A Language for Feedback Loops in Self-Adaptive Systems: Executable Runtime Megamodels

Zurich, Switzerland, June 4-5, 2012

Thomas Vogel and Holger Giese
System Analysis and Modeling Group
Hasso Plattner Institute
University of Potsdam, Germany
Engineering Self-Adaptive Software

- **Internal vs. external** approach
  [Salehie and Tahvildari, 2009]
- **Feedback Loop (MAPE-K)**
  [Kephart and Chess, 2003]
- **Multiple, flexible** feedback loops
  - Different concerns
    [Vogel et al., 2010a, Vogel and Giese, 2010]
  - Hierarchical structures
    [Hestermeyer et al., 2004, Kramer and Magee, 2007]
  - Uncertainty [Esfahani and Malek, 2012]
- **Models@run.time** for K and MAPE
Interplay of Runtime Models?

if self.name = 'TShop'
then self.components.size() <= 1
else true
endif

name = InvalidTX
f1:
name = IWarehousing
i2: Interface
Failure
name = InvalidTX
f3:
failures
Failure
name = InvalidTX
f2: failures Failure
failures
Specifying and Executing Feedback Loops

Specification — Modeling language

- Capturing the interplay of multiple runtime models
  [Vogel et al., 2010b, Vogel et al., 2011]
- Making feedback loops explicit in the design of self-adaptive systems
  [Müller et al., 2008, Brun et al., 2009]

Execution — Model interpreter

- Coordinated execution/usage of multiple runtime models
- Flexible solutions and structures for feedback loops
  ~ Adaptable feedback loops
Specifying and Executing Feedback Loops

**Specification — Modeling language**
- Capturing the interplay of multiple runtime models [Vogel et al., 2010b, Vogel et al., 2011]
- Making feedback loops **explicit** in the design of self-adaptive systems [Müller et al., 2008, Brun et al., 2009]

**Execution — Model interpreter**
- Coordinated execution/usage of multiple runtime models
- **Flexible** solutions and structures for feedback loops
  \[\Rightarrow\] Adaptable feedback loops

**Executable Runtime Megamodels**
A *megamodel* is a model that contains models and relations by means of model operations between those models.

In general:

![Diagram showing model operation between models and models']

**Model-Driven Architecture (MDA) example:**

- Research on model-driven software development (MDA, MDE) [Favre, 2005, Bézivin et al., 2003, Bézivin et al., 2004, Barbero et al., 2007]
- “Toward Megamodels at Runtime” [Vogel et al., 2010b]
An Example: Self-repair
An Example: Self-repair

Start

<<Monitor>>
Update
updated model

<<Analyze>>
Check for failures
detailed results

no failures

Analyzed

<<Analyze>>
Deep check for failures

<<Plan>>
Repair

<<Execute>>
Effect

repaired
done

done

Legend
(concrete syntax)

• Initial state

○ Final state

Model Operation
t1 t2
An Example: Self-repair

Legend
(concrete syntax)

- Initial state
- Final state

Control flow

Model
Operation

T. Vogel, H. Giese | Executable Runtime Megamodels | SEAMS 2012 | June 4-5, 2012
An Example: Self-repair

Failure analysis rules
<<EvaluationModel>>

Deep analysis rules
<<EvaluationModel>>

Architectural Model
<<ReflectionModel>>

TGG Rules
<<Monitor>>

Repair strategies
<<ChangeModel>>

Repair
<<Plan>>

Effect
<<Execute>>

done

Legend
(concrete syntax)

Initial state
Final state
Model
Operation
Control flow

T. Vogel, H. Giese | Executable Runtime Megamodels | SEAMS 2012 | June 4-5, 2012
An Example: Self-repair

Legend (concrete syntax)

- Initial state
- Final state
- Model
- Operation
- Control flow
- [condition]
- [else]
- Model usage
Modularity and Composition

Analysis step for self-repair

Self-repair

T. Vogel, H. Giese | Executable Runtime Megamodels | SEAMS 2012 | June 4-5, 2012
Modularity and Composition

Analysis step for self-repair

Complex model operations

Self-repair

T. Vogel, H. Giese | Executable Runtime Megamodels | SEAMS 2012 | June 4-5, 2012
Modularity and Composition

Analysis step for self-repair

Complex model operations

Self-repair
Modularity and Composition

Analysis step for self-repair

Shared runtime models

Self-repair
Modeling Interacting Feedback Loops

Self-repair

- **Update** <<Monitor>> Architectural Model
- **TGG Rules**
- **Analyzed**
- **Repair** <<Plan>> repaired
- **Effect** <<Execute>> done

Self-optimization

- **Update** <<Monitor>> ARCHITECTURAL Model
- **TGG Rules**
- **Analyzed**
- **Adjust** params
- **Effect** <<Execute>> done

Two example solutions:

1. **Linearizing Complete Feedback Loops**
2. **Linearizing Analysis and Planning Steps of Feedback Loops**

by using **complex model operations** and shared **runtime models**
(1) Linearizing Complete Feedback Loops

Self-repair

Self-optimization

Analyzed

Start

Effected

M
AP

Analyzed

Start

Effected

M
AP

M
AP

M
AP

Analyze

Self-managed

Self-managed
(1) Linearizing Complete Feedback Loops

**Self-repair**

Start

Analyzed

M

A

P

E

Effected

**Self-optimization**

Start

Analyzed

M

A

P

E

Effected

Analyze

Self-optimization.
(2) Linearizing Analysis and Planning Steps

Shared M+E

Self-repair

Analyzed

Planned

Self-optimization

Analyzed

Planned

T. Vogel, H. Giese | Executable Runtime Megamodels | SEAMS 2012 | June 4-5, 2012
(2) Linearizing Analysis and Planning Steps

**Shared M+E**

**Self-repair**

- Analyzed
- Planned

**Self-optimization**

- Analyzed
- Planned

---

**Architecture**

- Architectural Model
- <<ReflectionModel>>
- TGG Rules
- <<Execute>>
- [else]
- [c since 'Self-repair.AP.Planned' = 0]
- Updated model
- <<ExecuteModel>>
- Effect
- done
- Effected

---

T. Vogel, H. Giese | Executable Runtime Megamodels | SEAMS 2012 | June 4-5, 2012
Hierarchy of Feedback Loops

Layer 0

Running System
Hierarchy of Feedback Loops

Layer \(_0\) Running System

Layer \(_1\)

Self-repair

<<EvaluationModel>>
Failure analysis rules

<<Analyze>>
Check for failures

<<Monitor>>
Update

<<ReflectionModel>>
Architectural Model

<<MonitoringModel>>
<<ExecutionModel>>
TGG Rules

<<ChangeModel>>
Repair strategies

<<Plan>>
Repair

<<Execute>>
done

Layer \(_2\)

directly uses the megamodel of Layer \(_1\)
• no specific sensors and effectors required
• adapts the models or control flow of the Layer \(_1\) megamodel
• interpreter (flexibility)!

Causal connection
• sensors + effectors required
• implementation efforts!
Hierarchy of Feedback Loops

Layer 1

Layer 0

Running System

Causal connection

- sensors + effectors required
- implementation efforts!
Hierarchy of Feedback Loops

Layer 0

Running System

Layer 1

Start

Self-repair

Layer 2

Self-repair-strategies

Causal connection

- sensors + effectors required
- implementation efforts!
Hierarchy of Feedback Loops

Layer_2 directly uses the megamodel of Layer_1

- no specific sensors and effectors required
- adapts the models or control flow of the Layer_1 megamodel
- interpreter (flexibility)!

Causal connection

- sensors + effectors required
- implementation efforts!
Focus

- Coordinated execution of operations (adaptation steps)
- Handling input/output models for these operations

Simple approach

- A megamodel as a singleton
- Execution information
  - count and time
- Expression language for conditions
- Synchronous, single-threaded execution

Implementation

- EMF, JavaCC
Discussion: Executable Megamodels (I)

- **Explicit** specification of feedback loops by megamodels

- **Modularity**: individual adaptation steps and feedback loops
  - Composing steps to a feedback loop
  - Composing multiple feedback loops

- Abstraction level similar to software architectures
  - Reusing implementations of adaptation steps
  - Coordinated interplay and execution of such steps
Discussion: Executable Megamodels (II)

- Executable specifications kept **explicit** and **alive** at runtime
  - Runtime megamodels

- **Interpreter**: flexibility to cope with megamodel changes at runtime

- Megamodels as reflection models for feedback loops
  - Hierarchical control/structures
  - No specific sensors and effectors required

⇝ Supports the design/engineering of self-adaptive systems
⇝ Eases development/implementation efforts
Related Work

**Frameworks** [Garlan et al., 2004, Schmidt et al., 2008]
- Focus on reducing development efforts for single feedback loops
- Rather prescribe static solutions for feedback loops

**Explicit Feedback Loops**
- Abstraction level of controllers, no runtime support [Hebig et al., 2010]
- Formal modeling and analysis of design alternatives for self-adaptive systems, no runtime support [Weyns et al., 2010]

**Multiple, Interacting Feedback Loops**
- Implementation framework for distributed self-adaptive systems [Vromant et al., 2011]

**Modeling Languages**
- Flowcharts and dataflow diagrams, like *UML Activities*
Conclusion and Future Work

Conclusion

• Modeling language for feedback loops based on runtime models (Adaptation steps, single and multiple feedback loops)
• Executable megamodes kept alive at runtime
• Flexibility to dynamically change megamodes (interpreter)
• Leverages advanced solutions, like layered feedback loops

Future Work

• Elaborate the modeling language
  • Formal interface definitions for models and model operations
  • Analysis of megamodels
• Discuss restrictions on the execution semantics (concurrency)
Traceability and Provenance Issues in Global Model Management.

MDA components: Challenges and Opportunities.
In First Intl. Workshop on Metamodelling for MDA, pages 23–41.

On the Need for Megamodels.

Engineering Self-Adaptive Systems through Feedback Loops.

In Software Engineering for Self-Adaptive Systems 2, LNCS. Springer.
to appear.

In Language Engineering for Model-Driven Software Development, number 04101 in Dagstuhl Seminar Proc. IBFI.

Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure.


Structured Information Processing For Self-optimizing Mechatronic Systems.

The Vision of Autonomic Computing.

In Future of Software Engineering (FOSE 2007), pages 259–268. IEEE.

Visibility of control in adaptive systems.
OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1.

Self-adaptive software: Landscape and research challenges.

Simplifying autonomic enterprise Java Bean applications via model-driven engineering and simulation.

Adaptation and Abstract Runtime Models.

Incremental Model Synchronization for Efficient Run-Time Monitoring.
In MoDELS 2009 Workshops, volume 6002 of LNCS, pages 124–139. Springer.

Toward Megamodels at Runtime.

The Role of Models and Megamodels at Runtime.
In MoDELS 2010 Workshops, volume 6627 of LNCS, pages 224–238. Springer.

On interacting control loops in self-adaptive systems.

FORMS: a formal reference model for self-adaptation.