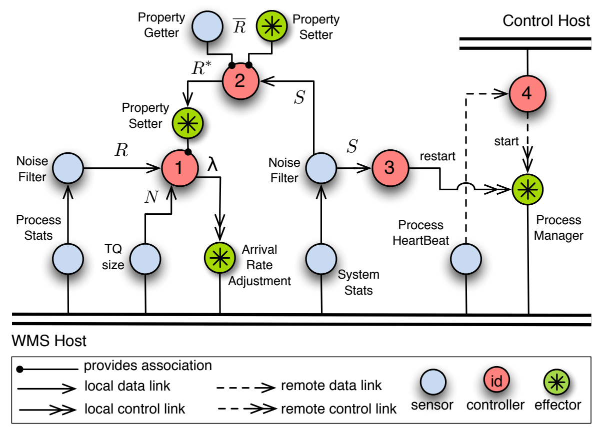

Figure 2: WMS Overload Control Architecture (*excerpt*)

A first runtime platform is based on an OSGi runtime with a DSL for model definition and code generation. Ongoing work deals with the distribution aspect of the system as well as with generating different compact representations in which a large number of model elements are merged at runtime. Additionally, SCA components and C++ code are targeted in two different grid middlewares, gLite and Condor[2] respectively. In the longer term, we expect the model to be the underlying basis for a library of loop elements and patterns.

Acknowledgements. The work reported in this paper is partly funded by the ANR SALTY project[3] under contract ANR-09-SEGI-012.

5. REFERENCES

[1] O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, and A. van Moorsel. The Self-star Vision. *Self-star Properties in Complex Information Systems*, pages 1–20, 2005.

[2] L. Broto, D. Hagimont, E. Annoni, B. Combemale, and J.-P. Bahsoun. Towards a model driven autonomic management system. In *Fifth International Conference on Information Technology: New Generations*, pages 63–69, 2008.

[3] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering Self-Adaptive Systems Through Feedback Loops. *Software Engineering for Self-Adaptive Systems*, pages 48–70, 2009.

[4] D. Garlan, B. Schmerl, and P. Steenkiste. Rainbow: architecture-based self-adaptation with reusable infrastructure. *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 276–277, 2004.

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

[6] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *Computer*, 36:41–50, 2003.

[7] M. Litoiu, M. Woodside, and T. Zheng. Hierarchical model-based autonomic control of software systems. *Proceedings of the 2005 workshop on Design and evolution of autonomic application software - DEAS '05*, page 1, 2005.

[8] M. Salehie and L. Tahvildari. Self-adaptive software. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, May 2009.

[2] `http://www.cs.wisc.edu/condor`
[3] `https://salty.unice.fr`