Engineering Self-Adaptive Software Systems with Runtime Models

Symposium on Future Trends in Service-Oriented Computing

Potsdam, Germany, June 14-15, 2012

Thomas Vogel
HPI Research School
System Analysis and Modeling Group
University of Potsdam, Germany

Motivation

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

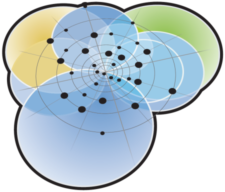
⇒ **Software evolution and maintenance**

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)
  - …

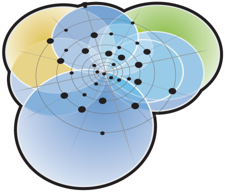


Motivation

- 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)
  - ...



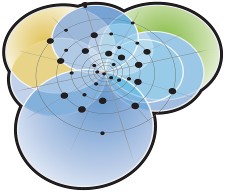
*"Evolution in ULS systems will rarely occur in discrete, planned steps in a closed environment; instead it will be continuous and dynamic. The rules for continuous evolution must therefore be built into ULS systems [...] so that they will be [...] able to cope with dynamically changing environments 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)
  - …



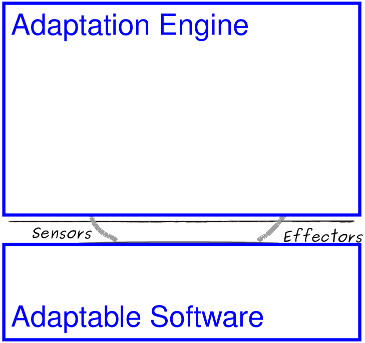
⇒ **Self-adaptive Software** [Cheng et al., 2009, de Lemos et al., 2012]

⇒ 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


Adaptation Engine

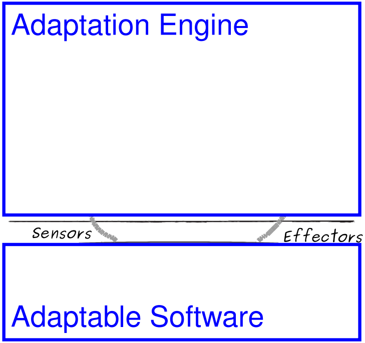
Adaptable Software

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



Related approaches, e.g.:

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

Models@run.time for engineering adaptation engines: (1)-(4)

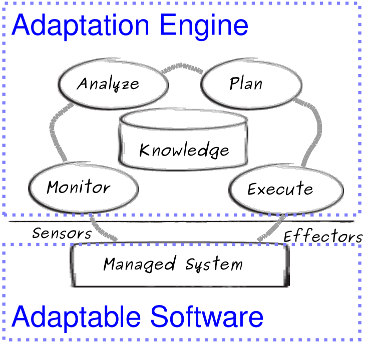
Adaptation Engine

Feedback Loop consisting of

- **Adaptation steps**

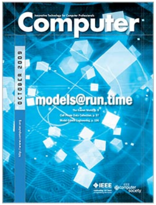
  Monitor, Analyze, Plan, Execute
- **Knowledge**

  about the managed system and its context
- MAPE-K [Kephart and Chess, 2003]



Adaptation Engine

Adaptable Software

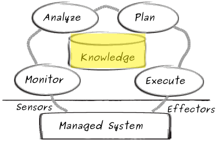


General goal: leverage MDE techniques and benefits to the runtime environment [France and Rumpe, 2007, Blair et al., 2009]

⇒ **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, . . .

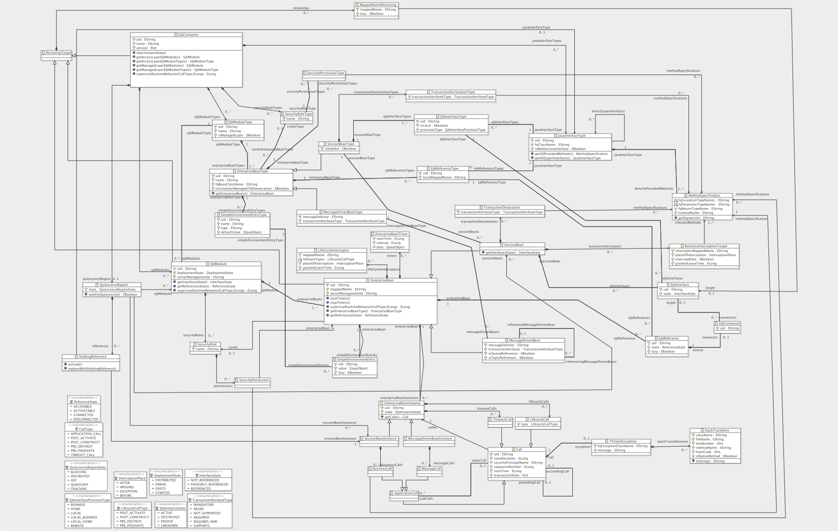
⇒ Different views on a running system

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

[ICAC09,MiSE10,SEAMS10]

Knowledge — Reflection Models

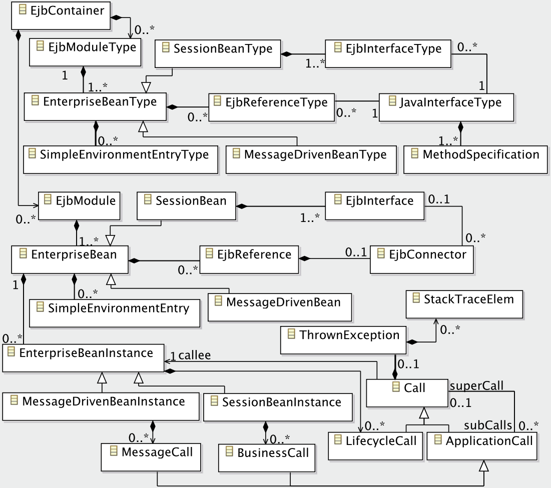
Metamodel of a PSM



Knowledge — Reflection Models

Metamodel of a PSM

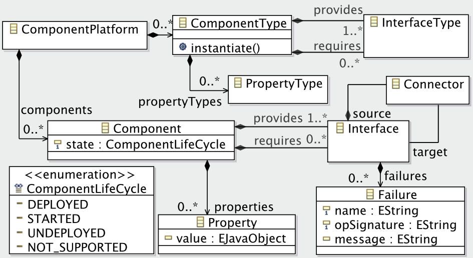
Simplified



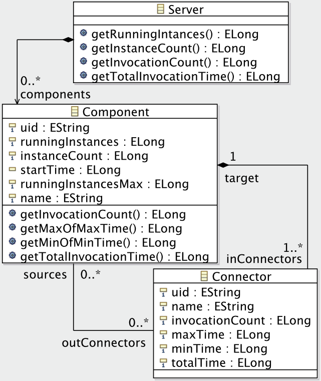
Knowledge — Reflection Models

Metamodels for PIMs

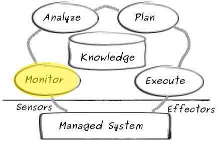
Failures



Performance



Monitor



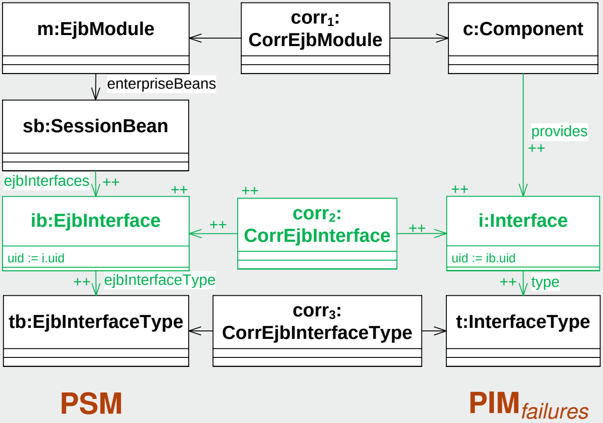
Synchronizing changes in the system to the reflection models

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

[MRT09,MiSE10]

Monitor — TGG Rules

TGG rule for PSM → PIM$_{failures}$



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

Monitor — Development costs

generated code from TGG rules

	Proposed solution			**Batch**
PIMs	#Rules	#Nodes/Rules	LOC	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 → PSM: 2685 LOC for the reusable adapter
  - PSM → 3 PIMs: 20 TGG rules (generated >33k LOC)
- **Batch** — creates PIMs directly from scratch (**non-incremental**)
  - 902 LOC (≈ 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 → PSM	0%	92.8%	94.1%	95.6%	95.2%	96.3%	-
PSM → 3 PIMs	0%	7.2%	5.9%	4.4%	4.8%	3.7%	-

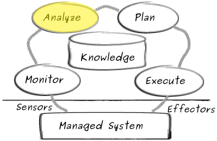
[ms]

- **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**

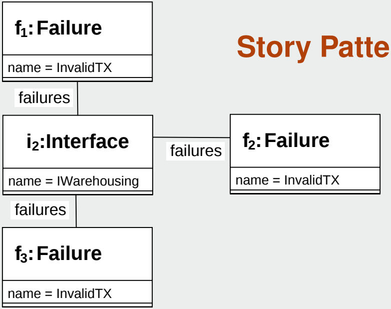
  (Story Pattern and Story Diagram Interpreter)
- Under certain conditions, **incremental** execution of Story Patterns
- Constraints expressed in the **Object Constraint Language (OCL)**

  (Existing engine from the Eclipse Model Development Tools)
- Model-based analysis techniques

[MRT11,MiSE12,SFM12]

Analyze — Evaluation Models

Identifying failures or violations of architectural constraints

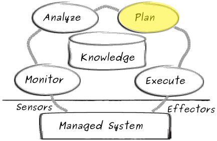


Story Pattern

OCL expression

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

Plan



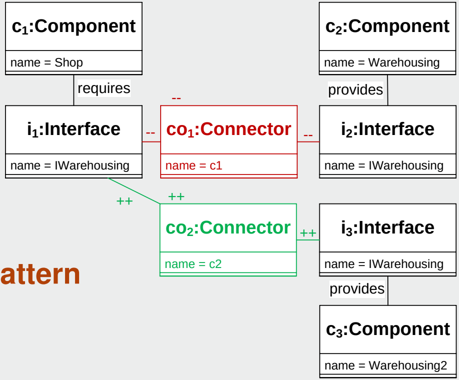
Planning adaptations based on analysis results

- 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)

[MRT11,MiSE12,SFM12]

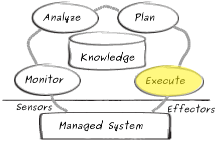
Plan — Change Models

Switching connections between components



Story Pattern

Execute



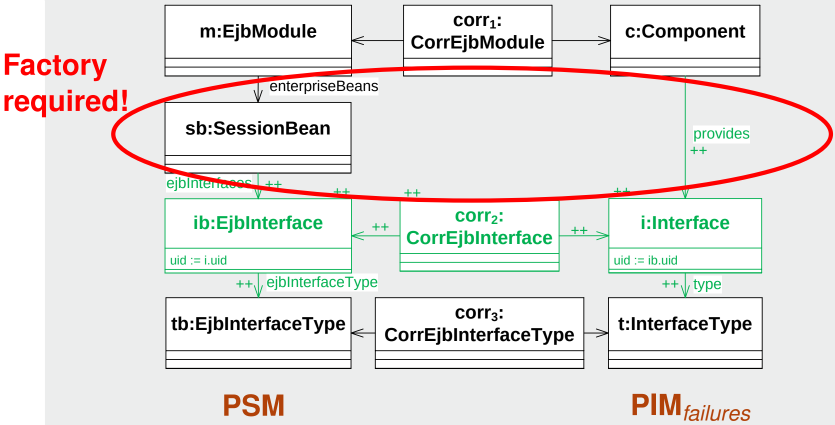
Synchronizing changes of reflection models to the system: PIMs → PSM → System

- 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)

[SEAMS10]

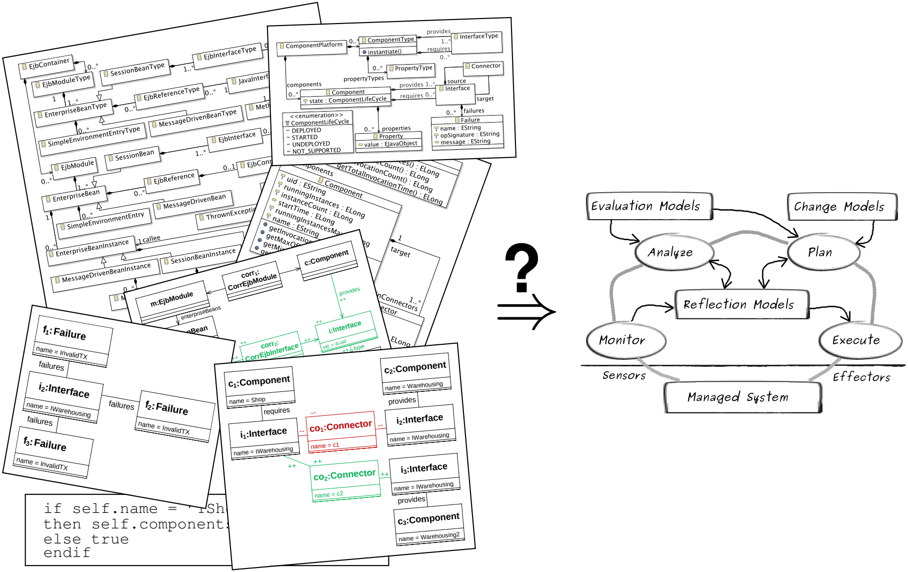
Execute — TGG Rules

TGG rule for PSM ↔ PIM$_{failures}$



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

Interplay of all those models?



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

[MRT10,MiSE11,SEAMS12]

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 (adaptive control)
  - 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., 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

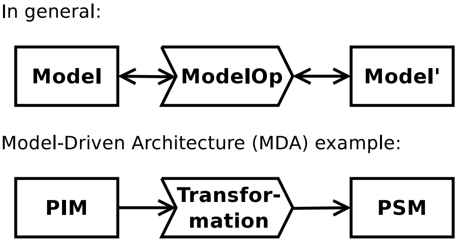
- Coordinated execution/usage of multiple runtime models
- **Flexible** solutions and structures for feedback loops
  - Adaptable feedback loops (adaptive control)
  - State-of-the-art frameworks often prescribe static solutions to single feedback loops (e.g., [Garlan et al., 2004, Schmidt et al., 2008])

Executable Runtime 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., 2010b]

An Example: Self-repair

Analyzed
Start
Effected

Legend

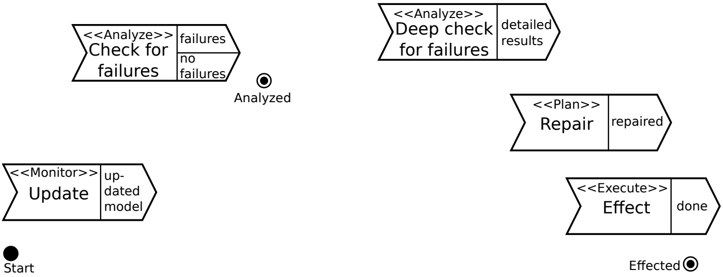
(concrete syntax)

● Initial state

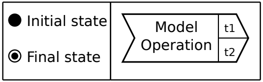
◉ Final state

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

An Example: Self-repair

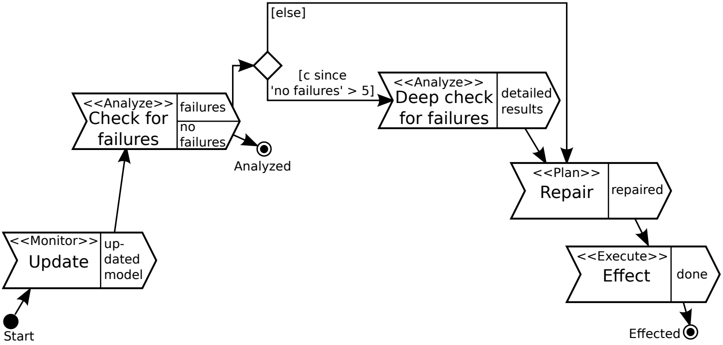


Legend
(concrete syntax)



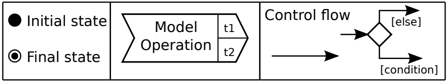
Remark: Abstract syntax defined by a metamodel [Vogel and Giese, 2012a]

An Example: Self-repair



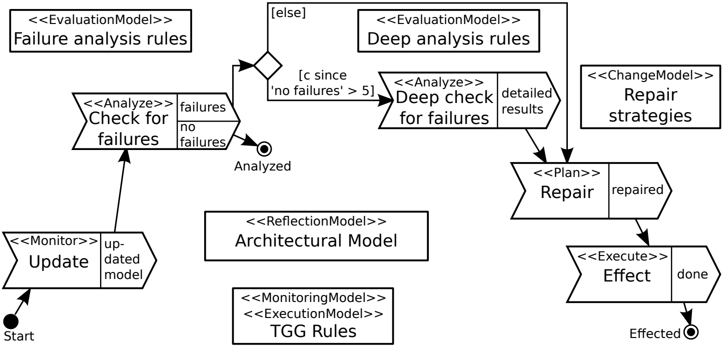
Legend

(concrete syntax)



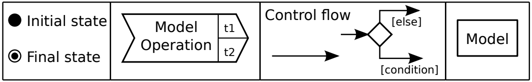
Remark: Abstract syntax defined by a metamodel [Vogel and Giese, 2012a]

An Example: Self-repair



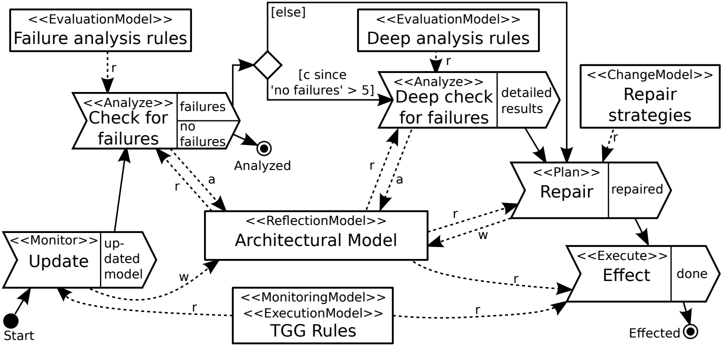
Legend

(concrete syntax)



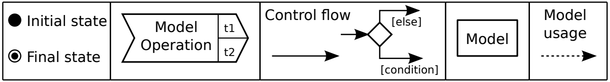
Remark: Abstract syntax defined by a metamodel [Vogel and Giese, 2012a]

An Example: Self-repair



Legend

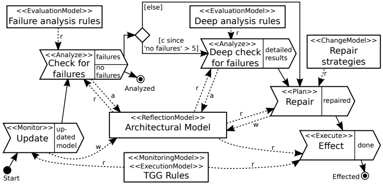
(concrete syntax)



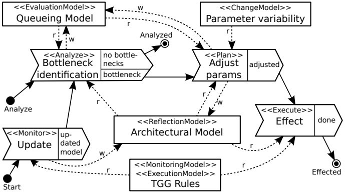
Remark: Abstract syntax defined by a metamodel [Vogel and Giese, 2012a]

Modeling Interacting Feedback Loops

Self-repair

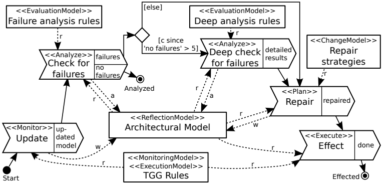


Self-optimization

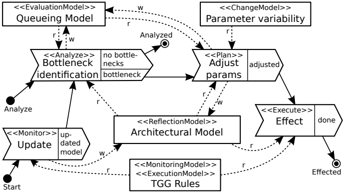


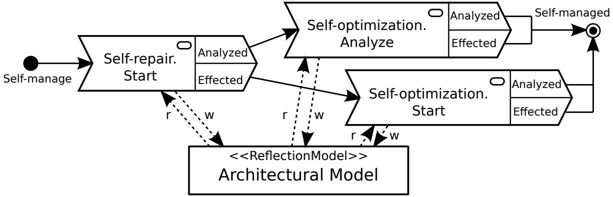
Modeling Interacting Feedback Loops

Self-repair



Self-optimization

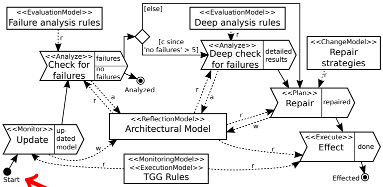




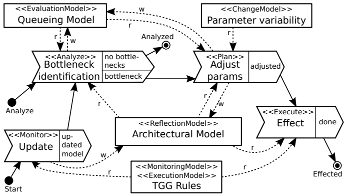
One solution: Linearizing Complete Feedback Loops

Modeling Interacting Feedback Loops

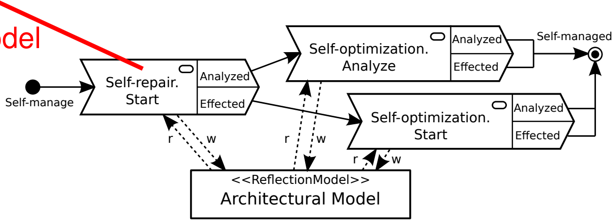
Self-repair



Self-optimization



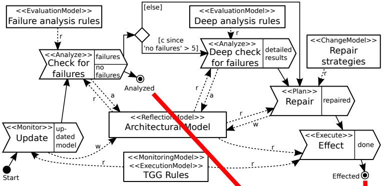
Complex model operations



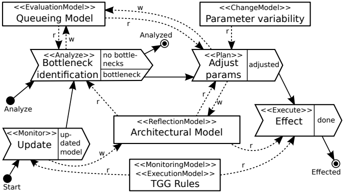
One solution: Linearizing Complete Feedback Loops

Modeling Interacting Feedback Loops

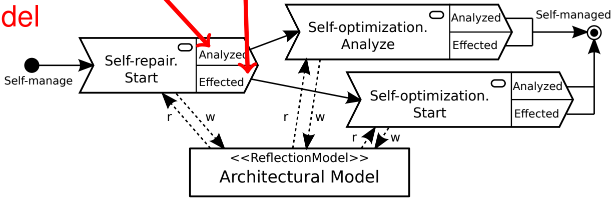
Self-repair



Self-optimization



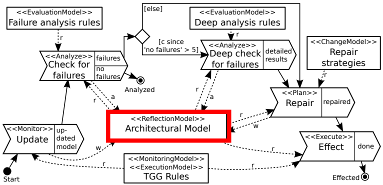
Complex model operations



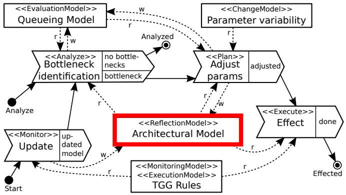
One solution: Linearizing Complete Feedback Loops

Modeling Interacting Feedback Loops

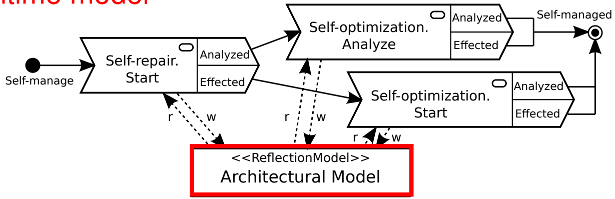
Self-repair



Self-optimization



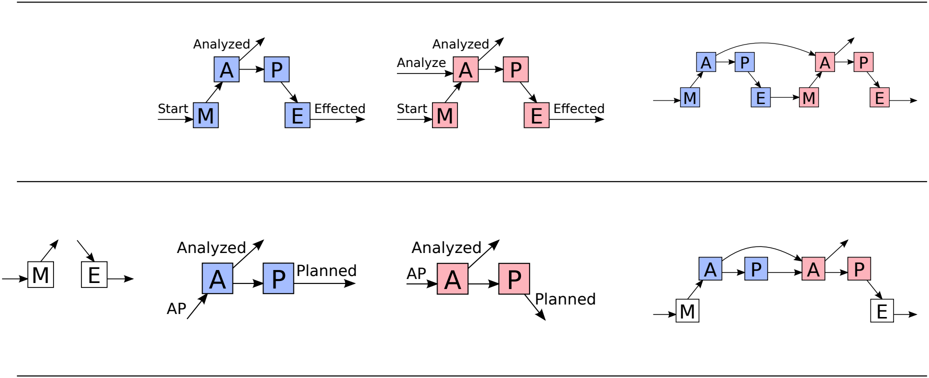
Shared runtime model



One solution: Linearizing Complete Feedback Loops

Other Solutions. . .

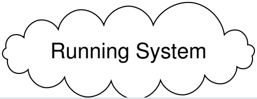
Generic Self-repair Self-optimization Composition



⇝ Patterns for control in self-adaptive systems [Weyns et al., 2012]

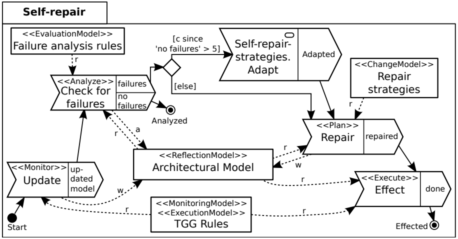
Modeling Hierarchies of Feedback Loops

Layer$_0$

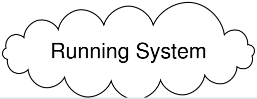


Modeling Hierarchies of Feedback Loops

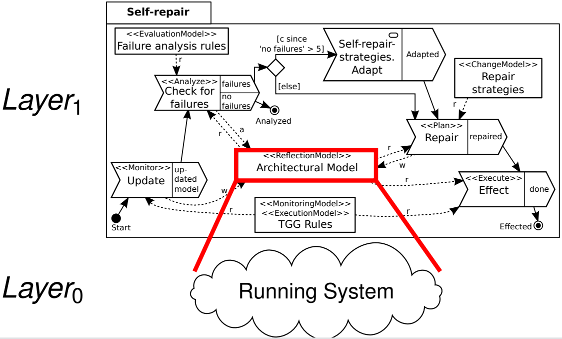
Layer$_1$



Layer$_0$



Modeling Hierarchies of Feedback Loops



Causal connection

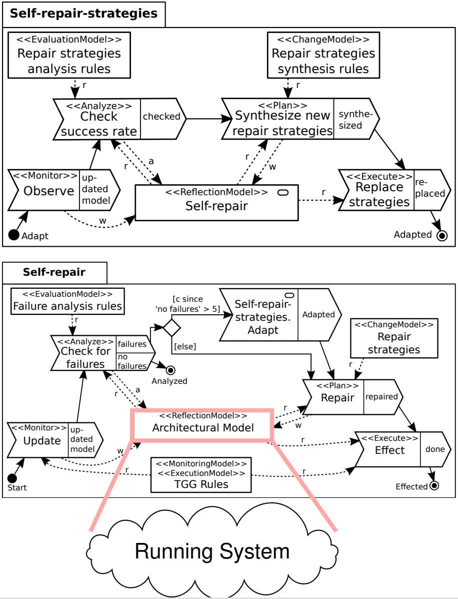
- sensors + effectors required
- implementation efforts!

Modeling Hierarchies 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
r
a
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
r
a
r
<<Plan>> Repair
repaired
r
w
<<ReflectionModel>> Architectural Model
<<Monitor>> Update
up-dated model
<<Execute>> Effect
done
w
r
r
r
<<MonitoringModel>> <<ExecutionModel>> TGG Rules
Start
Effected
Running System

Causal connection

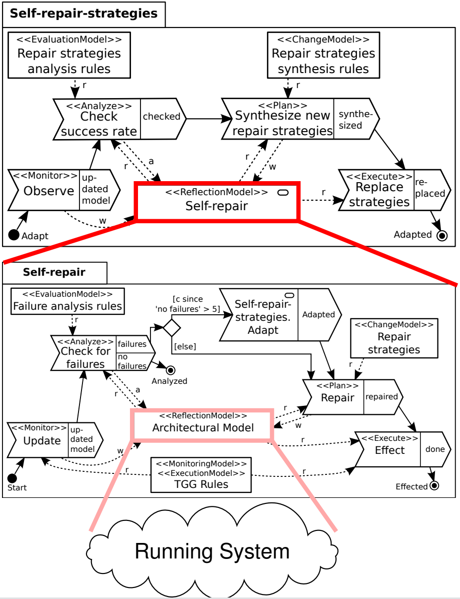
- sensors + effectors required
- implementation efforts!

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)!

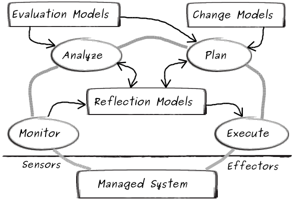
Causal connection

- 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)

References I

[Andersson et al., 2012] Andersson, J., Baresi, L., Bencomo, N., de Lemos, R., Gorla, A., Inverardi, P., and Vogel, T. (2012).
Software engineering processes for self-adaptive systems.
In de Lemos, R., Giese, H., Müller, H., and Shaw, M., editors, *Software Engineering for Self-Adaptive Systems 2*, volume tbd of *Lecture Notes in Computer Science (LNCS)*, page tbd. Springer-Verlag.
(to be published).

[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.

[Bruhn et al., 2008] Bruhn, J., Niklaus, C., Vogel, T., and Wirtz, G. (2008).
Comprehensive support for management of Enterprise Applications.
In *Proceedings of the 6th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2008), Doha, Katar*, pages 755–762. IEEE Computer Society.

[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.
(to be published).

[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.

References II

[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.

[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.

[Giese et al., 2012] Giese, H., Lambers, L., Becker, B., Hildebrandt, S., Neumann, S., Vogel, T., and Wätzoldt, S. (2012).
Graph Transformations for MDE, Adaptation, and Models at Runtime, volume 7320 of *LNCS*.
Springer.
(to be published).

[Giese et al., 2009] Giese, H., Seibel, A., and Vogel, T. (2009).
A Model-Driven Configuration Management System for Advanced IT Service Management.
In Bencomo, N., Blair, G., France, R., Jeanneret, C., and Munoz, F., editors, *Proceedings of the 4th International Workshop on Models@run.time at the 12th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS 2009), Denver, Colorado, USA*, volume 509 of *CEUR Workshop Proceedings*, pages 61–70. CEUR-WS.org.

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

[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.

[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 et al., 2008] Vogel, T., Bruhn, J., and Wirtz, G. (2008).
Autonomous Reconfiguration Procedures for EJB-based Enterprise Applications.
In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE 2008), San Francisco, CA, USA*, pages 48–53. Knowledge Systems Institute Graduate School.

References III

[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 and Giese, 2011] Vogel, T. and Giese, H. (2011).
Language and Framework Requirements for Adaptation Models.
In Bencomo, N., Blair, G., Cheng, B. H. C., France, R. B., and Jeanneret, C., editors, *Proceedings of the 6th International Workshop on Models@run.time at the 14th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS 2011), Wellington, New Zealand*, volume 794 of *CEUR Workshop Proceedings*, pages 1–12. CEUR-WS.org.
(best paper).

[Vogel and Giese, 2012a] Vogel, T. and Giese, H. (2012a).
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.

[Vogel and Giese, 2012b] Vogel, T. and Giese, H. (2012b).
Requirements and Assessment of Languages and Frameworks for Adaptation Models.
In *MoDELS 2011 Workshops*, volume 7167 of *LNCS*, pages 167–182. Springer.

[Vogel et al., 2009a] Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., and Becker, B. (2009a).
Incremental Model Synchronization for Efficient Run-time Monitoring.
In Bencomo, N., Blair, G., France, R., Jeanneret, C., and Munoz, F., editors, *Proceedings of the 4th International Workshop on Models@run.time at the 12th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS 2009), Denver, Colorado, USA*, volume 509 of *CEUR Workshop Proceedings*, pages 1–10. CEUR-WS.org.

[Vogel et al., 2009b] Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., and Becker, B. (2009b).
Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems.
In *Proceedings of the 6th IEEE/ACM International Conference on Autonomic Computing and Communications (ICAC 2009), Barcelona, Spain*, pages 67–68. 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.

[Weyns et al., 2012] Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., and Göschka, K. (2012).
On Patterns for Decentralized Control in Self-Adaptive Systems.
In de Lemos, R., Giese, H., Müller, H. A., and Shaw, M., editors, *Software Engineering for Self-Adaptive Systems 2*, volume tbd of *Lecture Notes in Computer Science (LNCS)*, page tbd. Springer-Verlag.
(to be published).