HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

13th International Workshop on Models@run.time

Towards software architecture runtime models for continuous adaptive monitoring

Thomas Brand, Holger Giese

14.10.2018

Agenda

- **Show why it is relevant** to investigate and support:
  - Continuous adaptive monitoring
  - Modeling languages for long living runtime model instances
- **Demonstrate the significance** of the modeling language
- **Describe the planned roadmap** for proposing an evaluated solution
- **Derive requirements** from illustrative scenarios and indicate how they are supported by two existing approaches
- **Questions and discussion**

Relevancy

How does the runtime model modeling language relate to this?

Why is monitoring adaptation without interruption important?

Why does monitoring need to be adaptive?

Setting the context

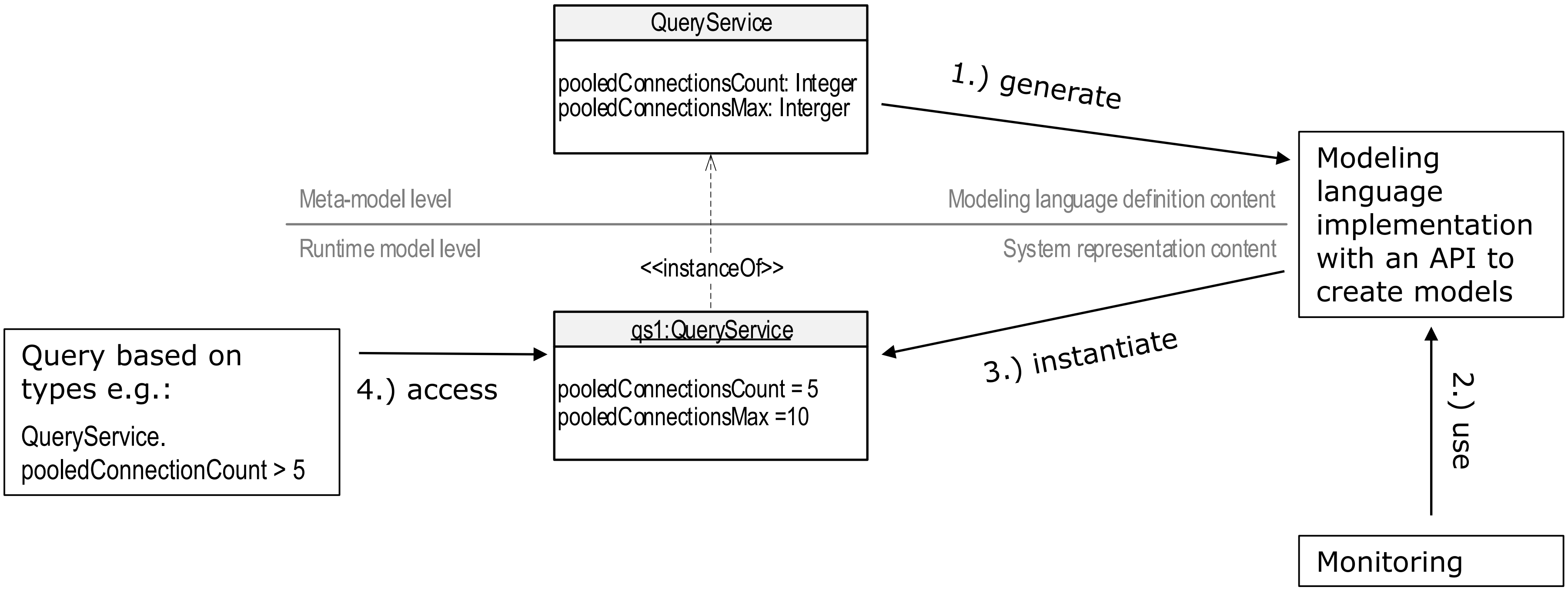
“models@run.time is an **abstraction** of a **running system** that is being **manipulated at runtime** for a **specific purpose**”

[Bencomo.2013]

Please imagine a software architecture runtime model thinking of:

- **graph in a datastore**
- **running system**
- **current monitoring results**
- **analysis and phenomena detection processes**

Classical Model-Driven Engineering approach


QueryService
pooledConnectionsCount: Integer
pooledConnectionsMax: Interger
1.) generate
Modeling language implementation with an API to create models
Meta-model level
Modeling language definition content
Runtime model level
System representation content
<<instanceOf>>
qs1:QueryService
pooledConnectionsCount = 5
pooledConnectionsMax =10
3.) instantiate
2.) use
Monitoring
Query based on types e.g.:
QueryService.
pooledConnectionCount > 5
4.) access

Motivation

- Monitored system and information demands **change over time**
  - Usage measurement and experimentation in software product development
  - Highly dynamic architectures based on microservices
  - Exploration and exploitation with machine learning …
- Modeling language determines **possible information types**
- Evolving the modeling language requires a **model re-instantiation**
- Re-instantiations interrupt the **monitoring and phenomena detection processes** and endanger continuous system operation

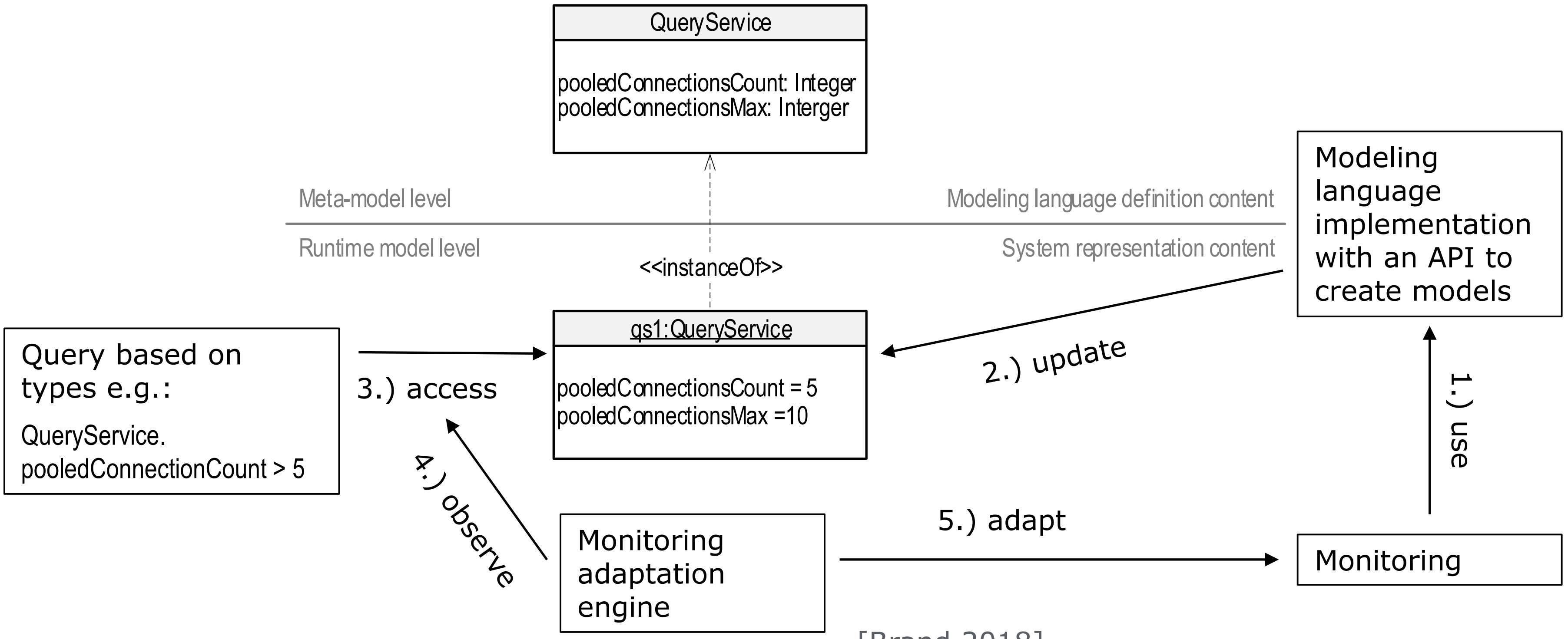
---

- A flexible **modeling language** regarding the types of information in the runtime model
  - Makes long living runtime model instances possible and supports continuous adaptive monitoring and system operation
  - Increases the feasibility of runtime models for additional fields of application

Significance of the modeling language

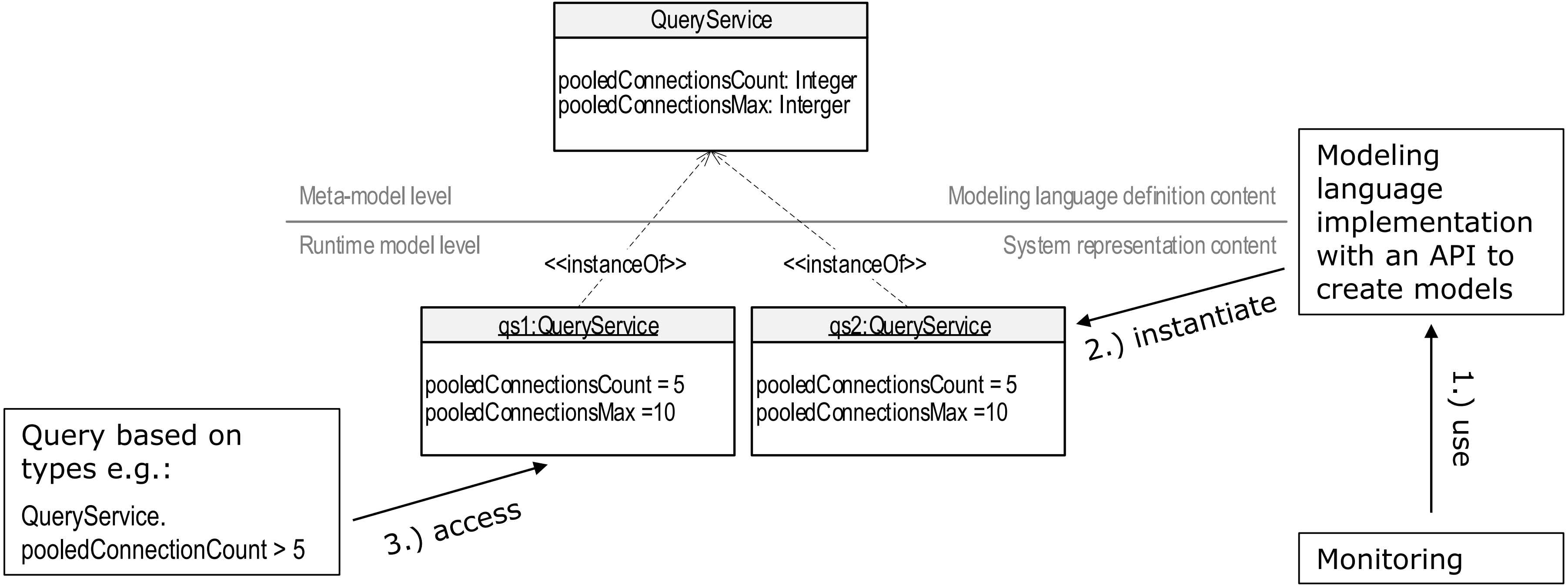
To better understand:
Can you show how the modeling language is actually significant?

Information demand changes - Filtering


QueryService
pooledConnectionsCount: Integer
pooledConnectionsMax: Interger
Meta-model level
Modeling language definition content
Runtime model level
System representation content
<<instanceOf>>
Modeling language implementation with an API to create models
qs1:QueryService
pooledConnectionsCount = 5
pooledConnectionsMax =10
Query based on types e.g.:
QueryService.
pooledConnectionCount > 5
3.) access
2.) update
1.) use
4.) observe
Monitoring adaptation engine
5.) adapt
Monitoring

[Brand.2018]

Running system changes - System adaptation



QueryService

pooledConnectionsCount: Integer
pooledConnectionsMax: Interger

Meta-model level

Modeling language definition content

Runtime model level

System representation content

<<instanceOf>>

<<instanceOf>>

qs1:QueryService

pooledConnectionsCount = 5
pooledConnectionsMax =10

qs2:QueryService

pooledConnectionsCount = 5
pooledConnectionsMax =10

Modeling language implementation with an API to create models

2.) instantiate

1.) use

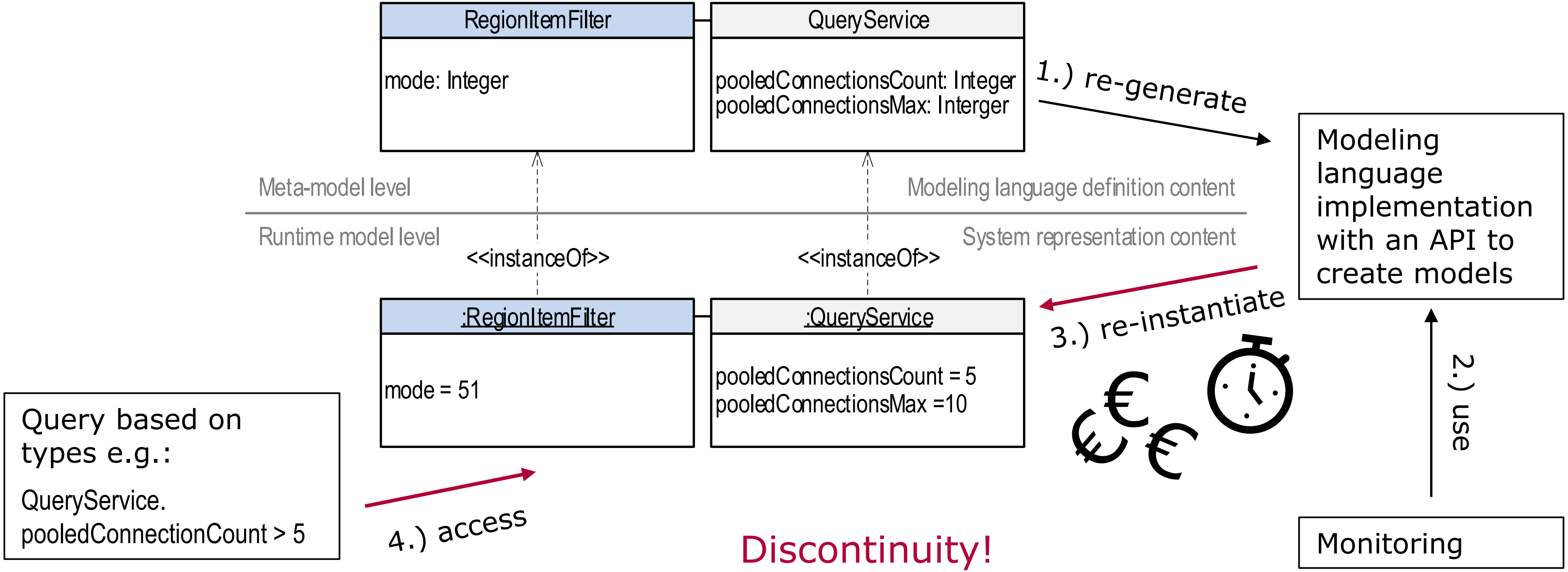
Monitoring

Query based on types e.g.:

QueryService.
pooledConnectionCount > 5

3.) access

Running system changes - System evolution



RegionItemFilter

mode: Integer

QueryService

pooledConnectionsCount: Integer
pooledConnectionsMax: Interger

1.) re-generate

Modeling language implementation with an API to create models

Meta-model level

Modeling language definition content

Runtime model level

System representation content

<<instanceOf>>

<<instanceOf>>

:RegionItemFilter

mode = 51

:QueryService

pooledConnectionsCount = 5
pooledConnectionsMax =10

3.) re-instantiate

2.) use

Monitoring

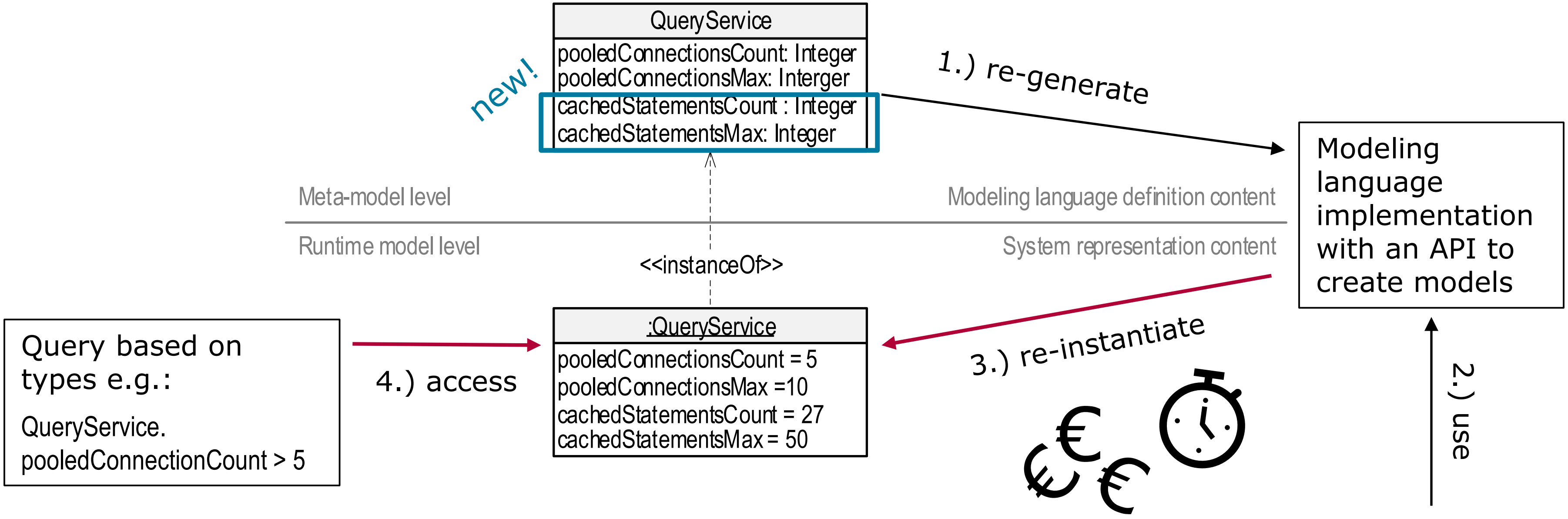
Query based on types e.g.:

QueryService.
pooledConnectionCount > 5

4.) access

Discontinuity!

Running system changes - Software evolution



QueryService

pooledConnectionsCount: Integer
pooledConnectionsMax: Interger
cachedStatementsCount : Integer
cachedStatementsMax: Integer

new!

1.) re-generate

Modeling language implementation with an API to create models

Meta-model level

Modeling language definition content

Runtime model level

System representation content

<<instanceOf>>

:QueryService

pooledConnectionsCount = 5
pooledConnectionsMax =10
cachedStatementsCount = 27
cachedStatementsMax = 50

3.) re-instantiate

2.) use

Monitoring

Query based on types e.g.:

QueryService.
pooledConnectionCount > 5

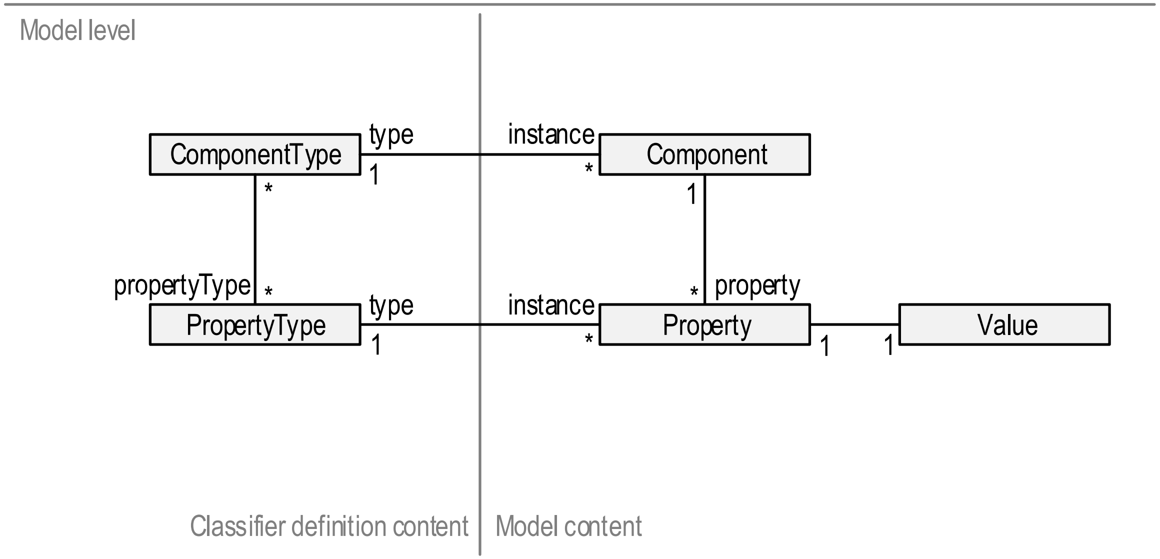
4.) access

Discontinuity!

The CompArch approach

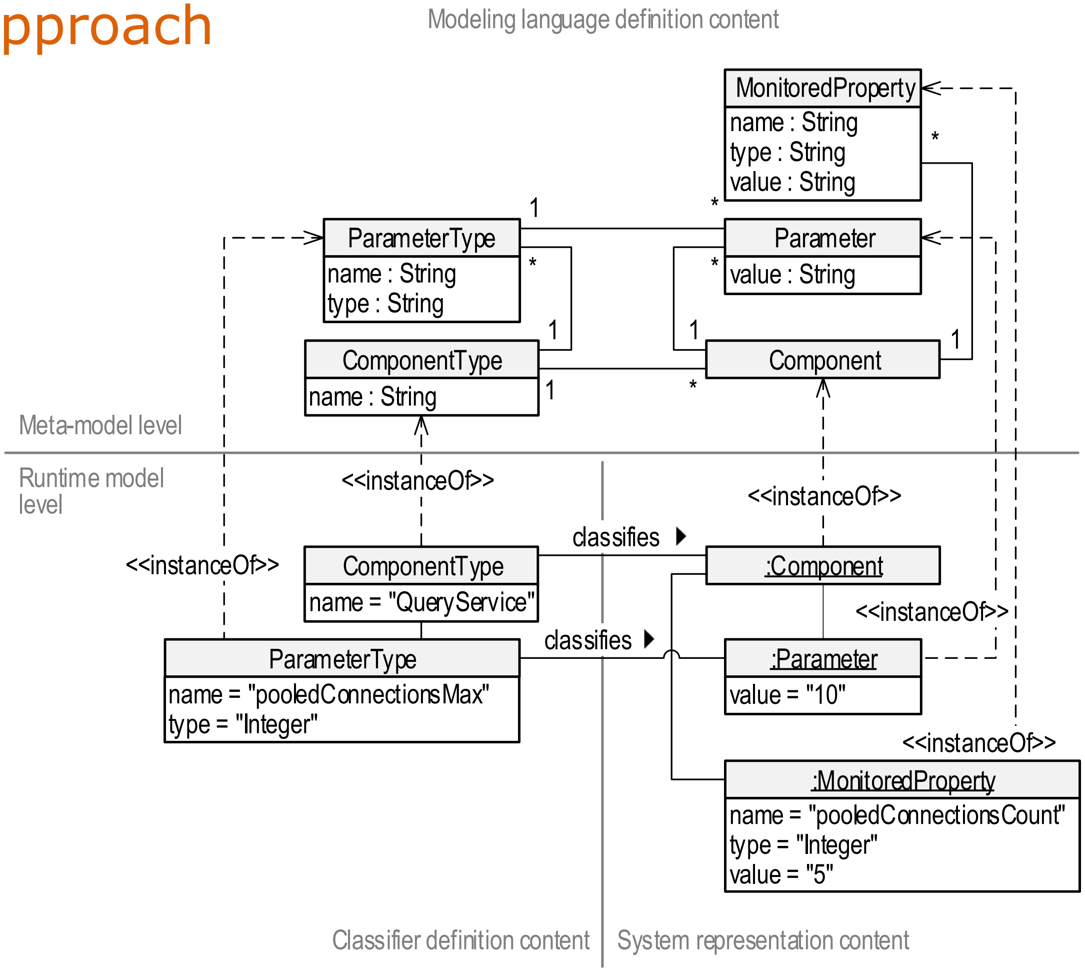
There is a way to improve the situation compared to the classical approach!?

Dynamic Object Model pattern



[Riehle.2005]

The CompArch approach



[Vogel.2018]

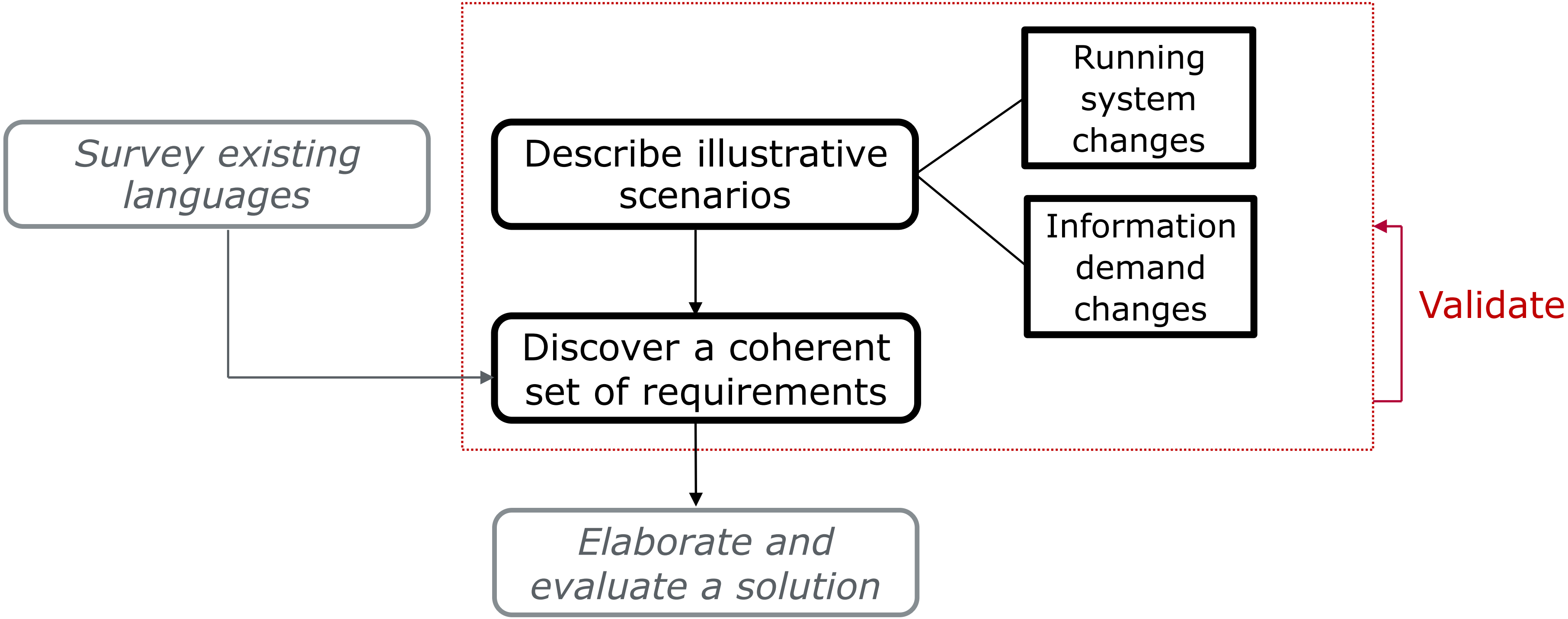
Planned roadmap towards a prospective solution

How shall the proposed solution be evaluated?

What actually are the important requirements?

Does this approach fulfill the requirements?

Planned roadmap towards a prospective solution


Survey existing languages
Describe illustrative scenarios
Running system changes
Information demand changes
Discover a coherent set of requirements
Validate
Elaborate and evaluate a solution

Illustrative scenarios and requirements

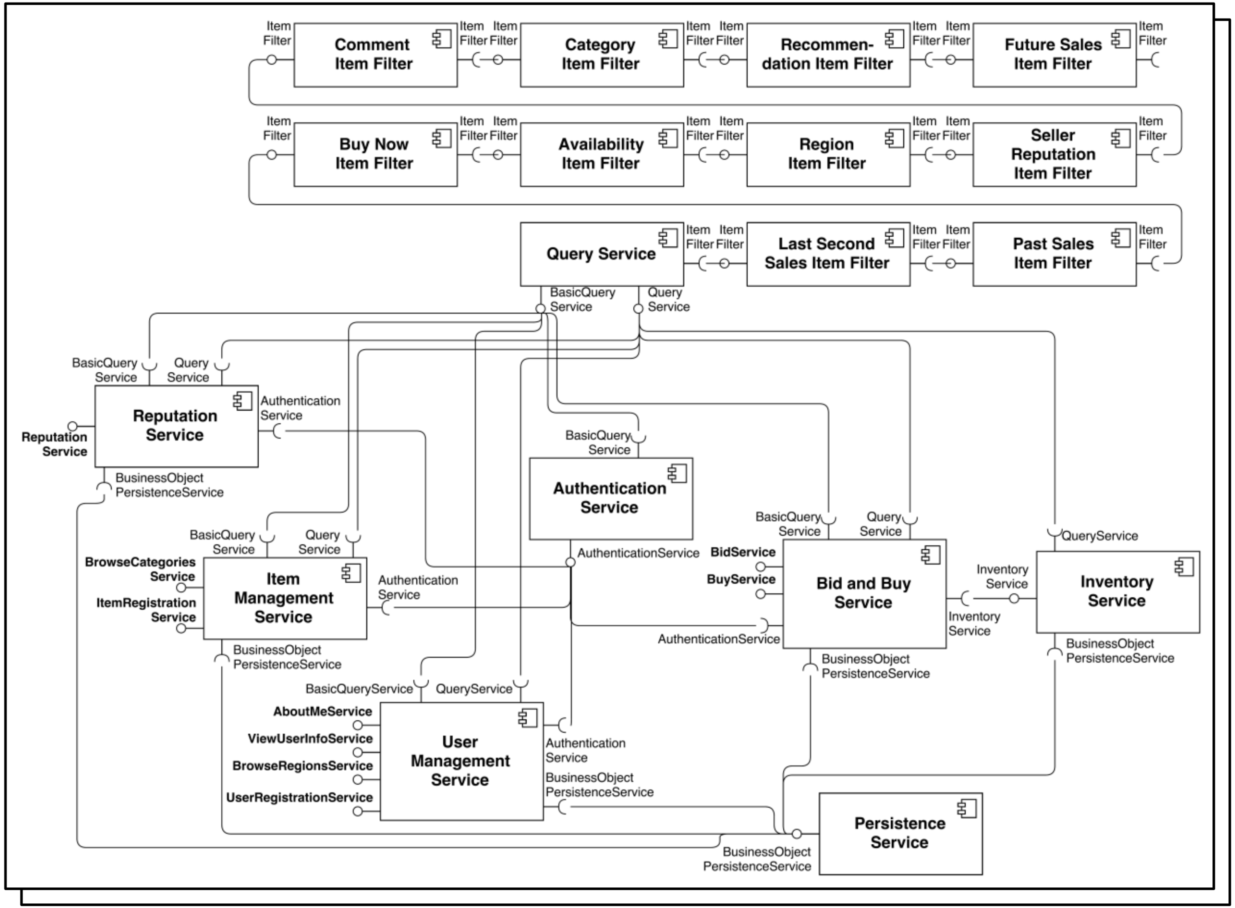
Can you give us some examples of scenarios and requirements?

Scenarios and requirements overview

	Illustrative scenarios	Requirements
Running system changes	S1 - System adaptation	R1 - Updating system representation structure and values
	S2 - System evolution	R2 - Indicating the actual information demand
	S3 - Software evolution	R3 - Introducing new classifiers including classifier versions
	S4 - Systems integration and division	R4 - Withdrawing obsolete classifiers
Information demand changes	S5 - Filtering	R5 - Establishing new kinds of relationships
	S6 - Aggregation	R6 - Assigning multiple classifiers progressively
	S7 - Itemization	R7 - Integrating multiple classifier systems
	S8 - Generalization and specialization	R8 - Introducing new logical elements and relationships

HPI Hasso Plattner Institut

Example system



Multiple tenants

Simplified mRUBiS runtime model

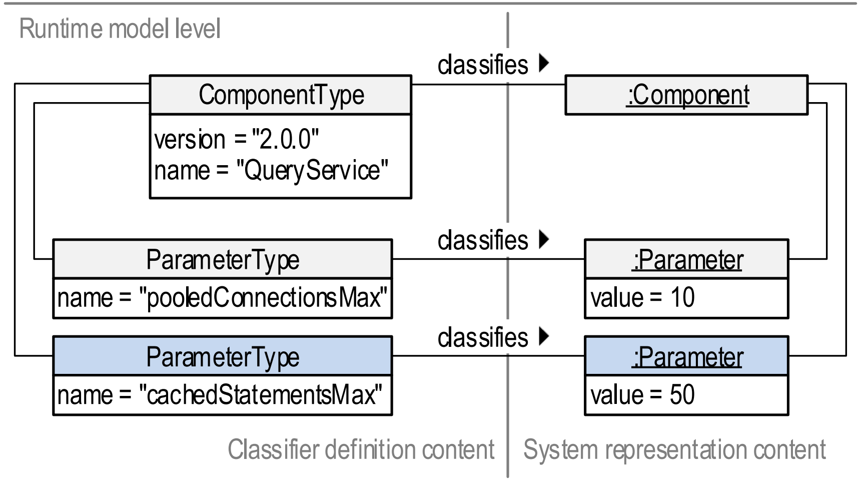
mRUBiS

[Vogel.2018]

Running system changes

S3 - Software evolution

- Conduct an experiment with new software product version
- Deploy a new version of the QueryService component to early adopter tenants
- Represent new component version with additional properties besides the old

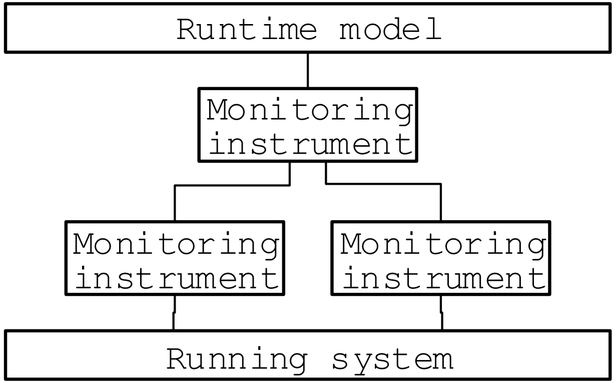


Requirements	Classic	ComArch
R3 - Introducing new classifiers including classifier versions	--	(✓)
S3 - Software evolution	--	(✓)

Information demand changes

S6 - Aggregation - Case 1: Invisible

Aggregation not visible in the runtime model
(on the monitoring instrument level)

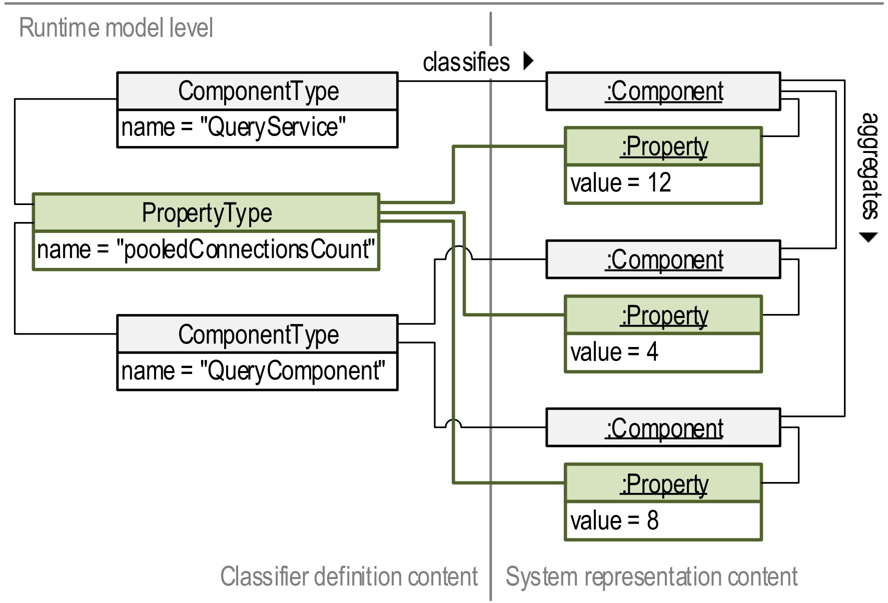


Information demand changes

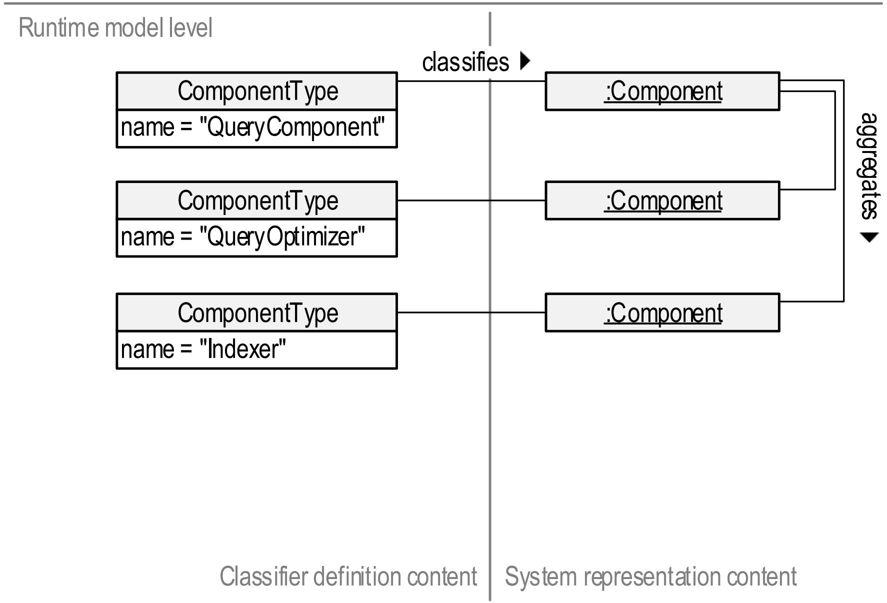
S6 - Aggregation - Case 2: Visible

Aggregation visible in the runtime model

Case 2.a: Functional aggregation



Case 2.b: Structural aggregation



Information demand changes

S6 - Aggregation

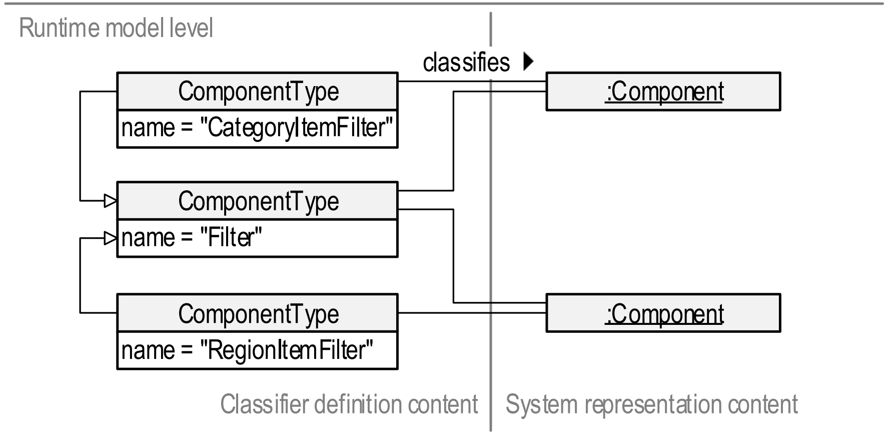
- Represent the service which all query component instances provide together
- Aggregate on the monitoring instrument level
- Provide the sum of exceptions for all early adaptors of query service v2.0.0
- Aggregate on the runtime model level

Requirements	Classic	ComArch
R3 - Introducing new classifiers including classifier versions	--	(✓)
R4 - Withdrawing obsolete classifiers	--	✓
R5 - Establishing new kinds of relationships	--	--
R8 - Introducing new logical elements and relationships	--	(✓)
S6 - Aggregation	--	--

Information demand changes

S8 - Generalization and specialization

- Indicate potential for configuration optimization by reporting two filters
- Query the number-of-filtered-items property which is common for all filter types
- Consider ten filters of different types in a general way for the query
- Have a specific and a more general classifier assigned to each filter


Runtime model level
classifies
ComponentType
name = "CategoryItemFilter"
ComponentType
name = "Filter"
ComponentType
name = "RegionItemFilter"
:Component
:Component
Classifier definition content
System representation content

Requirements	Classic	ComArch
R6 - Assigning multiple classifiers progressively	--	--
S8 - Generalization and specialization	--	--

Illustrative scenarios and requirements

How far are the requirements covered by the two approaches you looked at?

What are the remaining scenarios and identified requirements?

Scenarios and requirements coverage overview

Scenarios
S1 - System adaptation
S2 - System evolution
S3 - Software evolution
S4 - Systems integration and division
S5 - Filtering
S6 - Aggregation
S7 - Itemization
S8 - Generalization and specialization

Requirements	Classical	ComArch
R1 - Updating system representation structure and values	✓	✓
R2 - Indicating the actual information demand	(✓)	(✓)
R3 - Introducing new classifiers including classifier versions	--	(✓)
R4 - Withdrawing obsolete classifiers	--	✓
R5 - Establishing new kinds of relationships	--	--
R6 - Assigning multiple classifiers progressively	--	--
R7 - Integrating multiple classifier systems	--	--
R8 - Introducing new logical elements and relationships	--	(✓)

Summary

- Saw that runtime model modeling languages for flexibility are worth investigating
- Discussed plans on how to elaborate and evaluate a prospective solution
- Discussed the identified requirements

Outlook

- Complete the definition of a coherent set of scenarios and requirements also based on analyzing existing modeling languages
- Elaborate a proposal
- Evaluate regarding cost-effectiveness and support for the requirements
- Consider co-evolution of queries and the runtime model modeling language

References

[Bencomo.2013] N. Bencomo, G. Blair, et al., “Report on the 7th International Workshop on Models@run.time,” in SIGSOFT Software Engineering Notes, ACM, New York, 2013.

[Brand.2018] T. Brand, H. Giese, “Towards Generic Adaptive Monitoring” in 2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), to appear, 2018.

[Riehle.2005] D. Riehle, M. Tilman, et al., “Dynamic Object Model,” in Pattern Languages of Program Design 5, Addison-Wesley, Upper Saddle River, 2005.

[Vogel.2018] T. Vogel, “An Exemplar for Model-Based Architectural Self-Healing and Self-Optimization,” in International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2018.

Thomas Brand and Holger Giese

Hasso Plattner Institute at the University of Potsdam, Germany

{firstname.lastname}@hpi.uni-potsdam.de