Engineering Self-Adaptive Software Systems with Runtime Models

Seminar on QoS Attributes in Service- and Cloud-based Systems

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik
May 20-25, 2012

Thomas Vogel
System Analysis and Modeling Group
Hasso Plattner Institute
University of Potsdam, Germany

HPI

Motivation

- Need to continuously change software
  - Lehman's laws of software evolution [Lehman and Belady, 1985]
  - Software aging [Parnas, 1994]

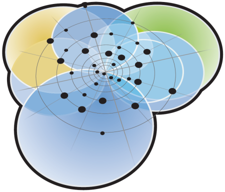
⇒ **Software evolution and maintenance**

Motivation

- Need to continuously change software
  - Lehman's laws of software evolution [Lehman and Belady, 1985]
  - Software aging [Parnas, 1994]

⇒ **Software evolution and maintenance**

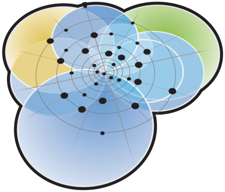
- Software systems that are...
  - self- or context-aware
  - mission-critical
  - ultra-large-scale (ULS)
  - ...



Motivation

- Need to continuously change software
  - Lehman's laws of software evolution [Lehman and Belady, 1985]
  - Software aging [Parnas, 1994]

⇒ **Software evolution and maintenance**



- Software systems that are...
  - self- or context-aware
  - mission-critical
  - ultra-large-scale (ULS)
  - ...

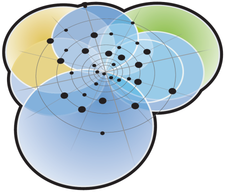
"Evolution in ULS systems will rarely occur in discrete, planned steps in a closed environment; instead it will be continuous and dynamic. The rules for continuous evolution must therefore be built into ULS systems [...] so that they will be [...] able to cope with dynamically changing environments without constant human intervention. Achieving this goal requires research on ***in situ control, reflection, and adaptation*** *to ensure continuous adherence to system functional and quality-of-service policies in the context of rapidly changing operational demands and resource availability."*
[Northrop et al., 2006, p.33]

Motivation

- Need to continuously change software
  - Lehman's laws of software evolution [Lehman and Belady, 1985]
  - Software aging [Parnas, 1994]

⇒ **Software evolution and maintenance**

- Software systems that are...
  - self- or context-aware
  - mission-critical
  - ultra-large-scale (ULS)
  - ...



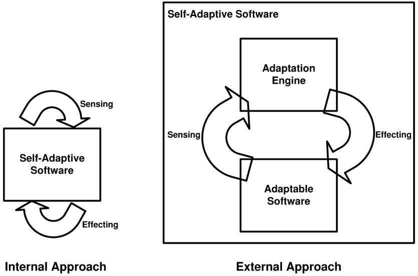
⇒ **Self-adaptive Software** [Cheng et al., 2009, de Lemos et al., 2012]

⇒ Autonomic Computing [Kephart and Chess, 2003]

Remark: Co-existence of evolution/maintenance and self-adaptation

Engineering Self-adaptive Software

(1) Cost-effective development
(2) Reflection capabilities
(3) Making feedback loops explicit
(4) Flexible (runtime) solutions



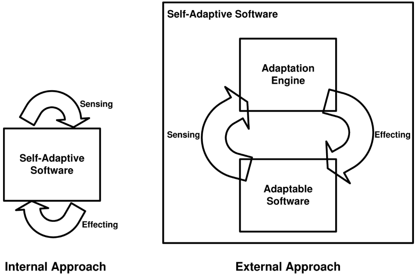
[Salehie and Tahvildari, 2009, p.14:15]

Related approaches, e.g.:

- *Rainbow* [Garlan et al., 2004] : (1), (2), (3), (4)
- *J3 Toolsuite* [Schmidt et al., 2008] : (1), (2), (3), (4)

Engineering Self-adaptive Software

(1) Cost-effective development
(2) Reflection capabilities
(3) Making feedback loops explicit
(4) Flexible (runtime) solutions



[Salehie and Tahvildari, 2009, p.14:15]

Related approaches, e.g.:

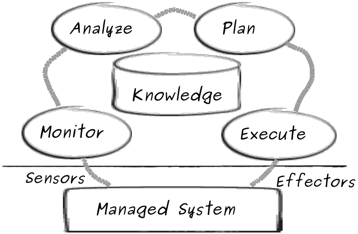
- *Rainbow* [Garlan et al., 2004] : (1), (2), (3), (4)
- *J3 Toolsuite* [Schmidt et al., 2008] : (1), (2), (3), (4)

Models at runtime for engineering adaptation engines: (1)-(4)

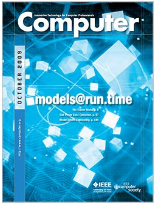
Adaptation Engine

Feedback Loop consisting of

- **Adaptation steps**
  Monitor, Analyze, Plan, Execute
- **Knowledge**
  about the managed system and its context



[Kephart and Chess, 2003]

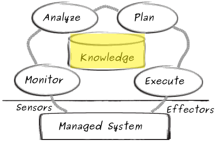


General goal: leverage MDE techniques and benefits to the runtime environment [France and Rumpe, 2007, Blair et al., 2009]

⇒ **Models@run.time for adaptation steps & knowledge**

Knowledge

Models causally connected to the running system



- Typically, **one** model is employed (often an architectural model emphasizing one concern)

  (cf. related work in [Vogel and Giese, 2010])

- Simultaneous use of multiple runtime models

→ **abstraction levels** — PSM vs. PIM (solution vs. problem space)

  - PSM: easier to connect to the running system
  - PIM: easier to use by adaptation steps

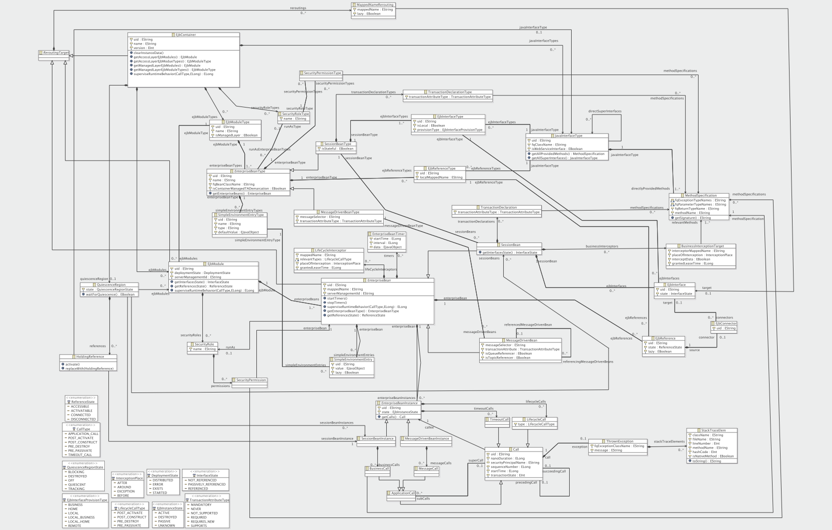
→ **concerns** — failures, performance, architectural constraints, . . .

⇒ Different views on a running system

⇒ **reflection capabilities** enabled and used by adaptation steps

Knowledge — Reflection Models

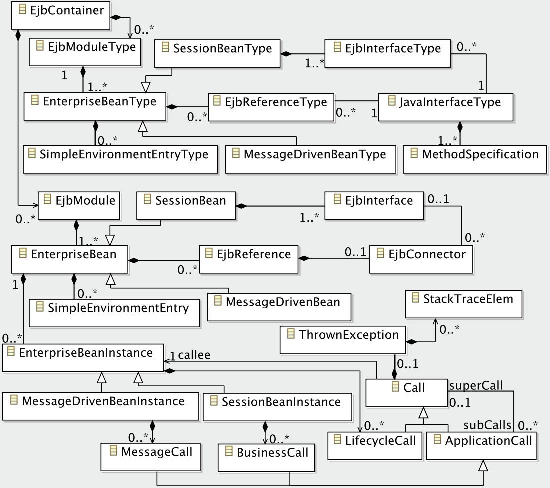
Metamodel of a PSM



Knowledge — Reflection Models

Metamodel of a PSM

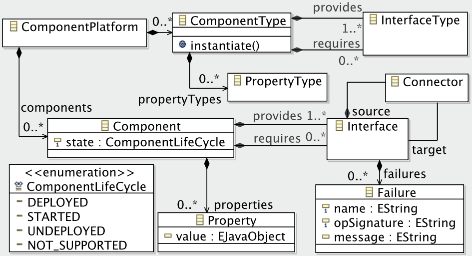
Simplified



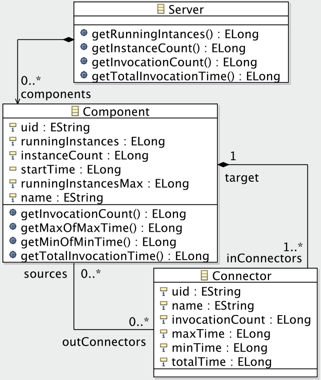
Knowledge — Reflection Models

Metamodels for PIMs

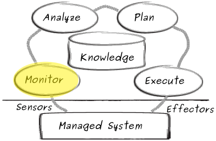
Failures



Performance



Monitor

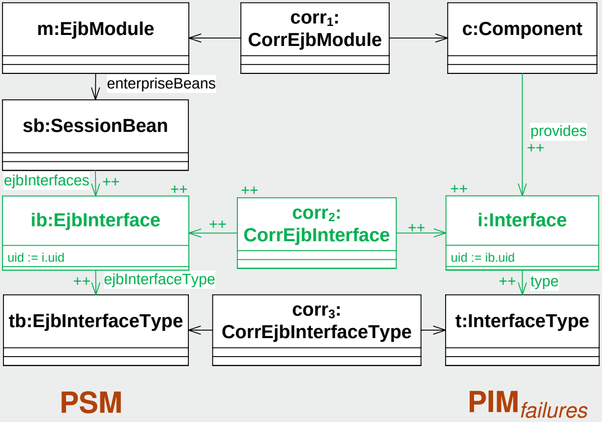


Synchronizing changes in the system to the reflection models

- Keeping runtime models up-to-date and consistent to each other
- Sensors (instrumentation): management APIs
- **Incremental**, event-driven updates: System $\rightarrow$ PSM
  (manually implemented adapter)
- **Incremental** model synchronization: PSM $\rightarrow$ PIM_1, PIM_2, ...
  (Model synchronization engine based on Triple Graph Grammars (TGG))

Monitor — TGG Rules

TGG rule for PSM → PIM$_{failures}$



- Overall, 11 rules for PSM → PIM$_{failures}$

Monitor — Development costs

generated code from TGG rules

PIMs	Proposed solution #Rules	Proposed solution #Nodes/Rules	Proposed solution LOC	Batch LOC
Simpl. Architectural Model	9	7,44	15259	357
Performance Model	4	6,25	5979	253
Failure Model	7	7,14	12133	292
Sum	20		33371	902

- **Proposed solution** — **incremental** synchronization
  - System $\rightarrow$ PSM: 2685 LOC for the reusable adapter
  - PSM $\rightarrow$ 3 PIMs: 20 TGG rules (generated >33k LOC)
- **Batch** — creates PIMs directly from scratch (**non-incremental**)
  - 902 LOC ($\approx$ 20 TGG rules)
- Declarative vs. imperative approaches

Remark: done for slightly different metamodels than shown here

Monitor — Performance

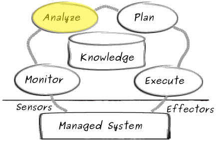
Size	Proposed Solution						Batch
	n=0	n=1	n=2	n=3	n=4	n=5	
5	0	163	361	523	749	891	8037
10	0	152	272	457	585	790	9663
15	0	157	308	472	643	848	10811
20	0	170	325	481	623	820	12257
25	0	178	339	523	708	850	15311
System $\rightarrow$ PSM	0%	92.8%	94.1%	95.6%	95.2%	96.3%	-
PSM $\rightarrow$ 3 PIMs	0%	7.2%	5.9%	4.4%	4.8%	3.7%	-

[ms]

- **Size**: number of deployed beans
- Structural monitoring through event-driven sensors
- Processing **n** events and invoking **once** the model synchronization engine

Remark: done for slightly different metamodels than shown here

Analyze

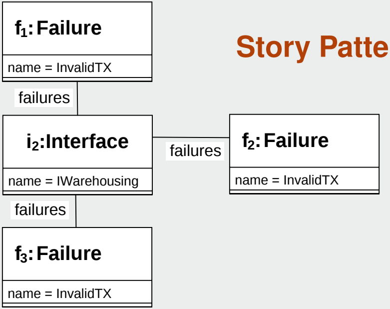


Analyzing the running system based on reflection models (PIMs)

- Identifying needs for adaptation (reactively)
- Structural checks expressed in **Story Patterns**
  (Story Pattern and Story Diagram Interpreter)
- Under certain conditions, **incremental** execution of Story Patterns
- Constraints expressed in the **Object Constraint Language (OCL)**
  (Existing engine from the Eclipse Model Development Tools)
- Model-based analysis techniques

Analyze — Evaluation Models

Identifying failures or violations of architectural constraints

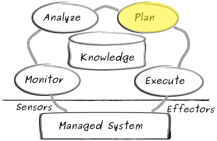


Story Pattern

OCL expression

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

Plan

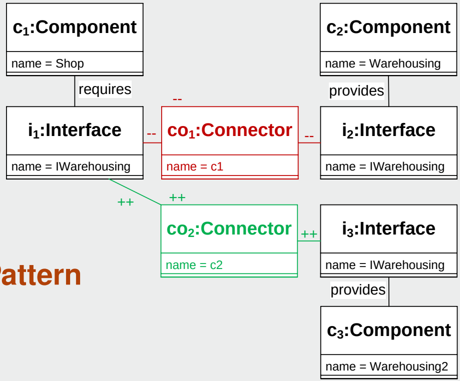


Planning adaptations based on analysis results

- Changing reflection models (PIMs) (and in the end the system)
- **Story Patterns** defining in-place transformations
  (Story Pattern and Story Diagram Interpreter)
- Under certain conditions, **incremental** execution of Story Patterns
- **OCL expression** to check and manipulate models
  (Existing engine from the Eclipse Model Development Tools)

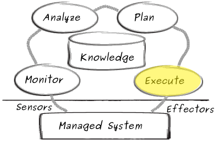
Plan — Change Models

Switching connections between components



Story Pattern

Execute

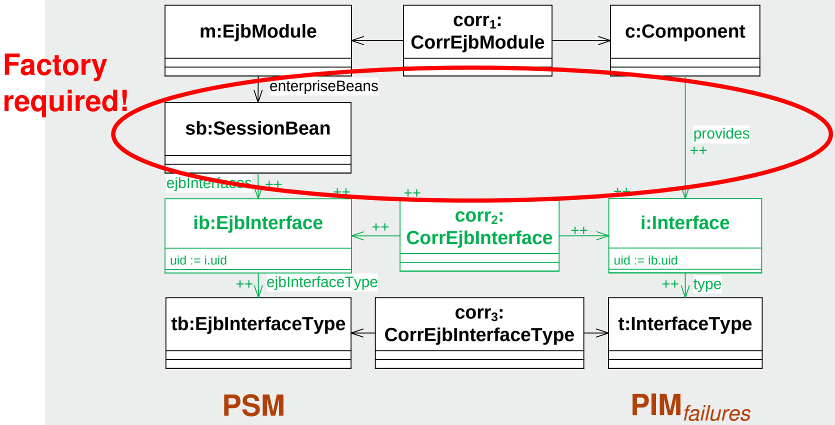


Synchronizing changes of reflection models to the system: PIMs → PSM → System

- PIM → PSM
  - **Incremental** model synchronization: same rules as for monitoring due to bidirectionality of TGG
  - Story Patterns for default creation patterns in refinement transformations (**Factories**)
- PSM → System
  - Observing PSM changes performed by the model synch. engine
  - Incrementally enacting these changes through effectors (management APIs)

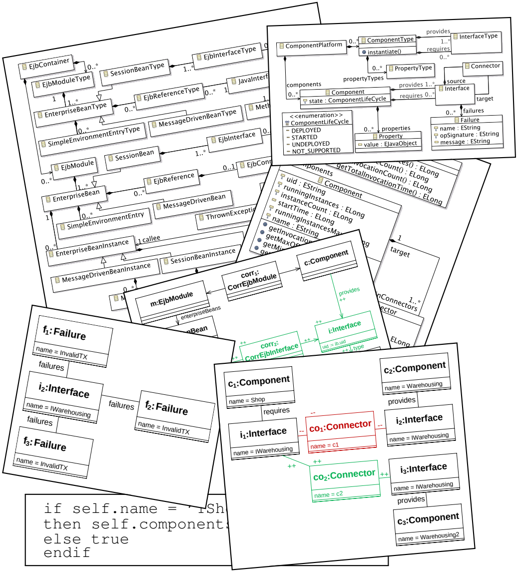
Execute — TGG Rules

TGG rule for PSM $\leftrightarrow$ PIM$_{failures}$

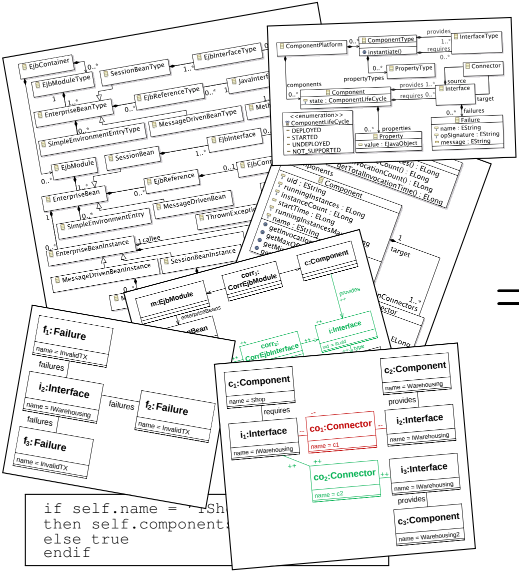


- Overall, 11 rules and 1 factory for PSM $\leftrightarrow$ PIM$_{failures}$

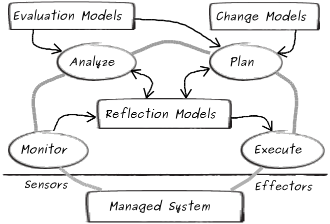
Interplay of all those models?



Interplay of all those models?



⇒



Specifying and executing feedback loops

Specification — Modeling language

- Capturing the interplay of multiple runtime models [Vogel et al., 2010, 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

- **Flexible** solutions and structures for feedback loops
  - Adaptive control [Kramer and Magee, 2007] ⇒ multiple loops
  - Uncertainty [Esfahani and Malek, 2012]
  - State-of-the-art frameworks often prescribe static solutions to single feedback loops (e.g., [Garlan et al., 2004, Schmidt et al., 2008])

Specifying and executing feedback loops

Specification — Modeling language

- Capturing the interplay of multiple runtime models
  [Vogel et al., 2010, 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

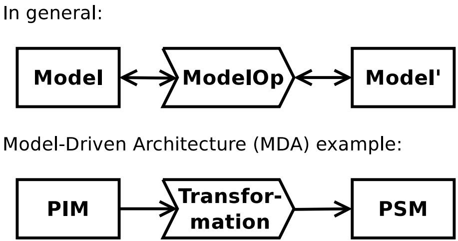
- **Flexible** solutions and structures for feedback loops
  - Adaptive control [Kramer and Magee, 2007] ⇒ multiple loops
  - Uncertainty [Esfahani and Malek, 2012]
  - State-of-the-art frameworks often prescribe static solutions to single feedback loops (e.g., [Garlan et al., 2004, Schmidt et al., 2008])

Executable Megamodels

Megamodels

Definition (Megamodel)

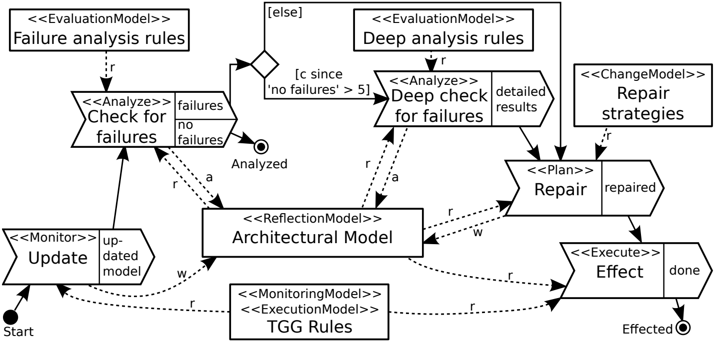
A *megamodel* is a model that contains models and relations by means of model operations between those models.



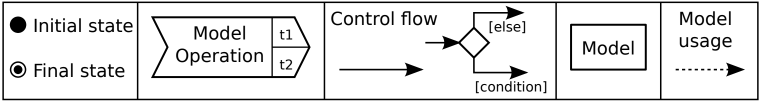
- Research on model-driven software development (MDA, MDE) [Favre, 2005, Bézivin et al., 2003, Bézivin et al., 2004, Barbero et al., 2007]
- “Toward Megamodels at Runtime” [Vogel et al., 2010]

Modeling a Single Feedback Loop

Self-repair



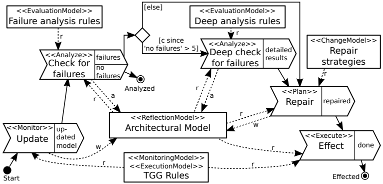
Concrete syntax:



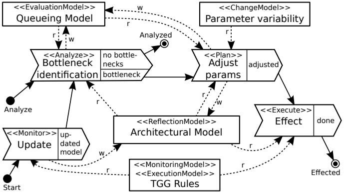
Remark: Abstract syntax defined by a metamodel [Vogel and Giese, 2012]

Modeling Interacting Feedback Loops

Self-repair

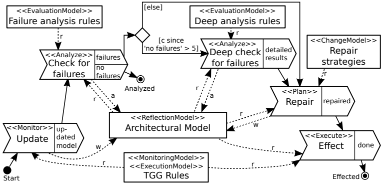


Self-optimization

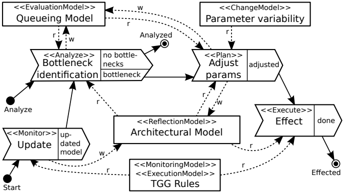


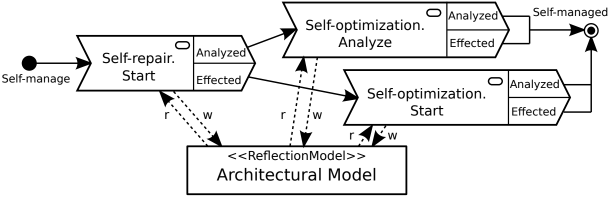
Modeling Interacting Feedback Loops

Self-repair



Self-optimization

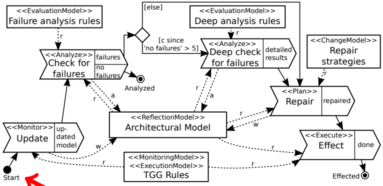




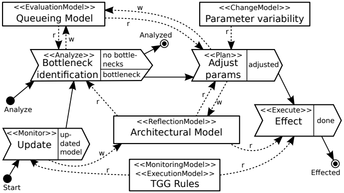
One solution: Linearizing Complete Feedback Loops

Modeling Interacting Feedback Loops

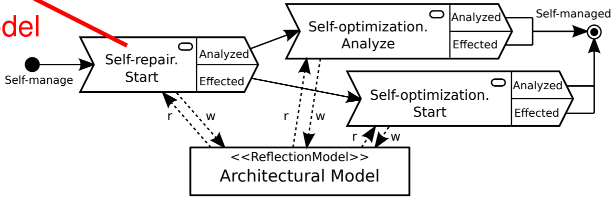
Self-repair



Self-optimization



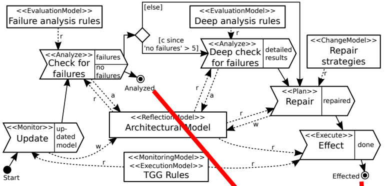
Complex model operations



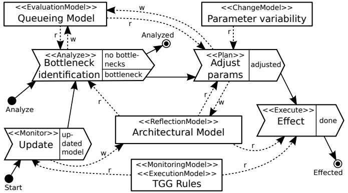
One solution: Linearizing Complete Feedback Loops

Modeling Interacting Feedback Loops

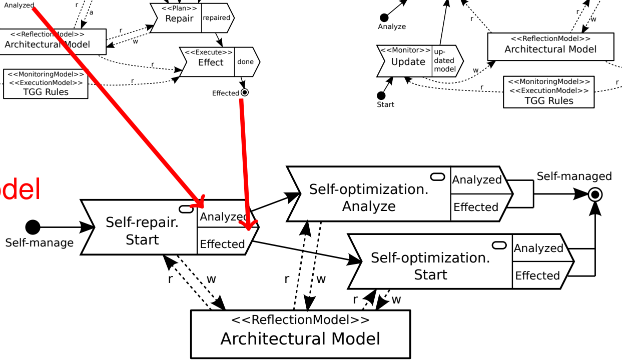
Self-repair



Self-optimization



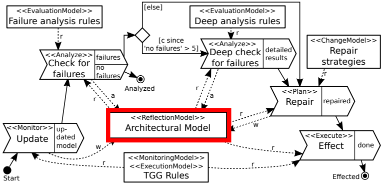
Complex model operations



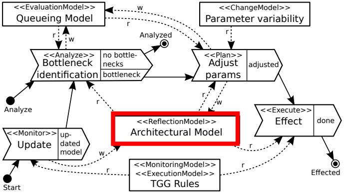
One solution: Linearizing Complete Feedback Loops

Modeling Interacting Feedback Loops

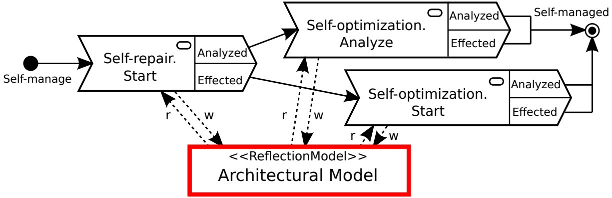
Self-repair



Self-optimization



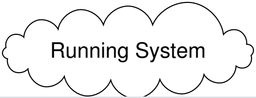
Shared runtime model



One solution: Linearizing Complete Feedback Loops

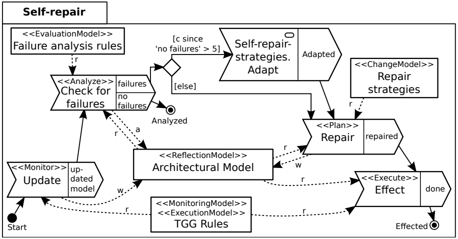
Modeling Hierarchies of Feedback Loops

Layer$_0$

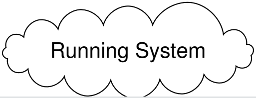


Modeling Hierarchies of Feedback Loops

Layer$_1$

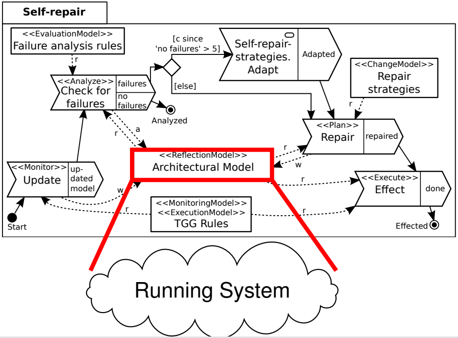


Layer$_0$



Modeling Hierarchies of Feedback Loops

Layer$_1$

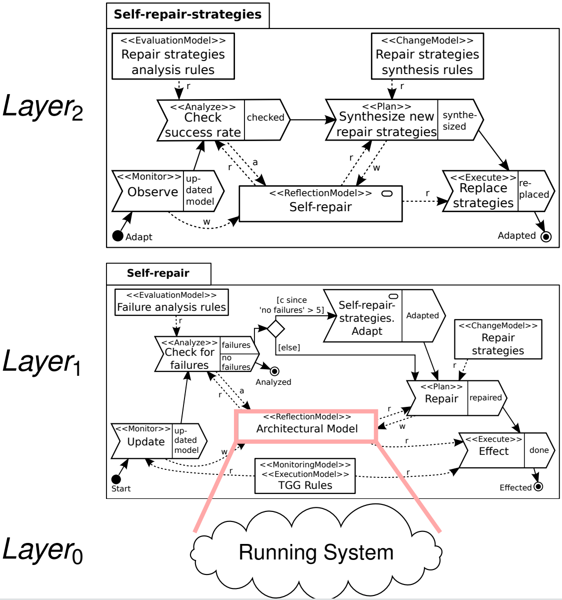


Layer$_0$

Causal connection

- sensors + effectors required
- implementation efforts!

Modeling Hierarchies of Feedback Loops



Causal connection

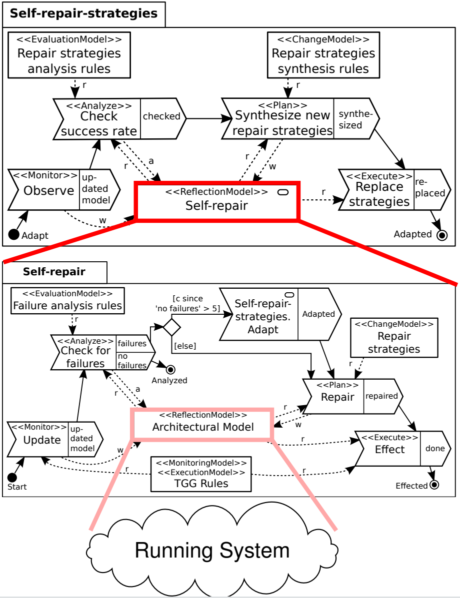
- sensors + effectors required
- implementation efforts!

Modeling Hierarchies of Feedback Loops

Layer$_2$

Layer$_1$

Layer$_0$



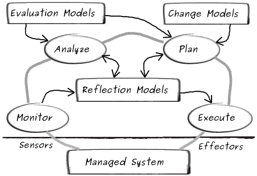
Layer$_2$ directly uses the megamodel of *Layer*$_1$

- 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!

Conclusion



Models at runtime

- Adaptation steps and knowledge
- Single and multiple feedback loops

Discussion

(1) Cost-effective development

(2) Reflection capabilities

(3) Making feedback loops explicit

(4) Flexible (runtime) solutions

... while being runtime efficient (incremental, on-line techniques)

Interests:

- Techniques, algorithms, models, and tools for QoS attributes
- Software architecture $\leftrightarrow$ multiple QoS attributes

Further Reading

- Thomas Vogel and Holger Giese. "*A Language for Feedback Loops in Self-Adaptive Systems: Executable Runtime Megamodels*". In SEAMS 2012. IEEE, 6 2012.
- Holger Giese, Leen Lambers, Basil Becker, Stephan Hildebrandt, Stefan Neumann, Thomas Vogel, and Sebastian Wätzoldt. "*Graph Transformations for MDE, Adaptation, and Models at Runtime*". In Formal Methods for Model-Driven Engineering, vol. 7320 of LNCS. Springer, 6 2012.
- Thomas Vogel and Holger Giese. "*Requirements and Assessment of Languages and Frameworks for Adaptation Models*". In Models in Software Engineering, vol. 7167 of LNCS. Springer, 4 2012.
- Thomas Vogel and Holger Giese. "*Language and Framework Requirements for Adaptation Models*". In Models@run.time 2011, vol. 794 of CEUR Workshop Proceedings. CEUR-WS.org, 10 2011 (best paper).
- Thomas Vogel, Andreas Seibel, and Holger Giese. "*The Role of Models and Megamodels at Runtime*". In Models in Software Engineering, vol. 6627 of LNCS. Springer, 5 2011.
- Thomas Vogel, Andreas Seibel, and Holger Giese. "*Toward Megamodels at Runtime*". In Models@run.time 2010, vol. 641 of CEUR Workshop Proceedings. CEUR-WS.org, 10 2010 (best paper).
- Thomas Vogel and Holger Giese. "*Adaptation and Abstract Runtime Models*". In SEAMS 2010. ACM, 5 2010.
- Thomas Vogel, Stefan Neumann, Stephan Hildebrandt, Holger Giese, and Basil Becker. "*Incremental Model Synchronization for Efficient Run-Time Monitoring*". In Models in Software Engineering, vol. 6002 of LNCS. Springer, 4 2010.
- Thomas Vogel, Stefan Neumann, Stephan Hildebrandt, Holger Giese, and Basil Becker. "*Incremental Model Synchronization for Efficient Run-time Monitoring*". In Models@run.time 2009, vol. 509 of CEUR Workshop Proceedings. CEUR-WS.org, 10 2009 (best paper).
- Holger Giese, Andreas Seibel, and Thomas Vogel. "*A Model-Driven Configuration Management System for Advanced IT Service Management*". In Models@run.time 2009, vol. 509 of CEUR Workshop Proceedings. CEUR-WS.org, 10 2009.
- Thomas Vogel, Stefan Neumann, Stephan Hildebrandt, Holger Giese, and Basil Becker. "*Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems*" (Poster Paper). In ICAC 2009. ACM, 6 2009.
- Thomas Vogel, Jens Bruhn, and Guido Wirtz. "*Autonomous Reconfiguration Procedures for EJB-based Enterprise Applications*". In SEKE 2008. Knowledge Systems Institute Graduate School, 7 2008.
- Jens Bruhn, Christian Niklaus, Thomas Vogel, and Guido Wirtz. "*Comprehensive support for management of Enterprise Applications*". In AICCSA 2008. IEEE, 3 2008.

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*.

[Blair et al., 2009] Blair, G., Bencomo, N., and France, R. B. (2009).
Models@run.time.
Computer, 42(10):22–27.

[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.

[Cheng et al., 2009] Cheng, B. H. C., Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Serugendo, G. D. M., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H. M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H. A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., and Whittle, J. (2009).
Software Engineering for Self-Adaptive Systems: A Research Roadmap.
In Cheng, B. H. C., Lemos, R., Giese, H., Inverardi, P., and Magee, J., editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer.

[de Lemos et al., 2012] de Lemos, R., Giese, H., Müller, H. A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N. M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Goeschka, K. M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovskii, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezze, M., Prehofer, C., Schäfer, W., Schlichting, R., Smith, D. B., Sousa, J. P., Tahvildari, L., Wong, K., and Wuttke, J. (2012).
Software Engineering for Self-Adaptive Systems: A Second Research Roadmap.
In de Lemos, R., Giese, H., Müller, H. A., and Shaw, M., editors, *Software Engineering for Self-Adaptive Systems 2*, Lecture Notes in Computer Science. 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.

[France and Rumpe, 2007] France, R. and Rumpe, B. (2007).
Model-driven Development of Complex Software: A Research Roadmap.
In *FOSE '07: 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA. IEEE Computer Society.

References II

[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.

[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.

[Lehman and Belady, 1985] Lehman, M. M. and Belady, L. A., editors (1985).
Program evolution: processes of software change.
Academic Press Professional, Inc., San Diego, CA, USA.

[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.

[Northrop et al., 2006] Northrop, L., Feiler, P. H., Gabriel, R. P., Linger, R., Longstaff, T., Kazman, R., Klein, M., and Schmidt, D. (2006).
Ultra-Large-Scale Systems: The Software Challenge of the Future.
Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[Parnas, 1994] Parnas, D. L. (1994).
Software aging.
In *ICSE '94: Proceedings of the 16th International Conference on Software Engineering*, pages 279–287, Los Alamitos, CA, USA. IEEE Computer Society Press.

[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 *Proceedings of the 5th Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2010) at the 32nd IEEE/ACM International Conference on Software Engineering (ICSE 2010), Cape Town, South Africa*, pages 39–48. ACM.

[Vogel and Giese, 2012] Vogel, T. and Giese, H. (2012).
A Language for Feedback Loops in Self-Adaptive Systems: Executable Runtime Megamodels.
In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2012)*. IEEE Computer Society.

References III

[Vogel et al., 2010] Vogel, T., Seibel, A., and Giese, H. (2010).
Toward Megamodels at Runtime.
In Bencomo, N., Blair, G., Fleurey, F., and Jeanneret, C., editors, *Proceedings of the 5th International Workshop on Models@run.time at the 13th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS 2010), Oslo, Norway*, 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 Dingel, J. and Solberg, A., editors, *Models in Software Engineering, Workshops and Symposia at MODELS 2010, Oslo, Norway, October 3-8, 2010, Reports and Revised Selected Papers*, volume 6627 of *Lecture Notes in Computer Science (LNCS)*, pages 224–238. Springer-Verlag.

Used Sources

- Slide 1: Dagstuhl figure from `http://www.dagstuhl.de/`.
- Slide 2: Ultra-large-scale systems [Northrop et al., 2006].