Karlsruhe Institute of Technology

Challenges in Evolving Metamodels

Misha Strittmatter, Robert Heinrich

EMLS Workshop, SE 2016, Wien

SOFTWARE DESIGN AND QUALITY GROUP
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS

Why use Modeling / Metamodels?

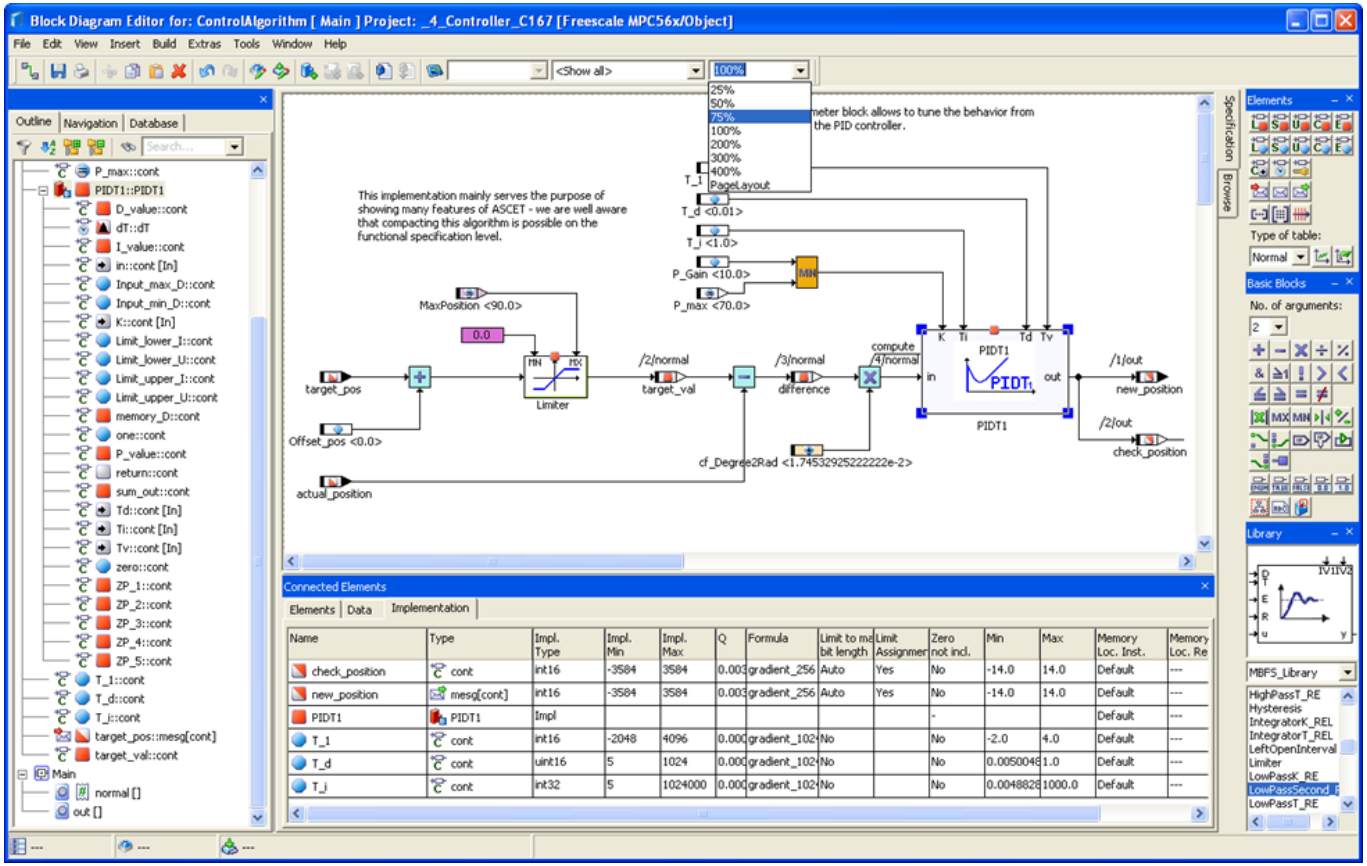
- Model-driven Engineering (MDE)
  - Use domain-specific models
  - Raise the abstraction level
  - „Hide“ complexity behind tooling
  - Use models for analysis, simulation, code generation

Why use Modeling / Metamodels?

- Model-driven Engineering (MDE)
  - Use domain-specific models
  - Raise the abstraction level
  - „Hide“ complexity behind tooling
  - Use models for analysis, simulation, code generation
- Metamodel
  - Abstract syntax of a (modelling) language
  - Defines set of its instances (models)

Example: ASCET

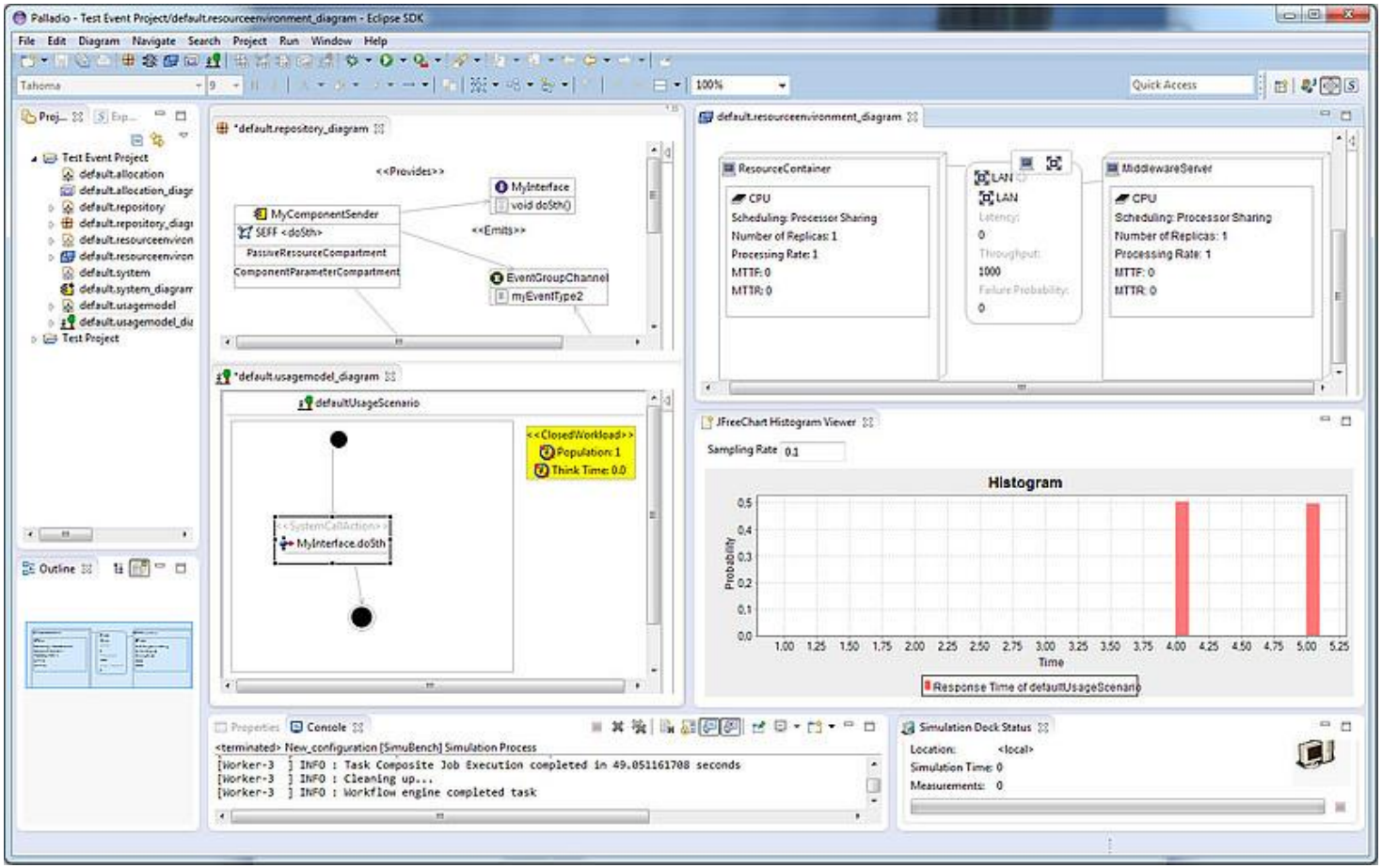
- **A**dvanced **S**imulation and **C**ontrol **E**ngineering **T**ool



(Wikipedia)

Example: PCM

- **P**alladio **C**omponent **M**odel



(http://www.palladio-simulator.com/)

Further (Industry Relevant) Examples

- UML
- AADL, AMALTHEA, EAST-ADL (automotive)
- ArchiMate
- Simulink
- Rapide
- …

(partly from a study about most used architecture description languages in industry [1])

Further (Industry Relevant) Examples

- UML
- AADL, AMALTHEA, EAST-ADL (automotive)
- ArchiMate
- Simulink
- Rapide
- …

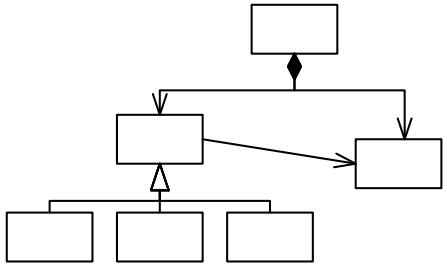
Examples for modeling languages (not necessarily metamodeled)

(partly from a study about most used architecture description languages in industry [1])

Metamodels

- Our focus: EMOF based metamodels (or similar)
- Classes, attributes, associations, inheritance, containment

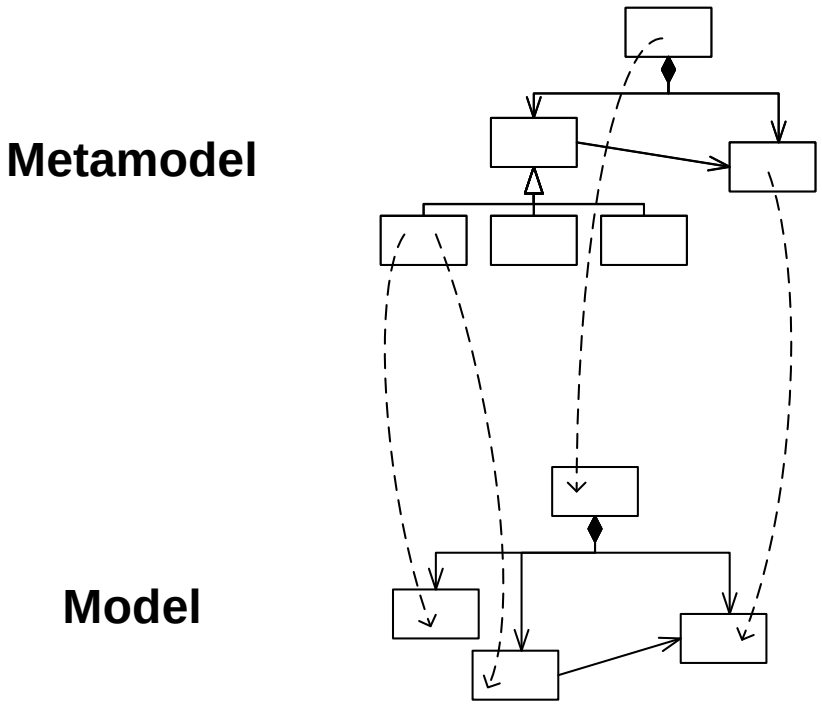
Metamodel



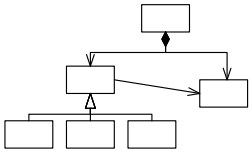
Software Design and Quality Group
Institute for Program Structures and Data Organization

Metamodels

- Our focus: EMOF based metamodels (or similar)
- Classes, attributes, associations, inheritance, containment

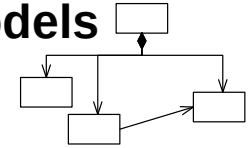


Usage

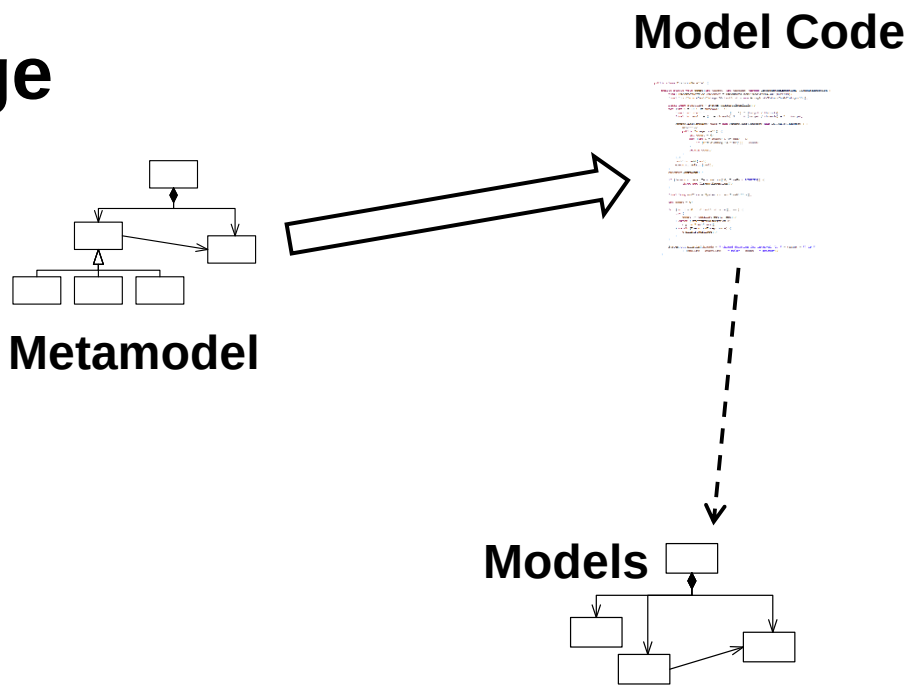


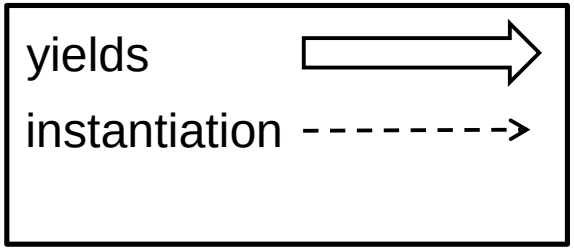
Metamodel

Models

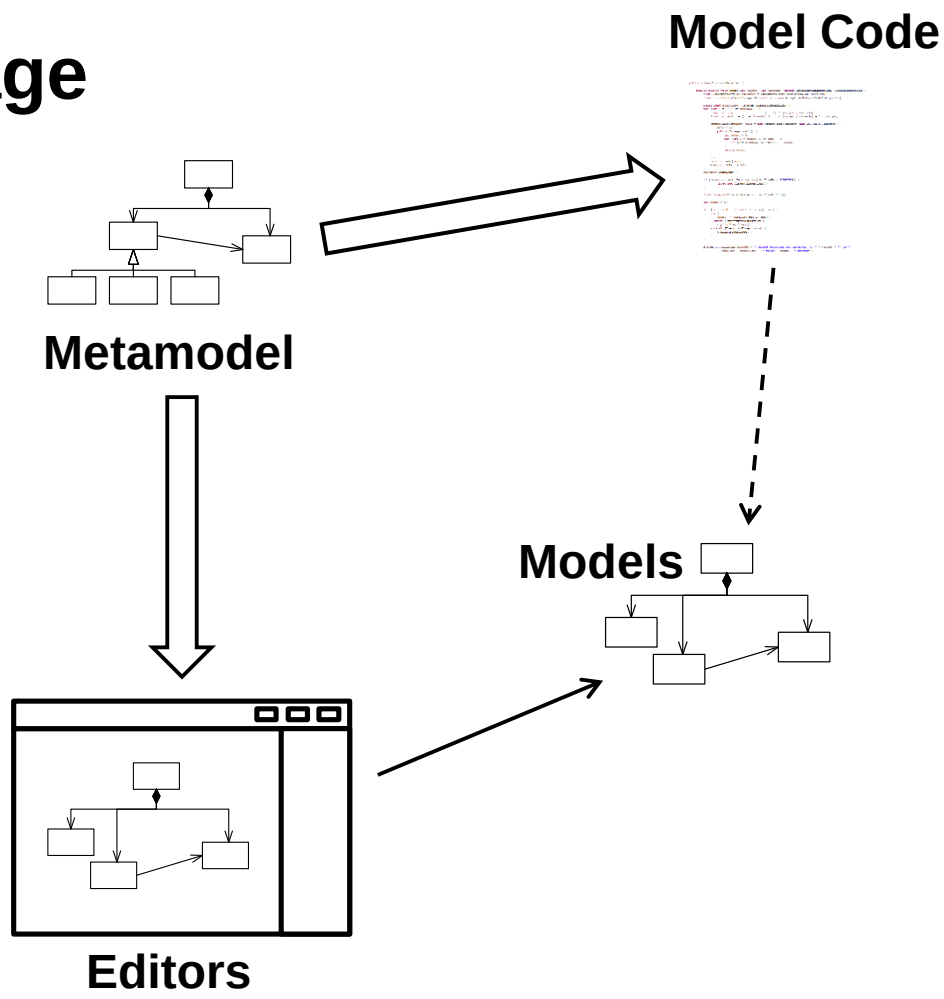


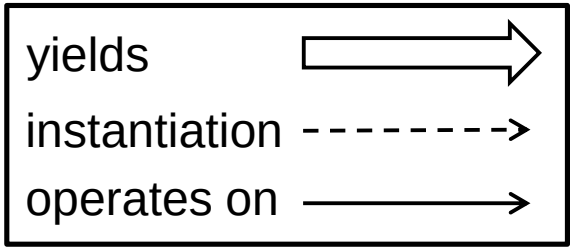
Usage



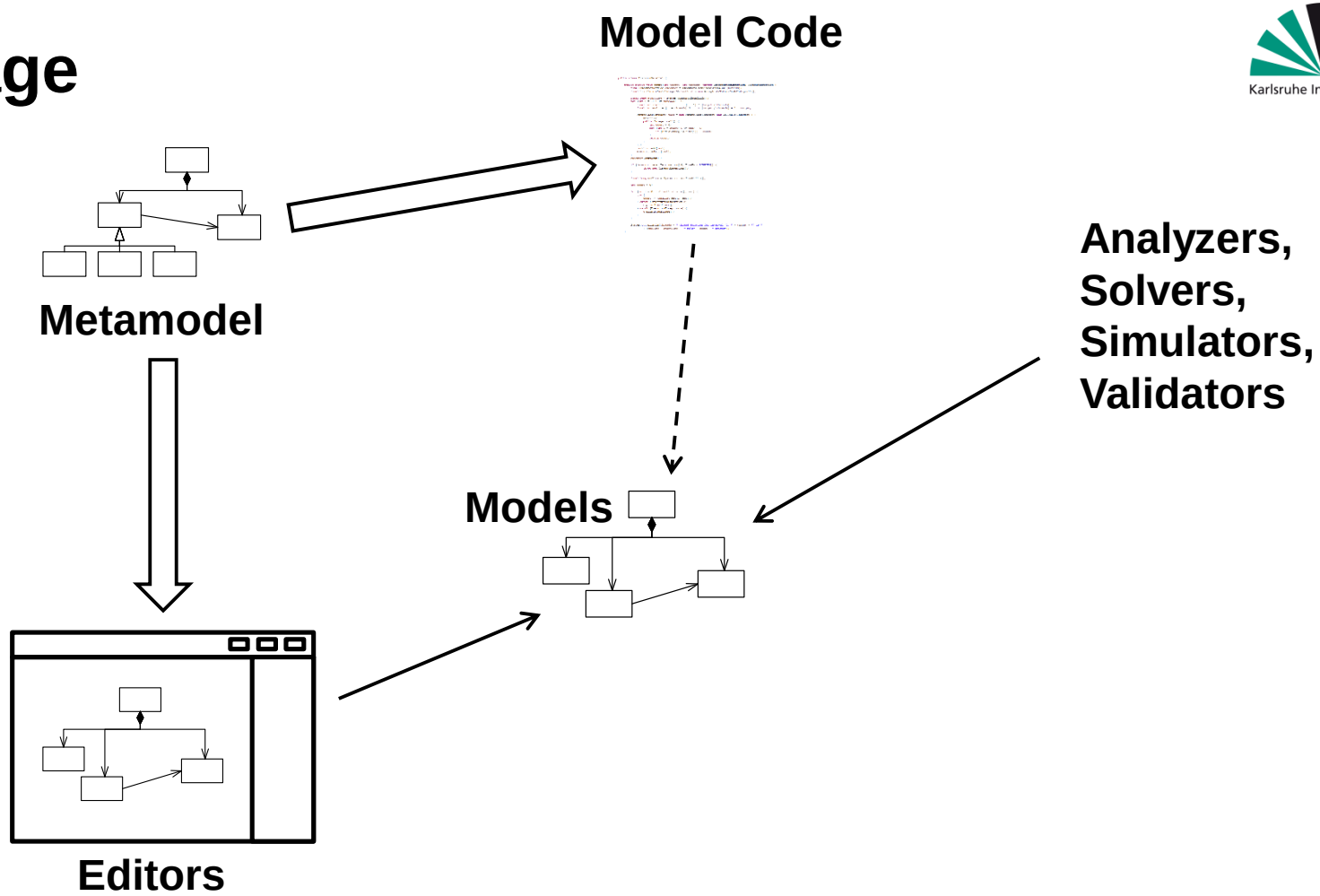


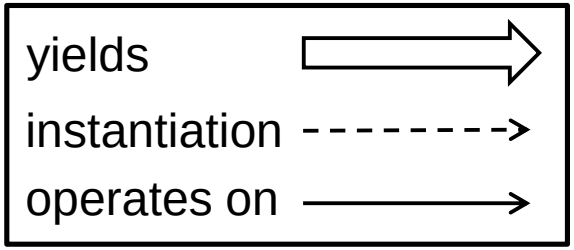
Usage


Model Code
Metamodel
Models
Editors

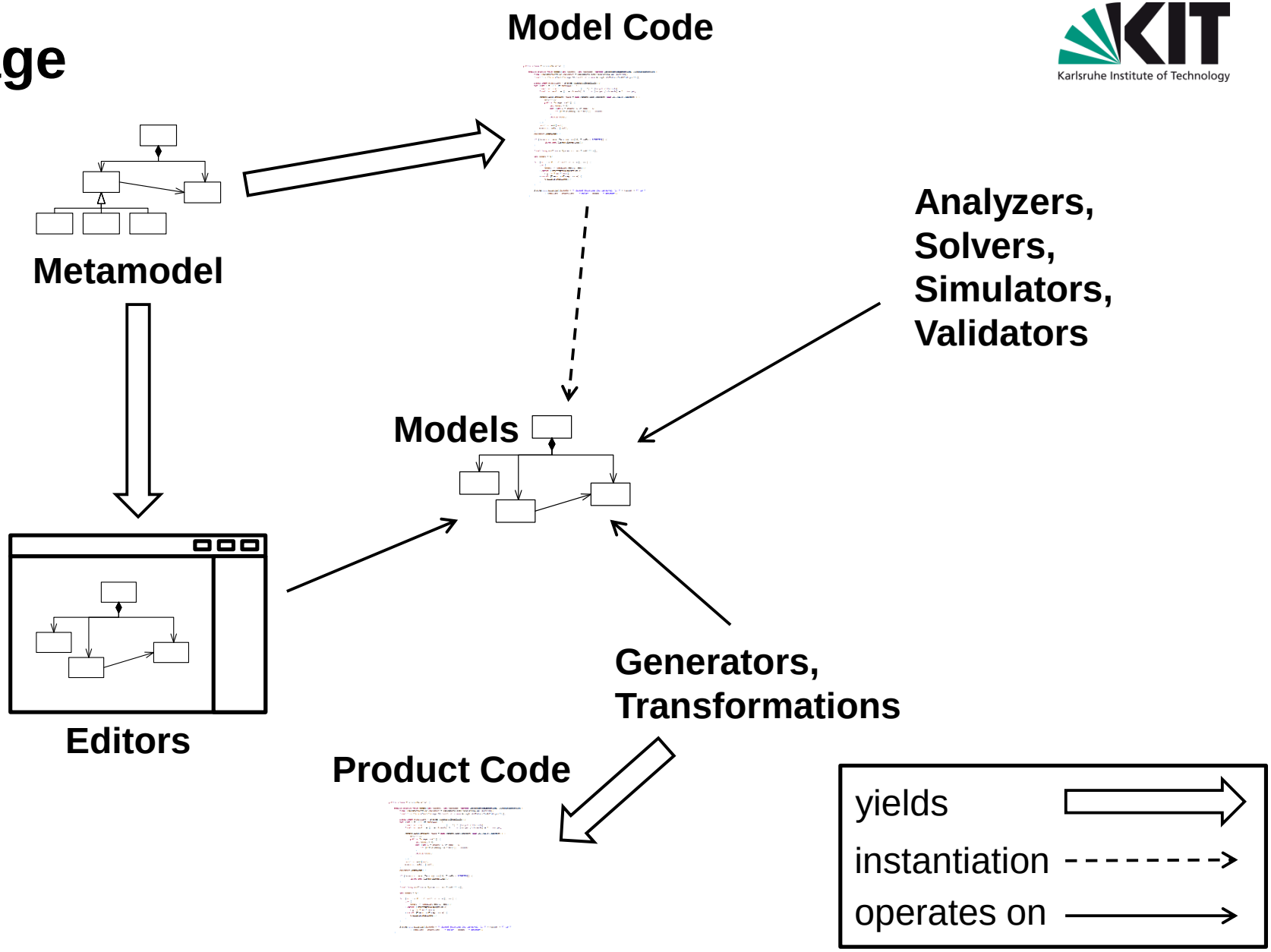

yields
instantiation
operates on

Usage

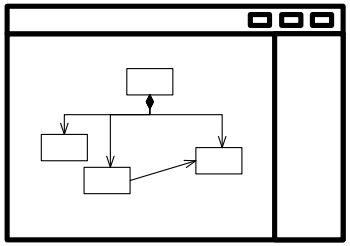

Model Code
Metamodel
Analyzers,
Solvers,
Simulators,
Validators
Models
Editors


yields
instantiation
operates on

Usage

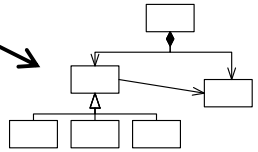

Model Code
Metamodel
Analyzers,
Solvers,
Simulators,
Validators
Models
Editors
Generators,
Transformations
Product Code
yields
instantiation
operates on

Metamodel-centric Software Systems



Editors

Analyzers, Solvers, Simulators, Validators



Metamodel

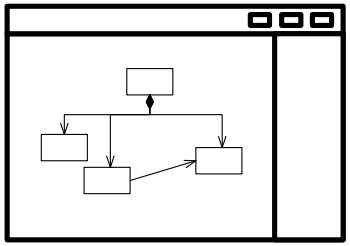
Product Code

Model Code

Generators, Transformations

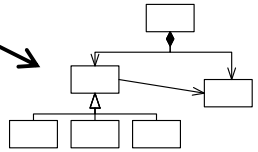
depends on ⟶

Metamodel-centric Software Systems



Editors

Analyzers,
Solvers,
Simulators,
Validators



Metamodel

Models

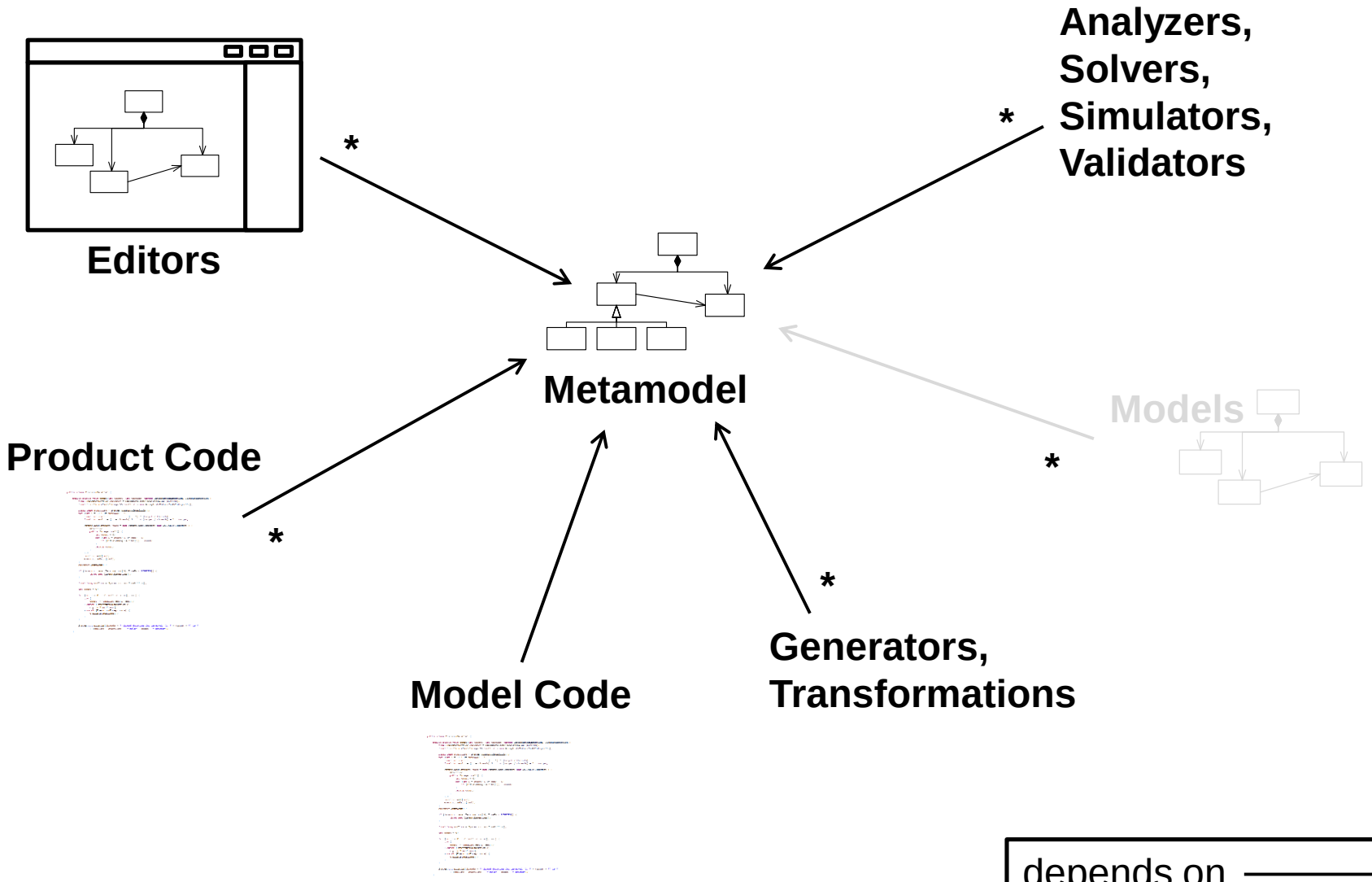
Product Code

Model Code

Generators,
Transformations

depends on ⟶

Metamodel-centric Software Systems

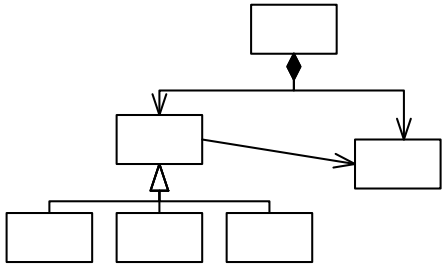


Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements

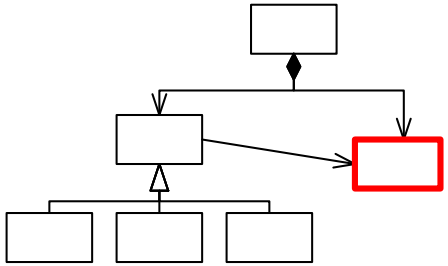
Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements
- Evolution types
  - Modification
  - Extension



Metamodel Evolution

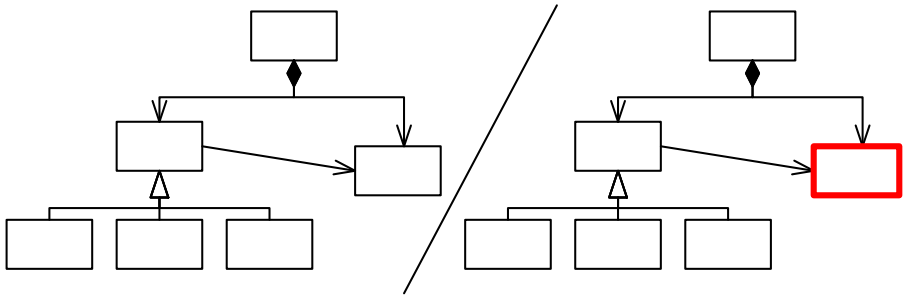
- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements
- Evolution types
  - Modification
    - Intrusive
  - Extension



Software Design and Quality Group
Institute for Program Structures and Data Organization

Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements
- Evolution types
  - Modification
    - Intrusive
    - Branch
  - Extension



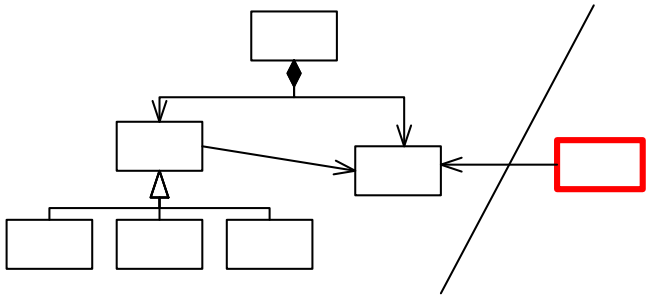
Software Design and Quality Group
Institute for Program Structures and Data Organization

Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements
- Evolution types
  - Modification
    - Intrusive
    - Branch
  - Extension
    - Intrusive
    - Branch

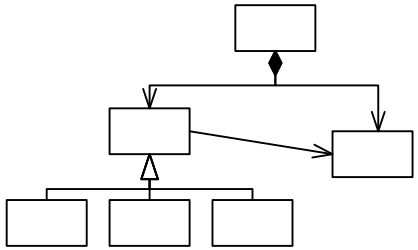
Metamodel Evolution

- Reasons
  - New feature
  - Feature changes
  - Bug fixing, improvements
- Evolution types
  - Modification
    - Intrusive
    - Branch
  - Extension
    - Intrusive
    - Branch
    - External



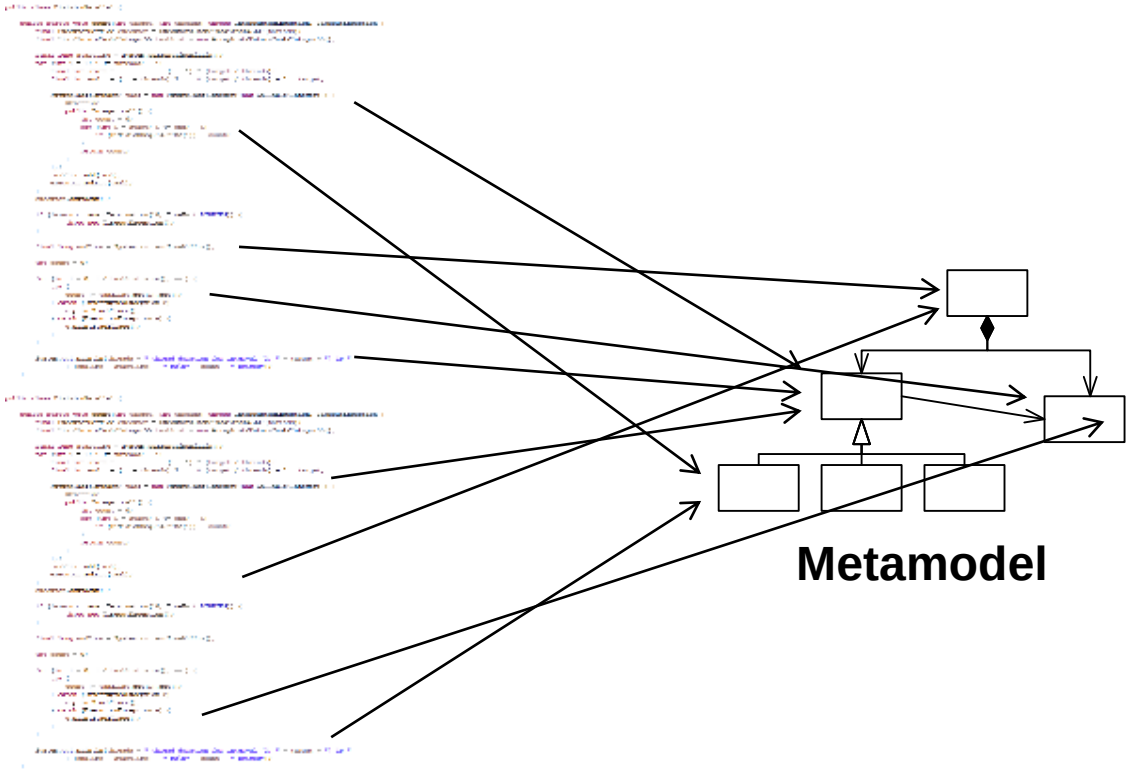
Challenges: Tight Coupling

Dependent Software



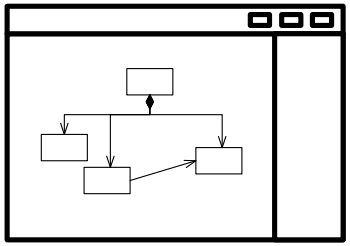
Metamodel

Challenges: Tight Coupling



Dependent Software

Challenge: Much Dependent Software

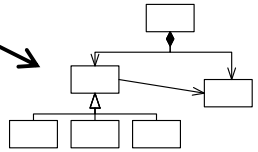


Editors

*

Analyzers,
Solvers,
Simulators,
Validators

*



Metamodel

Product Code

*

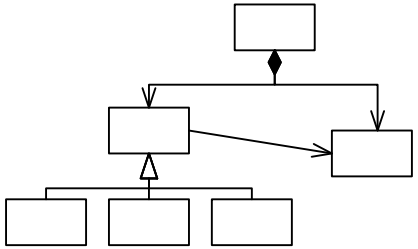
*

Generators,
Transformations

Model Code

depends on ⟶

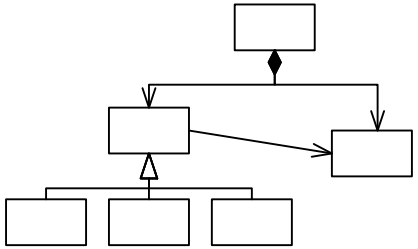
Challenge: Modifications in Generated Code



Metamodel

Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenge: Modifications in Generated Code

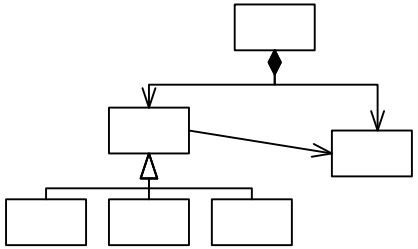


Metamodel

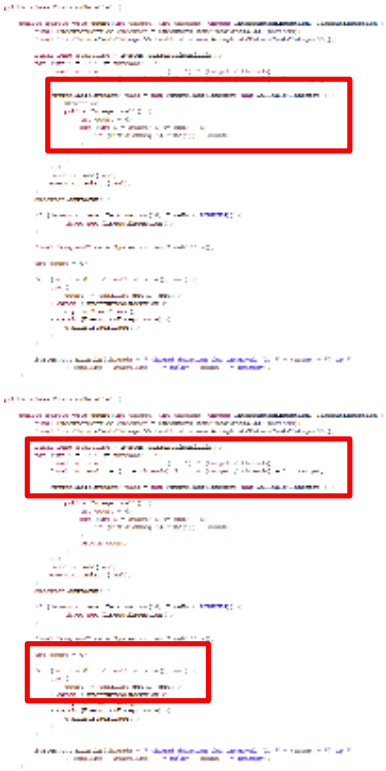
Generation

Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenge: Modifications in Generated Code

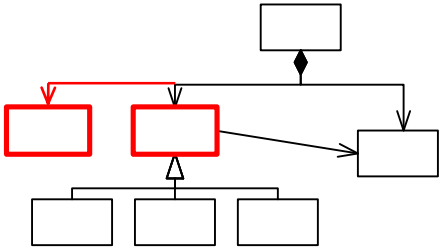


Metamodel

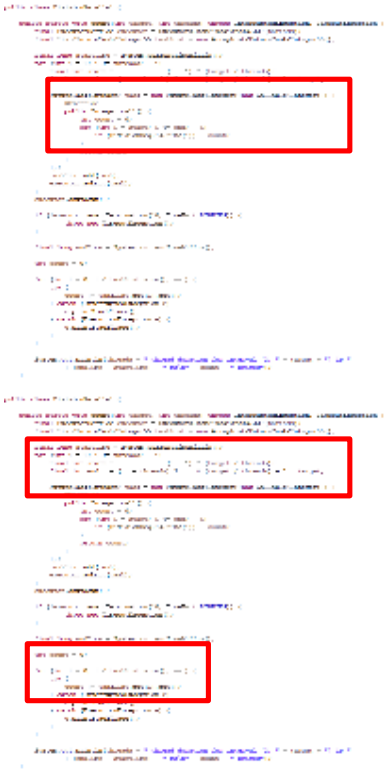


Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenge: Modifications in Generated Code

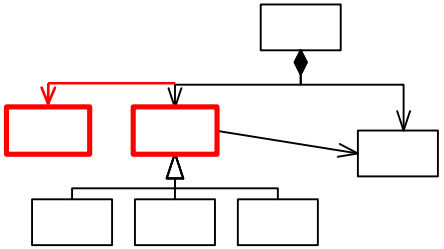


Metamodel



Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenge: Modifications in Generated Code

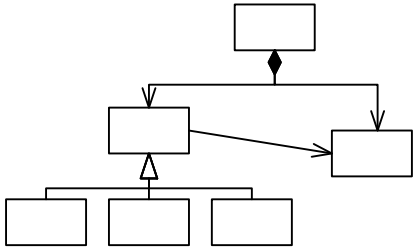


Metamodel

Generation

Software Design and Quality Group
Institute for Program Structures and Data Organization

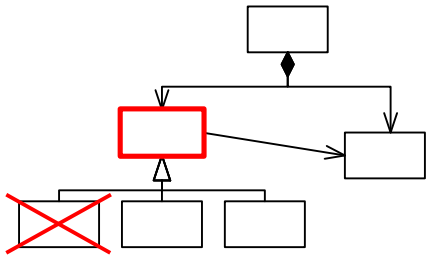
Challenge: Regeneration Remnants



Metamodel

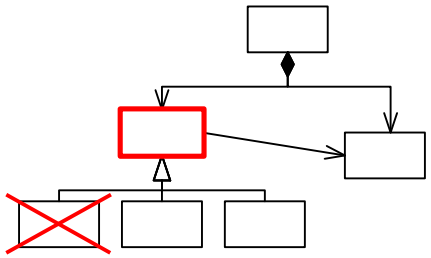
Generation

Challenge: Regeneration Remnants



Metamodel

Challenge: Regeneration Remnants

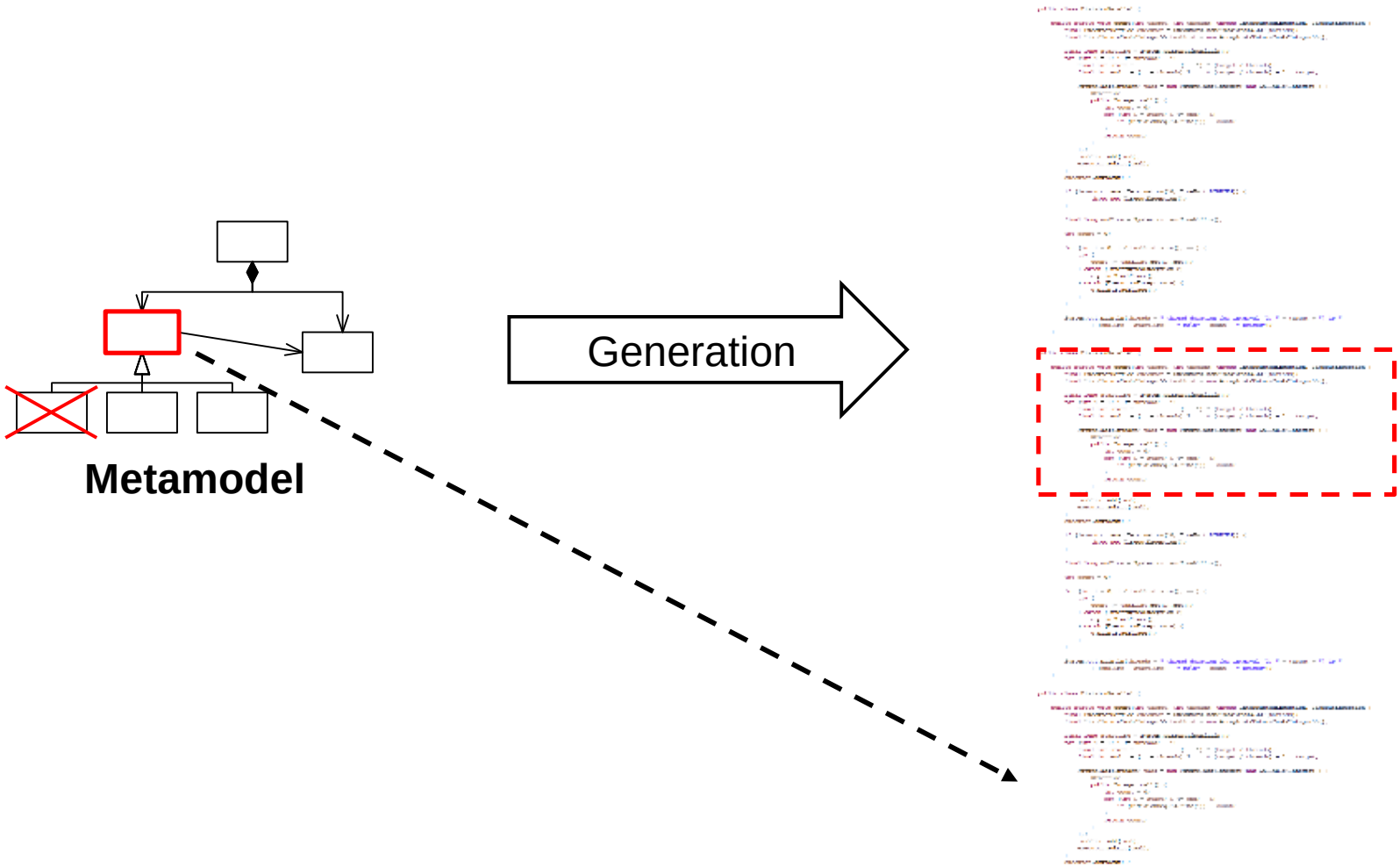


Metamodel

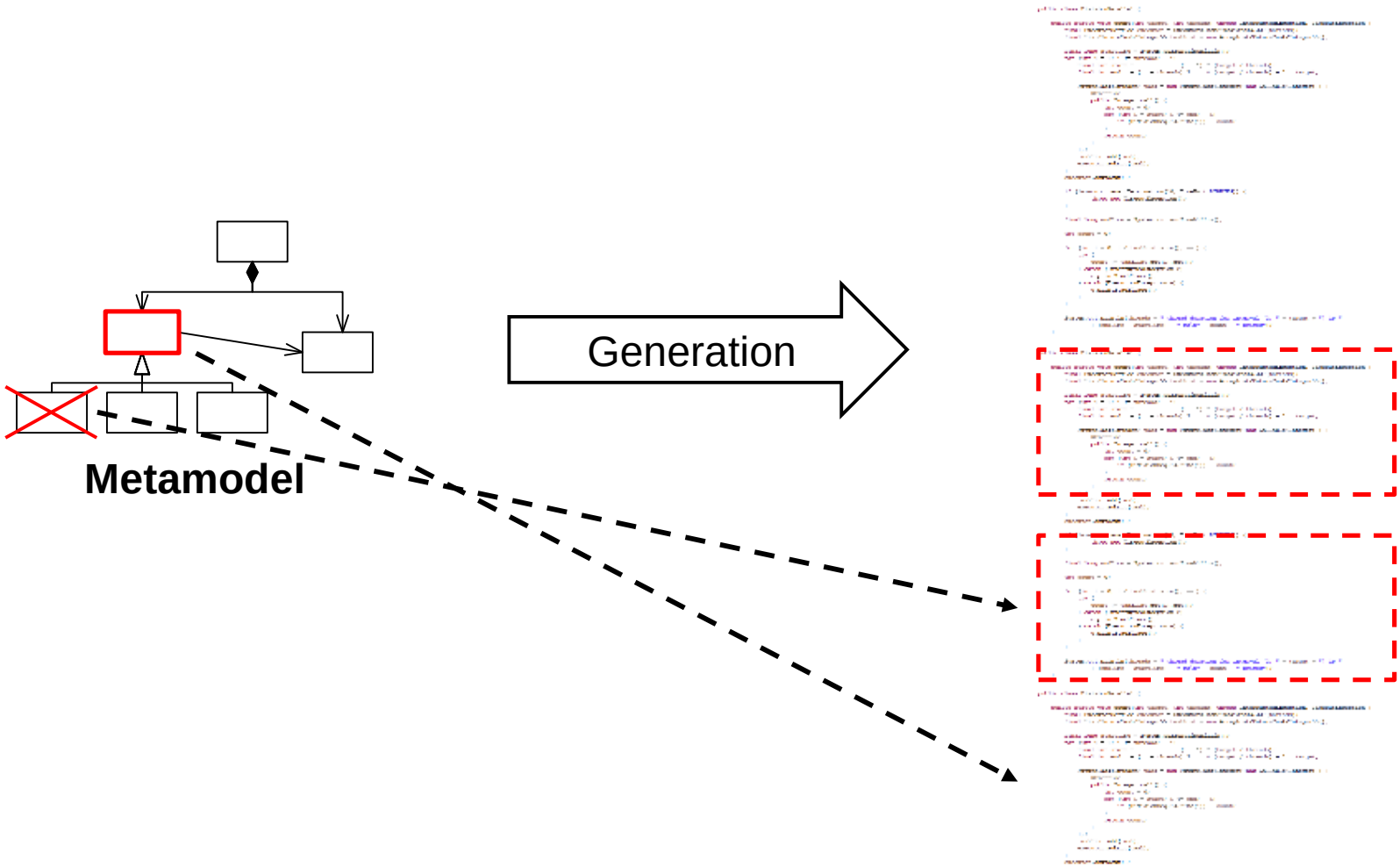
Generation

Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenge: Regeneration Remnants



Challenge: Regeneration Remnants



Challenges

- Tight coupling
- Much dependent software

Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code

Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise

Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise

} (also in design)

Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise

(also in design)

Meta-model specific

Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise
- Structural degradation
- Loss of knowledge

(also in design)

Meta-model specific

Software Design and Quality Group
Institute for Program Structures and Data Organization

Challenges

- Tight coupling
- Much dependent software
- Regeneration remnants
- Modifications in generated code
- Tooling compromise
- Extensibility compromise
- Structural degradation
- Loss of knowledge
- …?

(also in design)

Meta-model specific

Further challenges? Challenges from similar domains?

Software Design and Quality Group
Institute for Program Structures and Data Organization

Solutions: Prescriptive

1. Good initial design
2. Proper assessment of requirements
3. Good documentation

Solutions: Prescriptive

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics

Solutions: Prescriptive

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. Modularity
8. Explicit structure / reference structures

Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration

Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions

Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
16. …?

Solutions: Technical

9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
16. …?

Further solutions?

Transfer from other domains? (e.g. database schema evolution)

Validation

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. **Modularity**
8. **Explicit structure / reference structures**
9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. **External extensions**
13. **Explicit inter-package/module dependencies**
14. Synchronization
15. Visibility

Validation

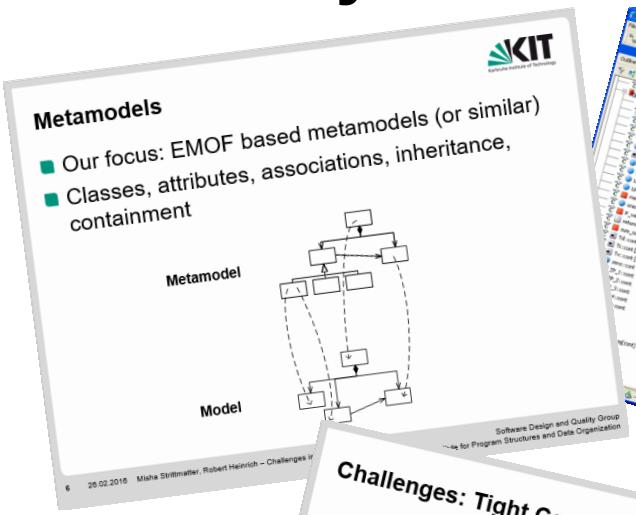
1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. **Modularity**
8. **Explicit structure / reference structures**
9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. **External extensions**
13. **Explicit inter-package/module dependencies**
14. Synchronization
15. Visibility

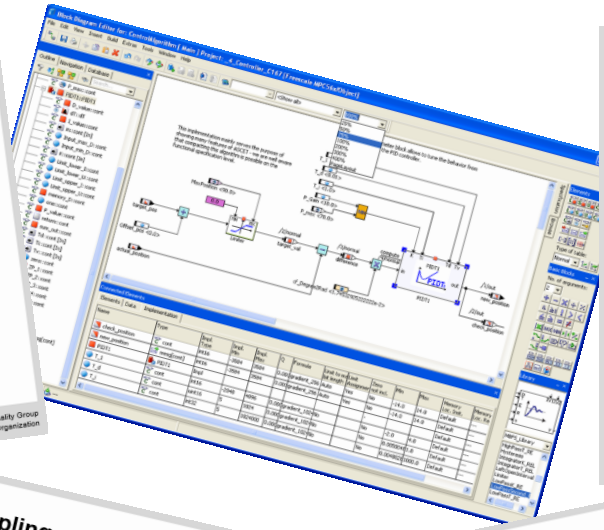
Validation approaches?

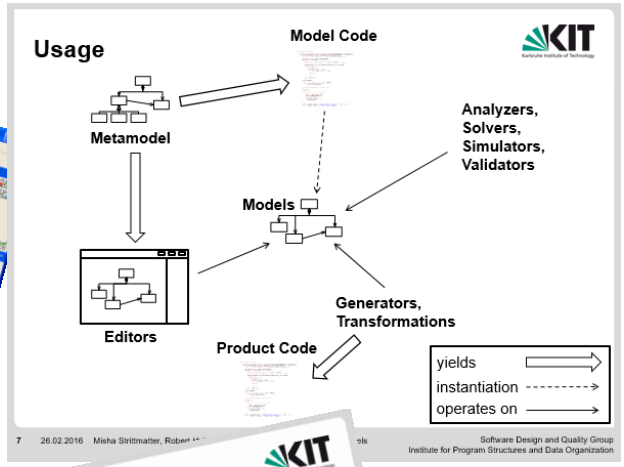
Questions

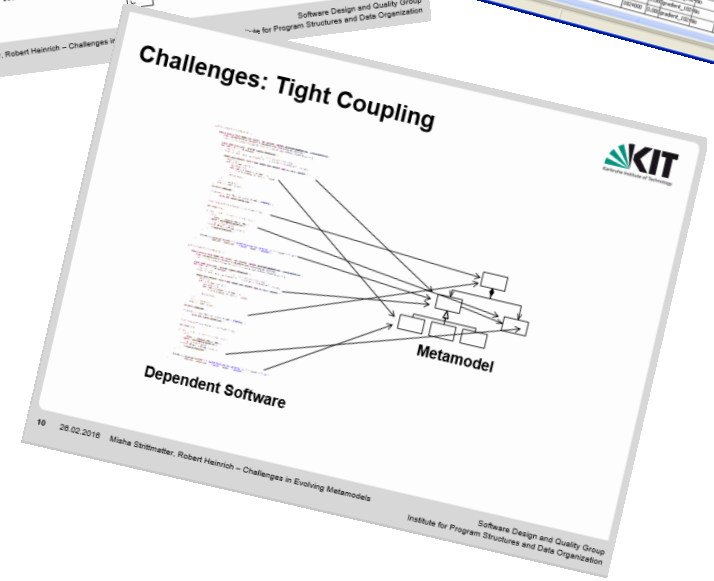
1. Are there further solutions?
2. How can the solutions be validated?
3. Are there further challenges?

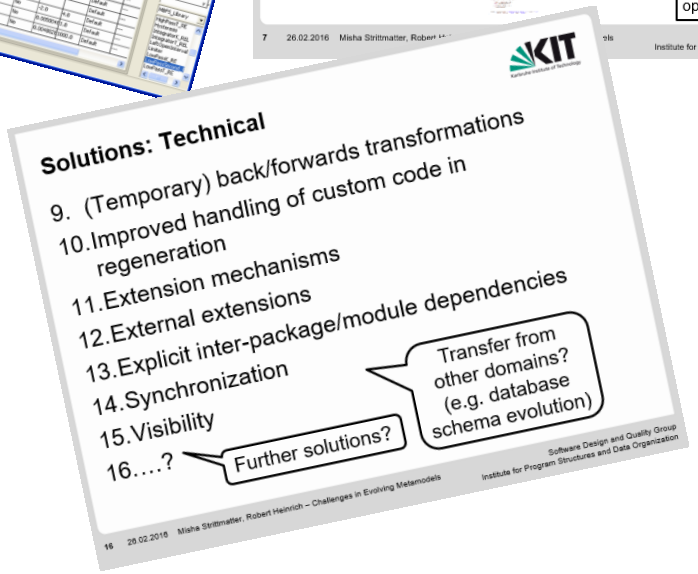
Summary


Metamodels
Our focus: EMOF based metamodels (or similar)
Classes, attributes, associations, inheritance, containment
Metamodel
Model




Usage
Model Code
Metamodel
Analyzers, Solvers, Simulators, Validators
Models
Editors
Generators, Transformations
Product Code
yields
instantiation
operates on


Challenges: Tight Coupling
Metamodel
Dependent Software


Solutions: Technical
9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility
16. ...?
Transfer from other domains? (e.g. database schema evolution)
Further solutions?

BACKUP

Solutions

1. Good initial design
2. Proper assessment of requirements
3. Good documentation
4. Conventions
5. Best practices
6. Metrics
7. Modularity
8. Explicit structure / reference structures
9. (Temporary) back/forwards transformations
10. Improved handling of custom code in regeneration
11. Extension mechanisms
12. External extensions
13. Explicit inter-package/module dependencies
14. Synchronization
15. Visibility

Basic Modifications

- Add, remove
  - Class, attribute, containment, inheritance, association, package, enum
- Change
  - Name, type, cardinality, property, …
- Move (into other package)
  - Class, package, enum

State of the Art

- Evolution of metamodels
- Co-evolution of models and metamodels
- Change impact prediction
- Rule-based evolution support tool

References

[1] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. 2013. What Industry Needs from Architectural Languages: A Survey. IEEE Trans. Softw. Eng. 39, 6 (June 2013), 869-891.