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

7th Intl. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012)

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

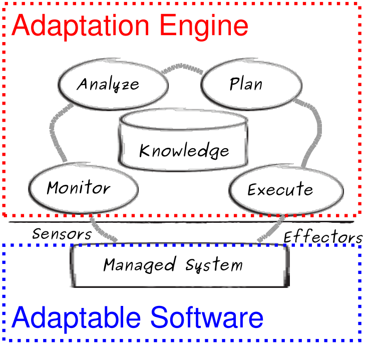
- Internal vs. **external** approach

  [Salehie and Tahvildari, 2009]
- Feedback Loop (MAPE-K)

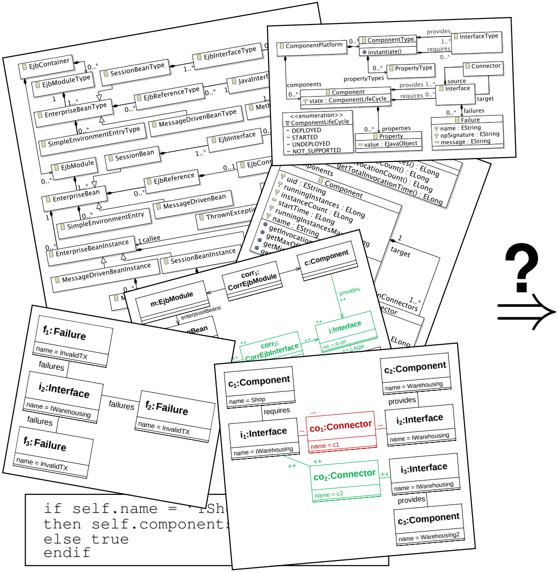
  [Kephart and Chess, 2003]
- **Multiple, flexible** feedback loops
  - Different concerns

    [Vogel et al., 2010a, Vogel and Giese, 2010]
  - Hierarchical structures

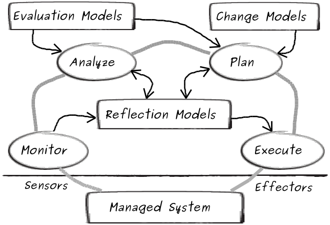
    [Hestermeyer et al., 2004,
    Kramer and Magee, 2007]
  - Uncertainty [Esfahani and Malek, 2012]
- **Models@run.time** for K and MAPE



Interplay of Runtime Models?



```
if self.name = 'iSh
then self.component
else true
endif
```



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
  - ⇝ Adaptable feedback loops

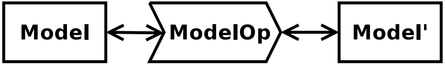
Executable Runtime Megamodels

Megamodels

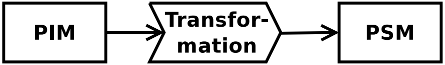
Definition (Megamodel)

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

In general:

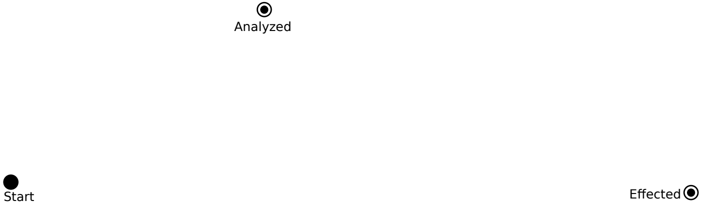


Model-Driven Architecture (MDA) example:



- 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

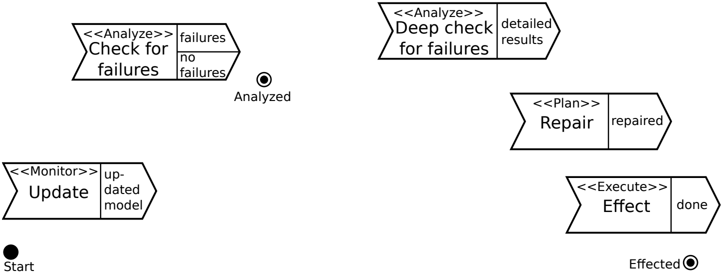


Legend
(concrete syntax)

● Initial state

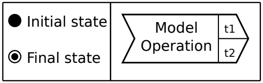
◉ Final state

An Example: Self-repair

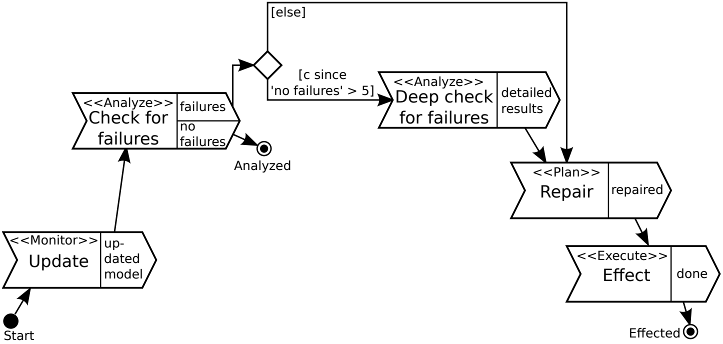


Legend

(concrete syntax)

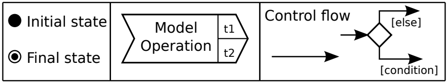


An Example: Self-repair

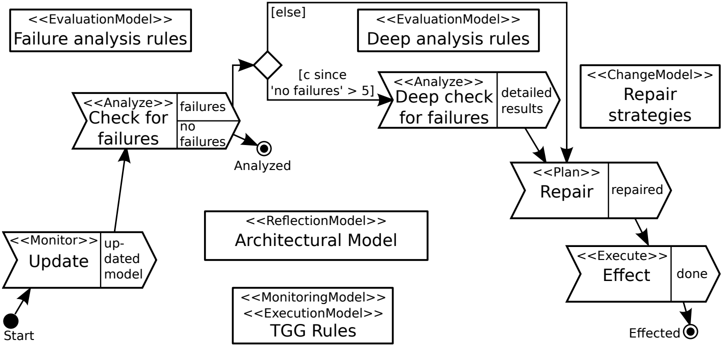


Legend

(concrete syntax)

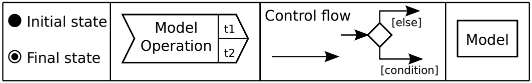


An Example: Self-repair

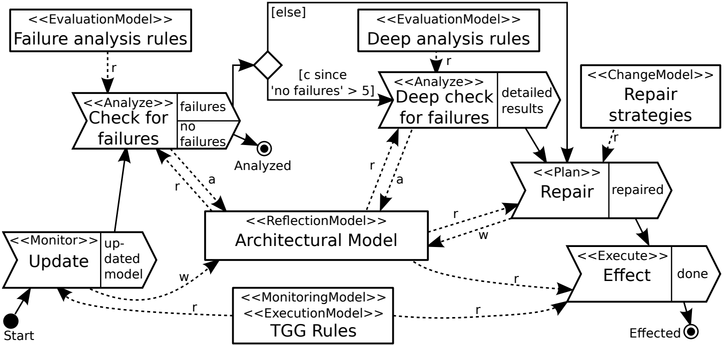


Legend

(concrete syntax)

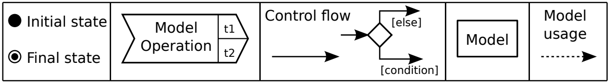


An Example: Self-repair



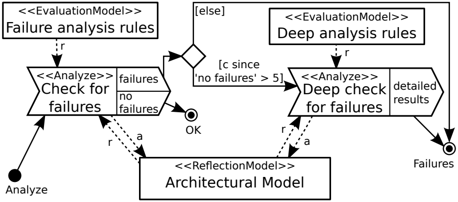
Legend

(concrete syntax)

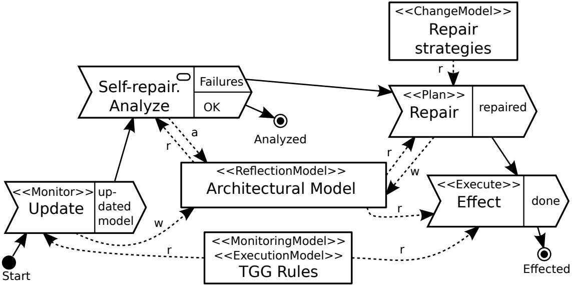


Modularity and Composition

Analysis step for self-repair



Self-repair

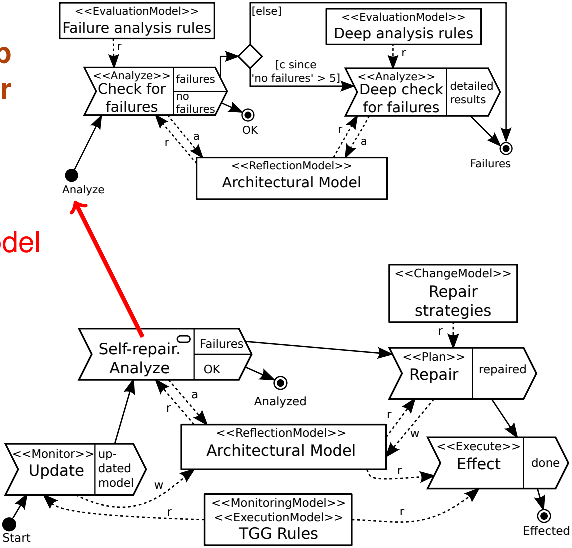


Modularity and Composition

Analysis step for self-repair

Complex model operations

Self-repair



<<EvaluationModel>>
Failure analysis rules

r

<<Analyze>>
Check for failures

failures

no failures

OK

[else]

[c since 'no failures' > 5]

<<EvaluationModel>>
Deep analysis rules

r

<<Analyze>>
Deep check for failures

detailed results

Failures

Analyze

r

a

r

a

<<ReflectionModel>>
Architectural Model

<<ChangeModel>>
Repair strategies

r

Self-repair. Analyze

Failures

OK

Analyzed

<<Plan>>
Repair

repaired

a

r

r

w

<<ReflectionModel>>
Architectural Model

<<Monitor>>
Update

up-dated model

<<Execute>>
Effect

done

w

r

r

r

Start

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

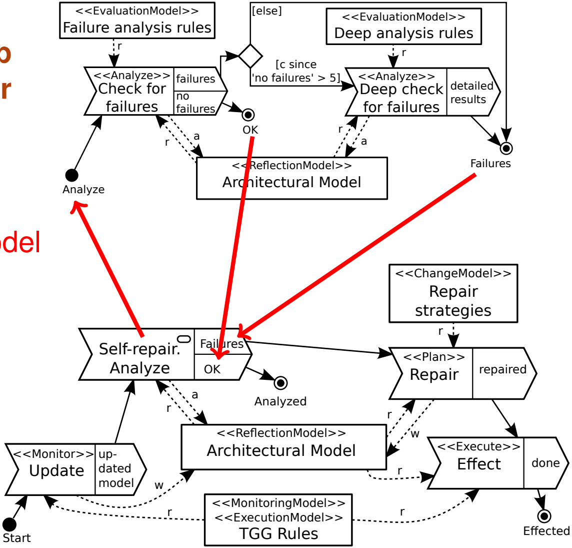
Effected

Modularity and Composition

Analysis step for self-repair

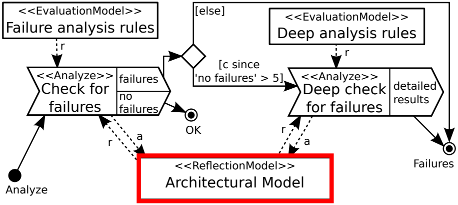
Complex model operations

Self-repair



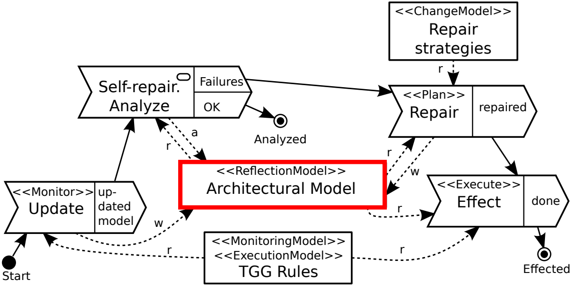
Modularity and Composition

Analysis step for self-repair



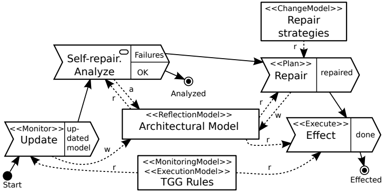
Shared runtime models

Self-repair

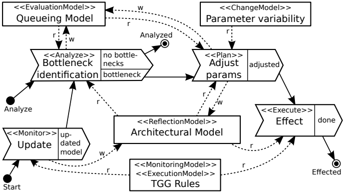


Modeling Interacting Feedback Loops

Self-repair



Self-optimization



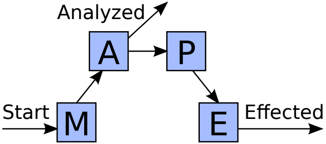
Two example solutions:

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

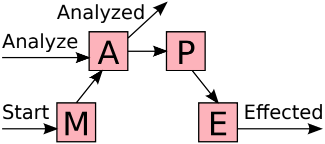
by using **complex model operations** and shared **runtime models**

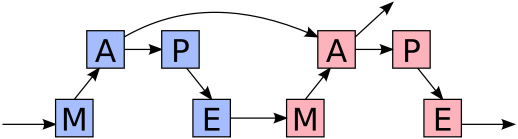
(1) Linearizing Complete Feedback Loops

Self-repair



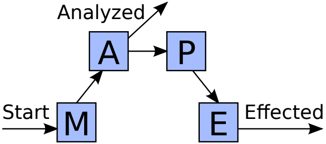
Self-optimization



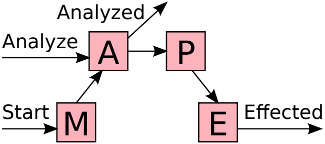


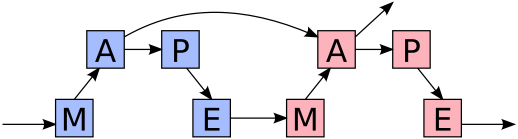
(1) Linearizing Complete Feedback Loops

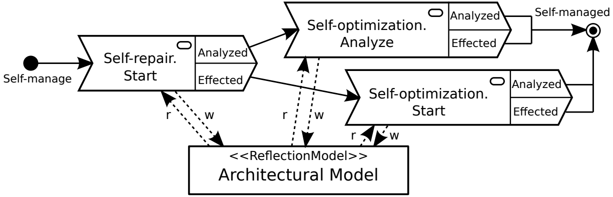
Self-repair



Self-optimization

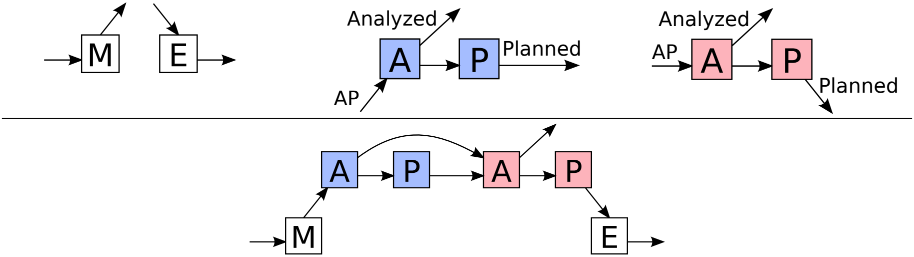






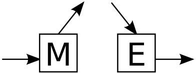
(2) Linearizing Analysis and Planning Steps

Shared M+E **Self-repair** **Self-optimization**

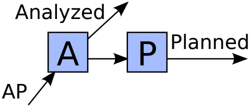


(2) Linearizing Analysis and Planning Steps

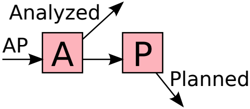
Shared M+E

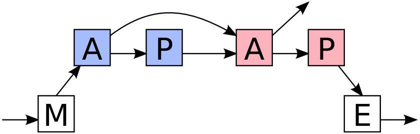


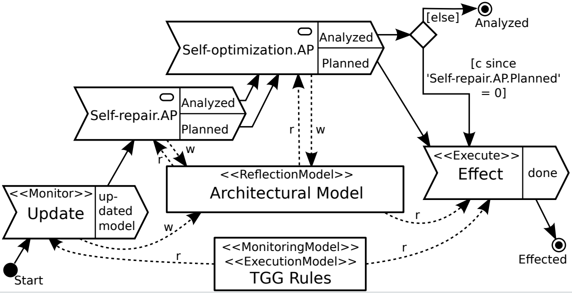
Self-repair



Self-optimization

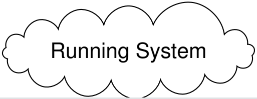






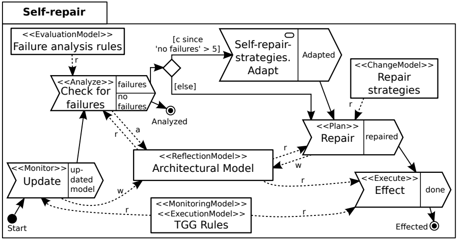
Hierarchy of Feedback Loops

Layer$_0$

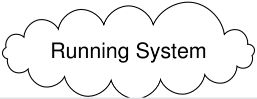


Hierarchy of Feedback Loops

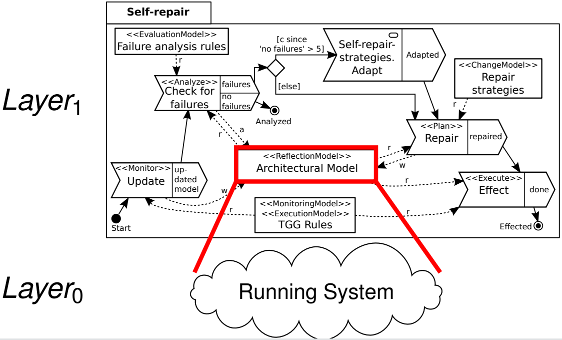
Layer$_1$



Layer$_0$



Hierarchy of Feedback Loops



Causal connection

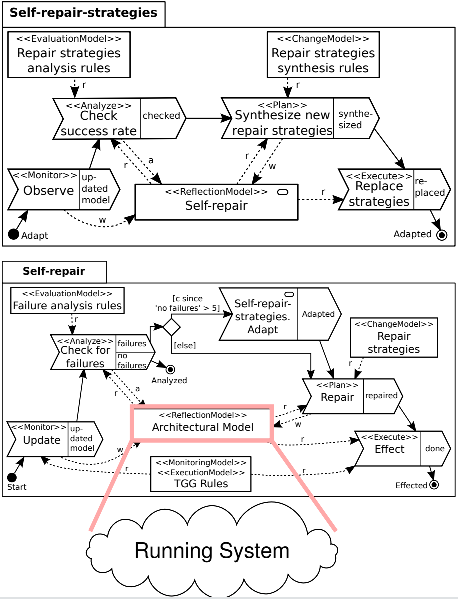
- sensors + effectors required
- implementation efforts!

Hierarchy of Feedback Loops

Layer$_2$

Layer$_1$

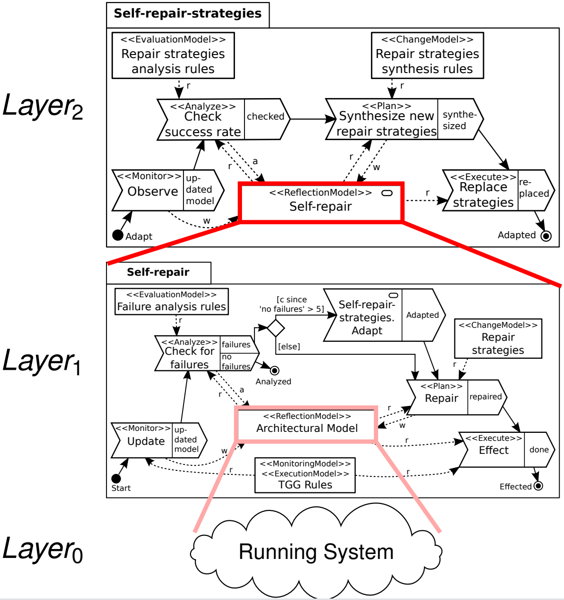
Layer$_0$


Self-repair-strategies
<<EvaluationModel>> Repair strategies analysis rules
<<ChangeModel>> Repair strategies synthesis rules
r
r
<<Analyze>> Check success rate
checked
<<Plan>> Synthesize new repair strategies
synthe-sized
a
r
r
w
<<Monitor>> Observe
up-dated model
<<ReflectionModel>> Self-repair
r
<<Execute>> Replace strategies
re-placed
w
Adapt
Adapted
Self-repair
<<EvaluationModel>> Failure analysis rules
r
[c since 'no failures' > 5]
Self-repair-strategies. Adapt
Adapted
<<ChangeModel>> Repair strategies
<<Analyze>> Check for failures
failures
no failures
[else]
Analyzed
a
r
r
<<Plan>> Repair
repaired
r
w
<<ReflectionModel>> Architectural Model
<<Monitor>> Update
up-dated model
w
r
<<Execute>> Effect
done
<<MonitoringModel>> <<ExecutionModel>> TGG Rules
r
r
Start
Effected
Running System

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!

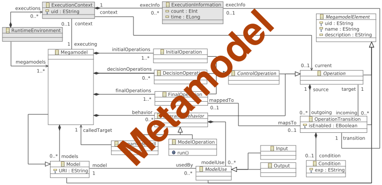
Execution Semantics and Interpreter

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

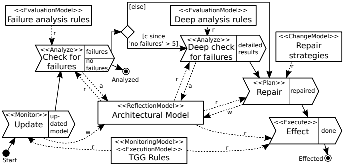

Metamodel

Implementation

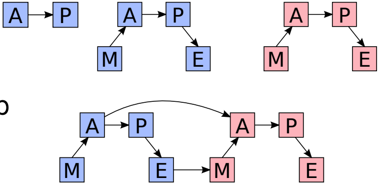
- EMF, JavaCC

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



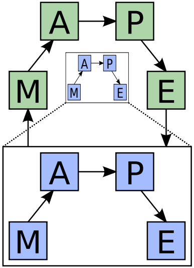
- Abstraction level similar to software architectures
  - Reusing implementations of adaptation steps
  - Coordinated interplay and execution of such steps

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 megamodels kept alive at runtime
- Flexibility to dynamically change megamodels (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)

References I

[Barbero et al., 2007] Barbero, M., Fabro, M. D., and Bézivin, J. (2007).
Traceability and Provenance Issues in Global Model Management.
In *Proc. of 3rd Workshop on Traceability (ECMDA-TW 2007)*, pages 47–55.

[Bézivin et al., 2003] Bézivin, J., Gerard, S., Muller, P.-A., and Rioux, L. (2003).
MDA components: Challenges and Opportunities.
In *First Intl. Workshop on Metamodelling for MDA*, pages 23–41.

[Bézivin et al., 2004] Bézivin, J., Jouault, F., and Valduriez, P. (2004).
On the Need for Megamodels.
In *Proc. of the Workshop on Best Practices for Model-Driven Software Development*.

[Brun et al., 2009] Brun, Y., Serugendo, G. D. M., Gacek, C., Giese, H. M., Kienle, H. M., Litoiu, M., Müller, H. A., Pezzè, M., and Shaw, M. (2009).
Engineering Self-Adaptive Systems through Feedback Loops.
In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 48–70. Springer.

[Esfahani and Malek, 2012] Esfahani, N. and Malek, S. (2012).
Uncertainty in Self-Adaptive Software Systems.
In *Software Engineering for Self-Adaptive Systems 2*, LNCS. Springer.
to appear.

[Favre, 2005] Favre, J.-M. (2005).
Foundations of Model (Driven) (Reverse) Engineering : Models – Episode I: Stories of The Fidus Papyrus and of The Solarus.
In *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proc. IBFI.

[Garlan et al., 2004] Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., and Steenkiste, P. (2004).
Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure.
Computer, 37(10):46–54.

[Hebig et al., 2010] Hebig, R., Giese, H., and Becker, B. (2010).
Making Control Loops Explicit When Architecting Self-Adaptive Systems.
In *Proc. of the 2nd Intl. Workshop on Self-Organizing Architectures (SOAR 2010)*, pages 21–28. ACM.

[Hestermeyer et al., 2004] Hestermeyer, T., Oberschelp, O., and Giese, H. (2004).
Structured Information Processing For Self-optimizing Mechatronic Systems.
In *Proc. of the 1st Intl. Conference on Informatics in Control, Automation and Robotics (ICINCO 2004)*, pages 230–237. INSTICC Press.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. (2003).
The Vision of Autonomic Computing.
Computer, 36(1):41–50.

[Kramer and Magee, 2007] Kramer, J. and Magee, J. (2007).
Self-Managed Systems: an Architectural Challenge.
In *Future of Software Engineering (FOSE 2007)*, pages 259–268. IEEE.

[Müller et al., 2008] Müller, H. A., Pezzè, M., and Shaw, M. (2008).
Visibility of control in adaptive systems.
In *Proc. of the 2nd Intl. Workshop on Ultra-large-scale Software-intensive Systems (ULSSIS 2008)*, pages 23–26. ACM.

References II

[OMG Specification, 2011] OMG Specification (2011).
OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1.

[Salehie and Tahvildari, 2009] Salehie, M. and Tahvildari, L. (2009).
Self-adaptive software: Landscape and research challenges.
ACM Trans. Auton. Adapt. Syst., 4(2):1–42.

[Schmidt et al., 2008] Schmidt, D., White, J., and Gokhale, A. (2008).
Simplifying autonomic enterprise Java Bean applications via model-driven engineering and simulation.
Software and Systems Modeling, 7(1):3–23.

[Vogel and Giese, 2010] Vogel, T. and Giese, H. (2010).
Adaptation and Abstract Runtime Models.
In *Proc. of the 5th ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2010)*, pages 39–48. ACM.

[Vogel et al., 2010a] Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., and Becker, B. (2010a).
Incremental Model Synchronization for Efficient Run-Time Monitoring.
In *MoDELS 2009 Workshops*, volume 6002 of *LNCS*, pages 124–139. Springer.

[Vogel et al., 2010b] Vogel, T., Seibel, A., and Giese, H. (2010b).
Toward Megamodels at Runtime.
In *Proc. of the 5th Intl. Workshop on Models@run.time*, volume 641 of *CEUR Workshop Proceedings*, pages 13–24. CEUR-WS.org.
(best paper).

[Vogel et al., 2011] Vogel, T., Seibel, A., and Giese, H. (2011).
The Role of Models and Megamodels at Runtime.
In *MoDELS 2010 Workshops*, volume 6627 of *LNCS*, pages 224–238. Springer.

[Vromant et al., 2011] Vromant, P., Weyns, D., Malek, S., and Andersson, J. (2011).
On interacting control loops in self-adaptive systems.
In *Proc. of the 6th Intl. Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011)*, pages 202–207. ACM.

[Weyns et al., 2010] Weyns, D., Malek, S., and Andersson, J. (2010).
FORMS: a formal reference model for self-adaptation.
In *Proc. of the 7th Intl. Conference on Autonomic Computing (ICAC 2010)*, pages 205–214. ACM.