Feedbacks Control Loops as 1st Class Entities

The SALTY Experiment

Filip Křikava, Romain Rouvoy, Lionel Seinturier
University of Lille - CRIStAL
Inria Lille
France

Philippe Collet
Université Nice Sophia Antipolis I3S - CNRS UMR 7271
France

Robert B. France
Colorado State University
Computer Science Department
USA
ABOUT MYSELF

• Middleware platforms (OpenCCM, FraSCAti)
  • Component-based Software Engineering
  • Aspect-Oriented Programming
  • Model-Driven Engineering
  • Domain-Specific Languages

• Distributed systems
  • Mobile/Ubiquitous computing
  • Wireless Sensor Networks
  • Cloud computing
OUR CONTRIBUTIONS SO FAR

Reflective Model
Pluggable Toolchain
Technology Mappings

Feedback control loop

System-centric  Application-centric  Control-centric
Ubiquitous computing  Legacy apps  Big data
COSMOS  MUSIC  Cappucino  SALTY  Datalyse

2007  2013
Feedback is a primary mean to enable self-adaptation.
CHALLENGES

- Engineering self-adaptive software systems is challenging
- Example: web server self-optimization

FCL control challenges

- Prepare experimental environment
  - identify system outputs (sensors)
  - identify system control inputs (effectors)
- Design decision mechanism
  - data collection
  - model construction and evaluation
  - controller design
- Implementation
  - integration into target system
  - consistent monitoring
  - coordinated reconfiguration

FCL integration challenges

- Prepare experimental environment
- Design decision mechanism
- Implementation

MEM

\[ MEM_{k+1} = 0.485 \cdot MEM_k + 3.63 \times 10^{-4} \cdot MC_k \]

\[ u_k = K_p e_k + K_i \sum_{j=1}^{k-1} e_j \]

[Hellerstain et al., 2004]
CHALLENGES

- Engineering self-adaptive software systems is challenging
- Example: web server self-optimization

FCL control challenges

- Prepare experimental environment
  - identify system outputs (sensors)
  - identify system control inputs (effectors)

- Design decision mechanism
  - data collection
  - model construction and evaluation
  - controller design

- Implementation
  - integration into target system
  - consistent monitoring
  - coordinated reconfiguration

FCL integration challenges

control engineers

[Image of an integrated software system with various components and interfaces]

U = K_P e_k + K_I \sum_{j=1}^{k-1} e_j

MEM_{k+1} = 0.485 \cdot MEM_k + 3.63 \times 10^{-4} \cdot MC_k
INTEGRATION CHALLENGES

- Forming the **connection** between **an adaptation mechanism** and a **target system**

| • Web service content adaptation [Abedzaher et al. ’99, ’02, ’06] |
| • Control theoretical approach |
| • **Matlab / C implementation directly in Apace code base** |

| • Self-healing in workflow execution on grids [Silva et al. ’13] |
| • Tuned analytical model |
| • **Java implementation directly in a workflow enacting engine** |

| • Scaling Hadoop Clusters [Berekmeri et al.’14] |
| • Control theoretical approach |
| • **Matlab / Bash** |

- Extensive handcrafting of a **non-trivial implementation** code
- **Low-level abstraction** - limited verification, reuse, maintainability
- Giving rise to **accidental complexities**
**OVERVIEW**

**Systematic integration** of self-adaptive mechanisms into software systems through architecture models and model-driven engineering techniques.
Derived requirements for integrating self-adaptation into software systems.

**REQUIREMENTS**

- **Generality**
  - Domain-agnostic
  - Technology-agnostic

- **Visibility**
  - Explicit FCLs, their process and interactions
  - Verification support

- **Reusability**
  - Reusable FCL parts across adaptation scenarios

- **Distribution**
  - Remote distribution of FCL

- **Complex control**
  - Composition
  - Reflection

- **Tooling**
  - Prototyping
  - Automating

[Babaoglu et al.'05, Salehie et al.'09, Cheng et al.'09, Brun et al.'09, Muller et al'09]
1 Feedback Control Definition Language
1. Raise the level of abstraction
2. Fine-grained decomposition of FCL elements
3. Explicit interactions
4. Provide reflection capabilities
5. Embed remoting

**Feedback Control Loop**
- Sequence of interconnected processes
- Input $\times$ State $\rightarrow$ Output
- Reactive
- Concurrent
- Dynamic

**The Actor Model**
- Message passing actor networks
- Message $\times$ State $\rightarrow$ Message(s)
- Reactive
- Concurrent
- Dynamic
- Scalable
- Remoting through location transparency

**Domain-Specific Modeling**
- Abstraction
- Automation
- Analysis
Apache adaptation example - adjusts the maximum number of connections to be processed simultaneously (MC) based on the difference between reference (MEM*) and measured memory usage (MEM) [Hellerstain et al.’04].
FEEDBACK CONTROL DEFINITION LANGUAGE - IN A NUTSHELL

Graphical Syntax

Controller

Apache Web Server

memUsage : ProcessMem

scheduler : PeriodTrigger

controller : IController

mcConf : SetApacheConf

(target system)

Graphical Syntax
FEEDBACK CONTROL DEFINITION LANGUAGE - IN A NUTSHELL

Adaptive Element

- Actor-like component
- Sensors
- Effectors
- Processors
- Controllers
- Input/output ports & properties
- Active / passive
- Implementation
  - Imperative code (e.g. Java)
  - CEP Rules (e.g. Drools)
  - STM (e.g. bzr/heptagon)
  - Matlab
  - BASH
  - ...

MEM

scheduler: PeriodTrigger
- initialPeriod

MEM

memUsage: ProcessMem
- pid

MEM

controller: IController
- refMem
- kI

MC

mcConf: SetApacheConf
- httpdConfPath

(target system)
Adaptive Element

- Actor-like component
- Sensors
- Effectors
- Processors
- Controllers
- Input/output ports & properties
- Active / passive
- Implementation
  - Imperative code (e.g. Java)
  - CEP Rules (e.g. Drools)
  - STM (e.g. bzr/heptagon)
  - Matlab
  - BASH
  - ...

\[ u(t) = u(k - 1) + K_i e(k) \]
FEEDBACK CONTROL DEFINITION LANGUAGE - IN A NUTSHELL

Adaptive Element

- Actor-like component
- Sensors
- Effectors
- Processors
- Controllers
- Input/output ports & properties
- Active / passive
- Implementation
  - Imperative code (e.g. Java)
  - CEP Rules (e.g. Drools)
  - STM (e.g. bzr/heptagon)
  - Matlab
  - BASH
  - ...
- Interaction contracts

\[ \alpha = \langle self; \downarrow \text{input}; \uparrow \text{output} \rangle \]
FEEDBACK CONTROL DEFINITION LANGUAGE - IN A NUTSHELL

Adaptive Element

- Actor-like component
  - Sensors
  - Effectors
  - Processors
  - Controllers
- Input/output ports & properties
- Active / passive
- Implementation
  - Imperative code (e.g. Java)
  - CEP Rules (e.g. Drools)
  - STM (e.g. bzr/heptagon)
  - Matlab
  - BASH
  - ...
- Interaction contracts

\[ \alpha = \langle \text{self}; \downarrow \text{(input)}; \uparrow \text{(output?)} \rangle \ || \ \langle \uparrow \text{(setPeriod)}; \emptyset; \uparrow \text{(period)} \rangle \]
2

ILLUSTRATION OF WEB SERVER QoS ADAPTATION IMPLEMENTATION
LOCAL CONTENT DELIVERY ADAPTATION

QoS management control of web servers by content delivery adaptation

Goal: maintain server load around some pre-set value

Idea: service time = fixed overhead + data-size dependent overhead

Prerequisite: preprocessed content (different quality and size)

\[ U = aR + bW \]
\[ G = G + k(U^* - U) \]

Using FCDL

Generality | Visibility | Reusability
Distribution | Complex control

Abdelzaher et al., 1999, 2002
1. Compute the number of requests ($r$) and size of responses ($w$)
Compute the number of requests ($r$) and size of responses ($w$)
2 Compute the requests rate ($R$), bandwidth ($W$) and utilization ($U$)

$$U = aR + bW = \frac{\sum_j r_j}{t} + \frac{b \sum_i w_i}{t}$$
LOCAL CONTENT DELIVERY ADAPTATION

2 Compute the requests rate ($R$), bandwidth ($W$) and utilization ($U$)

$$U = aR + bW = \frac{\sum_j r_j}{t} + b \frac{\sum_i w_i}{t}$$

![Diagram showing the computation of requests rate, bandwidth, and utilization with components like accessLog, requestCounter, responseSizeCounter, loadMonitor, and scheduler. The diagram also includes active processor code for a period trigger.]
Compute severity of adaptation ($G$)

$$G = G + k(U^* - U)$$

**Controller**

```
controller IController {
    in push port input: double
    out push port output: double

    property KI: double
    property reference: double
    property loBnd: double
    property upBnd: double
}
```
composite ApacheQOS {
  feature accessLog = new FileTailer {
    file = "/var/log/apache2/access_log"
  }
  feature accessLogParser = new AccessLogParser
  feature requestCounter = new Accumulator
  feature responseSizeCounter = new Accumulator
  feature loadMonitor = new LoadMonitor
  feature scheduler = new PeriodTrigger<Double>
  feature utilController = new IController {
    reference = 0.8
  }
  feature adaptor = new ContentAdaptor
  connect accessLog.lines to accessLogParser.lines
  connect accessLogParser.size to responseSizeCounter.input
  connect accessLogParser.requests to requestCounter.input
  connect requestCounter.output to loadMonitor.requests
  connect responseSizeCounter.output to loadMonitor.size
  connect loadMonitor.utilization to scheduler.input
  connect scheduler.output to utilController.utilization
  connect utilController.contentTree to adaptor.contentTree
}
LOCAL CONTENT DELIVERY ADAPTATION - COMPOSITION

QOSControl

utilController : IController

utilization : UtilizationMonitor

scheduler : PeriodTrigger

ApacheQOS

control : QOSControl

system layer

ApacheWebServer

accessLogParser : AccessLogParser

adaptor : ContentAdaptor

ApacheWebServer

accessLog : FileTailer

data=var/log/apache2/access_log
ILLUSTRATION - SYSTEM IDENTIFICATION

- Support for FCL design - black-box modelling
- **Open control loops** for data collection
ILLUSTRATION - ADAPTIVE CONTROL

- Using the reflection support for adaptive control

\[ G = G + k(U^* - U) \]
3

The ACTRESS Modeling Environment
IMPLEMENTATION

• Reference implementation of FCDL based on Eclipse Modeling Framework
• Eclipse IDE-based prototype to facilitate the use of FCDL - ACTRESS

- Reference implementation of FCDL based on Eclipse Modeling Framework
- Eclipse IDE-based prototype to facilitate the use of FCDL - ACTRESS

- Textual DSL for authoring FCDL models
- Modularity, Java interoperability, Xbase
- Eclipse IDE support

- Generates Actors from FCDL Adaptive Elements
- ACTRESS runtime based on Akka
- Maintain traceability
ACTRESS - VERIFICATION SUPPORT

- Model well-formedness through meta-model constraints
  - Data-types, port connections, required properties, ...

```ocl
@OCL(invDifferentSource="self.ports
->select(p | p.name = 'size' || p.name = 'requests') // select ports
->collect(p | p.connections) // select their connects
->collect(p | p.parent) // select owning instances
->asSet()->size() == 2 // there must be two different ones)
```

processor LoadMonitor {

- User-defined structural constraints, e.g., XFCDL OCL annotations

- User temporal constraints
  - Connectivity, reachability
  - FCDL to PROMELA transformation verified by SPIN model checker
Conclusions
SUMMARY

• Combining self-adaptive software systems with principles of MDE to provide **systematic** and **tooled approach** for **integrating adaptation mechanisms** into software systems

• Address ACTRESS limitations - MPS-based implementation
• Improvements in FCDL (e.g. data units, IO assertions, modeling assumptions)
• A library of reusable Adaptive Elements
• Executable models using Ptolemy 2
• Integration with Matlab/Simulink/Modelica
• Explore **models@run.time** for a systematic implementation of touchpoints
Thank you

Romain Rouvoy
romain.rouvoy@inria.fr