Karlsruhe Institute of Technology

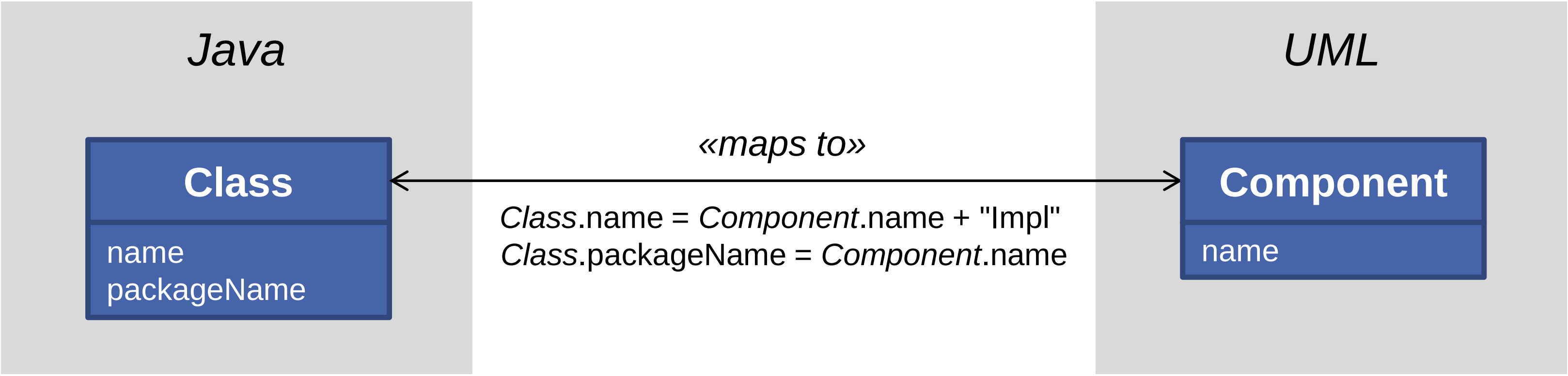
Commonalities for Preserving Consistency of Multiple Models

Heiko Klare, Joshua Gleitze

VoSE Workshop @ MODELS 2019, 15.09.2019

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

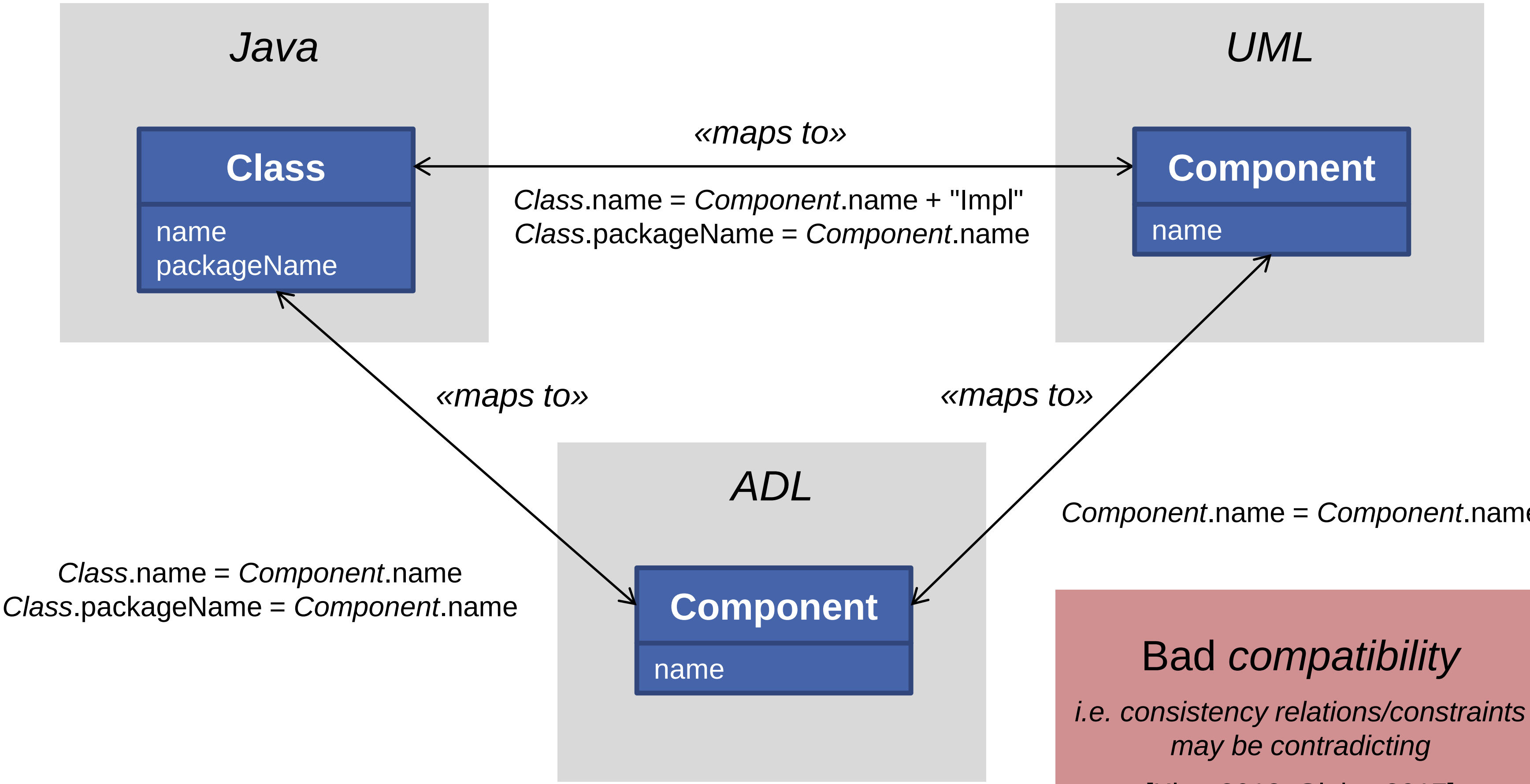
A Simple Consistency Scenario



A consistency-preserving transformation:

```
relation Class2ADLComponent {
   componentName : String;
   domain java class : Class {
     name = componentName + "Impl"
     packageName = componentName
   }
   domain uml component : Component {
      name = componentName
   }
}
```

Multi-Model Consistency: Dense Graphs



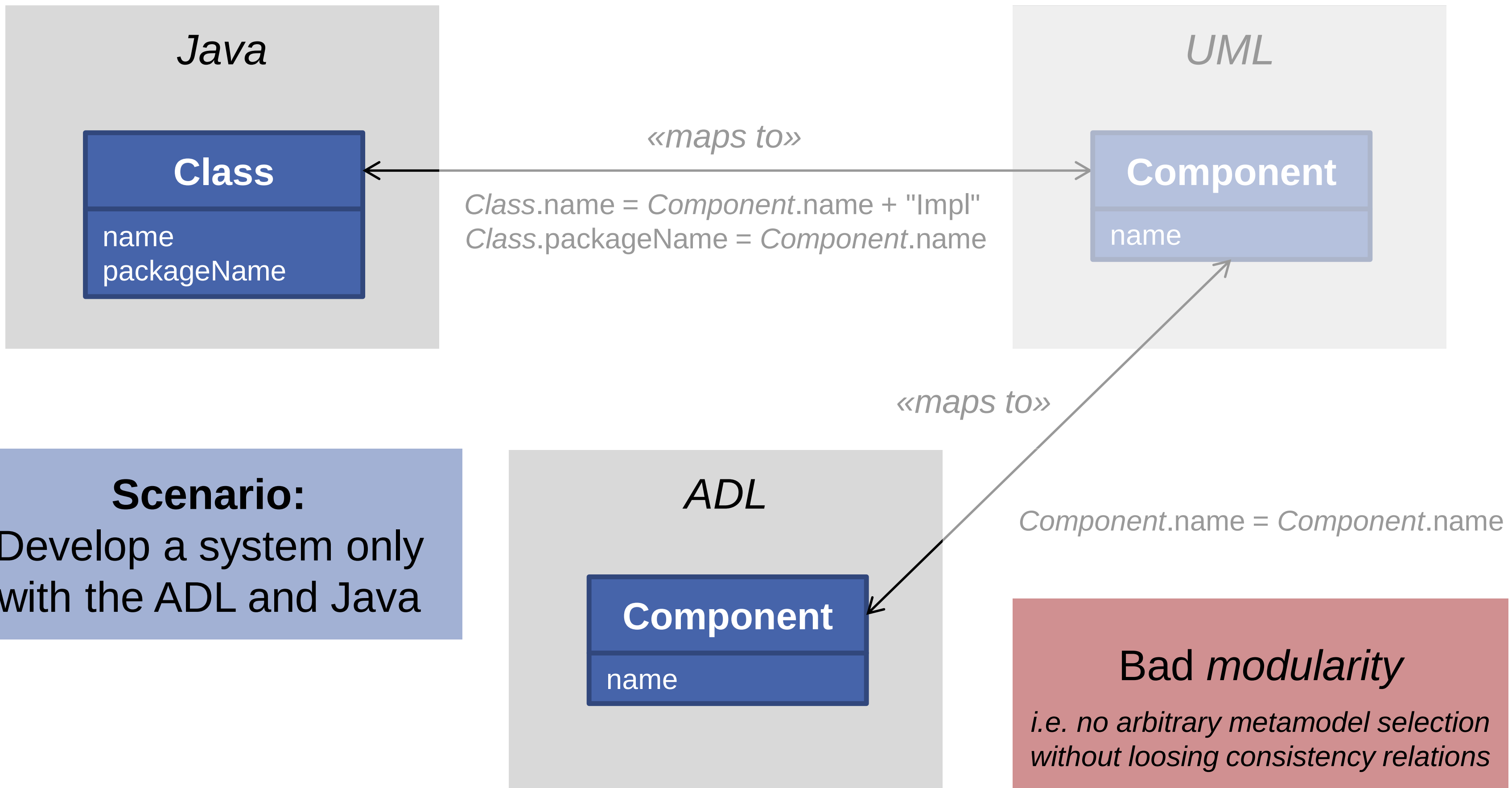
Bad *compatibility*

i.e. consistency relations/constraints may be contradicting

[Klare2018, Gleitze2017]

Option 1: Define relations between all pairs of metamodels

Multi-Model Consistency: Trees



Scenario:
Develop a system only with the ADL and Java

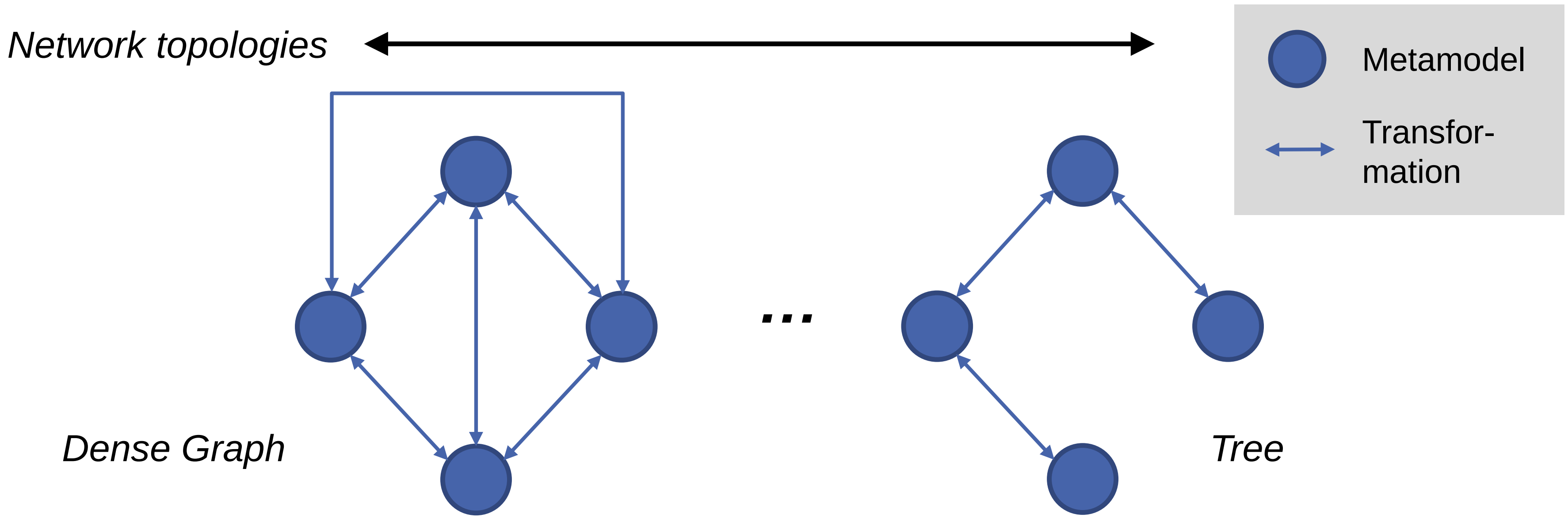
Bad *modularity*

i.e. no arbitrary metamodel selection without loosing consistency relations

[Klare2018, Gleitze2017]

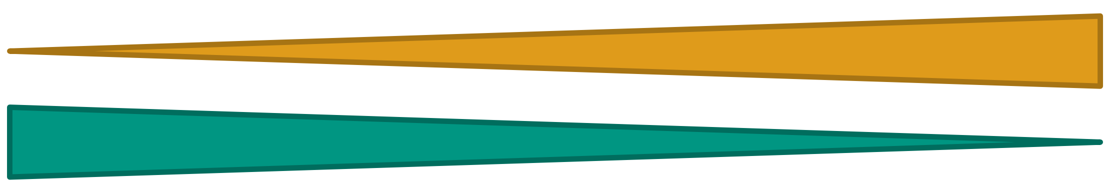
Option 2: Define relations between metamodels to form a tree

Trade-off: Compatibility vs. Modularity



Compatibility
(Relations/transformations do not contradict each other)

Maximize: $\theta(n^2)$ paths between each metamodel pair



Maximize: only one path between each metamodel pair

Modularity
(Possibility to select arbitrary metamodel subset without loosing consistency relations)

[Klare2018, Gleitze2017]

Contributions and Expected Benefits

Problem

Trade-off between *compatibility* and *modularity* in bidirectional transformation networks.

Idea

Resolve the trade-off between *compatibility* and *modularity* for consistency of multiple models by making common concepts of metamodels explicit.

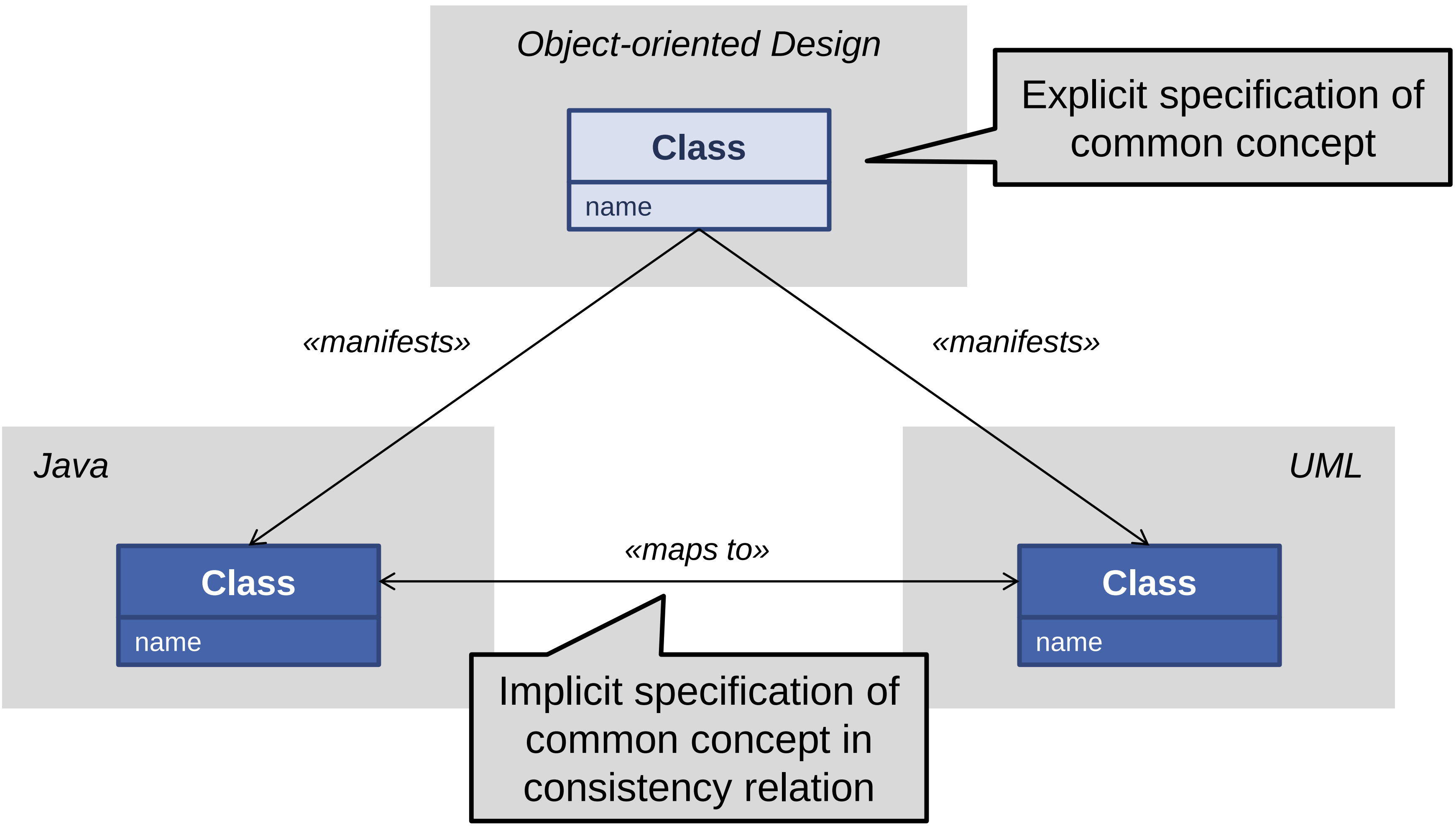
Contributions

- Commonalities Approach
- Commonalities Language
- Proof of Concept

Expected Benefits

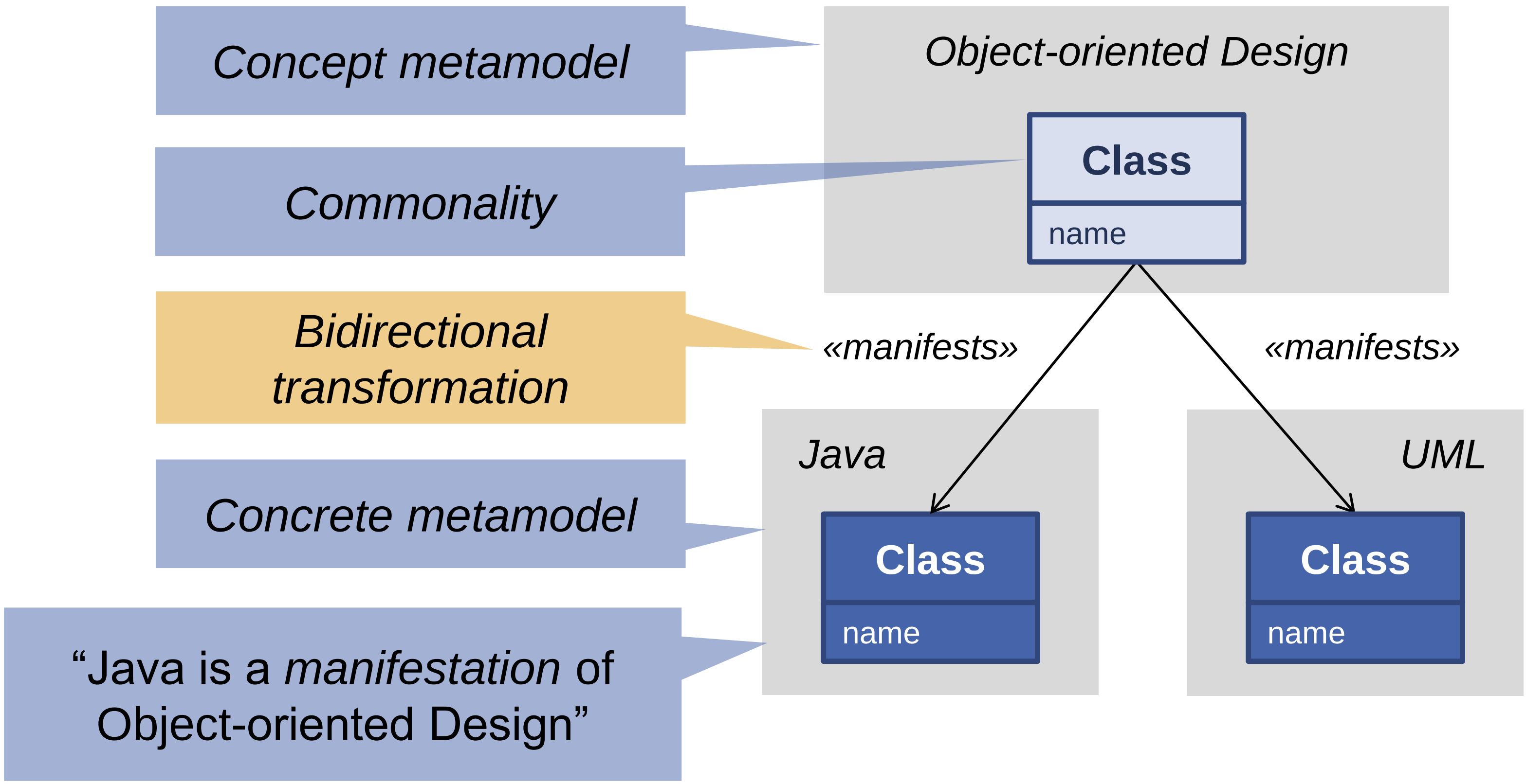
- Improved comprehensibility
- Reduced specification effort
- Improved compatibility and modularity

From Relations to Explicit Common Concepts



The Commonalities Approach

- Encode commonalities in metaclasses of a conceptual metamodel
- Represent common information in features of the conceptual metamodel



Heiko Klare, Joshua Gleitze

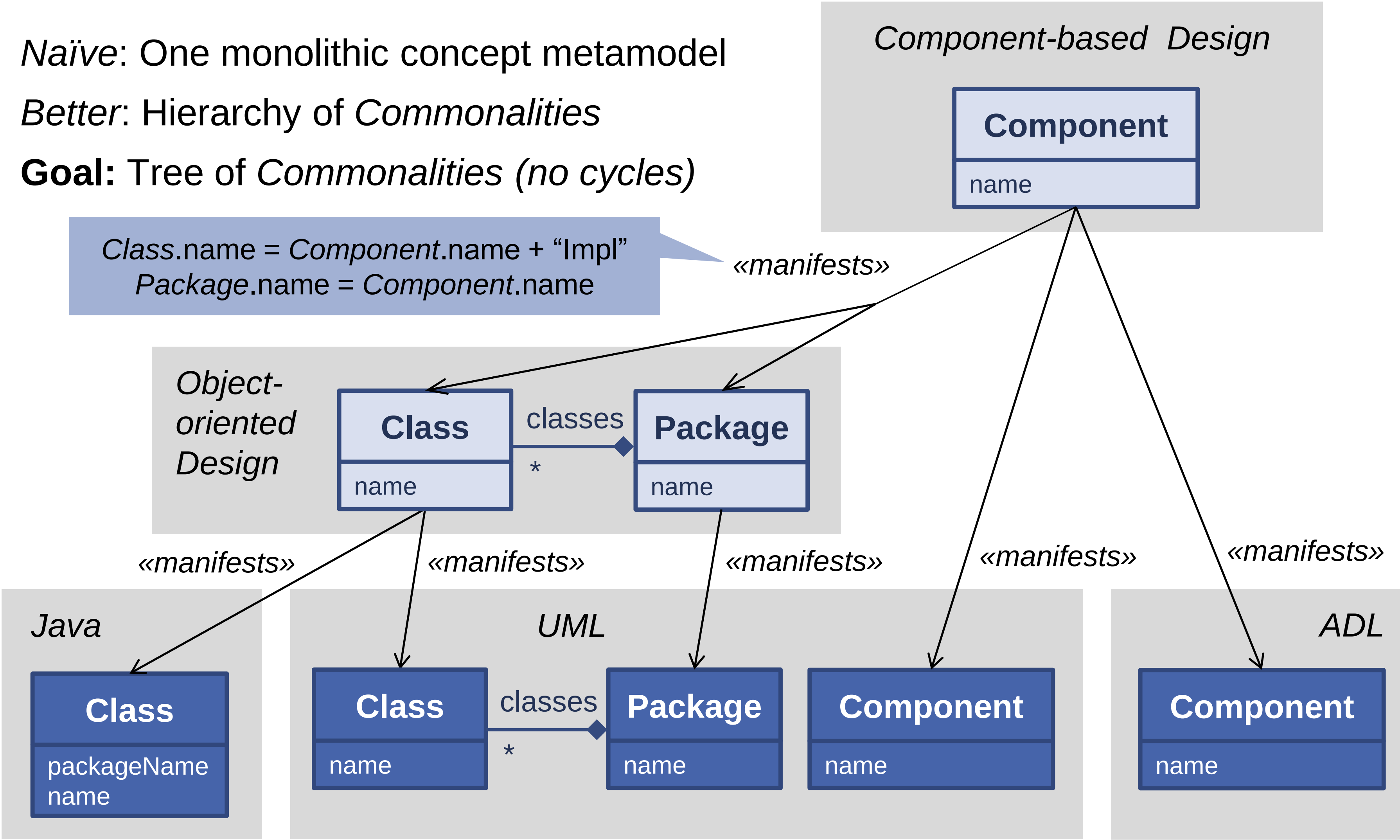
Software Design and Quality Group
Institute for Program Structures and Data Organization

Hierarchic Composition of Commonalities

Naïve: One monolithic concept metamodel

Better: Hierarchy of *Commonalities*

Goal: Tree of *Commonalities (no cycles)*

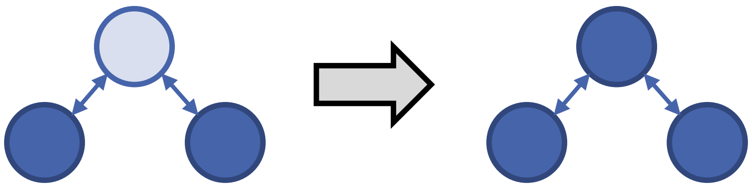


Design Decisions

Artifact Generation

(transparent to user)

Concept metamodels as additional metamodels



Concept metamodel → Metamodel
Commonality → Metaclass
Manifestation specification → Transformation

Alternative: Derive direct transformations between concrete metamodels

Benefits:

- Easy to achieve
- High expressiveness (*n*-ary relations)

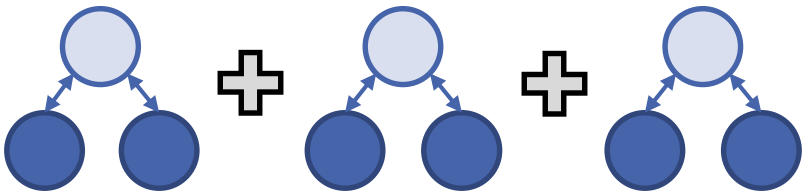
Drawback:

- Management of additional artifacts

Commonalities Specification

(visible to user)

Internal specification



Integrated definition of concept metamodels with manifestations

Decomposition dimension: Commonalities

Alternative: External specification (Decomposition dimension: Transformations)

Benefits:

- Easy to add Commonalities
- Improved locality / conciseness

Drawback:

- More difficult to add metamodels

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

Commonalities Language

```
concept Components

commonality Component {
   with UML:Component
   with ObjectOrientation:(Class in Package)

   has name {
      = UML:Component.name
      = ObjectOrientation:Package.name
      = suffix(ObjectOrientation:Class.name, "Impl")
   }

   has subcomponent referencing Components:Component {
      = UML:Component.packagedElement
      = ObjectOrientation:Package.subpackages
   }

}
```

Concept metamodel

Commonality

Manifestation (Concrete)

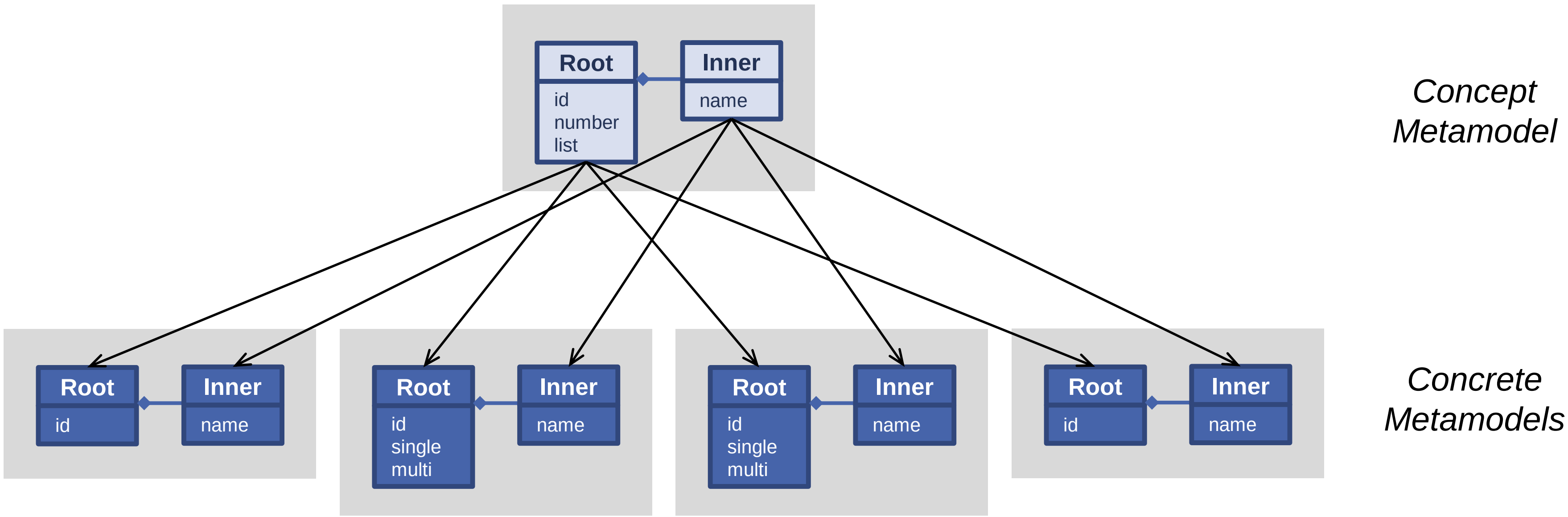
Manifestation (Concept)

Attribute Mapping

Reference Mapping

Proof of Concept

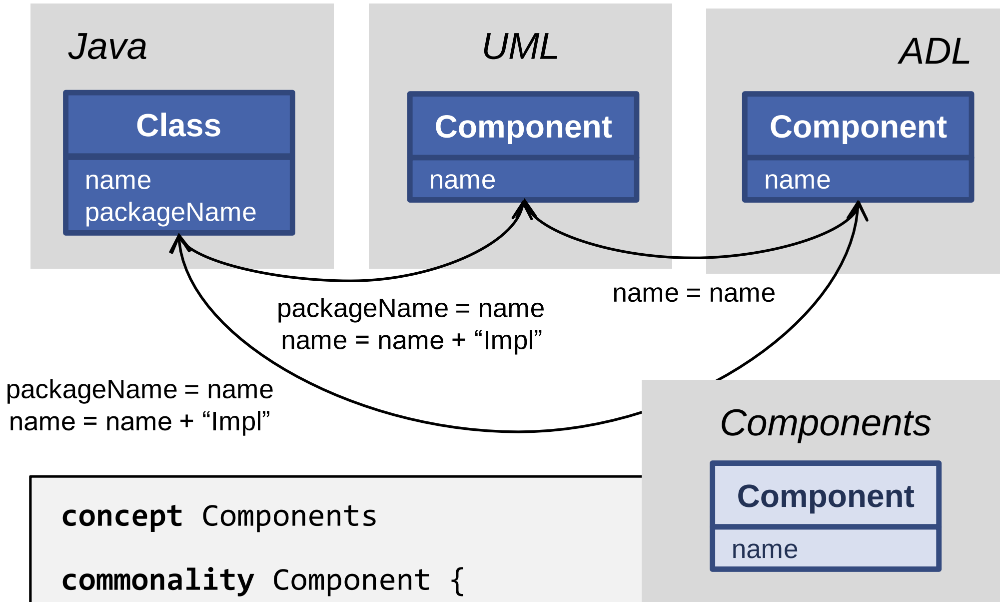
Case Study *(schematic)*



Feasibility

- Test cases performing all possible types of model modifications
- Correct propagation of all changes → indicator for functional correctness

Benefit: Comprehensibility



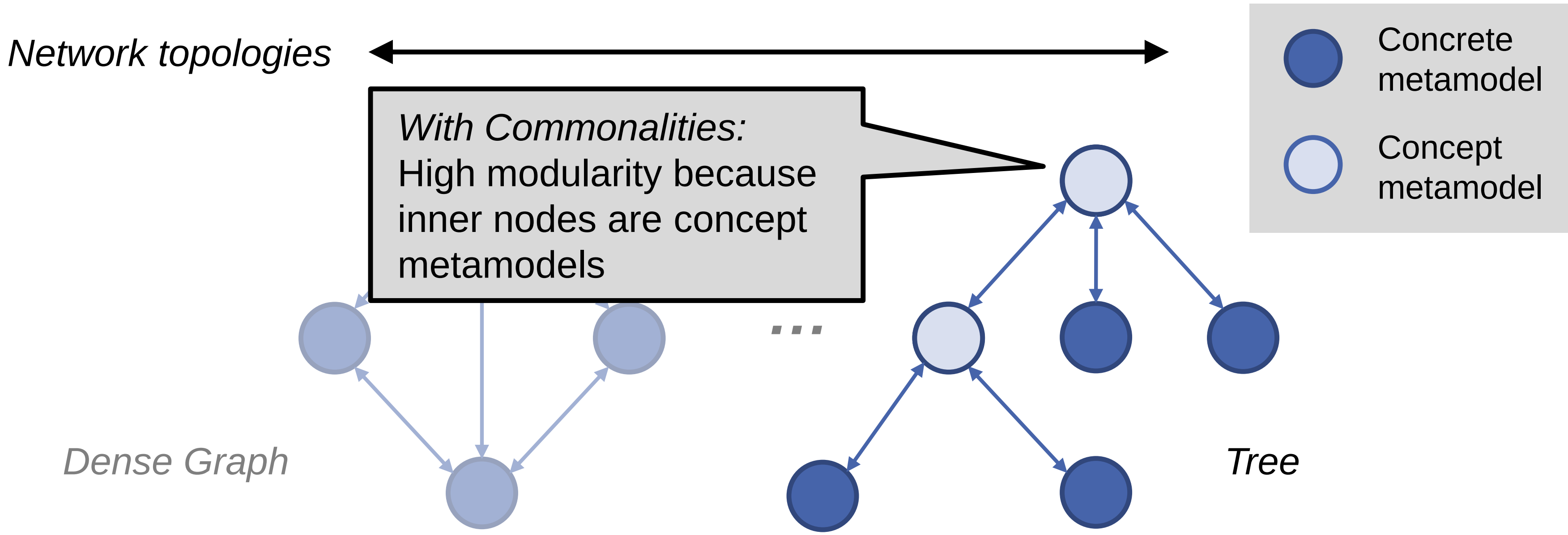

```
concept Components

commonality Component {
    with uml:Component
    with adl:Component
    with java:Class

    has name {
        = uml:Component.name
        = adl:Component.name
        = java:Class.packageName
        = suffix(java:Class.name, "Impl")
    }
}
```

```
relation UMLComponent2ADLComponent {
   componentName : String;
   domain uml component : Component {
     name = componentName;
   }
   domain adl component : Component {
      name = componentName;
   }
}
```

```
relation Class2ADLComponent {
   componentName : String;
   domain java class : Class {
     name = componentName + "Impl";
     packageName = componentName;
   }
   domain adl component : Component {
      name = componentName;
   }
}
```

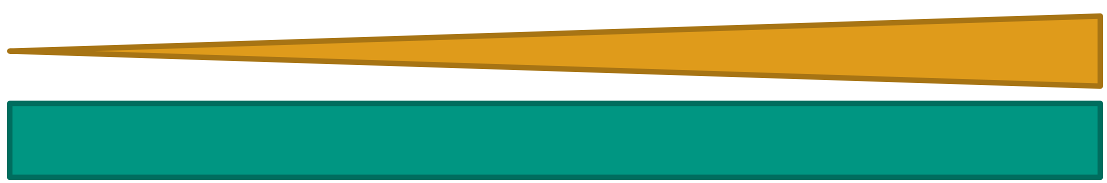
```
relation Class2UMLComponent {
   componentName : String;
   domain java class : Class {
     name = componentName + "Impl";
     packageName = componentName;
   }
   domain uml component : Component {
      name = componentName;
   }
}
```

Benefit: Compatibility and Modularity



Compatibility
(Relations/transformations do not contradict each other)

Maximize: only one path between each metamodel pair



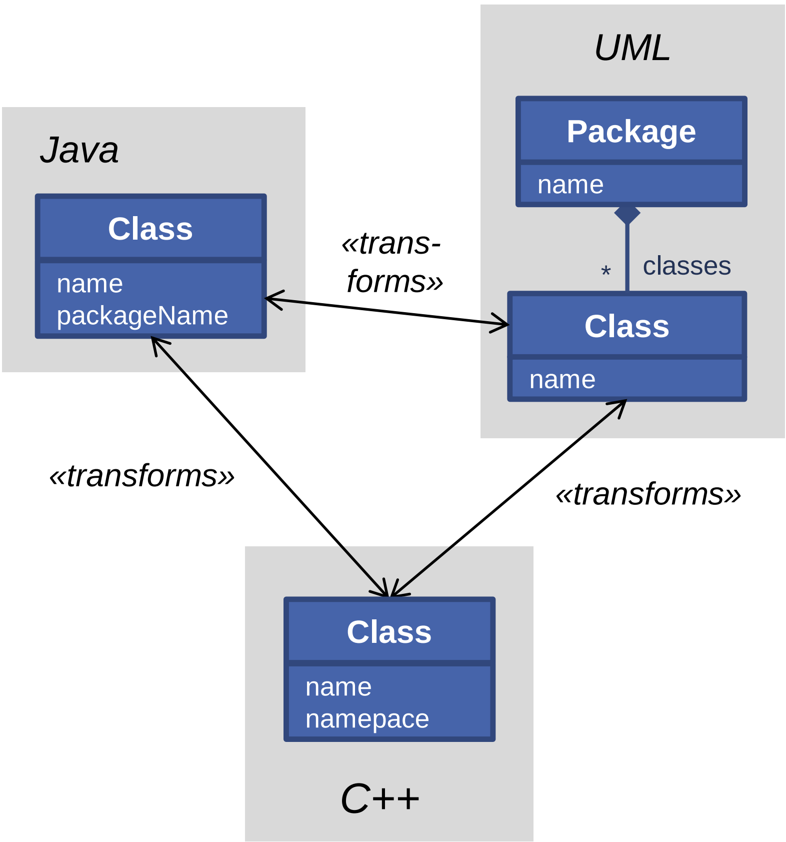
Maximize: $\theta(n^2)$ paths between each metamodel pair

Modularity
(Possibility to select arbitrary metamodel subset without loosing consistency relations)

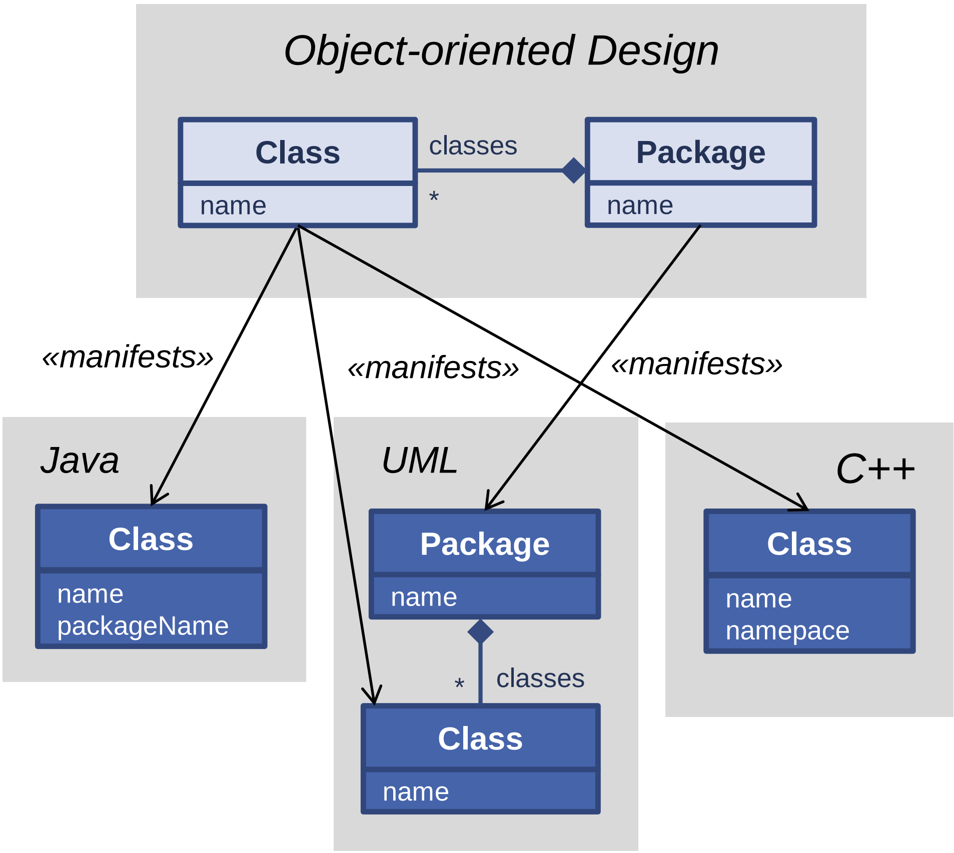
[Klare2018, Gleitze2017]

Benefit: Specification Effort

Ordinary Transformations



Commonalities

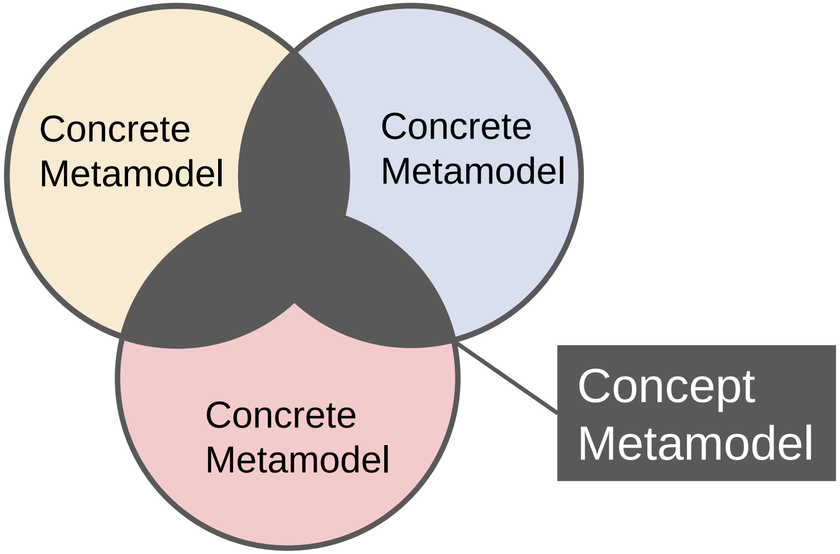


Number of relations grows linearly with *Commonalities* but quadratically with transformations

Comparison with the SUM(M) Approach

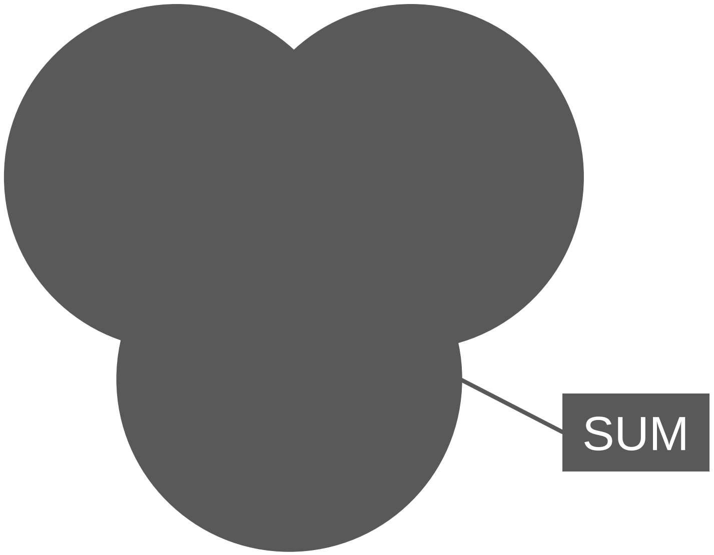
Commonalities Approach

Concept metamodel is union of pairwise intersections of concepts



SUM(M) Approach

SUM is union of concepts of all metamodels



Heiko Klare, Joshua Gleitze

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

Related Work

Commonalities Approaches

Practical approaches

- Sophisticated commonalities language [Gleitze2017]
- Role-oriented SUM [Werner2018]
- Domain-specific: DUALLy [Malavolta2010, Eramo2012]

Theoretic considerations

- Multiary Delta Lenses [Diskin2018]
- Commonalities for *n*-ary constraints [Stünkel2018]

Multidirectional Transformations and Networks of Bidirectional Transformations

- Dagstuhl Seminar [Cleve2019]
- Constraint decomposition problems [Stevens2017]
- Language-specific: QVT-R [Macedo2014], TGG [Trollmann2016]

Conclusion and Future Work

Goal

Resolve the trade-off between *compatibility* and *modularity* for multi-model consistency.

Contributions

Commonalities Approach

- *Concept metamodels* of *Commonalities*
- Manifestation relations
- Hierarchic composition of Commonalities

Commonalities Language

- Design options
  - Artifact generation
  - Commonalities specification
- Proof of concept implementation

Expected Benefits

General	Comprehensibility ↑
Multi-model case	Effort ↓ Errors ↓ Modularity ↑

Future Work

- Extend language capabilities
- Evaluate benefits
  - *Applicability*: case study
  - *Comprehensibility*: experiment
- Validate practicality of hierarchic composition

Bib: Commonalities

[Gleitze2017] J. Gleitze, “A Declarative Language for Preserving Consistency of Multiple Models,” Bachelor’s Thesis, Karlsruhe Institute of Technology (KIT), 2017.

[Diskin2018] Z. Diskin, H. König, and M. Lawford, “Multiple Model Synchronization with Multiary Delta Lenses,” in *Fundamental Approaches to Software Engineering*, Springer International Publishing, 2018, pp. 21–37.

[Stünkel2018] P. Stünkel, H. König, Y. Lamo, and A. Rutle, “Multimodel Correspondence Through Inter-model Constraints,” in *Conference Companion of the 2Nd International Conference on Art, Science, and Engineering of Programming*, ser. Programming’18 Companion, ACM, 2018, pp. 9–17.

[Eramo2012] R. Eramo, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio, “A model-driven approach to automate the propagation of changes among Architecture Description Languages”, *Software and Systems Modeling*, vol. 11, pp. 29–53, 1 2012.

[Malavolta2010] I. Malavolta, H. Muccini, P. Pelliccione, and D. A. Tamburri, “Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies”, *IEEE Transactions of Software Engineering*, vol. 36, no. 1, pp. 119–140, 2010.

[Werner2018] C. Werner and U. Assmann, “Model Synchronization with the Role-oriented Single Underlying Model,” in *Proceedings of MODELS 2018 Workshops, co-Located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages*, CEUR-WS.org, 2018, pp. 62–71

Bib: Multidirectional Transformations/Networks

[Klare2018] H. Klare, “Multi-model Consistency Preservation,” in *21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS): Companion Proceedings*, 2018, pp. 156–161.

[Cleve2019] A. Cleve, E. Kindler, P. Stevens, and V. Zaytsev, “Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491),” *Dagstuhl Reports*, vol. 8, no. 12, pp. 1–48, 2019.

[Macedo2014] N. Macedo, A. Cunha, and H. Pacheco, “Towards a frame- work for multi-directional model transformations,” in *3rd International Workshop on Bidirectional Transformations - BX*, vol. 1133, CEUR-WS.org, 2014.

[Stevens2017] P. Stevens, “Bidirectional Transformations in the Large,” in *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MOD-ELS)*, 2017, pp. 1–11.

[Trollmann2016] F. Trollmann and S. Albayrak, “Extending Model Synchronization Results from Triple Graph Grammars to Multiple Models,” in *Theory and Practice of Model Transformations*, Springer International Publishing, 2016, pp. 91–106.

Bib: Other Related Topics

[Lúcio2013] L. Lúcio, S. Mustafiz, J. Denil, H. Vangheluwe, and M. Jukss, “FTG+PM: An Integrated Framework for Investigating Model Transformation Chains,” in *SDL 2013: Model-Driven Dependability Engineering*, Springer Berlin Heidelberg, 2013, pp. 182–202.

[Eramo2008] R. Eramo, A. Pierantonio, J. R. Romero, and A. Valle- cillo, “Change Management in Multi-Viewpoint System Using ASP,” in *Enterprise Distributed Object Computing Conference Workshops*, 2008, pp. 433–440.

[Atkinson 2010] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. “Orthographic Software Modeling: A Practical Approach to View-Based Development”. In: *Proceedings of the 3rd and 4th International Conferences on Evaluation of Novel Approaches to Software Engineering (ENASE 2008/2009)*. Vol. 69. Communications in Computer and Information Science. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 206 – 219

Bib: Vitruvius Approach

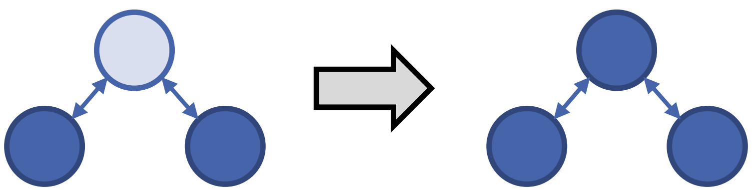
[Kramer 2013] Max E. Kramer, Erik Burger, and Michael Langhammer. “View-centric Engineering with Synchronized Heterogeneous Models”. In: *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling (VAO 2013)*. New York, NY, USA: ACM, 2013, 5:1–5:6.

[Kramer 2015] Max E. Kramer et al. *Realizing Change-Driven Consistency for Component Code, Architectural Models, and Contracts in Vitruvius*. Tech. rep. Karlsruhe: Karlsruhe Institute of Technology, Department of Informatics, 2015.

Design Decision: Artifact Generation

Artifact generation is transparent to user

Concept metamodels as additional metamodels

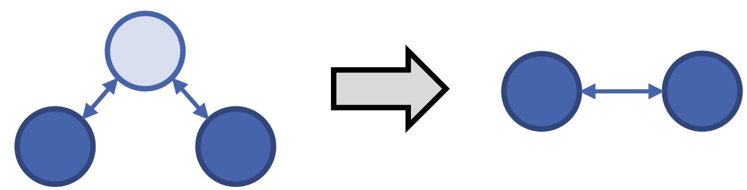


Concept metamodel → Metamodel
Commonality → Metaclass
Relation specification → Transformation

Benefits:

- Easy to achieve
- High expressiveness (*n*-ary relations)

Transformations between concrete metamodels



Indirect relations across concept metamodels → Transformations between pairs of concrete metamodels

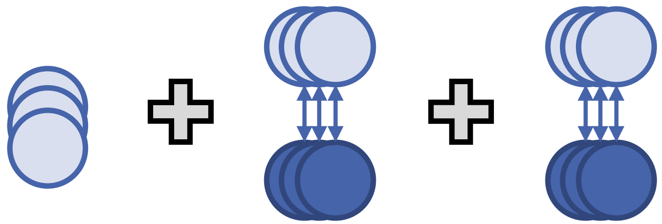
Benefits:

- No management of additional artifacts
- Easier to understand direct relations

Design Decision: Commonalities Specification

Commonalities specification is visible to user

External specification



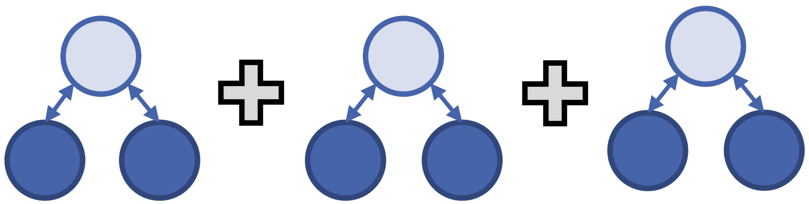
Independent definition of concept metamodels and transformations

Decomposition dimension: Transformations

Benefits:

- Easy to add concrete metamodels
- Reuse existing tooling

Internal specification



Integrated definition of concept metamodels with manifestations

Decomposition dimension: Commonalities

Benefits:

- Easy to add Commonalities
- Improved locality / conciseness

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

Operators in the Commonalities Language

- The Commonalities language can be extended by operators that allow bidirectional propagation of information, e.g. ***`in`*** and ***`suffix`***
- To have well-defined bidirectional transformations, operators must be
  - Correct
  - Hippocratic
  - Undoable