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

Filip Kříkava
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 SYSEWEVER [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 community1, 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

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](image)

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](image)

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\(^\ast\) 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 \((\text{self}; \triangledown \langle \text{input} \rangle; \triangledown \langle \text{output} \rangle)\). 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 pre-processed 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 \(R = \sum r\) and delivered bandwidth \(W = \sum w\), where \(a\) and \(b\) are platform constants 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 \(G = G + K_i E = G + K_i (U^\ast - U)\) where \(K_i\) is the controller integral gain, \(U^\ast\) 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, Controller, 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 (\(U^\ast\)) are represented as the controller properties. The **monitoring** part periodically computes server utilization \(U\). Both the \(R\)

\(^3\)cf. Abdelzaher et al. [2,3]
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 `FileTailer` 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.

```java
Listing 1: xFCDL code of PeriodicTrigger AE
```

**4.2 Distributed Adaptation**

Next, we extend the adaptation to cover distributed \( Zm.com \) 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: \( \text{content_tree}(s) = \max(\text{content_tree}(S)) \).

**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.
4.3 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.

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 AdaptiveController 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 properties. 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].
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 Zm.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.

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 websitehttp://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


