41. Data Integration of Tools by Role-Based Composition of Materials (RoleCore-Based Metamodel Composition on M2)

for Tool Interoperability on M1-Models and M0-Repositories

Prof. Dr. Uwe Aßmann
Mirko Seifert, Christian Wende, Thomas Kühn
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de
Version 19-1.2, 27.01.20

Obligatory Literature

- Deep Rolling and RoleCore:
  - Mirko Seifert, Christian Wende and Uwe Aßmann. Anticipating Unanticipated Tool Interoperability using Role Models. In Proceedings of the 1st Workshop on Model Driven Interoperability (MDI'2010) (co-located with MODELS 2010), 5th October 2010, Oslo, Norway
  - https://github.com/DevBoost/EMFText-Zoo/tree/master/BreedingStation/RoleCore
- Course "Design Patterns and Fra[illegible]e modeling)
- http://www.langems.org

emftext

Literature on Roles

T. Reenskaug, P. Wold, O. A. Lehne. Working with objects. Manning. The OOram Method. http://heim.ifi.uio.no/~trygver/documents/book11d.pdf

H. Allert, P. Dolog, W. Nejdl, W. Siberski, F. Steimann. Role-Oriented Models for Hypermedia Construction – Conceptual *Modelling for the Semantic Web*. citeseer.org.

N. Guarino, M. Carrara, and P. Giaretta. An ontology of meta-level categories. In Proceedings of the Fourth International Conference on Knowledge Representation and Reasoning, pages 270–280. Morgan Kaufmann, San Mateo, 1994.

F. Steimann. On the representation of roles in object-oriented and conceptual modelling. Data and Knowledge Engineering. 2000.

T. Reenskaug, P. Wold, O. A. Lehne. Working with objects. Manning. http://heim.ifi.uio.no/~trygver/documents/book11d.pdf

D. Riehle, T. Gross. Role Model Based Framework Design and Integration. OOPSLA 1998.

U. Aßmann, J. Henriksson, I. Savga, J. Johannes: Composition of Ontologies and Rule Sets. REASONING WEB Summer School, LNCS 4126

Christian Wende. Language Family Engineering. PhD thesis, Technische Universität Dresden, Fakultät Informatik, March 2012, http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-88985.

40.1 Different Factors Drive Business Models – How to Vary a Business Model Canvas

[BMG p 142]

The Role-Based Architectural Language SMAGs

- Role-based Architectural language: Smart Apps (SMAPPs) and Smart Application Grids (SMAGs) http://st.inf.tu-dresden.de/smags
  - Development by Christian Piechnick
- C. Piechnick, S. Richly, S. Götz, C. Wilke, U. Aßmann. Using Role-Based Composition to Support Unanticipated, Dynamic Adaptation – Smart Application Grids. Adaptive and Self-adaptive Systems and Applications (Adaptive 2012)

Works in the Last Years

- Henrik Lochmann. HybridMDSD: Multi-Domain Engineering with Model-Driven Software Development using Ontological Foundations. PhD thesis, Technische Universität Dresden, Fakultät Informatik, 2009, http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-27380
- Mirko Seifert. Designing Round-Trip Systems by Model Partitioning and Change Propagation. PhD thesis, Technische Universität Dresden, Fakultät Informatik, June 2011, http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-71098
- Konrad Voigt. Structural Graph-based Metamodel Matching. PhD thesis, Technische Universität Dresden, Fakultät Informatik, November 2011, http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-81671
- Jendrik Johannes. Component-Based Model-Driven Software Development. PhD thesis, Technische Universität Dresden, Fakultät Informatik, December 2010. http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-63986
  - www.reuseware.org
- Birgit Grammel. Automatic Generation of Trace Links in Model-driven Software Development. PhD thesis, Technische Universität Dresden, Fakultät Informatik, February 2014

41.1. Language Composition

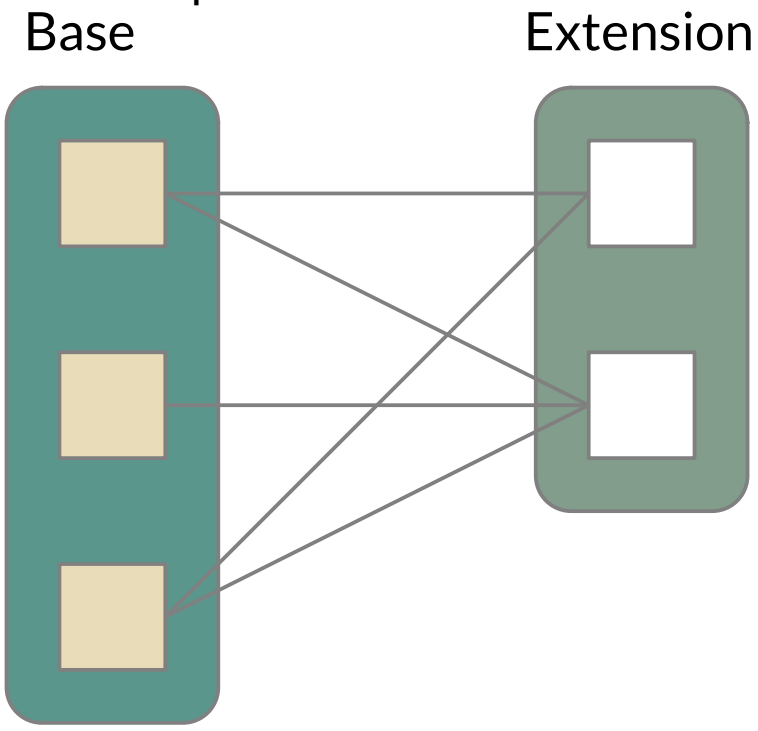
Language Composition

- We have learned in chapter “Metamodeling” that metamodels can be composed by a merge operator
  - Then metamodel-driven repositories can be generated
- So far, the integration was based on *merge of metamodel packages*, i.e., the metaclasses stayed as they are during composition
- In this chapter, we will merge metaclasses during composition by *role merge*
  - This achieves a much tighter integration of data for several tools (data sharing, Datenteilung)

Hypothesis: Language integration works well for Role-based Metaclasses, in particular for Material Metaclasses

Metamodel Mappings and the Composition of Languages

- **Problem**: Language mappings for transformation bridges should be modular
  - But the composition of two languages (and two metamodels) is difficult
- Examples:
  - Extension of a base language with a domain-specific extension
  - Design of a *language family* of related languages
  - Specification of a crosscut in the semantics
- Language Composition is traditionally done with declarative specifications
  - Composition of Attribute grammars
    - JastAdd, ELI, fnc-2, LISA, Silver
  - Composition of Natural Semantics
    - Typol, RML
  - Composition of Logic Specifications
    - Datalog, OWL



The Problem of Language Composition

Ideally, a language designer would like to build a language simply be reusing language definition modules...

This approach is common to component-based programming where components can be simply plug-ins.

This cannot be done now.

[Mernik, Wu, Bryant, 2004]

DSL development is hard.

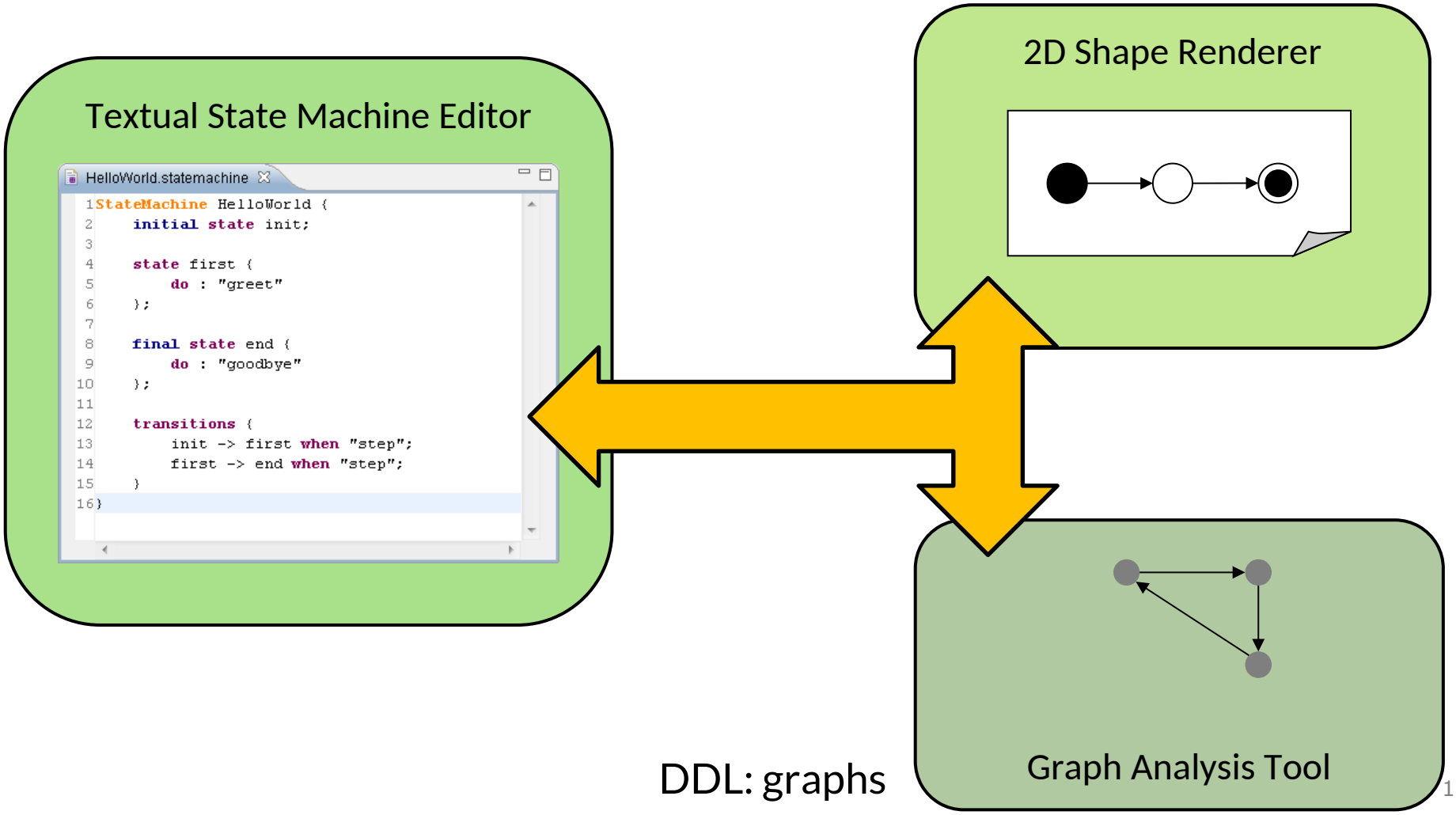
[Mernik, Heering, Sloane, 2005]

Motivational Example for Data Sharing in Tool Integration

- Tools may rely on different DDL, which represent *similar concepts*

DDL: state machines

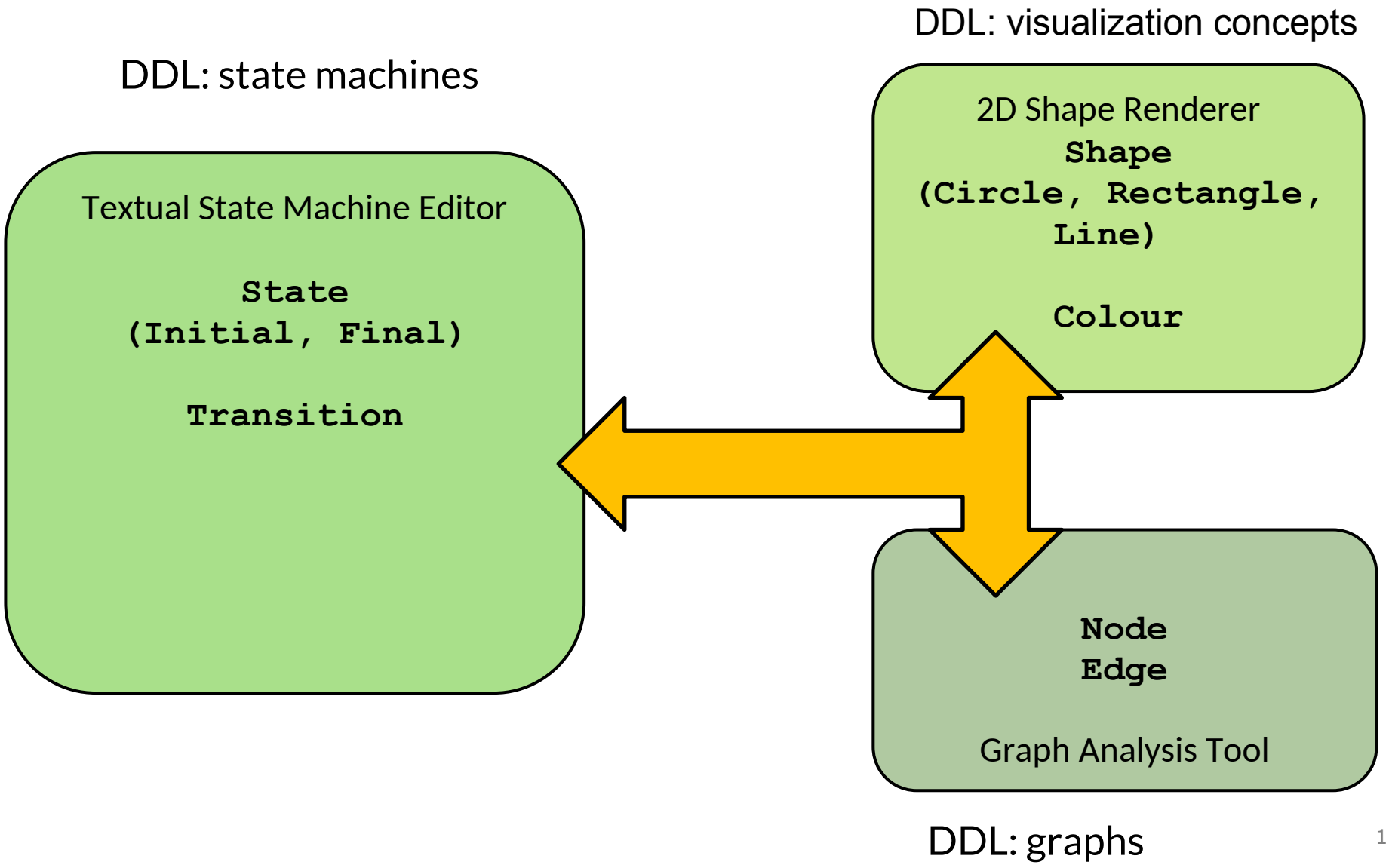
DDL: visualization concepts



DDL: graphs

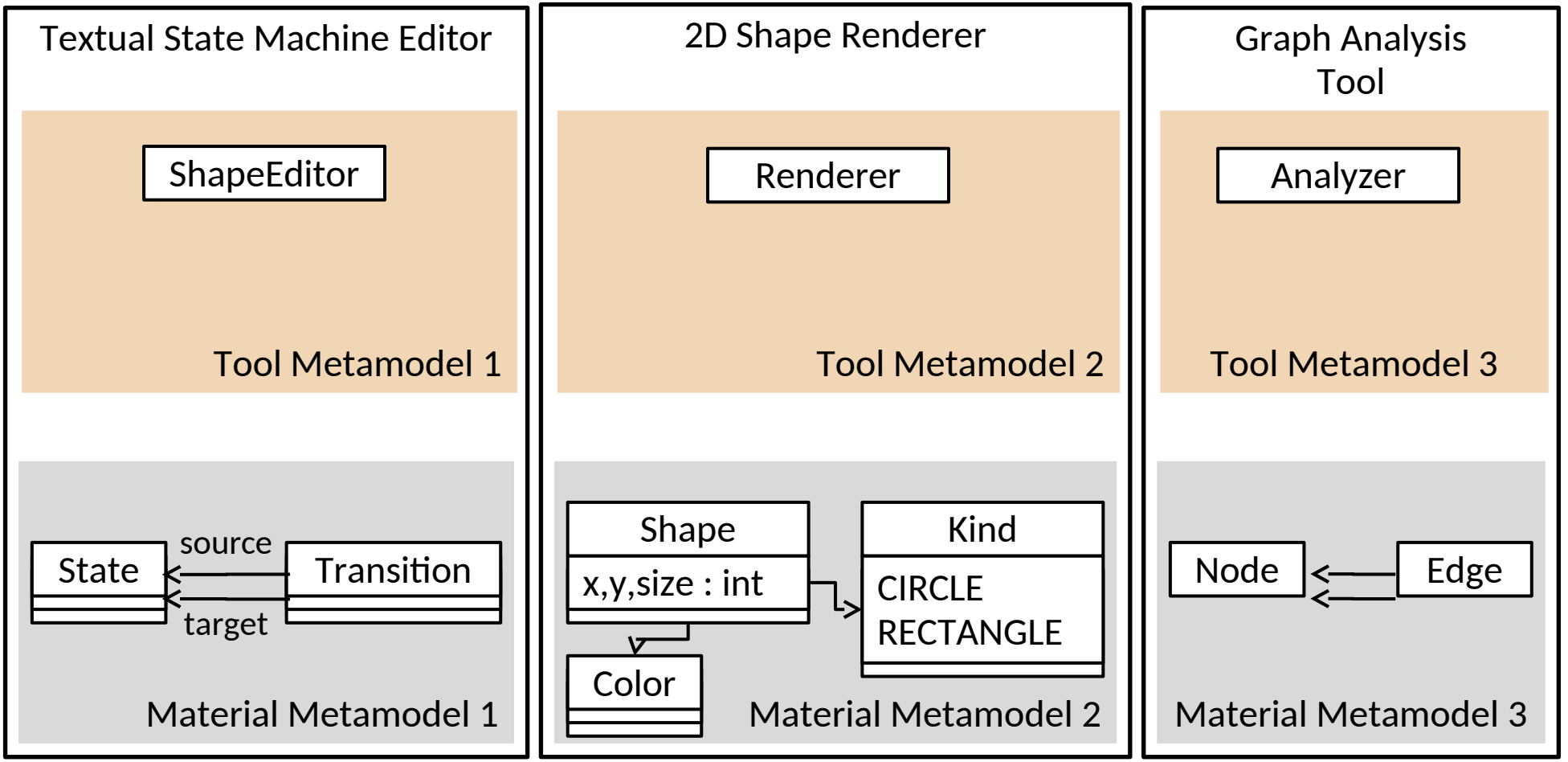
Example – Language Concepts in Metamodels of the Involved Tools

- Then, tools rely on different DDL metamodels with *overlapping concepts*



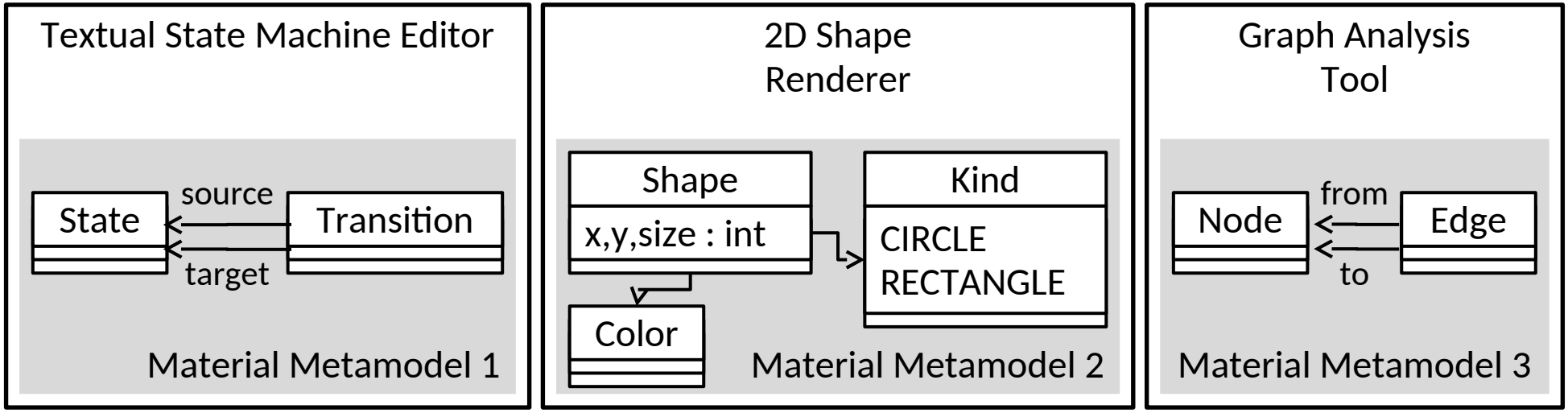
How Can these TAM-Metamodels be Integrated?

- Scenario: Data Integration (Material Integration) of 3 tools, with overlapping Material metamodels



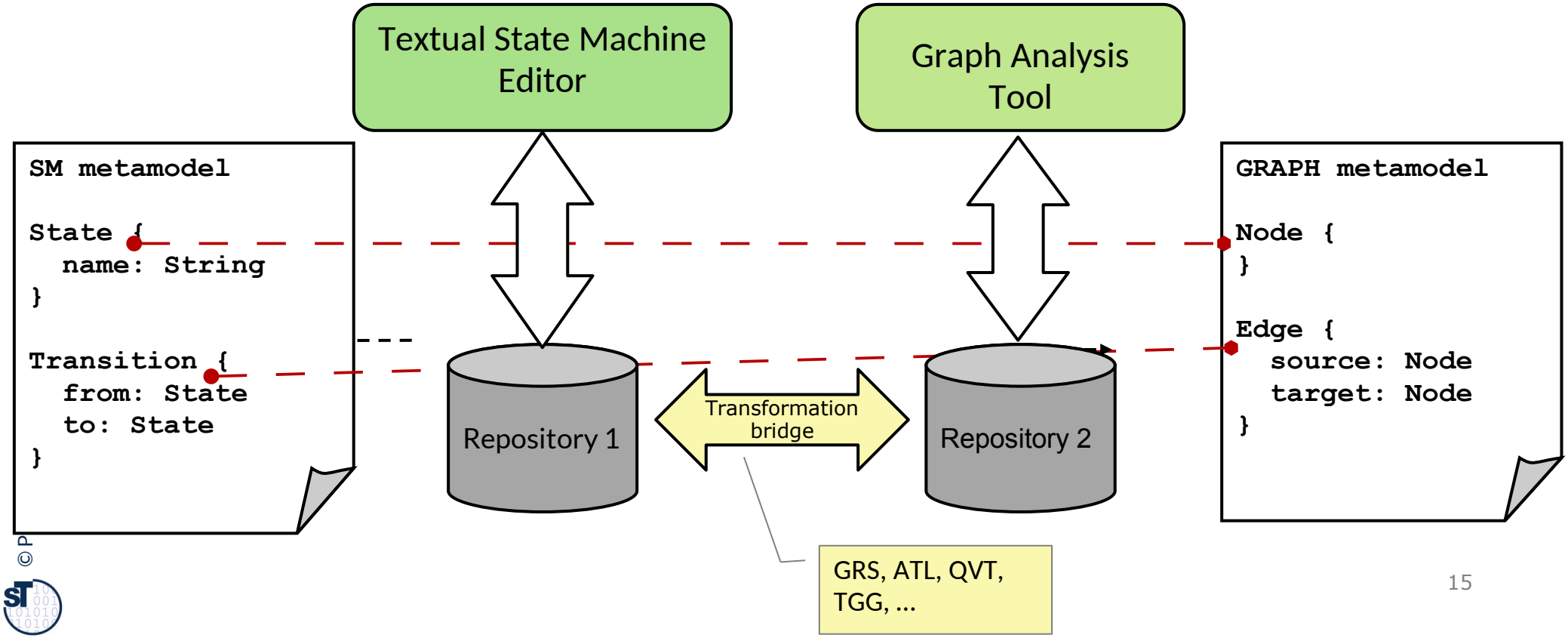
How Can these Metamodels of Materials be Integrated?

- Scenario: Data Integration (Material Integration) of 3 tools



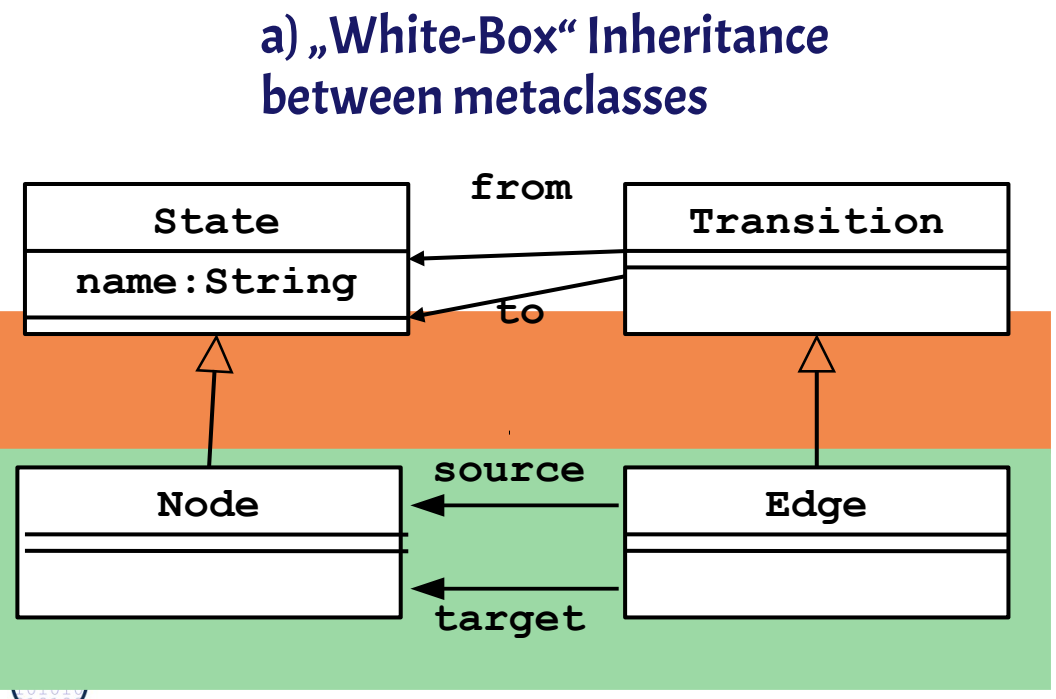
Retroactive Tool Integration on Repositories by **Data Connection**

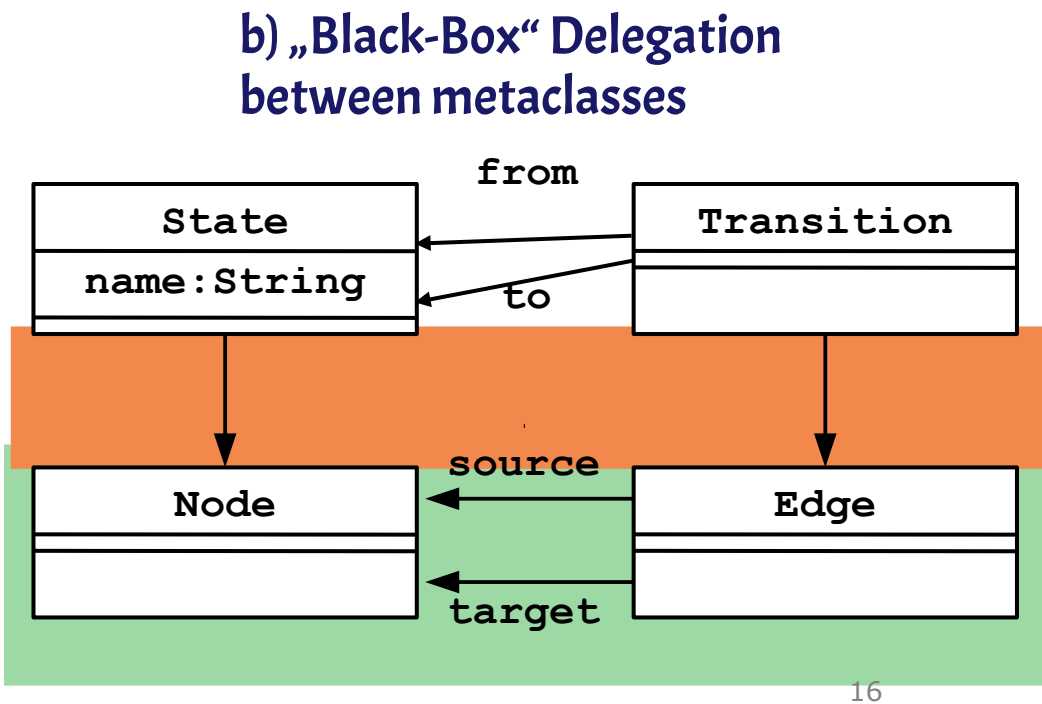
- Often, tools, their metamodels, and the metamodel-driven repositories already exist
- **Data connection** via **transformation** converting data from one tool to another (data exchange via transformation bridge, Datenverbindung) requires
- **Metamodel mapping (language mapping):** map the concepts of one DDL to the other



Classic Proactive Material Integration by Material Inheritance and Material Delegation

- Sometimes, metamodels and repositories are not fixed yet and can be integrated
- Use **metamodel extension (integration)** to make material from one tool accessible to another
  - **Extension by inheritance ("white-box"):** Submetaclasses are formed; language concepts are integrated, but no extension of supermetaclasses possible
  - **Extension by delegation ("black-box"):** Language concepts stay separate, but are connected; no real integration

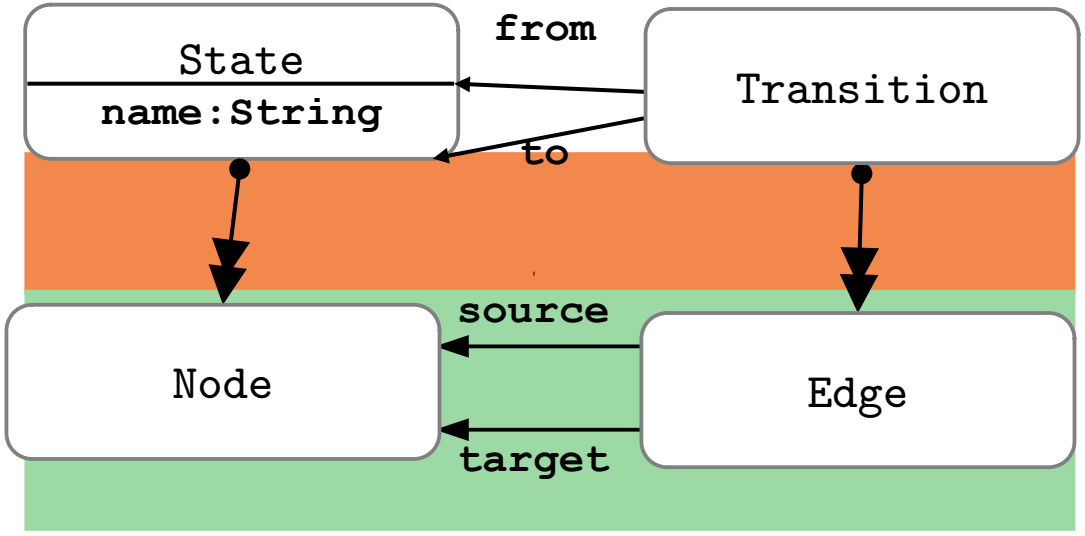




Role-based Material Integration by Material Role-Play

- On role metamodels, Use **metamodel extension (and integration)** can be done by letting *roles play other roles*
  - Role-play makes material from one tool accessible to another. Result: deep roles (chains and trees of roles)
  - **Extension by deep role-play ("deep-rolling")** is a subtechnology of **extension by delegation ("black-box"):** Language role concepts stay separate, but are connected; integration via role play of deep roles
- Grounding of roles to classes ("role mapping") is done later
- Role types are service types; composition of service types

c) „Role-Play" between role metaclasses



Proactive vs. Retroactive Tool Integration

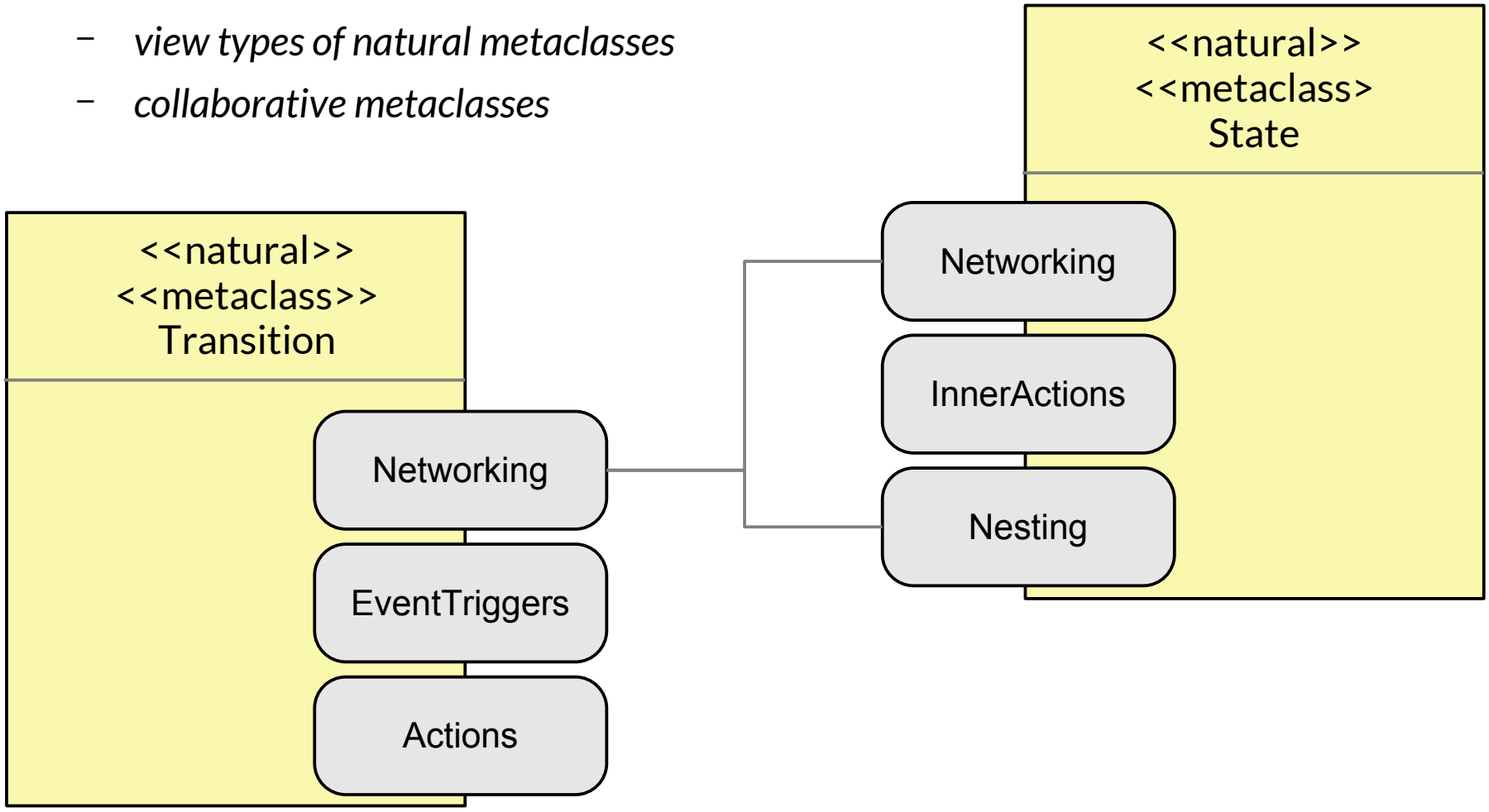
	Proactive (Material Integration)	Retroactive Integration with Data Connection	Deep Rolling
Technique	Inheritance	Transformation	Role binding
	Delegation		
Appropriate Abstraction	Metamodels need to be adapted	Metamodels unaffected	Role metamodels unaffected; class metamodels affected
Tool Independence	Strong coupling	No coupling	No coupling
Shared Data	Sharing among all integrated tools	Replicated Data, Synchronization needed	Sharing among all integrated tools
Tool Interaction	Support for anticipated interaction only	Transformations hinder interaction	Support for anticipated interaction only
Test effort	Inheritance: high Delegation: bit lower	Hopefully none	Hopefully low

41.2 Roles in Models, Metamodels, and Metalanguages

What are Role Metaclasses (on M2)? (Rpt.)

Role metaclasses are

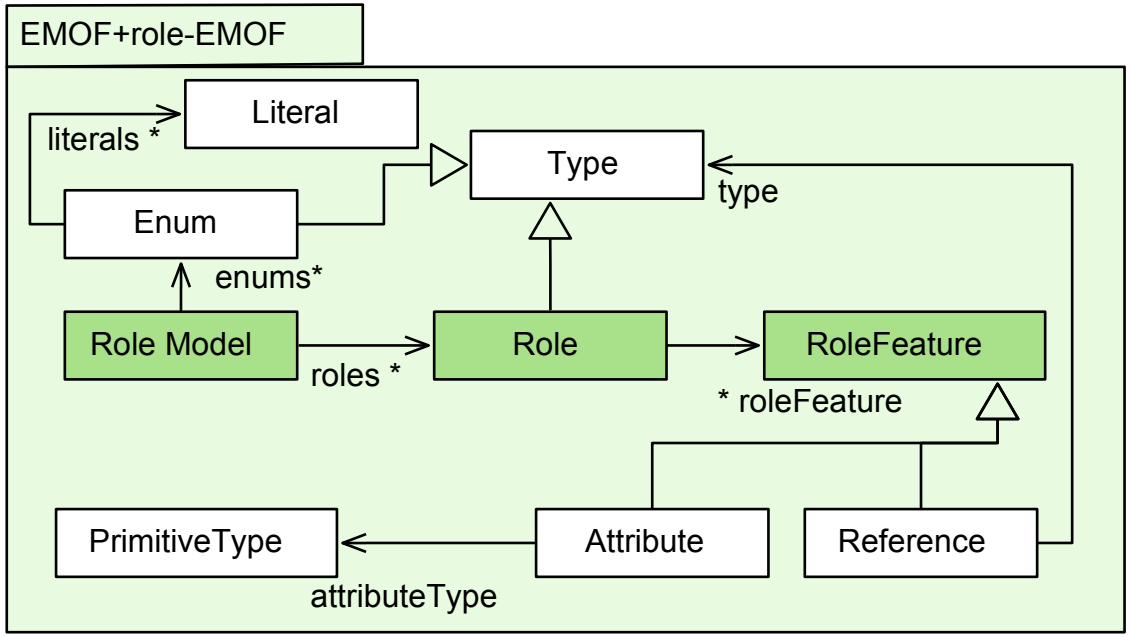
- *service types of natural metaclasses*
- *view types of natural metaclasses*
- *collaborative metaclasses*

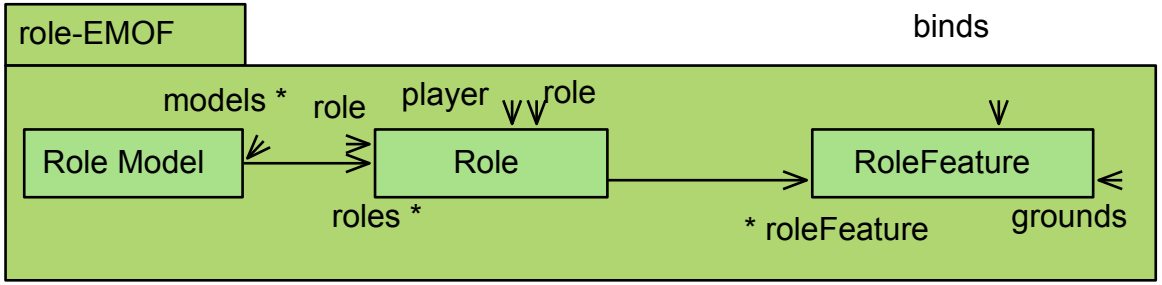


Roles in the Metalanguage (Metametamodel) Role-EMOF

- Roles can be introduced as modeling concept in M3
- ***Role-EMOF*** is an extension of EMOF with roles:

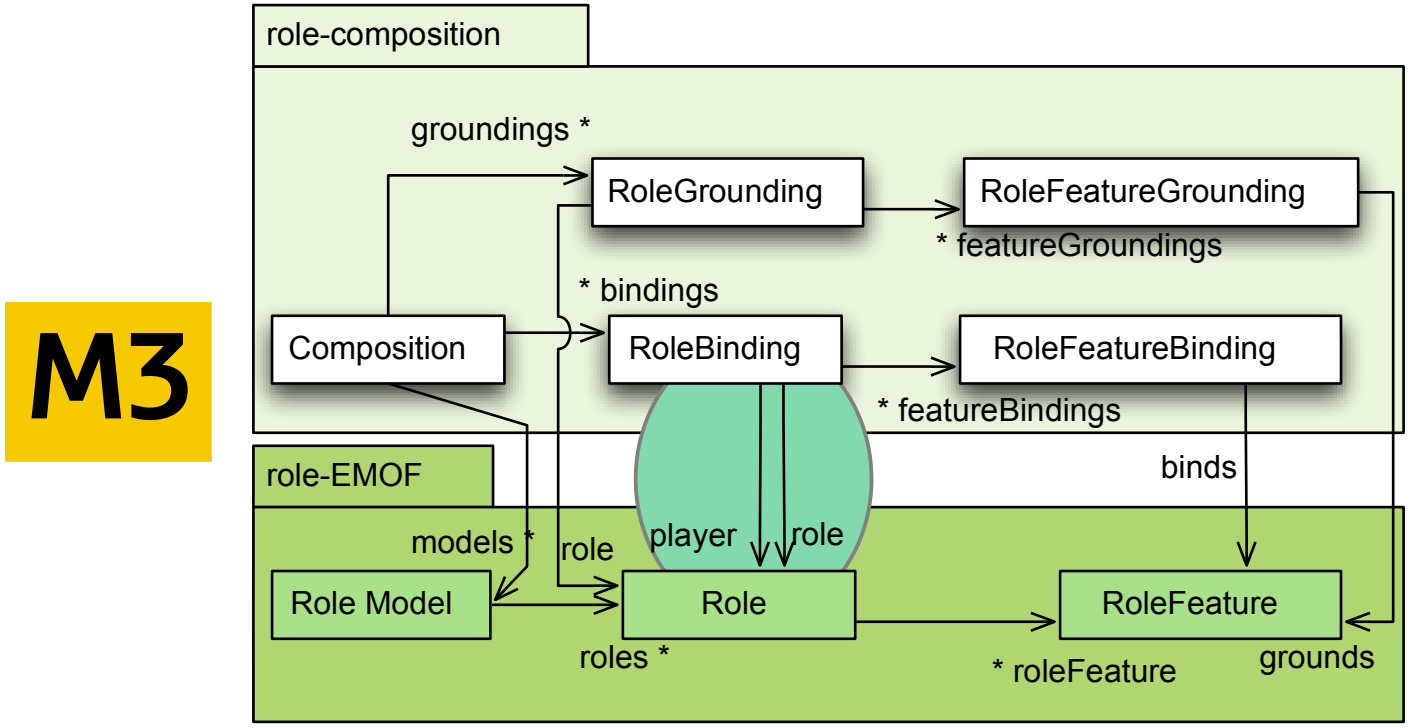
M3





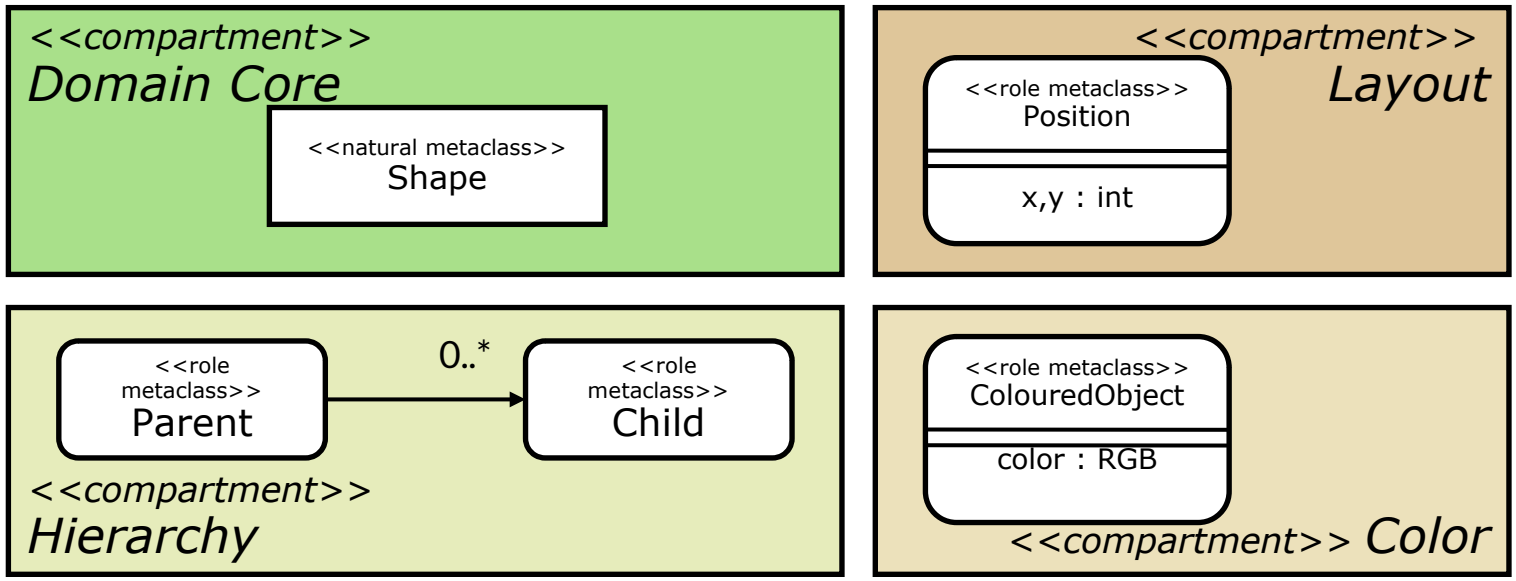
“Deep-Role-EMOF”, a Metametamodel for Deep Role Composition

- **Flat roles** do not play roles
- **Deep roles** are roles playing roles, i.e., can delegate work to other roles
- The Rolecore approach has a *composition technique* for composition of roles and classes, specified by a role-composition metamodel, allows for deep roles
- **Grounding** of a role describes how to bind the role to a Java class



Example: The Material Metamodel of Tool ShapeRenderer with Roles

- Roles of Material-metaclasses adhere to a *compartment*, an explicitly defined context
  - A context is a specific *concern* (here: colors); a compartment a reified context
- Only one natural metaclass, many role metaclasses

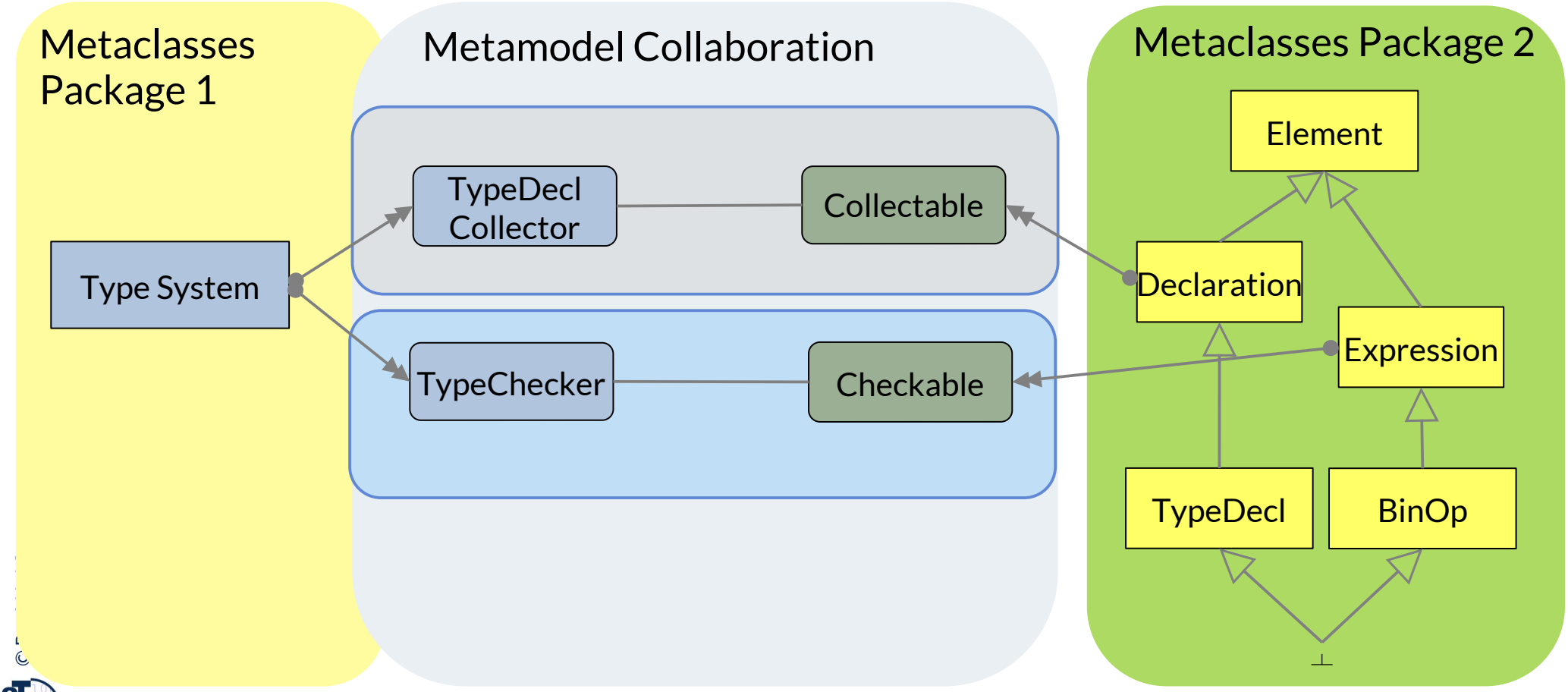


Hypothesis: Language integration works well for Role-based Metaclasses, in particular for Material Metaclasses

41.3. Role-Based Language Composition

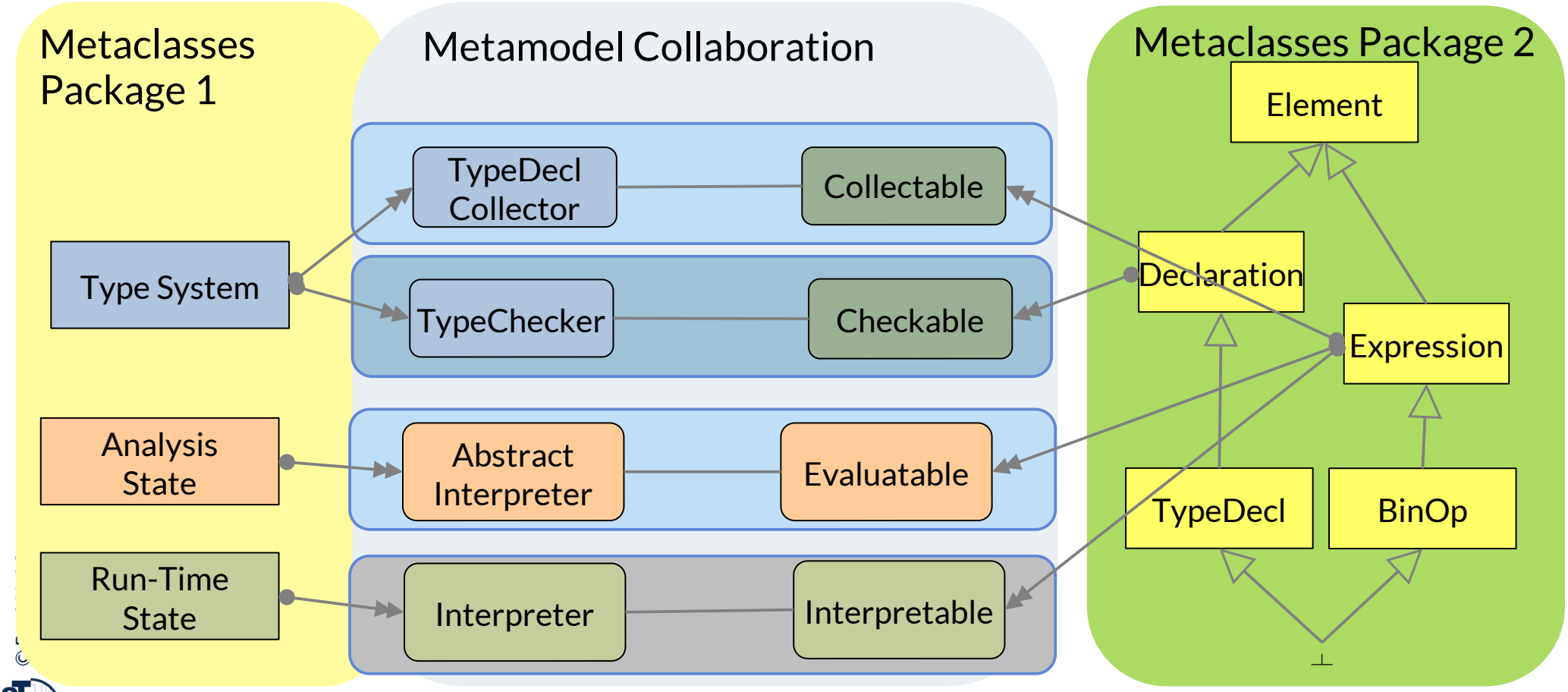
Good News: Role-Based Collaboration of Metaclasses and Their Hierarchies is Simple

- Given a metaclass hierarchy, metaclass roles can be added in new views
- Addition of new metaclasses (blue) easy, because of role extension



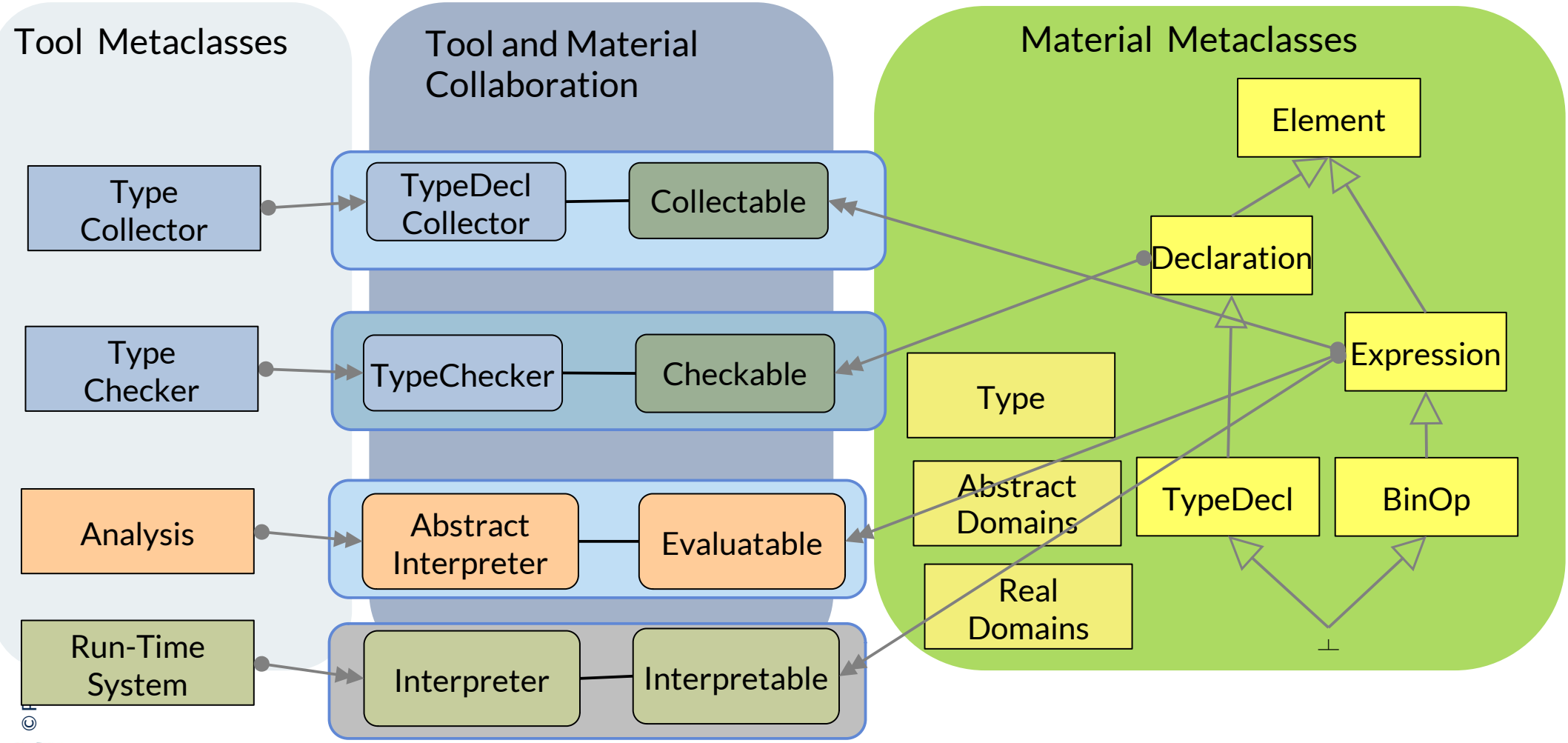
Good News: Role-Based Collaboration of Metaclass Hierarchies with New Metaclass Collaborations is Simple

- Given a metaclass hierarchy, new metaclass collaborations can be added



TAM Metaphor in MDSD Tools on M2 is also Easily Extensible with New Tools

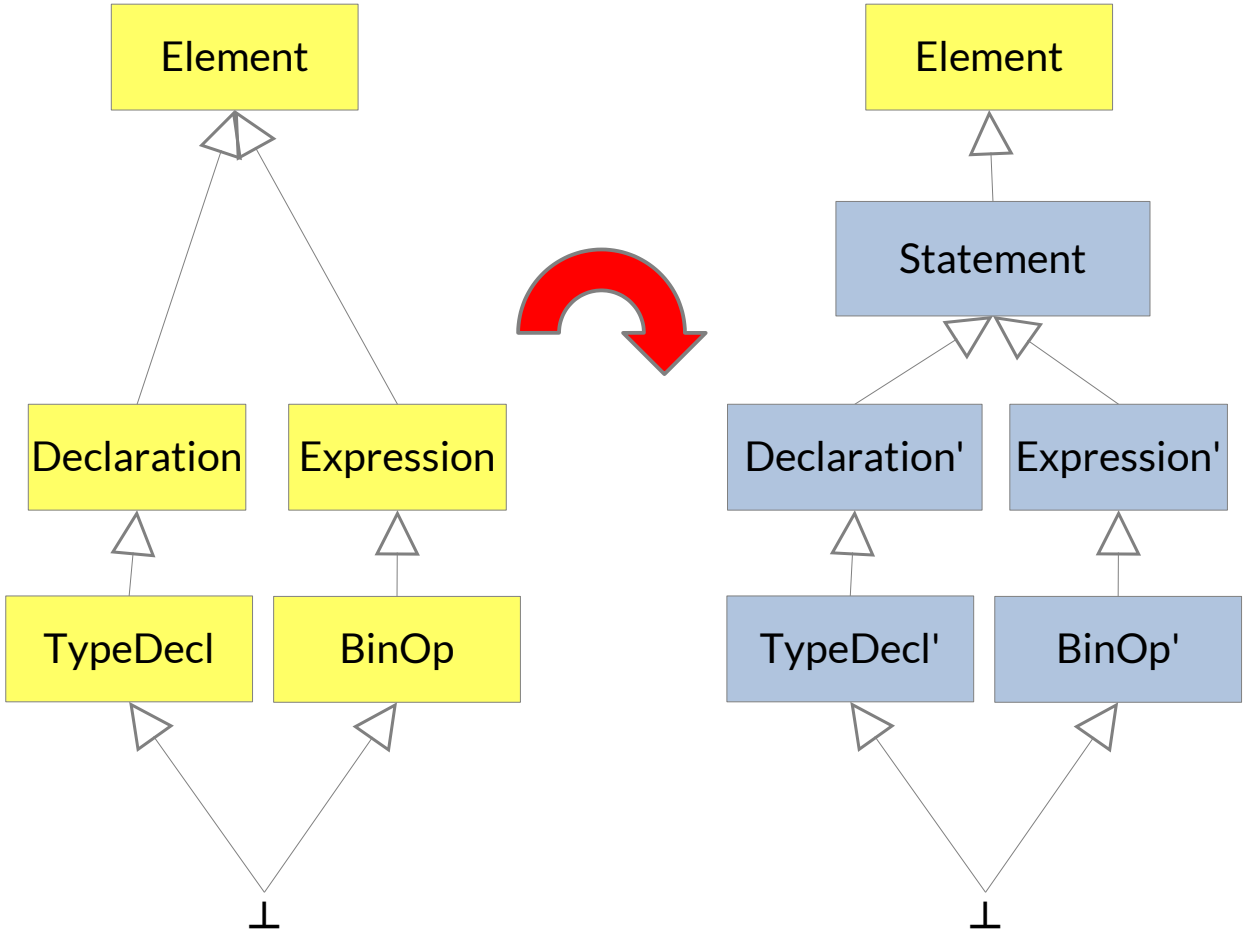
- Given a metaclass hierarchy, metaclass roles can be added in new tool-material collaborations



Superimposition of Material Natural Superclasses Changes Many Subclasses

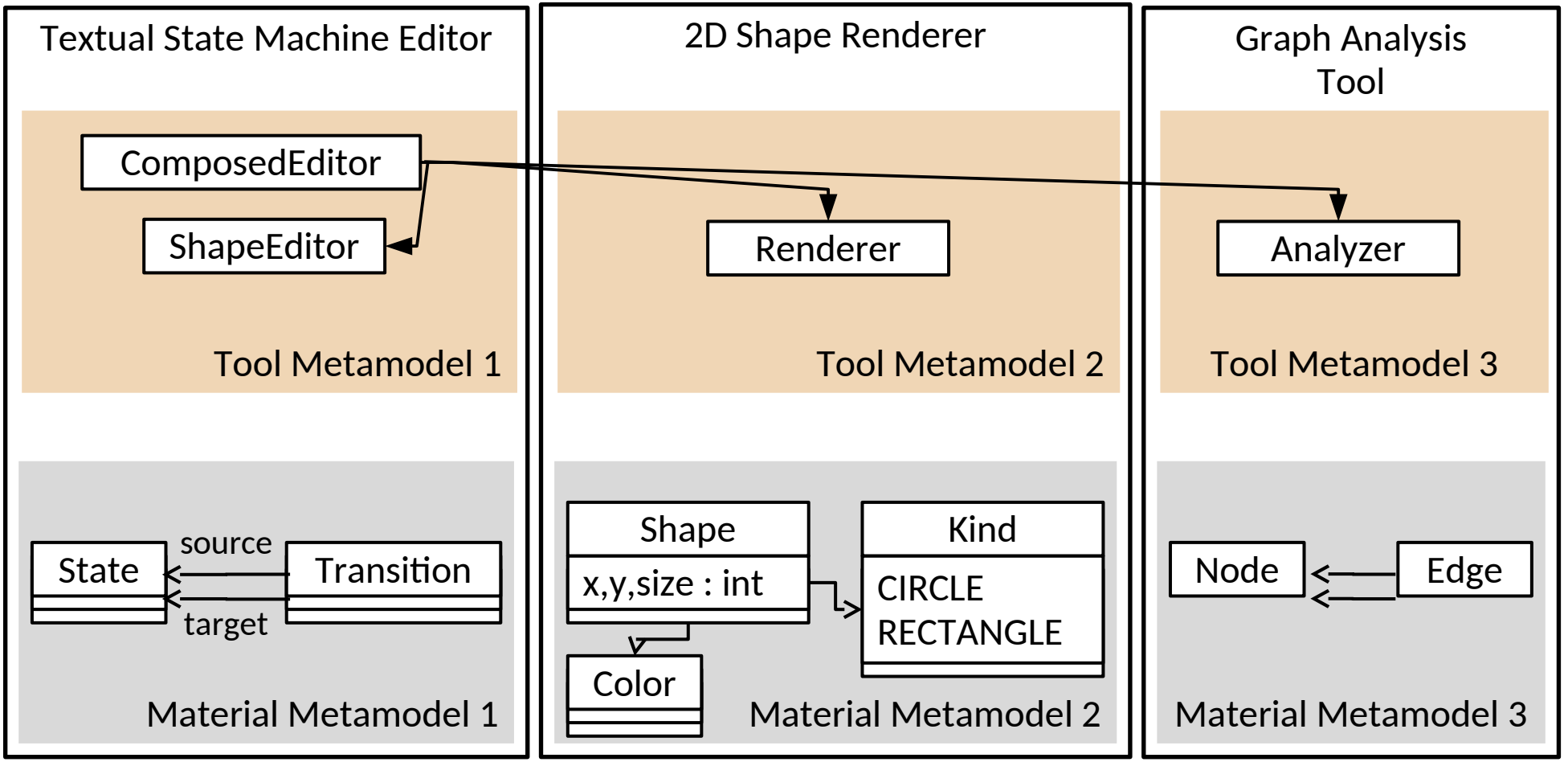
Identity of all derived subclasses changes

- Declaration --> Declaration' under-a Statement
- Expression --> Expression' under-a Statement



Tool Composition is Different than Material Composition

- Materials are passive, with CRUD-like interfaces: Role composition is easy
- Tools are not passive; they must be control-wired by calls or streaming

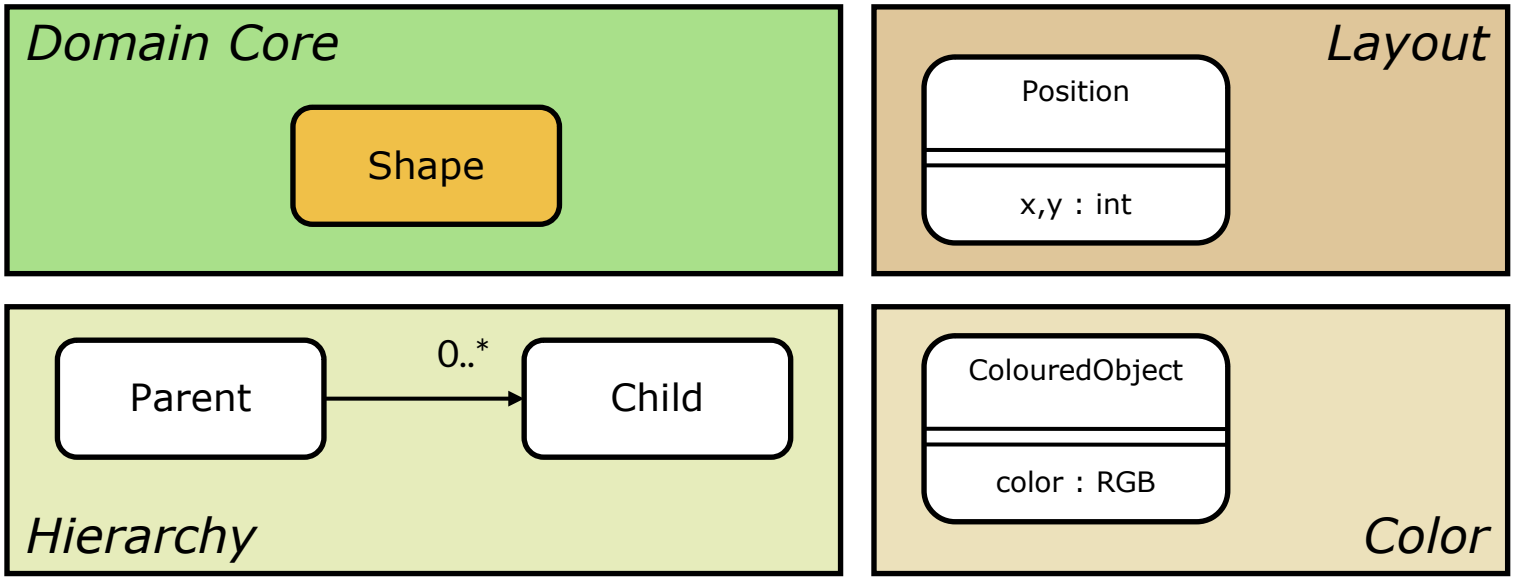


41.4 DeepRolling: an Approach for Metamodel Integration with Deep Role Metaclasses

- Rolecore is a domain-specific language (DSL), with a preprocessor generating Java (standard language)
- Employs Role-Object Pattern for roles in the generated code
- Maximal runtime flexibility, but slow
- Developed by Christian Wende and Mirko Seifert

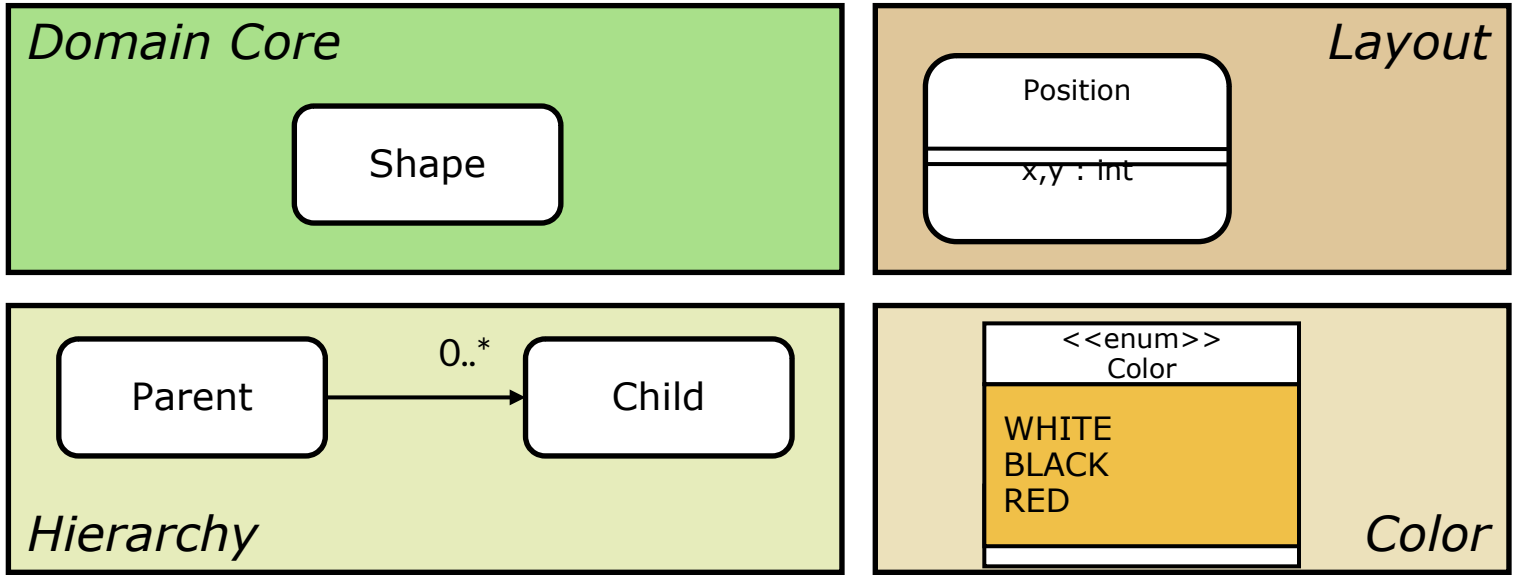
Example: ShapeRenderer's Metamodel with Deep Roles

- The DeepRolling approach integrates Material Metamodels with Role-Type Binding
  - works on role metamodels (see DPF course)
  - The Rolecore-DSL is a textual DSL for DeepRolling and the specification of Deep-Role-EMOF based metamodels
- In Rolecore-DSL, the choice of natural metaclasses is being delayed
  - We first specify all metamodels with deep roles
  - Other materials' metamodels might provide the natural metaclasses
  - Then, they can be played by the naturals of other materials
- Example: Role-metamode of `ShapeRenderer`:



Example: Tool ShapeRenderer's Material Metamodel in RoleCore, with Deep Roles and Enum Classes

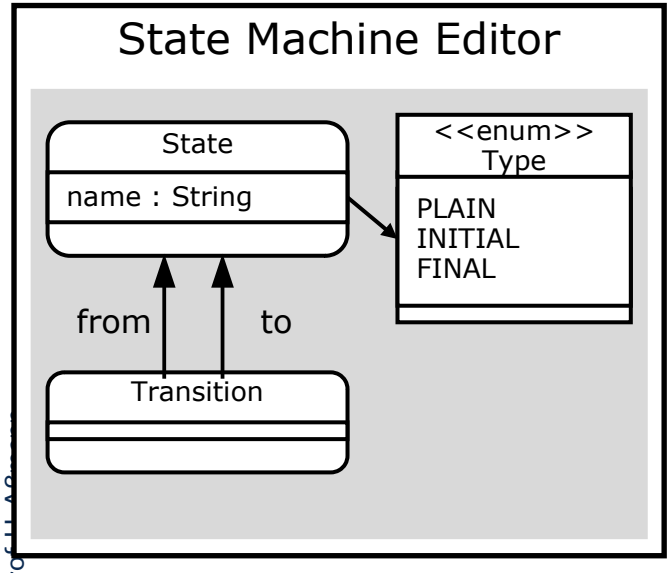
- In the DeepRolling approach, some roles can be represented as *enum classes* if their attributes have finite value ranges
- Then they will become natural classes in the implementation

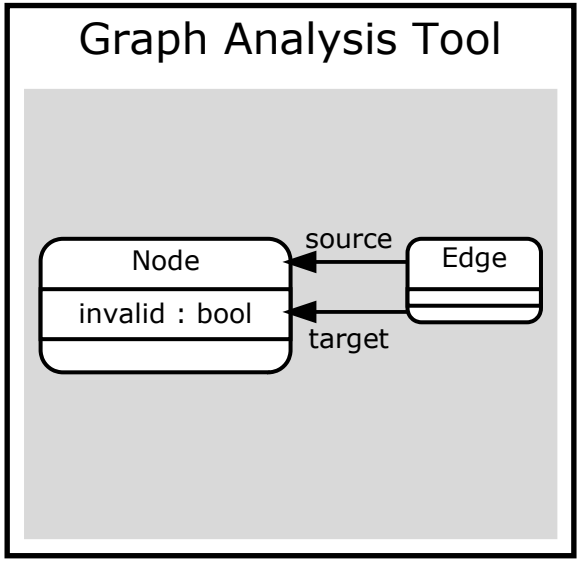


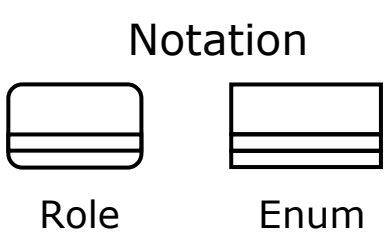
41.4.1 Proactive Material Integration with Deep Roles in RoleCore

Tool Integration using Deep-Role-Model Based Integration of Material Metamodels on M2

- Specify M2-metamodels also with role metaclasses (abilities), not only classes
- Difference to classical role modeling:
  - First specify everything as deep role
  - Select those roles which should become enums
  - Naturals are selected last

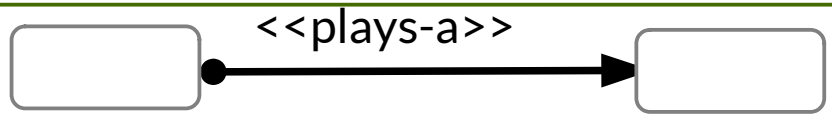






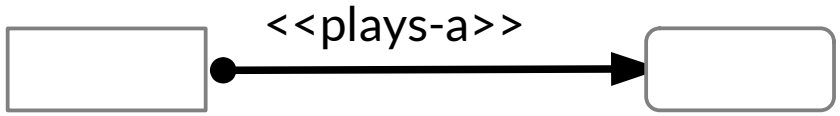
For Tool Integration, the DeepRolling Approach Uses Role Bindings and Role Grounding

- **Role Bindings** on the logical level



  - with relationship “plays-a”
  - Connect roles and role players, producing deep roles
  - Define how to obtain value of attribute or reference
  - Allow to create views on other classes

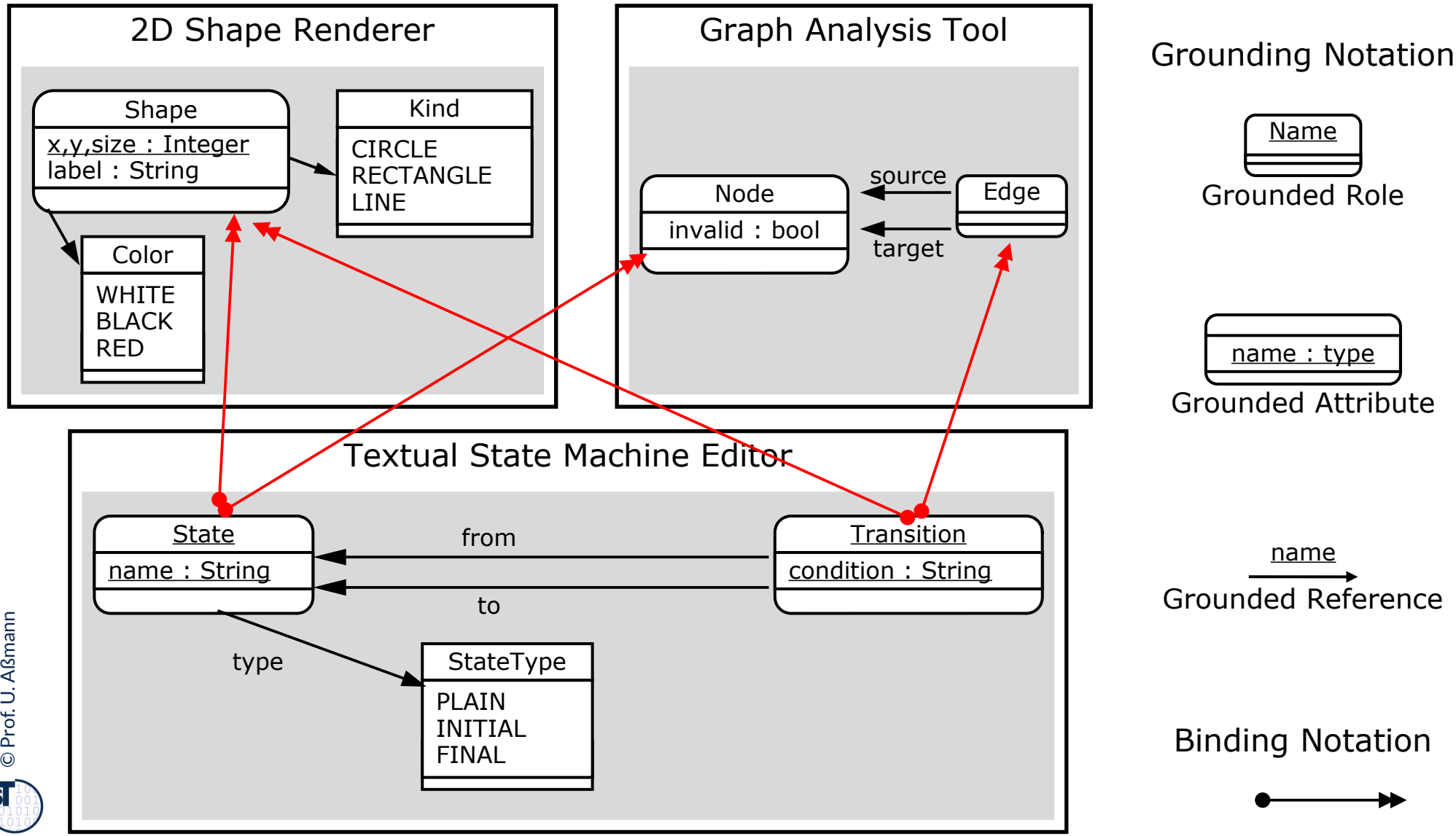
- **Role Grounding** on the physical level



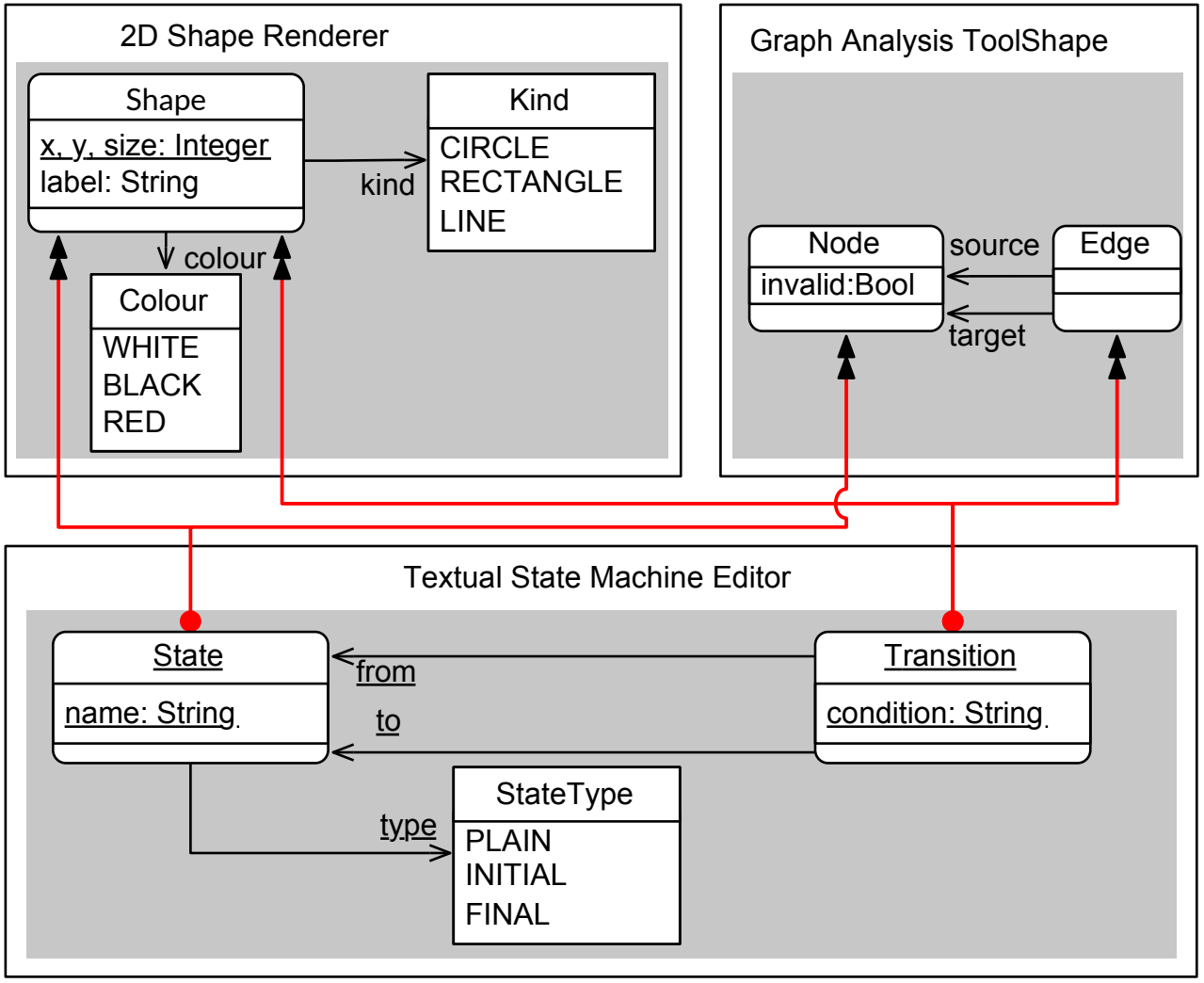
  - Defines which attributes/classes are represented physically
  - Select natural metaclasses
  - Ground to implementation by design patterns or other role-implementations (see course Design Patterns and Frameworks)
- The decision (about which data is derived and which is not) is done at tool integration time!

Metamodel Composition based on Deep Role-Type Binding

- Composition by *deep role-type binding*
- We defer the decision "what is a natural" (*role grounding)* to later



Deep Rolling



41.5. Rolecore: a DSL for Metamodel Integration with Deep Role Metaclasses

Grounding (Realization): Mapping Role Metaclasses to Programming Languages

RoleCore DSL for Integration of Materials with DeepRolling (EMFText-Based Language)

- Role binding (playing) and role grounding can be described by the RoleCore DSL.
- RoleCore generates class models with appropriate role-play and grounding

```
integrate statemachine, 2dShapes, graph {
    State plays Shape {
     label: name
     kind: if (player.type == PLAIN) return RECTANGLE
           else return CIRCLE
     colour: if (player.type == INITIAL) return WHITE
           else return BLACK
    }
    Transition plays Shape {
     label: condition
     kind: return LINE
     colour: return BLACK
    }
    State plays Node {}
    Transition plays Edge {
     source: from
     target: to
    }

    ground State { name, type }
    ground Transition { condition, from, to }
}
```

Role Binding Specification

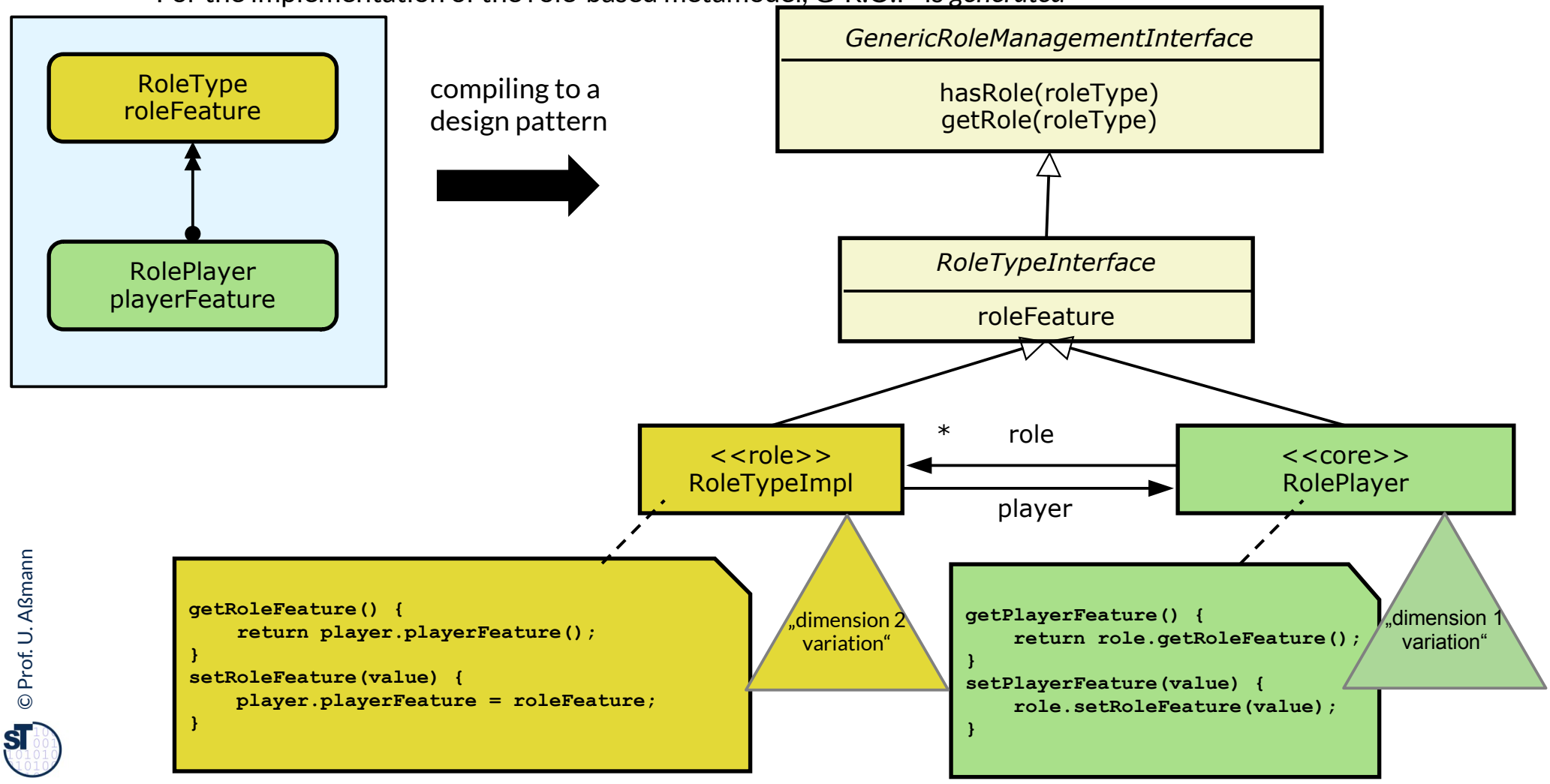
Link mapping from one role to the other

Grounding Specification

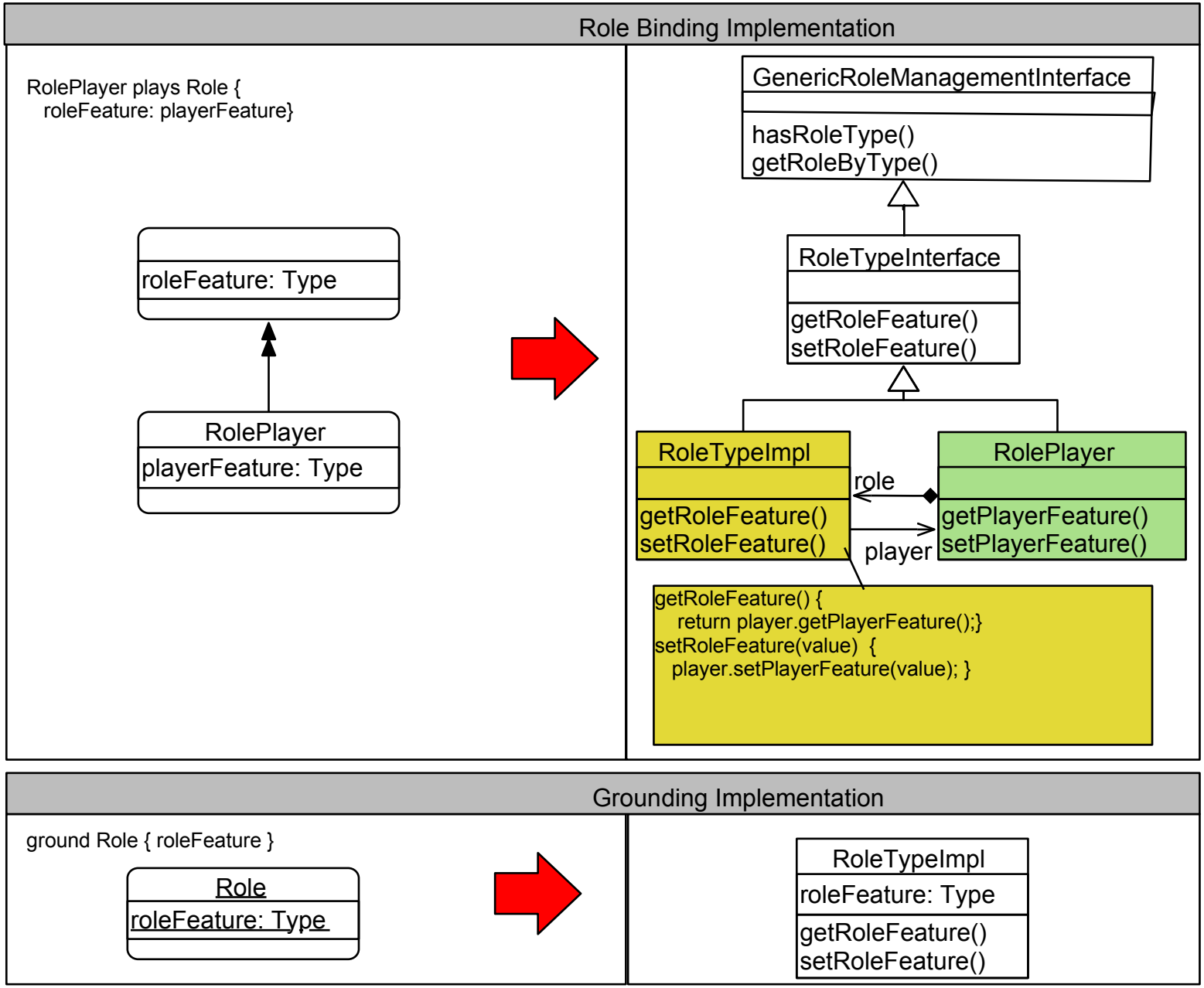
https://github.com/DevBoost/EMFText-Zoo/tree/master/BreedingStation/RoleCore

Realisation of Role Binding and Grounding: Design Pattern "Generative Role Object Pattern"

- Grounding is straightforward with many design patterns for role implementations
- The constructs of RoleCore can be easily expanded to design patterns (code generation), e.g., MultiBridge, Flat or Deep Role-Object Pattern
- For the implementation of the role-based metamodel, G-R.O.P is *generated*



Role Binding Implementation with Generative Role Object Pattern (GROP)



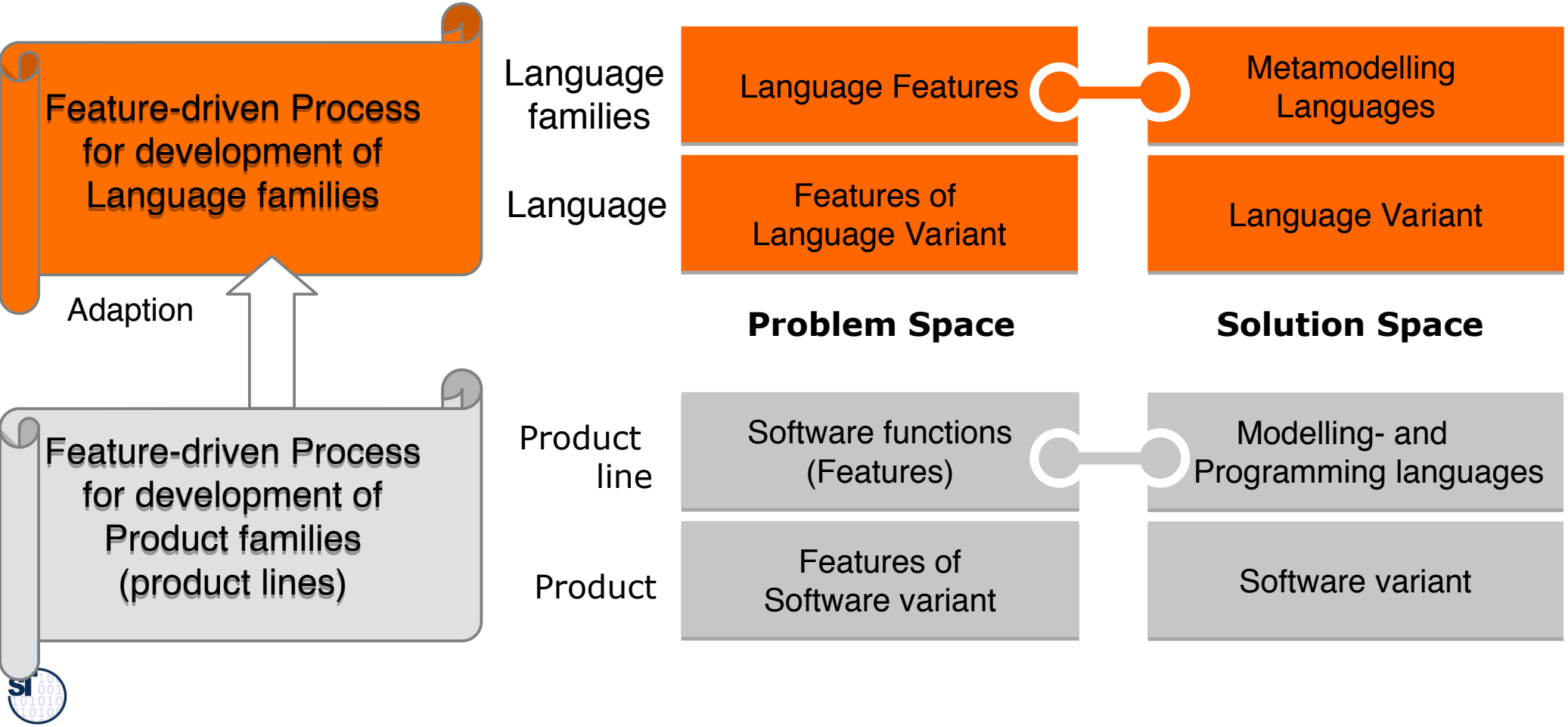
41.6. Role-Based Language Composition with LanGems

Why Why are Roles Good in a Metamodel Mapping?

1) Role metaclasses compartmentalize metaclasses
   - If mappings are defined on role metaclasses, they can be defined much more fine-grainedly
   - For role-based metamodels, language mappings can be much more precise!
2) Role-based metamodel packages are simpler to reuse
   - Because their roles can be merged into metaclasses of other packages
   - Data integration and sharing becomes simpler because data is composed from “data components”

Feature-Driven Development of Language Families PhD Christian Wende (2012)

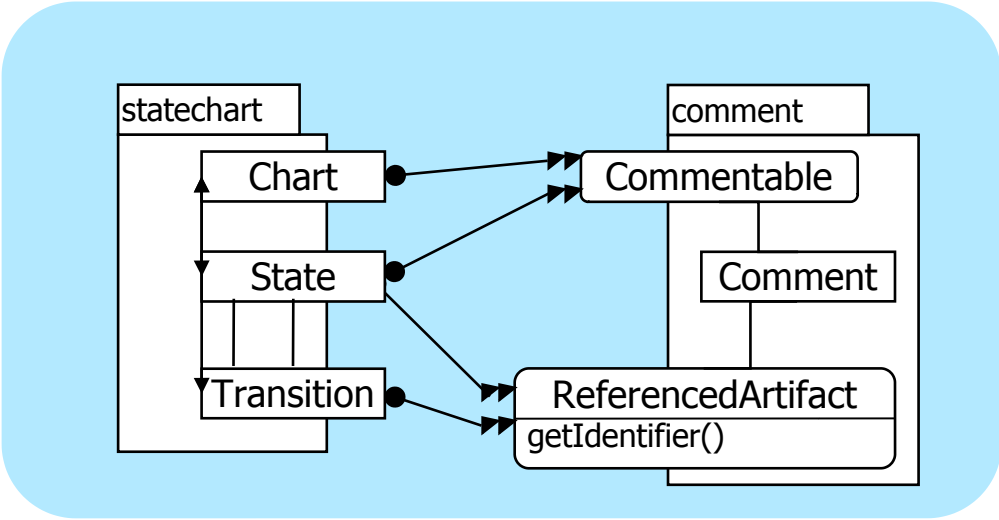
Systematic variability management for language families with feature models



Role-Based Language Composition with LanGems Language Components

RoleBinding:

- Employ EMOF packages as language components
- One natural metaclass can play several roles
- A role metaclass can be played by different natural metaclasses
- Interfaces of EMOF packages:
  - Natural metaclasses looking for played roles
  - Offered roles to be bound on naturals



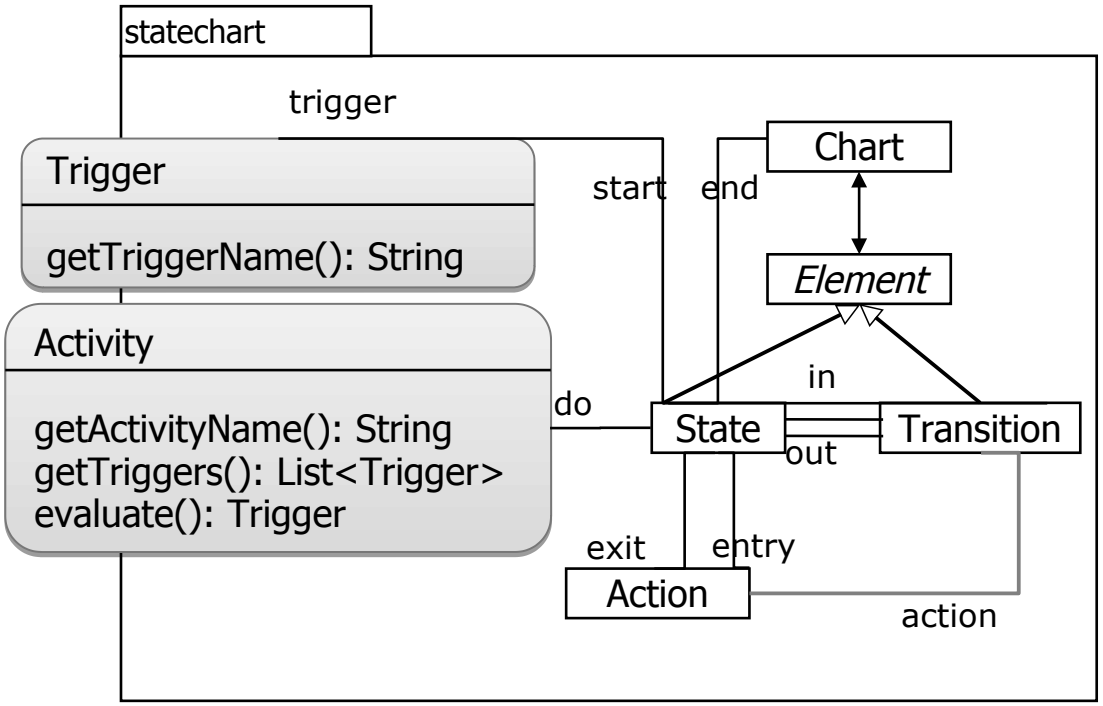

```
composition chartWithComments {
  …
  State plays ReferencedArtifact (
     // example OCL-RoleOpBinding
     getIdentifier() : player.stateName;
  )
}
```

Christian Wende. Language Family Engineering. PhD thesis, Technische Universität Dresden, Fakultät Informatik, March 2012. www.qucosa.de http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-88985

Example: Statecharts and Forms

Language Component: statechart

Abstract Syntax



concrete Syntax (offered by EMFText)

```
chart example
  Init: init
  End: finish cancel
{
  state init {…}

  from init to data
     when login do {}

  state data {…}
  …
}
```

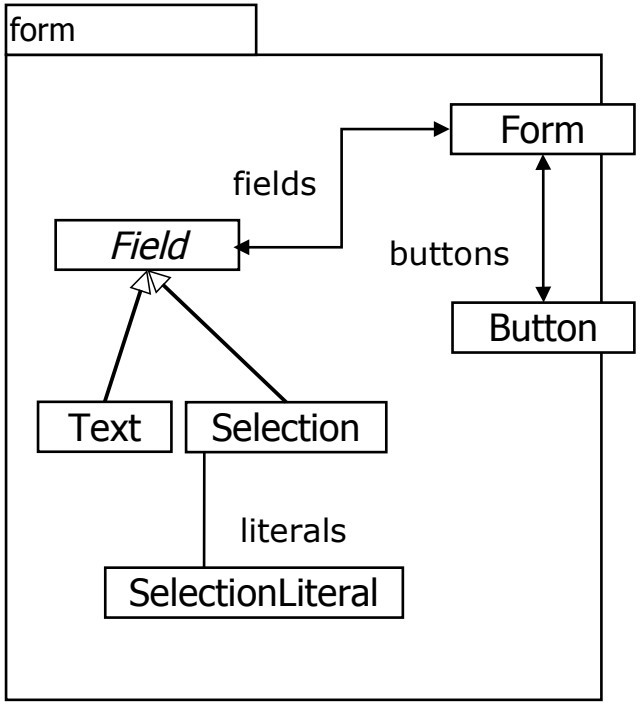
Semantics:

Operational Semantik written in Java based on the interfaces of abstract syntax

Example (2): Forms

Language Component: form

Abstract Syntax



concrete Syntax (with EMFText)

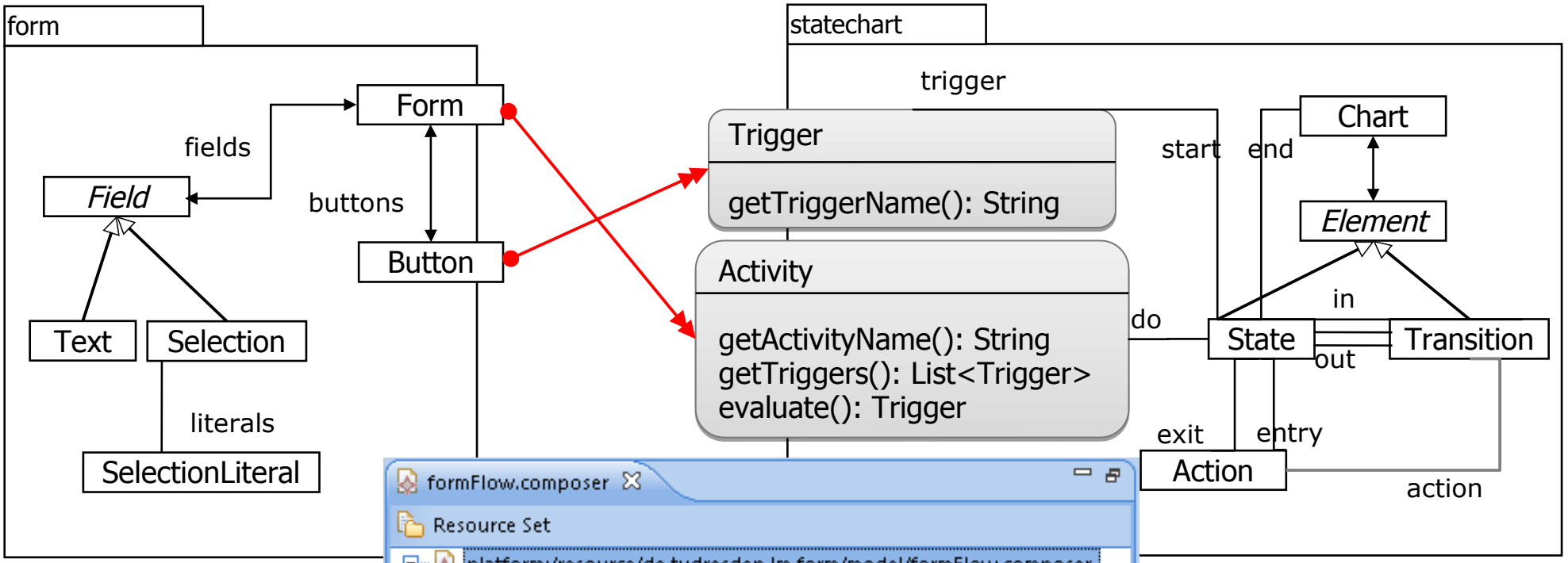
```
form [ welcome ]
-( Please enter your
     login data. )-
{
     name
     password
}
     buttons     > login
                 > cancel
```

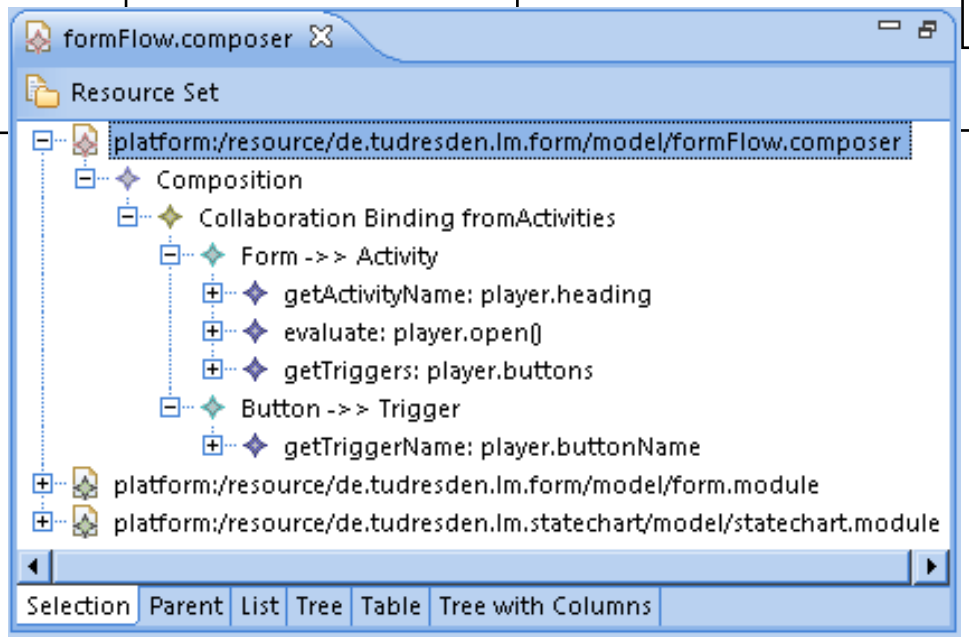
Semantics:

Operational Semantics with Java

Example (3) Composition of Statechart and Form

Language Composition: formFlow := form -->> statechart

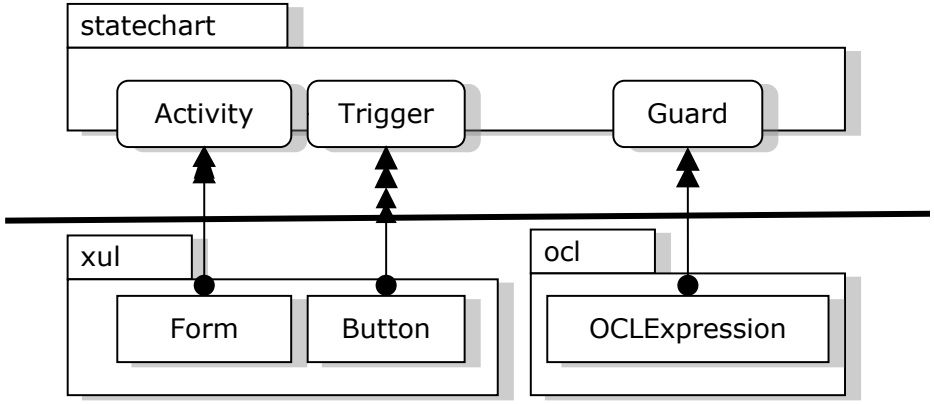




Language Composition Editor

LanGems Module Composition Language

- Types contribute the composition interface of language modules
  - Role Types: required interface
  - Natural Types: provided interface
- Language Composition is described by superimposition of the collaborations of several modules where *RoleBinding* connects role player and role
- Binding of RoleOperations in the context of a role player (*RoleOperationBindings)* contributes structual and semantic adaptation of the role player w.r.t the role contract

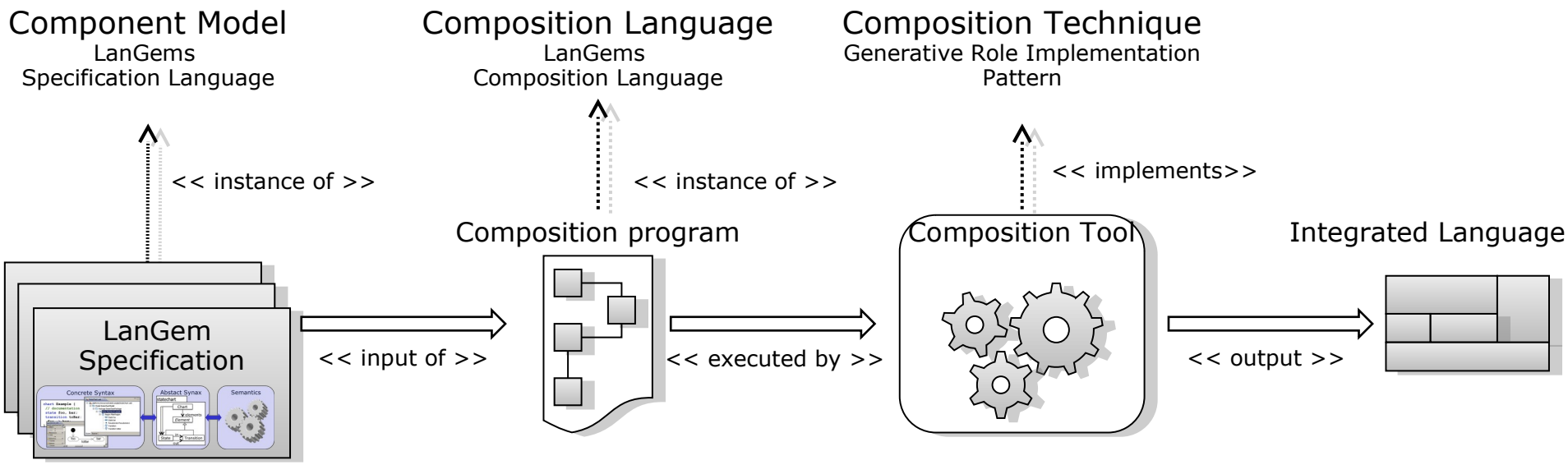



```
Form plays Activity {
   getActivityName(): player.getTitle();
   executeActivity (): player.open();
}

Button plays Trigger {
  getTriggerName(): player.getText();
}

OCLExpression plays Guard {
  evaluate(Object context) : if (player.type = boolean)
                             then player.interpret(context)
                             else false;
}
```

LanGems Composition System



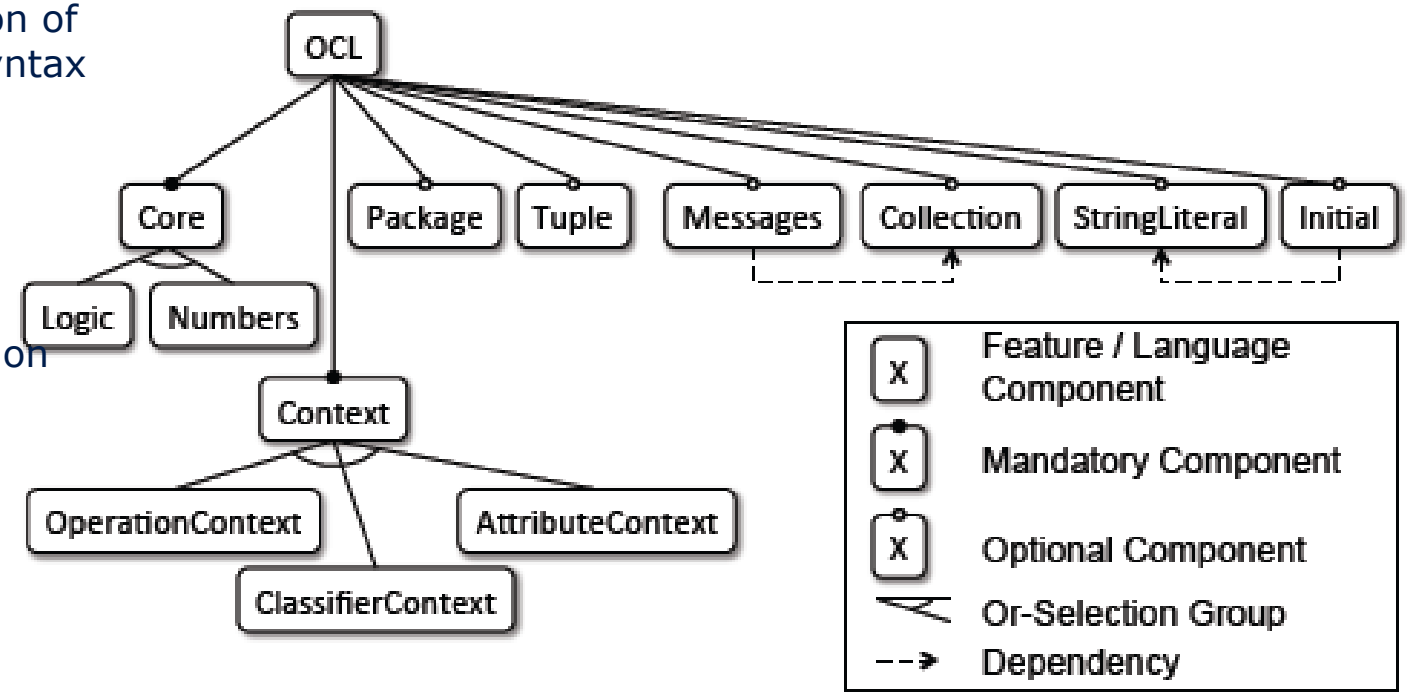
Case Study: Modularisation of OCL

OCL

- Complex language
- Applied at different abstraction layers and environments
- Several proposals for extension of OCL

Activities

- Separation of 13 language modules
- Each contributes specification of abstract syntax, concrete syntax and static semantics
- Language adaptation to use OCL on different metamodels (Ecore, UML, MOF)
- Exemplary language extension with temporal logic



Case Study Statecharts: The Problem Space Perspective

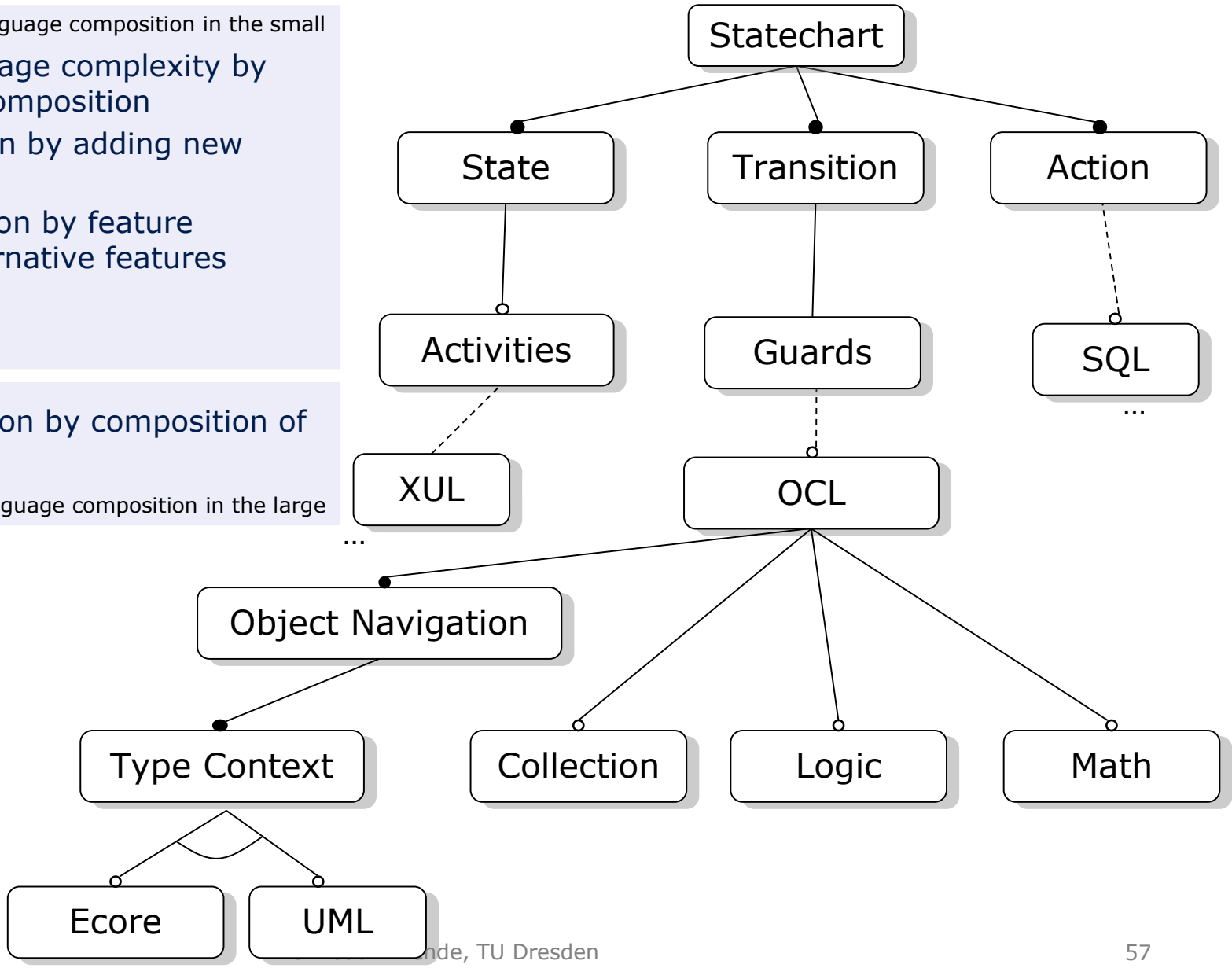
Feature-Based Family Specification

language composition in the small

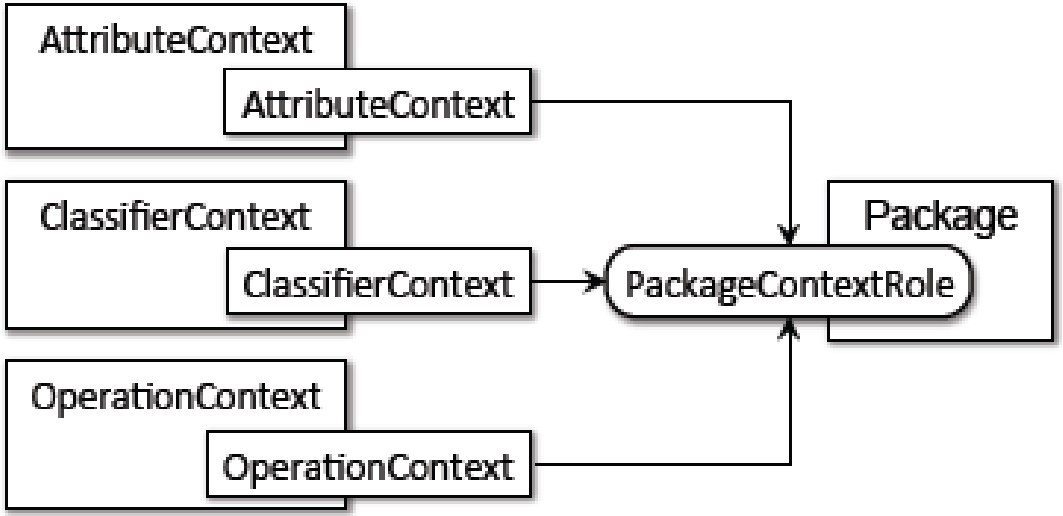
- Reduction of language complexity by feature-based decomposition
- Language extension by adding new features
- Language adaptation by feature exchange and alternative features

- Language integration by composition of their features

language composition in the large



Evaluation



Experienced Benefits

- Self-contained comprehensible modules
  - Independent Development and Maintenance
  - Explicit language component interfaces decouple language modules
  - Adaptation of OCL by variation on language modules
  - Extension of OCL by adding language modules
- Role-based modularisation and composition supports for concrete syntax and language semantics
  - Composition did not invalidate module syntax and semantics
  - Composition provides means for semantic (and structural) adaptation

Problems & Open Issues

- Operator priorities needs to be considered during composition
- Context-free parsing required adjustment of token definitions among modules
- Dynamic Semantics not implemented yet

What Did We Learn?

- Deep Role Modelling allows for unanticipated material integration, but needs to be applied at material design time
- Clean separation of required interface (to access tool-specific data) and realization of this interface (to obtain data)
- Physical representation define at integration time by design patterns for role implementation
- If ROP is used as a pattern in the code generator, a role-based access layering of the repository results naturally.

The End

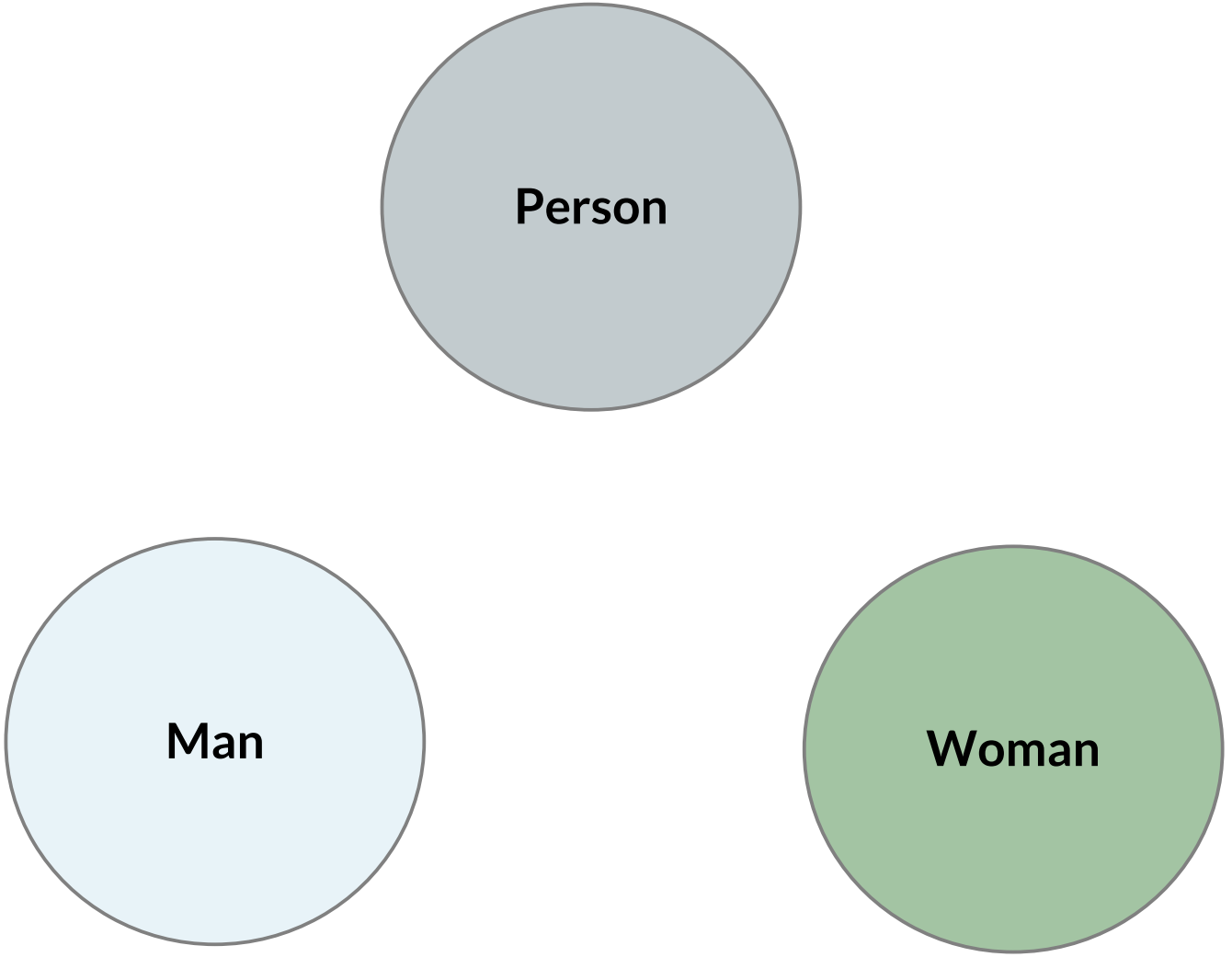
- Explain the difference of a role metaclass and a natural metaclass.
- Why is it beneficial to use roles on M2 as role metaclasses?
- Describe the differences in the development process of RoleCore and LanGems
- Why is it easy to extend role-based metamodels?

Db.inf.tu-dresden.de/rosi

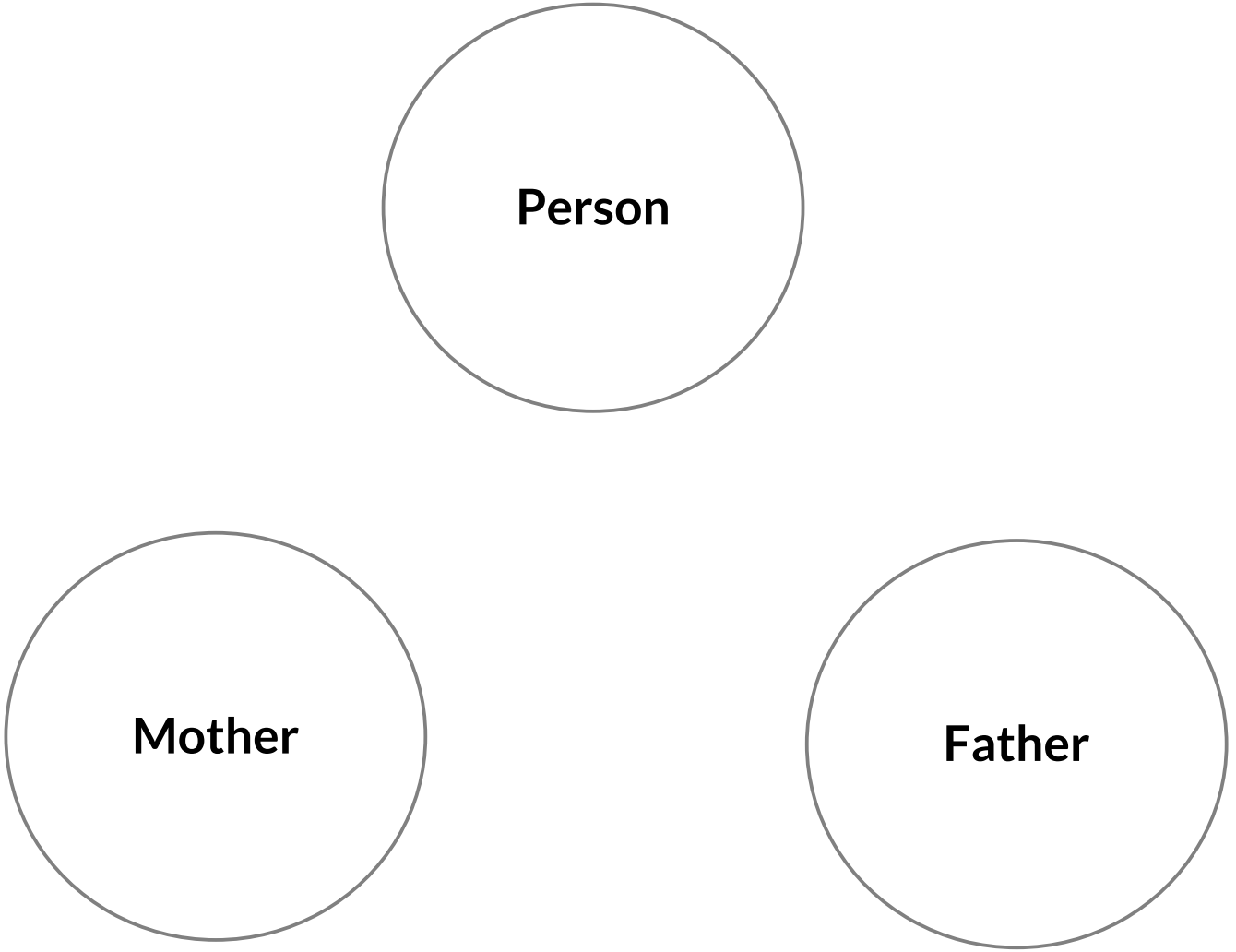
http://st.inf.tu-dresden.de/

Appendix: Role Modeling

A Riddle..



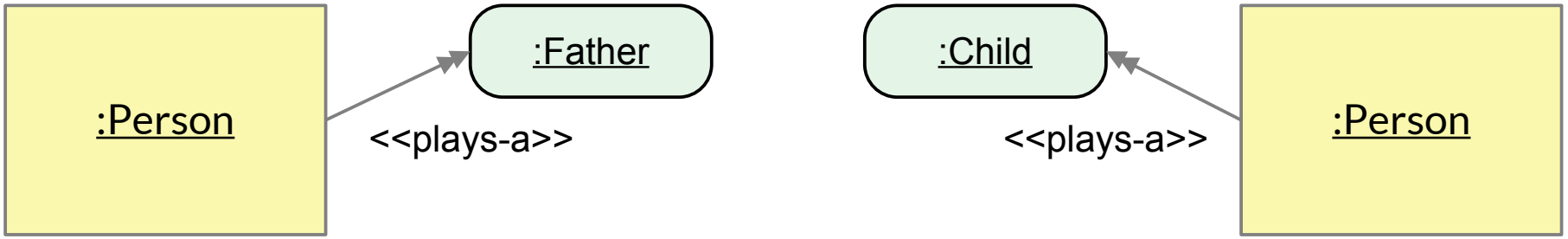
Another Riddle..



Collaboration-Based Modeling (Role Modeling) (Rpt.)

Roles are first-class modeling concepts in modern object-oriented languages

- Databases [Bachmann], Object-Role Modeling [Halpin]
- Factorization [Steimann]
- Research in Design Patterns [Reenskaug, Riehle/Gross]



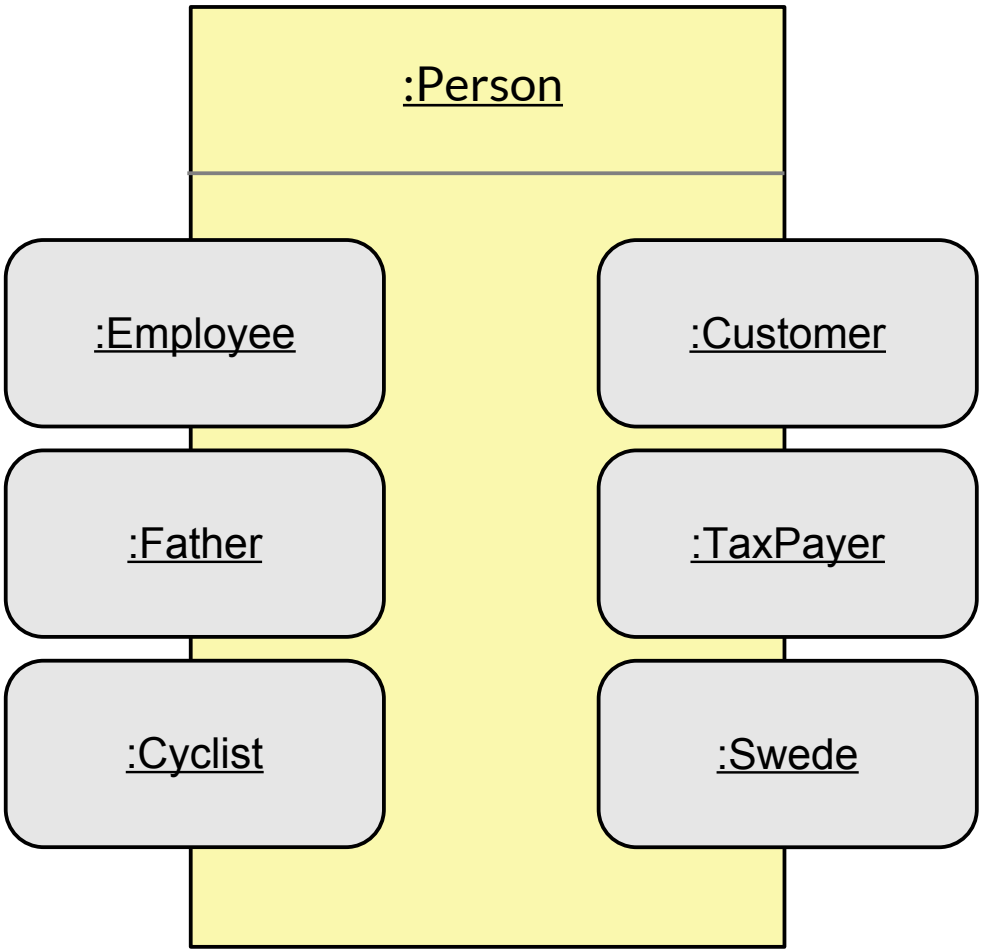
What are Roles? (Rpt.)

A *role* is a *dynamic view* onto an object

- Roles are *played* by the objects (the object is the *player* of the role)
- A *partial object*

Roles are tied to *collaborations*

- Do not exist standalone, depend on a partner



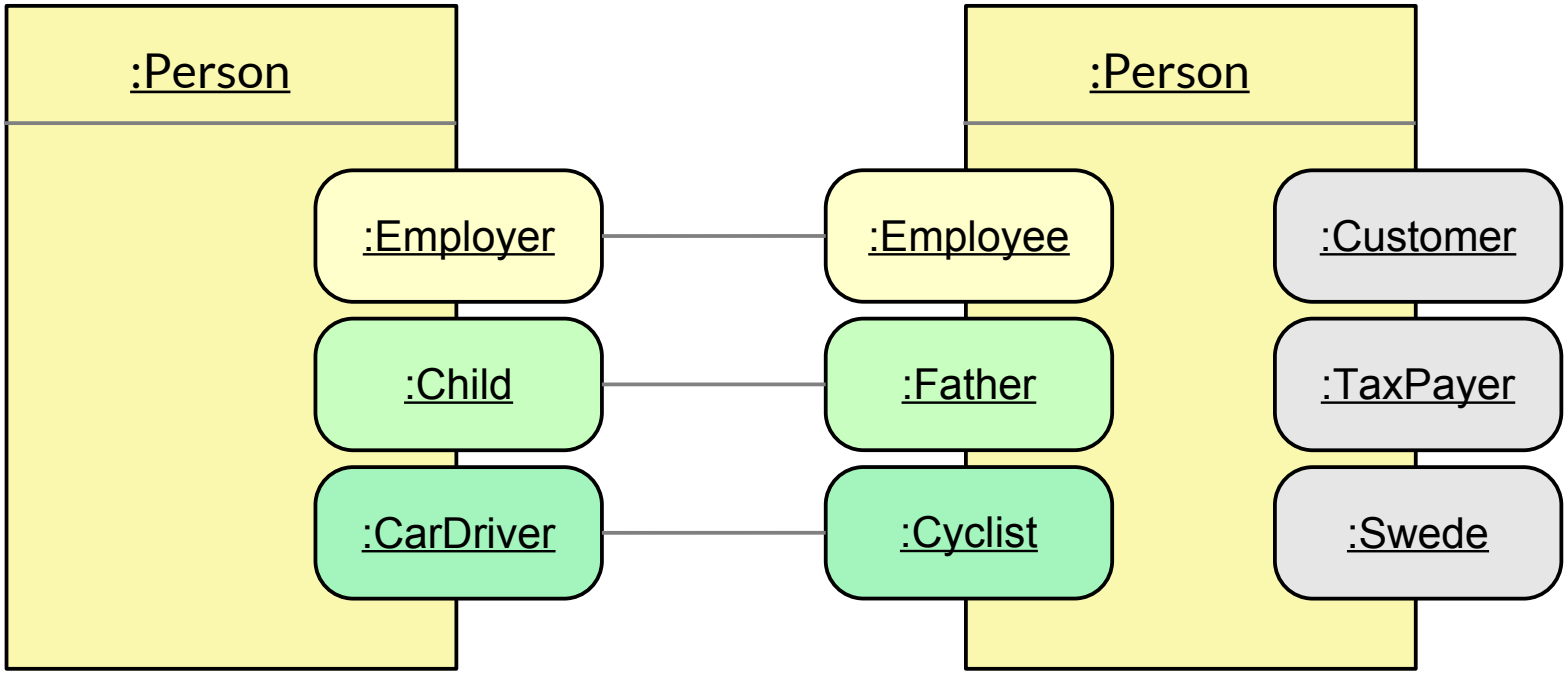
What are Roles? (Rpt.)

Roles are *services* of an object in a context

- Roles can be connected to each other
- A role has an *interface*

Roles form *role models*, capturing an area of *concern* [Reenskaug]

- Role models are *collaborative aspects*



What are Role Types? (Rpt.)

Role types (abilities) are

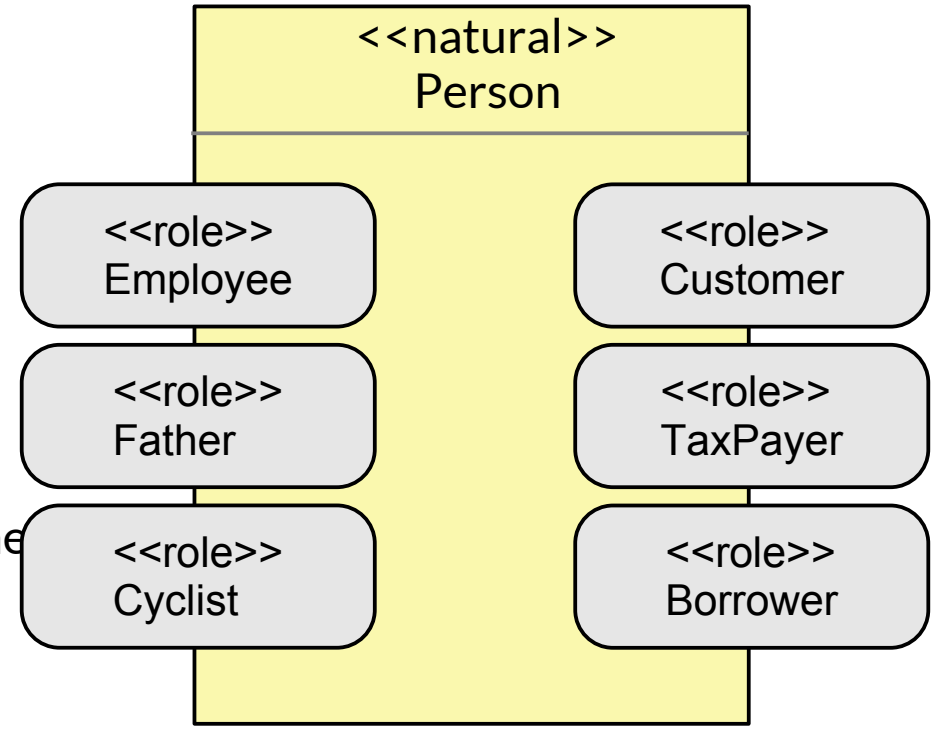
- *service types*
- *dynamic types*
- *collaborative types*

Roles are context-sensitive

Natural classes are context-free

Problem:

- The word “role” is also used on the class level, i.e., for a “role type”



Collaboration Schemas (Role-Type Model) (Rpt.)

Collaboration schema (role type model, ability model):

- Set of object collaborations abstracted by a set of role types
- A constraint specification for classes and object collaborations

Ex: A figure can play many roles in different *collaboration schemas*