Integrating Adaptation Mechanisms Using Control Theory Centric Architecture Models: A Case Study

Filip Křikava, *University of Lille 1 and Inria;* Philippe Collet, *Université Nice Sophia Antipolis;* Romain Rouvoy, *University of Lille 1 and Inria*

https://www.usenix.org/conference/icac14/technical-sessions/presentation/krikava

This paper is included in the Proceedings of the 11th International Conference on Autonomic Computing (ICAC '14).
June 18–20, 2014 • Philadelphia, PA
ISBN 978-1-931971-11-9

Integrating Adaptation Mechanisms Using Control Theory Centric Architecture Models: A Case Study

Filip Křikava
University Lille 1 / Inria

Philippe Collet
Université Nice Sophia Antipolis

Romain Rouvoy
University Lille 1 / Inria

Abstract

Control theory provides solid foundations for developing reliable and scalable feedback control for software systems. Although, feedback controllers have been acknowledged to efficiently solve common classes of problems, their adoption by state-of-the-art approaches for designing self-adaptation in legacy software systems remains limited and at best consists in *ad hoc* integrations, which are usually engineered manually.

In this paper, we revisit the *Znn.com* case study and we present an alternative implementation based on classical feedback controllers. We show how these controllers can be easily integrated into software systems through control theory centric architecture models and domain-specific modeling support. We also provide an assessment of the resulting properties, quality attributes and limitations.

1 Introduction

Feedback control is acknowledged as one of the viable solutions for self-adaptive software systems engineering [8, 31, 34]. It provides solid foundations and a systematic approach for designing reliable and robust adaptation mechanisms, *controllers*, which drive the system adaptation [3]. However, integrating such controllers into legacy software systems remains challenging [8,10]. In particular, this requires selecting the appropriate target system outputs and control inputs (*touchpoints*), devising the actual controller design, and finally a software architecture integrating the controller into the target system [21]. As a matter of example, well-established feedback controllers for common and recurring problems (*e.g. Quality of Service* (QoS) management [2, 20] or performance guarantees [1, 3, 4]), are being integrated into target systems and tuned manually. Even though supporting tools, such as MATLAB, SIMULINK, or SYSWEAVER [35], provide code generation capabilities, the controller integration into the target system still requires an extensive handcrafting of a non-trivial code that results in significant accidental complexities. Moreover, these tools mostly target embedded real-time systems rather than distributed enterprise systems.

In this paper, we revisit the *Znn.com* case study [12], an acknowledged case study from the self-adaptive software systems community[1], and we describe an elegant solution integrating classical feedback controllers using control theory centric architecture models [27]. *Znn.com* is a web-based N-tier client-server system that models a news service provider like *cnn.com*. The main control objective is to make *Znn.com* to serve its content within acceptable response time and quality even in the event of traffic spikes caused by highly popular news by using *content adaptation* (*e.g.* serving reduced content quality). This paper contributes to demonstrate a systematic integration of a control theory based approach that addresses the *Znn.com* control objective.

Our solution is based on a technologically agnostic *Domain-Specific Modeling Language* (DSML) for defining *Feedback Control Loops* (FCLs). It supports composition, distribution and reflection, thereby enabling coordination and composition of multiple distributed FCLs using control schemes. It raises the level of abstraction at which the FCL architectures are defined, and a support is provided for automated implementation code synthesis and verification. The application to *Znn.com* enables us to demonstrate the model capabilities to progressively refine adaptation mechanisms, going from local content delivery adaptation using an existing and proven control algorithm [2, 3] to distributed content adaptation, and finally to adaptive control.

2 Related Work

IBM proposed MAPE-K decomposition of a FCL [24] which has become a widely referenced model for autonomic systems, followed by number of framework-based approaches [34]. *The Rainbow framework* [18] provides

[1] http://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/model-problem-znn-com

an architecture-based approach for self-adaptive software systems using utility theory for an optimal adaptation strategy selection. *The DYNAMICO model* [38] defines a fixed three-layers architecture with three FCLs for managing control objectives, target system adaptation and dynamic monitoring. *The StarMX framework* [6] designs self-managing Java-based applications using JMX for target system touchpoints and a policy-rule language for adaption engine. *The Zanshin Framework* [5] uses requirements engineering and goal models for self-adaptive software development.

The advantage of the above framework based solutions is that they provide an architecture basis of an application and therefore they can simplify its development. However, the adaptation mechanisms within these frameworks mostly use a simple, fixed threshold-based event-condition actions, not providing support for control theory based controllers. For example, both DYNAMICO and Zanshin implements *Znn.com* decision policies by using simple conditions such as `experiencedRespTime > MAX_RESPTIME` [11, p. 187]. As a result, the target system is more likely to experience instability due to oscillations (*e.g.* continuously enlisting and discharging servers). Furthermore, frameworks always impose the use of a specific technological stack. The level of abstraction and formal reasoning is also usually limited since the adaptation is an integral part of the implementation. Finally, except DYNAMICO, they are primarily designed for scenarios that can be solved by centralized control loop and do not allow hierarchical control schemes nor adaptive control as they do not support runtime modifications of adaptation strategies or thresholds.

The *model@run.time* approaches are using models to represent abstractions of running systems and MDE techniques for their adaptation at runtime [15]. For example, Vogel *et al.* [40] propose runtime executable megamodels with a language for adaptation logic modeling and a runtime interpreter. Similarly to our approach they also support hierarchically organization and FCL coordination, however, they present only a high-level overview of how the actual adaptations look like.

A lot of effort has been also invested in tools for engineering feedback control for real-time embedded systems. *Ptolemy II* [13] is an extensive framework for the simulation of concurrent actor-oriented systems allowing to combine heterogeneous models of computation. We follow a similar actor-oriented approach and our execution semantics is derived from Ptolemy push-pull model of computation (*cf.* Section 3.2). However, Ptolemy focus rather on simulation of the executable models and their transformations to the embedded systems. SIMULINK is an industry standard tool for developing feedback control targeting primarily embedded systems. SYSWEAVER extends SIMULINK code generation capabilities for distributed real-time systems.

3 Control Theory Centric Architecture Models

This section outlines our approach for integrating adaptation mechanisms into software systems through control theory centric architecture models. A detailed description is provided in [26].

3.1 Principles and Design Decisions

Generality (applicability to a wide range of target platforms and adaptation scenarios), *visibility* (explicit FCLs, their processes and interactions), and *composability* (fine-grained reusable elements representing the FCL processes) are all well-identified requirements for FCL engineering [8, 10, 32, 34]. In order to meet these requirements, we structure the approach around a DSML with an actor-oriented design. The key advantage of a DSML is the possibility to raise the level of abstraction at which the FCLs are described and directly use the FCL domain concepts. Moreover, DSMLs are particularly suitable for automated reasoning and implementation code synthesis [25]. Since FCLs are inherently concurrent, we choose an actor-oriented design [22] representing the FCL processes as message-passing actors. The actor model allows to implement FCLs without worrying about thread safety, it is scalable [19] and seamlessly supports remote distribution.

For illustration, we use the Apache overload control FCL (*cf.* Figure 1) from Hellerstain *et al.* [21, §4.6.2][2], which can be considered as a simple *Znn.com* adaptation mechanism. It adjusts the maximum number of simultaneous connections (*MC*) based on the difference between reference (MEM^*) and actual (*MEM*) memory usage.

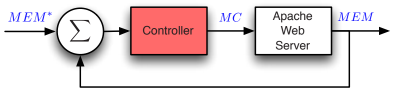

Figure 1: Apache overload control block diagram

3.2 Feedback Control Definition Language

Our approach is based on an actor-oriented component meta-model for representing FCLs abstractions, called *Feedback Control Definition Language* (FCDL) [27]. The components are actor-like entities called *Adaptive Elements* (AE) that are connected into hierarchically composed networks that form closed FCLs.

Syntax. An AE defines properties and input/output ports through which it communicates with other AEs using either data-driven (*push*) and demand-driven (*pull*) mode.

[2]For simplicity, we only use the case with one controller.

Once an AE receives a message, it executes its associated behavior whose result may or may not be sent further to the connected downstream elements which in turn will cause them to react and so on and so forth. An AE can be *passive*, *i.e.* triggered by a message, or *active*, *i.e.* triggered by an external event (*e.g.* a file modification). The ports and properties data values are statically typed and FCDL further supports parametric polymorphism. We recognize the following types of AE: *a sensor* (raw information collection), *an effector* (changes propagation), *a processor* (data processing and analyzing), and *a controller* (decision making). FCDL also contains a *composite* type that can be created from both atomic AEs and other composites. It can define ports, which are used to promote ports of the contained elements. Furthermore, a composite is also the primary unit of deployment.

Figure 2 shows an FCDL model implementing the FCL from Figure 1. The figure uses an informal FCDL graphical notation (a formal textual syntax is presented further in Section 4).

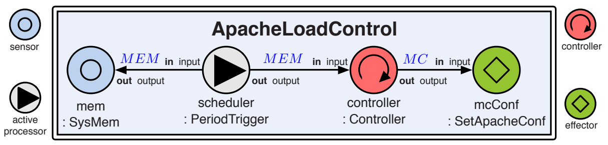

Figure 2: A FCDL model of Apache overload control

The `PeriodicTrigger` is an active processor. It periodically pulls memory utilization (MEM) from `SysMem` sensors and in turn pushes the value to the `Controller` that computes a new MC configuration to be applied by the `SetApacheConf` effector. The MEM^* value is modeled as a property of the controller.

Conceptually, each AE can be seen as a target system itself, and as such it can provide sensors and effectors enabling the AE reflection. This is a crucial feature permitting to hierarchically organize multiple FCL [36] in an uniform way and therefore realize complex control schemes from elementary building blocks.

Semantics. The execution semantics is based on the Ptolemy [13] push-pull model of computation [42]. We further adapt a notion of *Interaction Contracts* (IC) to precisely define allowed interactions of AE [9]. An IC specifies what ports activate an AE, what inputs might be pulled during AE execution, and what outputs might push results. For example, the IC associated with `PeriodicTrigger` is $\langle self; \Downarrow$ (`input`); $\Uparrow$ (`output?`)$\rangle$. It denotes an interaction caused by a *self* activation, pulling data from the `input` port and conditionally pushing data to the `output` port. ICs allow for asserting certain architectural properties (*e.g.*, consistency, determinacy, completeness) and they denote the type of the associate activation function making the generated source code both *prescriptive* (guiding developers) and *restrictive* (limiting developers to what the architecture allows).

4 Application to ZNN.COM

The main *Znn.com* control objective is content adaptation whereby the delivered content quality (*e.g.* degraded image quality) is reduced when the server is under heavy load. This has been well studied by Abdelzaher *et al.* [1–3], providing a control theoretic approach, which we integrate into *Znn.com* using FCDL.

4.1 Local Content Delivery Adaptation

The aim of the adaptation is to maintain web server load at a certain pre-set value. The server content is preprocessed and stored in M trees where each one offers the same content, but of a different quality and therefore size. At runtime, a given URL request, *e.g.* `photo.jpg`, is served from either `/full/photo.jpg` or `/degraded/-photo.jpg` depending on the current load of the server. Since the resource utilization is proportional to the size of the content delivered, offering the content from the degraded tree helps to reduce the server load.

Controller Design. Abdelzaher *et al.* [2,3] proposes two controllers: a simple integral controller and a more sophisticated proportional integral controller. Due to the space limitations, in this paper we only consider the former one, however, from the software architecture perspective, the only difference between them is the type of AE that is instantiated. The focus of FCDL is to facilitate the controller integration into software system not to develop of the controller itself.

The controller input is the web server utilization $U = aR + bW$ that is periodically computed using request rate $R = \frac{\sum r}{t}$ and delivered bandwidth $W = \frac{\sum w}{t}$, where a and b are platform constants[3] and $\sum r$, $\sum w$ are the number of requests and the amount of bytes sent over some period of time t, respectively. The controller output is the severity of the adaptation action $G = G + K_I E = G + K_I(U^* - U)$ where K_I is the controller integral gain, U^* is the target utilization (set by a system administrator) and U is the observed utilization. It determines which content tree should be used ranging from $G = M$, servicing all requests using the highest quality content tree to $G = 0$ in which case all requests are rejected.

Architecture. Figure 3 shows one possible integration of the above controller into the target system using FCDL.

For the *decision-making* part we create an AE, `IController`, that implements a general integral controller. Once a new value (U) is pushed into its input, it computes and pushes the control input (G). Both the integral gain (K_I) and the reference input (U^*) are represented as the controller properties. The *monitoring part* periodically computes server utilization U. Both the R

[3] *cf.* Abdelzaher *et al.* [2,3]

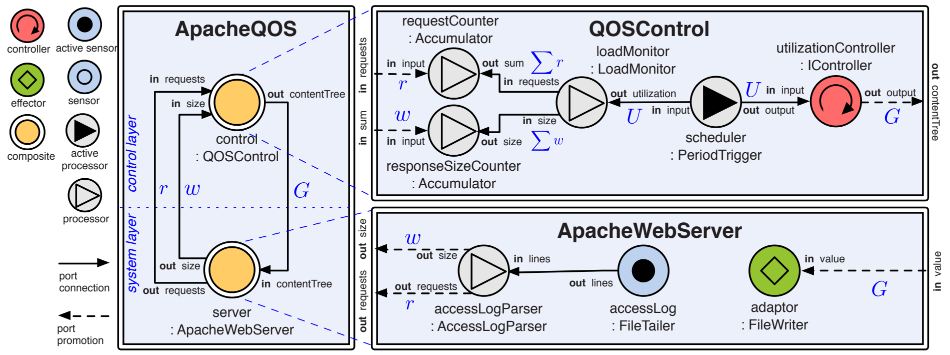

Figure 3: Apache content delivery control

and W can be obtained from Apache access log file. We create an active sensor, `FileTailer`, that activates every time a file content changes pushing out the modified part. The connected `AccessLogParser` extracts the number of requests r, the size of the responses w and pushes the values into the connected counters `requestCounter` and `responseSizeCounter`. To compute utilization U, the sum of requests $\sum r$ and response size $\sum w$ has to be converted into request rate R and bandwidth W—*i.e.*, the number of requests and sent bytes over certain time period t. We reuse the periodic trigger, which by pulling its input causes `LoadMonitor` to compute U using the accumulated $\sum r$, $\sum w$ sums. In the *reconfiguration part*, the `FileWriter` updates the web server URL rewrite rules reflecting the newly computed content tree.

To demonstrate composition, the presented elements are assembled into three composites `ApacheQOS`, `QOSControl` and `ApacheWebServer`, representing the main composite that will be deployed, the control, and the target system, respectively. This makes a clear separation of concerns and easy to switch from web server implementation to another.

Implementation. FCDL models are implemented in a domain-specific language called *Extended Feedback Control Definition Language* (XFCDL). It is a textual DSL for authoring FCDL models that further supports modularization and AE implementation using a Java-like expression language Xbase[4]. Listing 1 shows an excerpt[5] of the `IController` AE. Line 1 defines a new active polymorphic processor type with data type parameter T, followed by ports declaration (lines 2-4) and property definition (line 6). Line 7 specifies an IC and line 10 provides its implementation directly in Xbase.

```
1  active processor PeriodicTrigger<T> {
2    push in port output: T
3    pull in port input: T
4    self port selfport: long // self port for self-activation
5
6    property initialPeriod: Duration = 10.seconds
7    act activate(selfport; input; output?)
8
9    implementation xbase {
10     act activate { output.put(input.get) }
11   }
12 }
```

Listing 1: XFCDL code of `PeriodicTrigger` AE

4.2 Distributed Adaptation

Next, we extend the adaptation to cover distributed *Znn.com* deployment on a pool of replicated servers with a load balancer.

Controller Design. The distributed deployment consists of a server pool S with n servers and one load balancer. Each server S_i runs locally the previously developed `ApacheQOS` FCL computing its target content tree G_i. In order to maintain the highest QoS, the load balancer dynamically schedules the arriving requests to a server $s \in S$ that provides the least degraded content: $\texttt{content\_tree}(s) = \texttt{max}(\texttt{content\_tree}(S))$.

Architecture. Figure 4 depicts the FCL architecture representing the distributed control. The `LocalApacheQOS` runs at each of the server S_i, encapsulating the local `ApacheQOS` FCL. The `LoadBalancerControl` runs on the load balancer controlling the scheduler using the above equation.

The load balancer FCL first collects the content tree (G) status of all the participating servers using distributed publish/subscribe event bus. An advantage of using an event bus is that it does not need to be *a priori* aware of all the participating servers. In FCDL, an event bus is facilitated by two AEs: the publisher (`EventBusPublisher`) and the subscriber (`EventBusSubscriber`). We use key-value tuples of servers S_i (server hostname) with their corresponding content trees G_i. The G_i is obtained from a newly promoted `ApacheQOS` port `contentTree` so that the G is available from the outside. The pushed (S_i, G_i) entries are received by the `EventBusSubscriber` and aggregated using the `MapStore` AE, which is a map storage. The server with the highest G is selected by the `MapMaxKey` AE and consequently used to update the load balancer scheduling rules.

[4] A statically typed Java-like expression language `http://bit.ly/1mr36bt`

[5] The complete XFCDL code is available from the companion website `http://fikovnik.github.io/Actress/ICAC14.html`

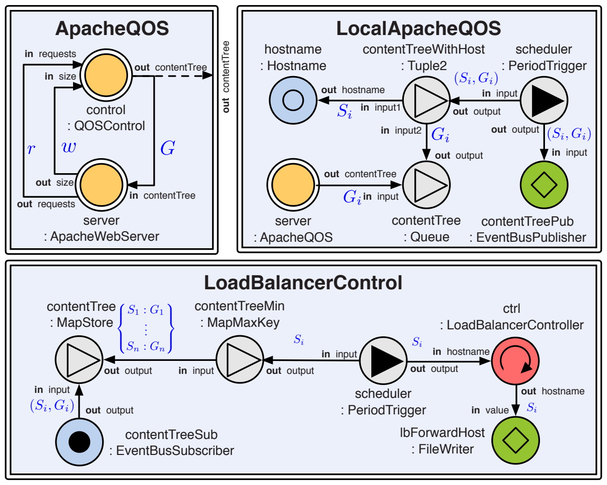
ApacheQOS
LocalApacheQOS
LoadBalancerControl
control : QOSControl
server : ApacheWebServer
hostname : Hostname
contentTreeWithHost : Tuple2
scheduler : PeriodTrigger
contentTree : Queue
contentTreePub : EventBusPublisher
contentTree : MapStore
contentTreeMin : MapMaxKey
ctrl : LoadBalancerController
contentTreeSub : EventBusSubscriber
lbForwardHost : FileWriter

Figure 4: Distributed QoS Management Control FCLs

4.3 System Identification

Controllers for software systems are usually driven by *"black box"* models derived from experimental runs collecting data and statistical model constructions. An experimental run consists of observing the effect of control inputs on the measured outputs. In FCDL, this can be facilitated by designing an open loop architecture in which target system touchpoints are used to set control inputs and observe/log corresponding system outputs. For example, Figure 5 shows an architecture model for tuning the controller from Section 4.1 into an open loop that can exercise the system on a range of inputs and log its outputs. Instead of connecting a controller output into the `ApacheWebServer` content tree input, we connect it directly to a value generator, a discrete sine wave.

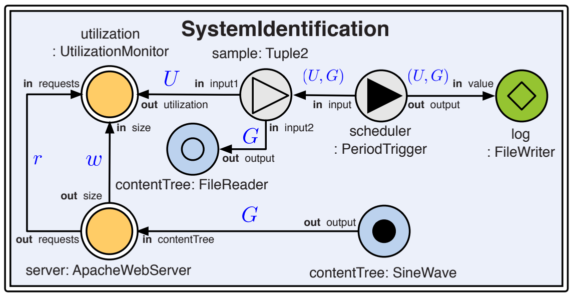

Figure 5: Apache content delivery control. The `UtilizationMonitor` contains the `requestCounter`, `responseSizeCounter` and `loadMonitor` elements from Figure 4.

4.4 Adaptive Control

An adaptive control improve FCL portability to load conditions and platform resource capacities that have not been anticipated during the system identification [4]. In FCDL, an adaptive control is facilitated by the model reflection. Figure 6 depicts an architecture of adaptive control for local content delivery adaptation FCL (*cf.* Section 4.1) that reuses part of the system identification developed above.

The aim is to perform an online profiling of the target system (relation between U and G), based on which we estimate the controller parameters (K_I). First the `IController` is extended with a provided effector to allow to change K_I at runtime. Next, we reuse the part of the architecture developed for the system identification and we create an `AdaptiveController` for the parameter estimation. It can be implemented using an adaptive controller as shown by Lu *et al.* [29] or by constructing a dynamic system model as proposed by Filieri *et al.* [37]. Finally, we encapsulate the corresponding elements into a new composite `AdaptiveControl` that can be placed on the top of the FCL developed in Section 4.1.

Adaptive control is one example of the FCDL reflection capabilities, which can also be used to design adaptive monitoring, or to organize multiple FCLs using various control schemes, such as hierarchical control.

5 Assessment and Discussion

In this section we assess our approach by discussing its properties and quality attributes. As such, the assessment is rather qualitative since we do not evaluate the controllers themselves.

Implementation. To facilitate the development using FCDL, we have implemented a prototype of a Java/EMF [14] based modeling environment called ACTRESS [27]. It provides support for FCDL modeling, verification (user defined constraints and temporal properties) and source code generation together with runtime platform.

Properties. The generality is addressed by using a fine-grained FCL decomposition which should be usable in any domain for various adaptation property. FCDL is built as a technologically-agnostic model and the Java based ACTRESS implementation provides only one technological solution. Generality is also obtained in adaptation scenarios, as they are captured at a conceptual level using the problem domain concepts, rather than the implementation concepts. The visibility property is tackled by having all the FCL processes represented as first-class entities with explicit interactions that are precisely guided by interaction contracts. Finally, we have shown that FCLs are composed from clearly structured fine-grained AEs using ICs to guide AE interactions and implementations (cf. Section 4.1).

Quality Attributes. Among many software quality attributes, the following are relevant for evaluating self-adaptive engineering approaches and have been already used by others, *e.g.* Asadollahi *et al.* [6].

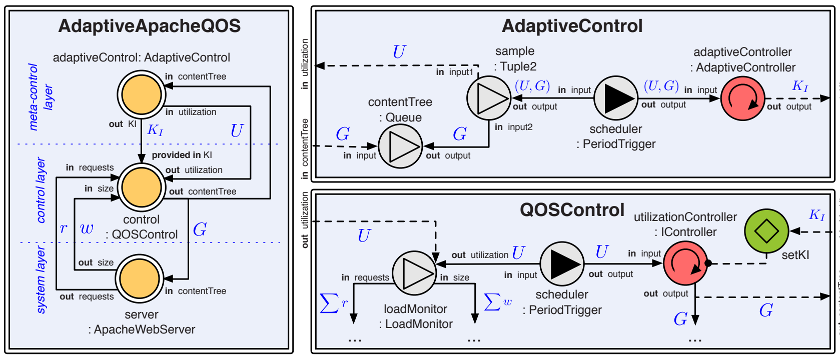

Figure 6: Adaptive control for Apache content delivery controller

– *Flexibility.* FCDL can represent both closed and open control loops and AE reflection allows for designing complex control schemes. Unlike most frameworks, FCDL does not dictate any system architecture nor any specific technology. Furthermore, it promotes separation of concerns in the sense that the FCL architecture and control mechanisms may be defined by control engineers while the technical/system-level processors and touchpoints may be implemented by software engineers. Next to the *Znn.com* case study, FCDL was also used to build overload control adaptation scenarios in the domain of high-throughput computing [26, §8.1].
– *Scalability.* The FCDL support for composition, polymorphic data types and ICs allowed us to incrementally refine the needed FCLs throughout the case study. These techniques are likely to allow for building larger models.
– *Usability.* Our approach relies on known concepts, as FCDL is using notions from control theory and component-based software engineering, XFCDL follows known concepts from Java, and the ACTRESS modeling environment is integrated in the Eclipse IDE, which might simplify adoption for the users already familiar with it. Furthermore, using on the actor-model simplifies AE implementation without the need to protect mutable states [19]. The implementation effort varies between 200-300 XFCDL lines of code per scenario[6].
– *Reusability.* There are two features that contributes to AE reusability: the FCDL support for data type polymorphism and the Xbase support for lambda expressions that allows to use functions types as properties. This results in higher-order polymorphic AEs definitions.
– *Extensibility.* FCDL and XFCDL are both defined using their respective EMF meta-models. Therefore, extending their core functionality is only possible by modifying the ACTRESS source code. On the other hand, thanks to MDE, it is possible to use the FCDL models and target different systems, providing new code generators, verification techniques and the like.
– *Performance.* The ACTRESS runtime is based on Akka[7] which with no AE deployed accounts for 1.5MB[8]. The memory overhead is about 400 bytes per actor instance with a possible throughput of 50 million messages per sec on a single machine[9]. The size and the CPU time of an AE is mostly affected by the amount of state it keeps and the complexity of its activation methods. However, the main potential performance issues are in the indirect load caused by the sensors and effectors.

6 Conclusions

While control theory provides solid foundations for designing self-adaptive systems, its mapping into implementation artifacts often results in the development of dedicated assets (*e.g.* code, models) which inevitably prevents their reuse and adoption at a larger scale. To overcome this limitation, we define FCDL, a domain-specific modeling language for integrating adaptation mechanisms into legacy software systems. We demonstrated its use on an implementation of local and distributed content delivery adaptation and distributed resource management. FCDL is a domain-specific and technologically-agnostic architecture model that provides an actor-based programming model.

Currently we are focusing in carrying more case studies, in particular targeting different self-adaptive properties and improvements such as support for distributed deployment and failure propagation. For future work, we intend to investigate adaptive feedback controllers and the principles of defensive programming in order to better control the execution of feedback control loops.

Acknowledgments. This work is partially supported by the Datalyse project `www.datalyse.fr`.

[6]The complete XFCDL code is available from the companion website`http://fikovnik.github.io/Actress/ICAC14.html`

[7]`http://akka.io`
[8]MacBook Pro 2.53 Ghz, 8GB RAM, Java 1.7_17 64bit, Akka 2.2
[9]`http://bit.ly/1gHM975`

References

[1] T. Abdelzaher. Modeling and performance control of Internet servers. In *39th IEEE Conference on Decision and Control*, volume 3, IEEE, 2000.

[2] T. Abdelzaher and N. Bhatti. Web server QoS management by adaptive content delivery. In *7th International Workshop on Quality of Service*, IWQoS, London, 1999. IEEE.

[3] T. Abdelzaher, K. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: a control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[4] T. Abdelzaher and J. Stankovic. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):1014–1027, Sept. 2006.

[5] K. Angelopoulos, V. E. Silva Souza, and J. Pimentel. Requirements and architectural approaches to adaptive software systems: A comparative study. In *8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, IEEE, May 2013.

[6] R. Asadollahi, M. Salehie, and L. Tahvildari. StarMX: A framework for developing self-managing Java-based systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Ieee, May 2009.

[7] K. J. Astöm and B. Wittenmark. Adaptive Control, 1995.

[8] 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*, 2009.

[9] D. Cassou, E. Balland, C. Consel, and J. Lawall. Leveraging software architectures to guide and verify the development of sense/compute/control applications. In *33rd International Conference on Software Engineering*, ICSE, 2011.

[10] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, J. Whittle, R. Lemos, and Others. Software Engineering for Self-Adaptive Systems: A Research Roadmap. *Software Engineering for Self-Adaptive Systems*, 2009.

[11] S.-w. Cheng. *Rainbow: Cost-Effective Software*. PhD thesis, Carnegie Mellon University, 2008.

[12] S.-W. Cheng, D. Garlan, and B. Schmerl. Evaluating the effectiveness of the Rainbow self-adaptive system. In *4th ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, IEEE, May 2009.

[13] J. Eker, J. Janneck, E. Lee, J. Ludvig, S. Neuendorffer, and S. Sachs. Taming heterogeneity - the Ptolemy approach. *IEEE*, 2003.

[14] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework (2nd Edition). Addison-Wesley Professional (2008)

[15] R. B. France and B. Rumpe. Model-driven Development of Complex Software: A Research Roadmap. In *Future of Software Engineering*, FOSE, 2007.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[17] N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache Web server. In *EEE/IFIP Network Operations and Management Symposium.*, pages 219–234. IEEE, 2002.

[18] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[19] P. Haller and M. Odersky. Scala Actors: Unifying thread-based and event-based programming. *Theoretical Computer Science*, 410(2-3):202–220, Feb. 2009.

[20] T. He, J. a. Stankovic, M. Marley, C. Lu, Y. Lu, T. Abdelzaher, S. Son, and G. Tao. Feedback control-based dynamic resource management in distributed real-time systems. *Journal of Systems and Software*, 2007.

[21] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury. *Feedback control of computing systems*. Wiley Online Library, 2004.

[22] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8(3):323–364, June 1977.

[23] G. J. Holzmann. *Spin Model Checker*. Addison-Wesley Professional, 1. edition edition, 2003.

[24] IBM. An Architectural Blueprint for Autonomic Computing, 4. edition. Technical report, IBM, 2006.

[25] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE, 2008.

[26] F. Krikava. *Domain-Specific Modeling Language for Self-Adaptive Software System Architectures*. PhD thesis, University of Nice Sophia-Antipolis, 2013.

[27] F. Krikava, P. Collet, and R. France. ACTRESS: Domain-Specific Modeling of Self-Adaptive Software Architectures. In *Symposium on Applied Computing (SAC), track on Dependable and Adaptive Distributed Systems (DADS)*, 2014.

[28] E. A. Lee. The Problem with Threads. *Computer*, 2006.

[29] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao. An adaptive control framework for QoS guarantees and its application to differentiated caching. In *10th International Workshop on Quality of Service*, IWQoS, IEEE, 2002.

[30] M. Luckey, B. Nagel, C. Gerth, and G. Engels. Adapt cases. In *6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, page 30, New York, New York, USA, 2011. ACM Press.

[31] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Decision making in autonomic computing systems. In *8th ACM international conference on Autonomic computing*, ICAC, 2011.

[32] H. Müller, M. Pezzè, and M. Shaw. Visibility of control in adaptive systems. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*, ULSSIS, 2008.

[33] A. J. Ramirez and B. H. C. Cheng. Design patterns for developing dynamically adaptive systems. In *2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, 2010.

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

[35] D. Niz, G. Bhatia and R. Rajkumar. Model-Based Development of Embedded Systems: The SysWeaver Approach. *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[36] J. Kephart and D. Chess. The Vision of Autonomic Computing. *Computer 36(1)*, 2003.

[37] A. Filieri, H. Hoffmann and M. Maggio. Automated Design of Self-Adaptive Software with Control-Theoretical Formal Guarantees. *Proc. 36th International Conference on Software Engineering*, 2014.

[38] G. Tamura, N. M. Villegas, H. A. Muller, L. Duchien, and L. Seinturier. Improving context-awareness in self-adaptation using the DYNAMICO reference model. In *8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS, 2013.

[39] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, volume 1 of *SEAM*, page 80, New York, New York, USA, 2011. ACM Press.

[40] T. Vogel and H. Giese. A Language for Feedback Loops in Self-Adaptive Systems: Executable Runtime Megamodels. In *7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, number 3 in SEAMS, IEEE, June 2012.

[41] K. Wu, D. J. Lilja, and H. Bai. The Applicability of Adaptive Control Theory to QoS Design: Limitations and Solutions. In *19th International Parallel and Distributed Processing Symposium*, IPDPS, pages 272b–272b. IEEE, 2005.

[42] Y. Zhao. A Model of Computation with Push and Pull Processing. Technical report, Technical Memorandum UCB/ERL M03/51, University of California, Berkeley, 2003.