

# 31. Data Integration of Tools by Role-Based Composition of Materials (Role-Based Metamodel Composition on M2) for Tool Interoperability on M1-Models and M0-Repositories

Prof. Dr. Uwe Aßmann

Mirko Seifert, Christian Wende

Technische Universität Dresden

Institut für Software- und Multimediatechnik

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

Version 15-1.4, 16.01.16

## 1) Motivational Example

Proactive vs. Retroactive Tool  
Integration

## 2) Roles in Metaclasses

## 3) Role-based composition of metamodels

## 4) Role-Based Composition of Metamodels with RoleCore

## 5) LanGems

## 6) CROM from RoSI (external)



DRESDEN  
concept  
nexus  
aus  
sellschaft  
Kultur



# Obligatory Literature

- ▶ 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
- ▶ Course “Design Patterns and Frameworks” (chapter about role modeling)
- ▶ <http://www.langems.org>



- ▶ <http://www.emftext.org/language/rolecore>

**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>.

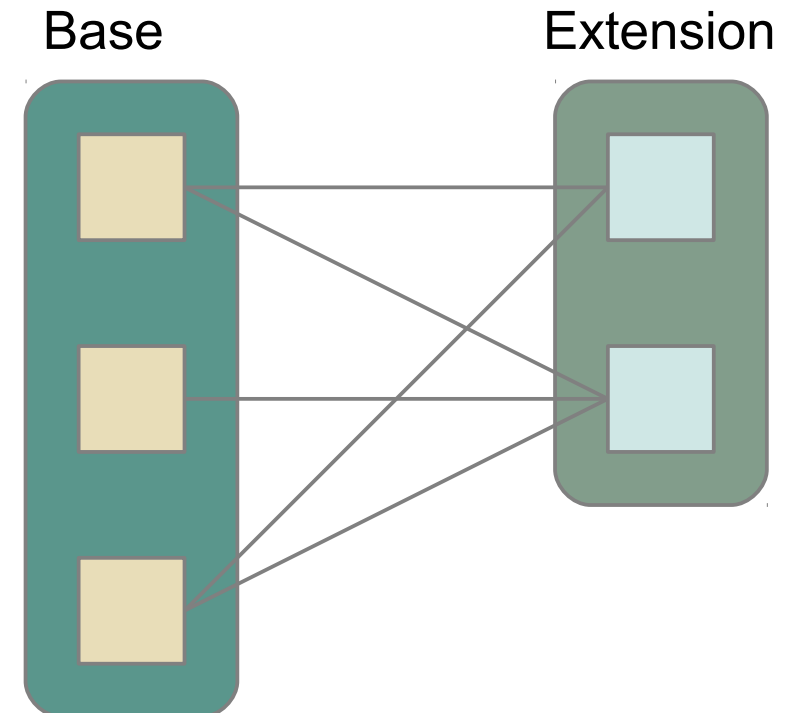
- Role-based Architectural language: Smart Apps (SMAPPs) and Smart Application Grids (SMAGs)
  - Under development by Christian Piechnick
- C. Piechinick, S. Richly, S. Götz, C. Wilke, U. Aßmann. Using Role-Based Composition to Support Unanticipated, Dynamic Adaptation – Smart Application Grids. Adaptive and Self-adaptive Systems and Applications (Adaptive 2012)

- 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](http://www.reuseware.org)
- Birgit Grammel. **Automatic Generation of Trace Links in Model-driven Software Development.** PhD thesis, Technische Universität Dresden, Fakultät Informatik, February 2014

- ▶ 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 (Datenteilung)

# 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
    - ELI, fnc-2, LISA, Silver, JastAdd
  - 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 by 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]



# 31.1 Motivational Example for Data Sharing in Tool Integration

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

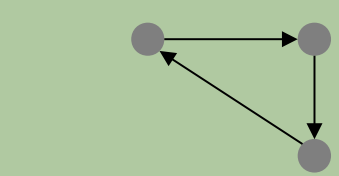
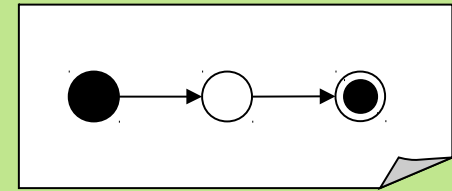
DDL: state machines

Textual State Machine Editor

```
HelloWorld.statemachine x
1 StateMachine HelloWorld {
2   initial state init;
3
4   state first {
5     do : "greet"
6   };
7
8   final state end {
9     do : "goodbye"
10  };
11
12  transitions {
13    init -> first when "step";
14    first -> end when "step";
15  }
16}
```

DDL: visualization concepts

2D Shape Renderer

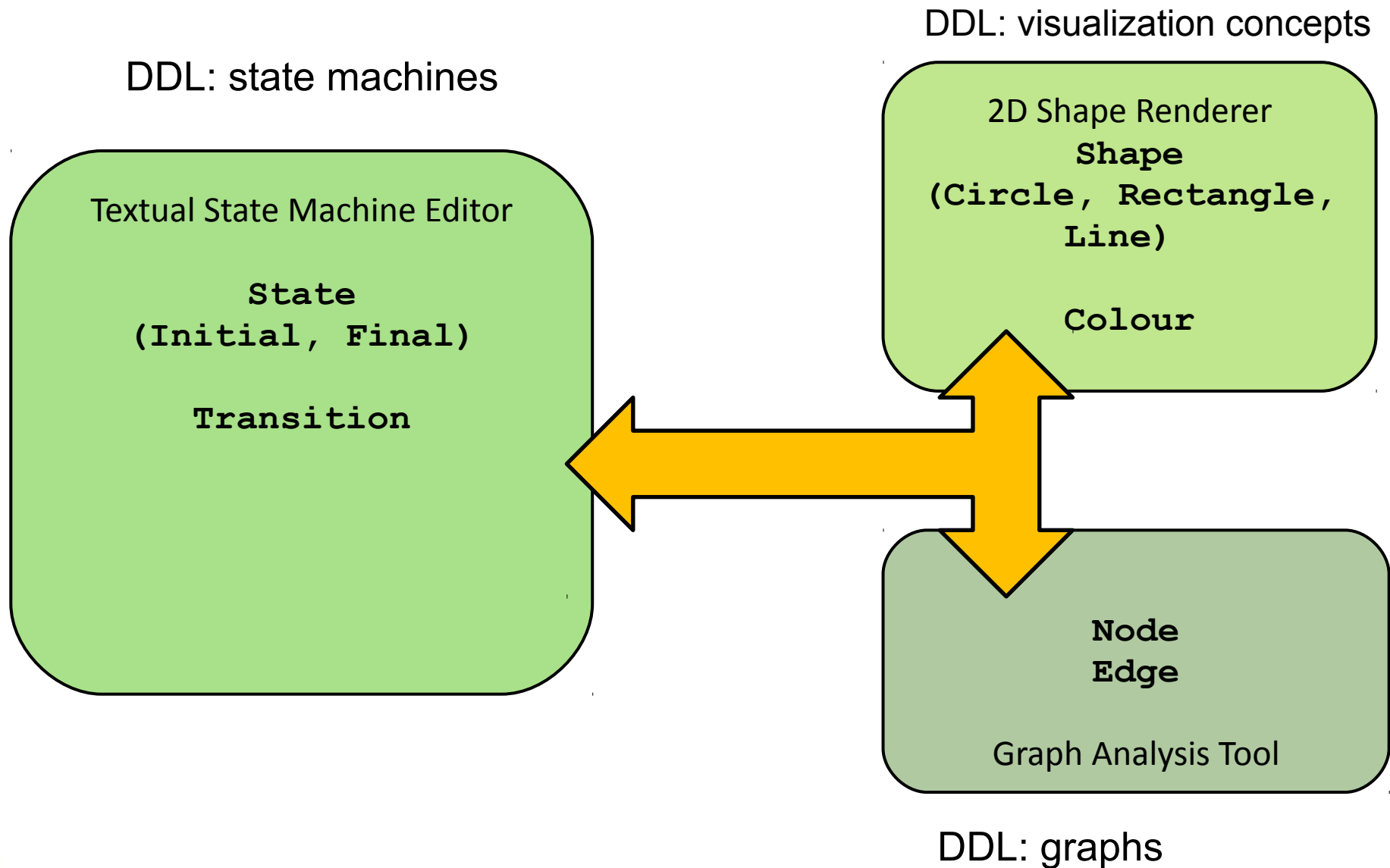


DDL: graphs

Graph Analysis Tool

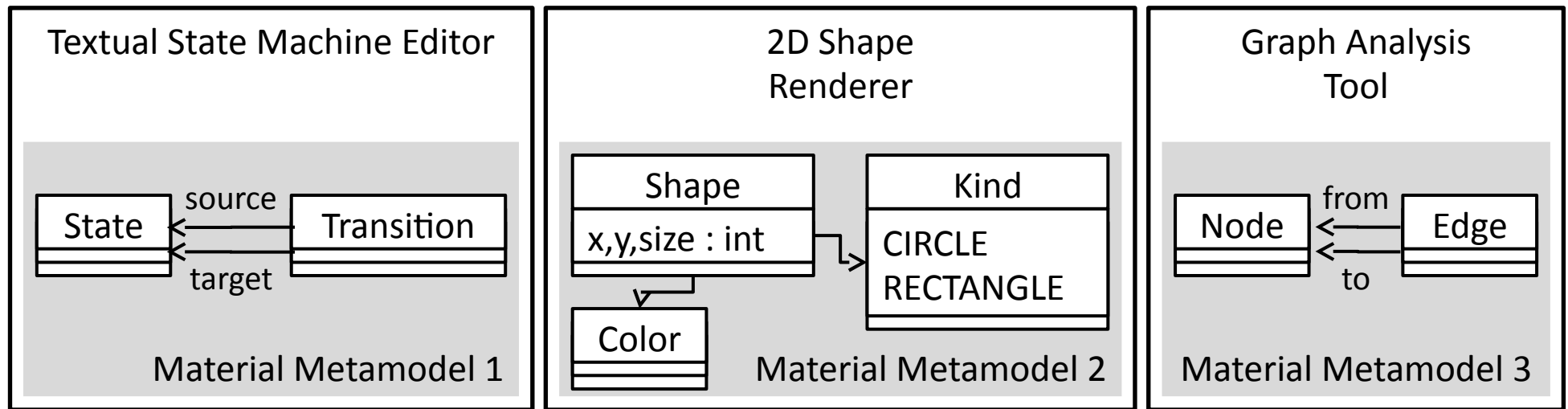
# Example – Language Concepts in Metamodels of the Involved Tools

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



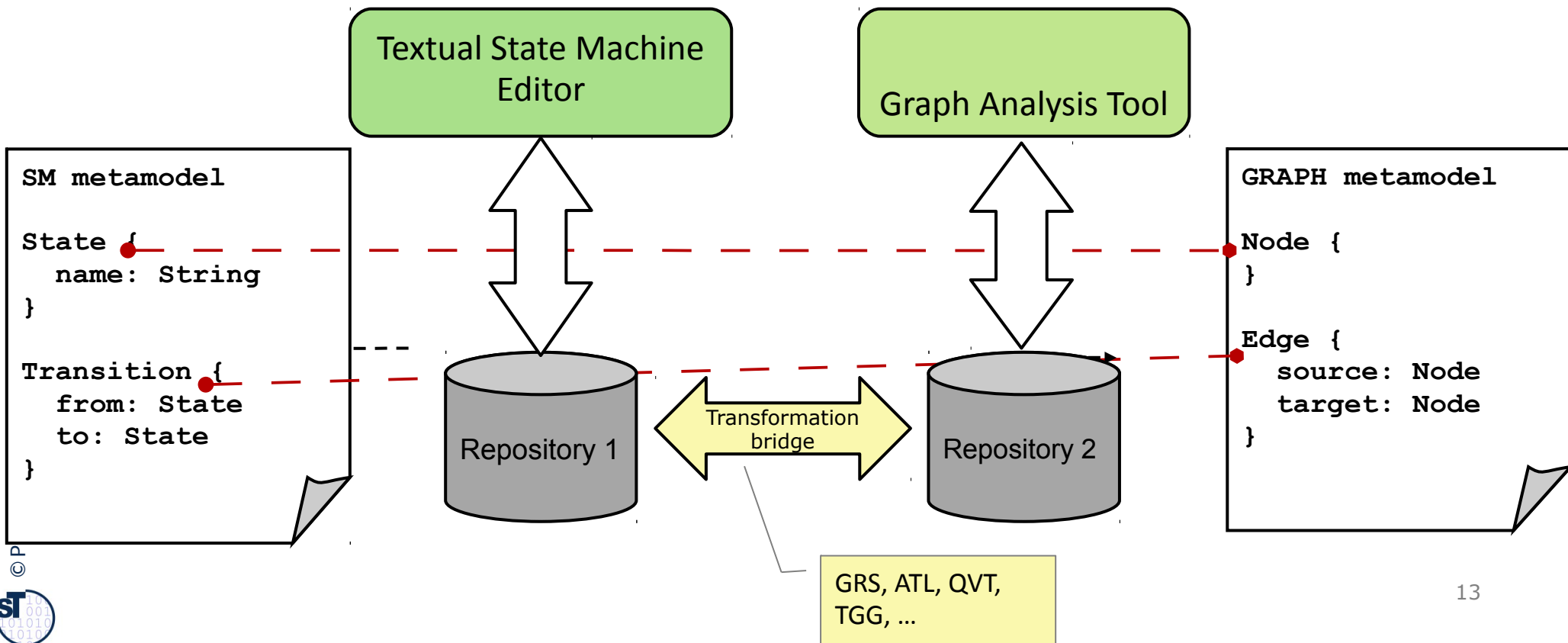
# How Can these Metamodels of Materials be Integrated?

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



# Retroactive Tool Integration on Repositories by Data Connection

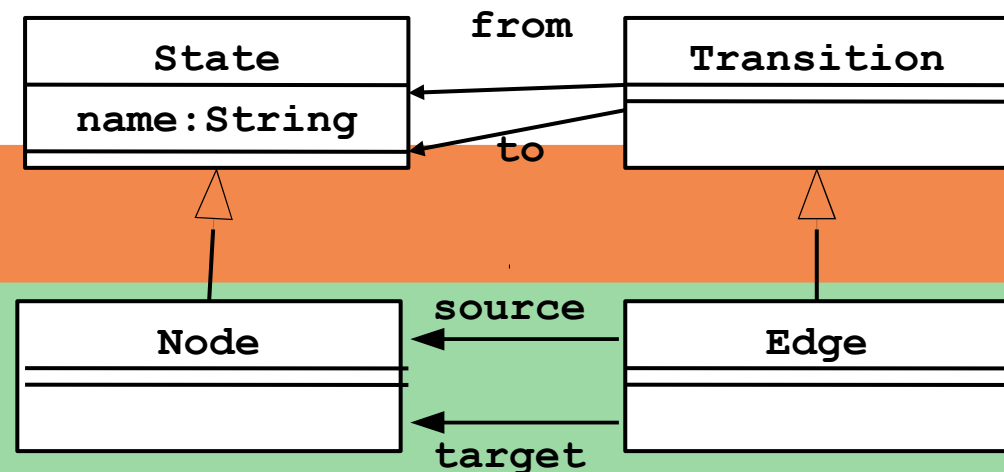
- ▶ Often, tools, their metamodels, and the metamodel-driven repositories already exist
- ▶ **Metamodel mapping (language mapping):** map the concepts of one DDL to the other
- ▶ Use transformations to convert data from one tool to another (data exchange via transformation bridge, Datenverbindung)



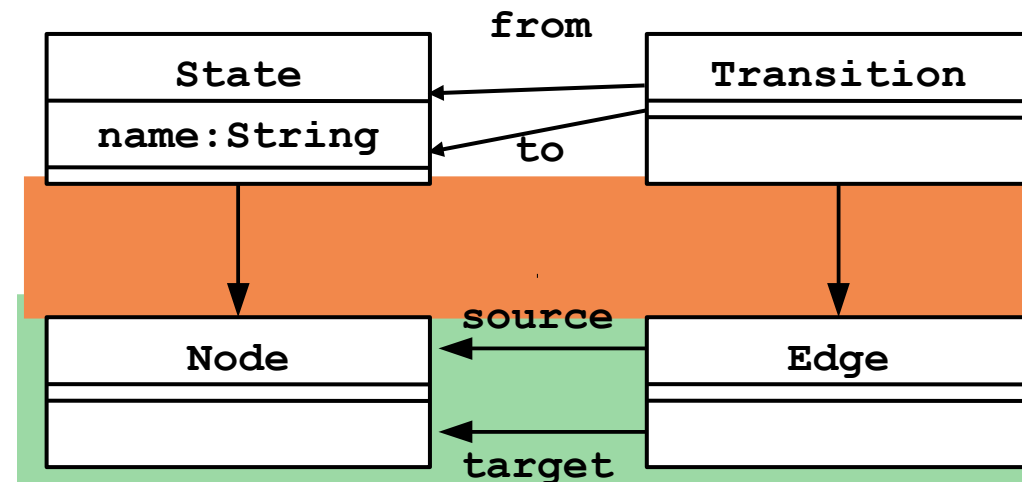
# Classic Proactive Tool Integration

- ▶ Sometimes, metamodels and repositories are not fixed yet and can be integrated
- ▶ Use **metamodel extension (integration)** to make data 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

## a) „White-Box“ Inheritance













## b) „Black-Box“ Delegation





# Proactive vs. Retroactive Tool Integration

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

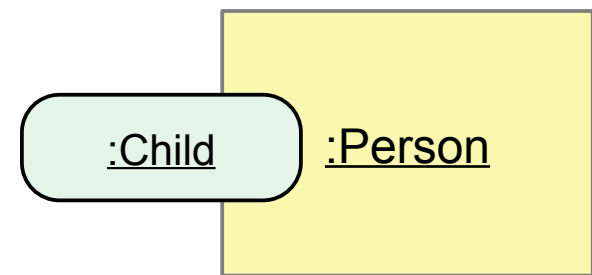
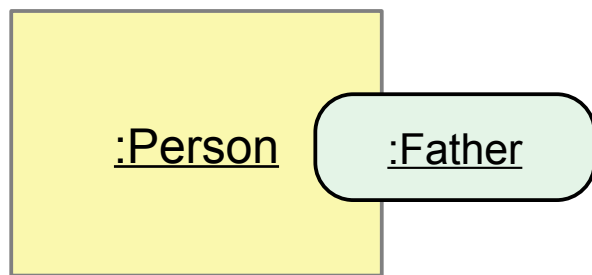
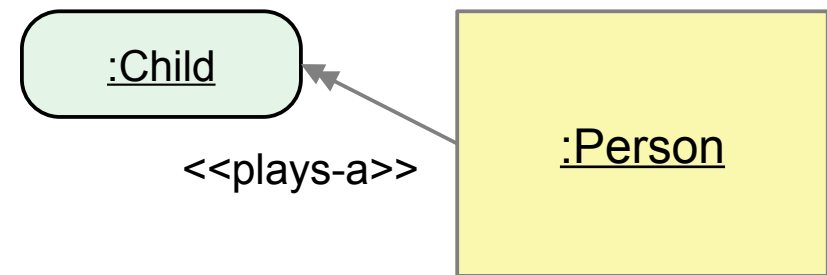
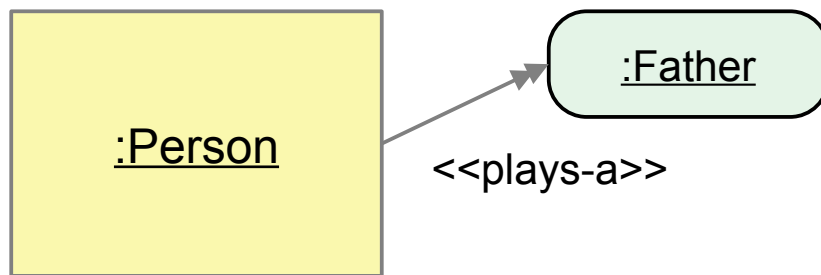
## 31.2 Roles in Models, Metamodels, and Metalanguages



# 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]





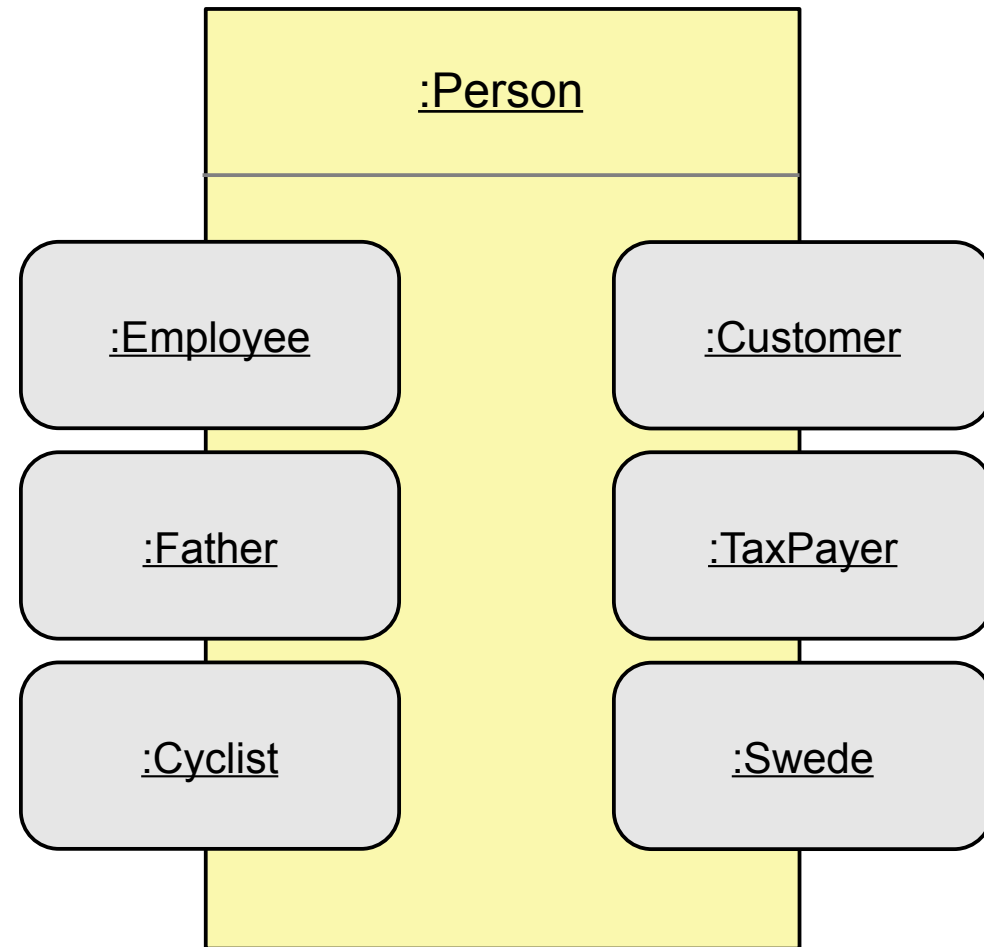
# What are Roles? (Rpt.)

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



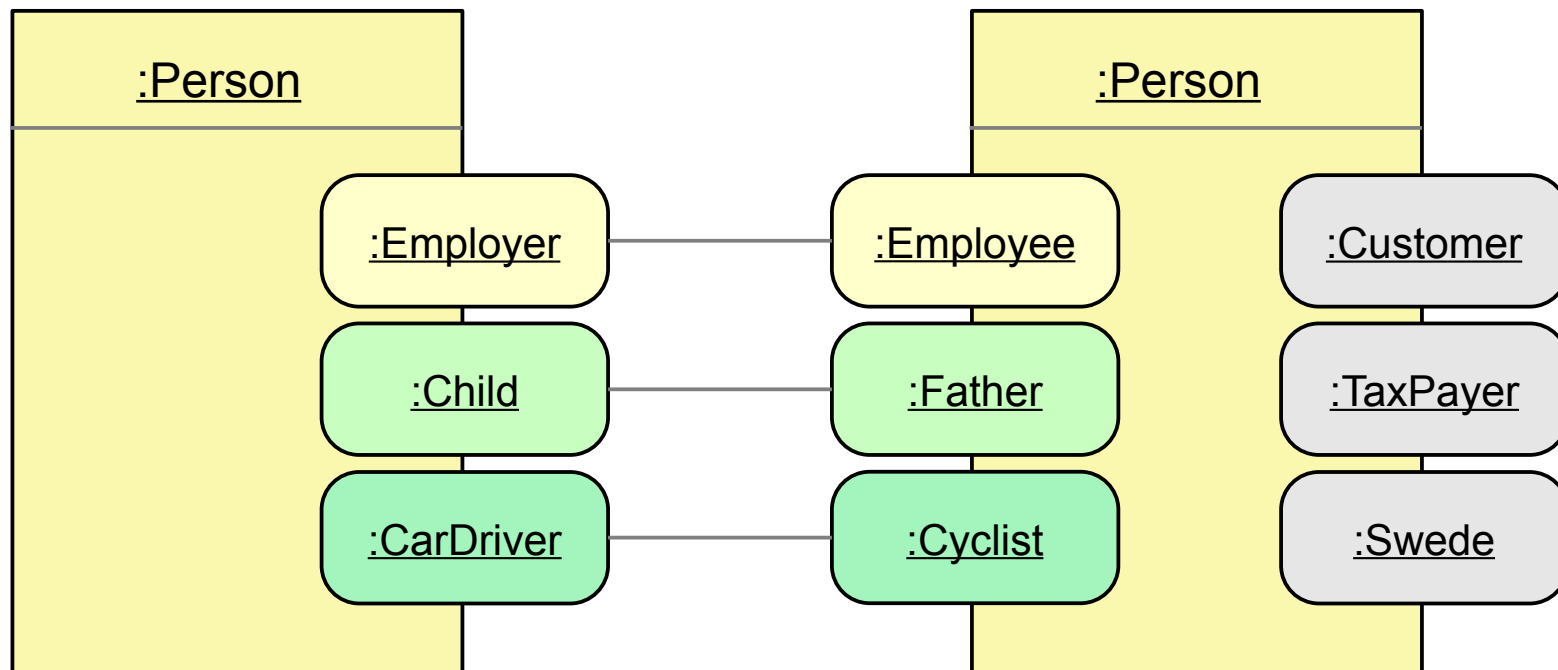
# What are Roles? (Rpt.)

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

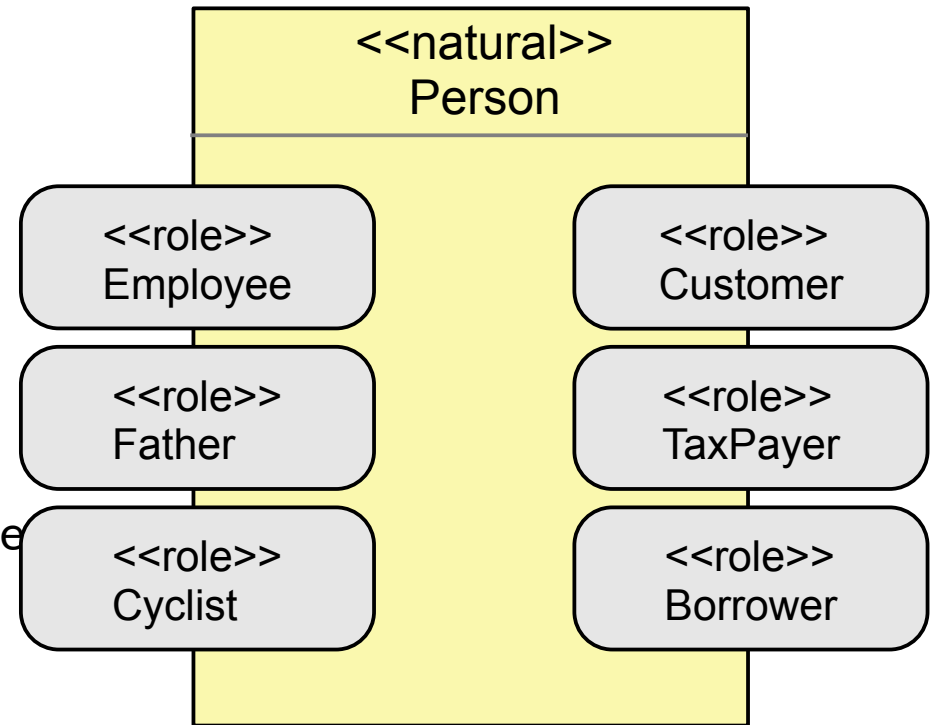
- *service types*
- *dynamic types*
- *collaborative types*

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”

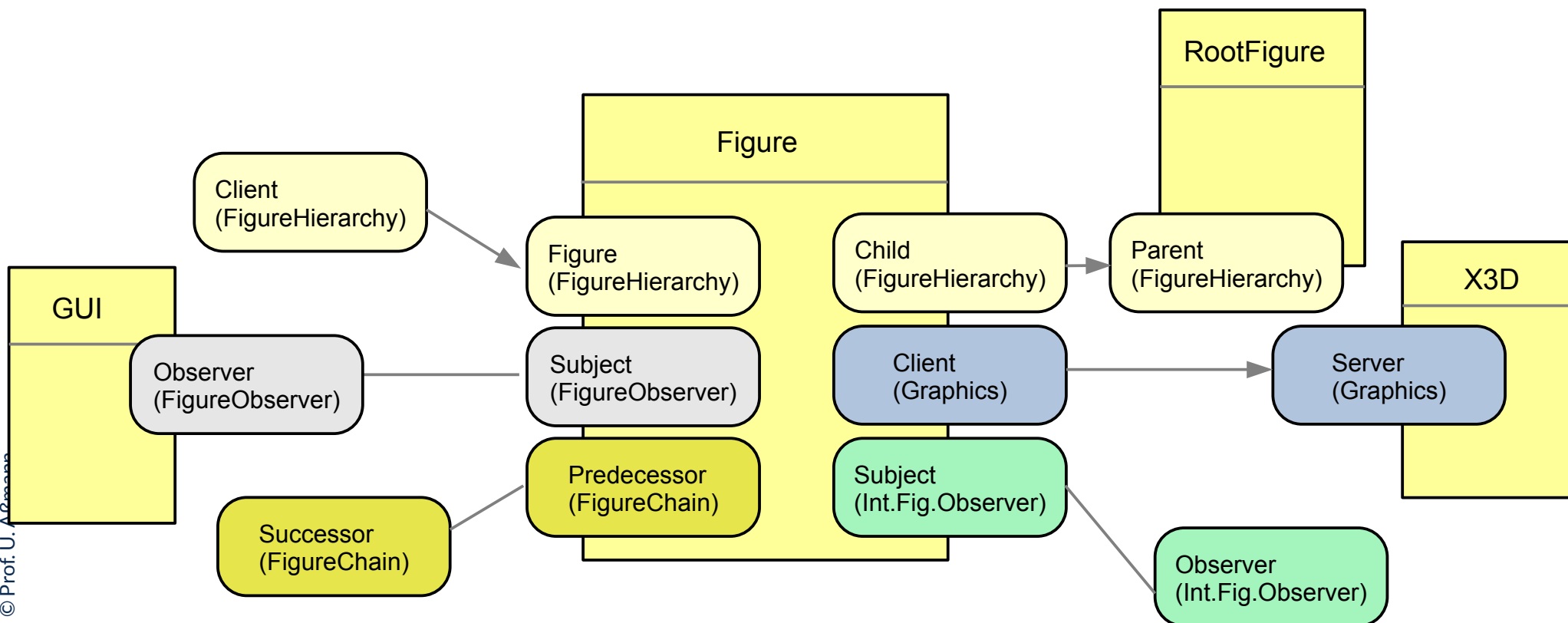


# Collaboration Schemas (Role-Type Model) (Rpt.)

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*



# Role- and Role-Type Models Underly Many Gray-Box Component Models

## Views

- Hyperspace (MDSOC)
- Orthographic Software Modeling

## Collaborative Aspects

- ObjectTeams [www.objectteams.org](http://www.objectteams.org)
- CaesarJ

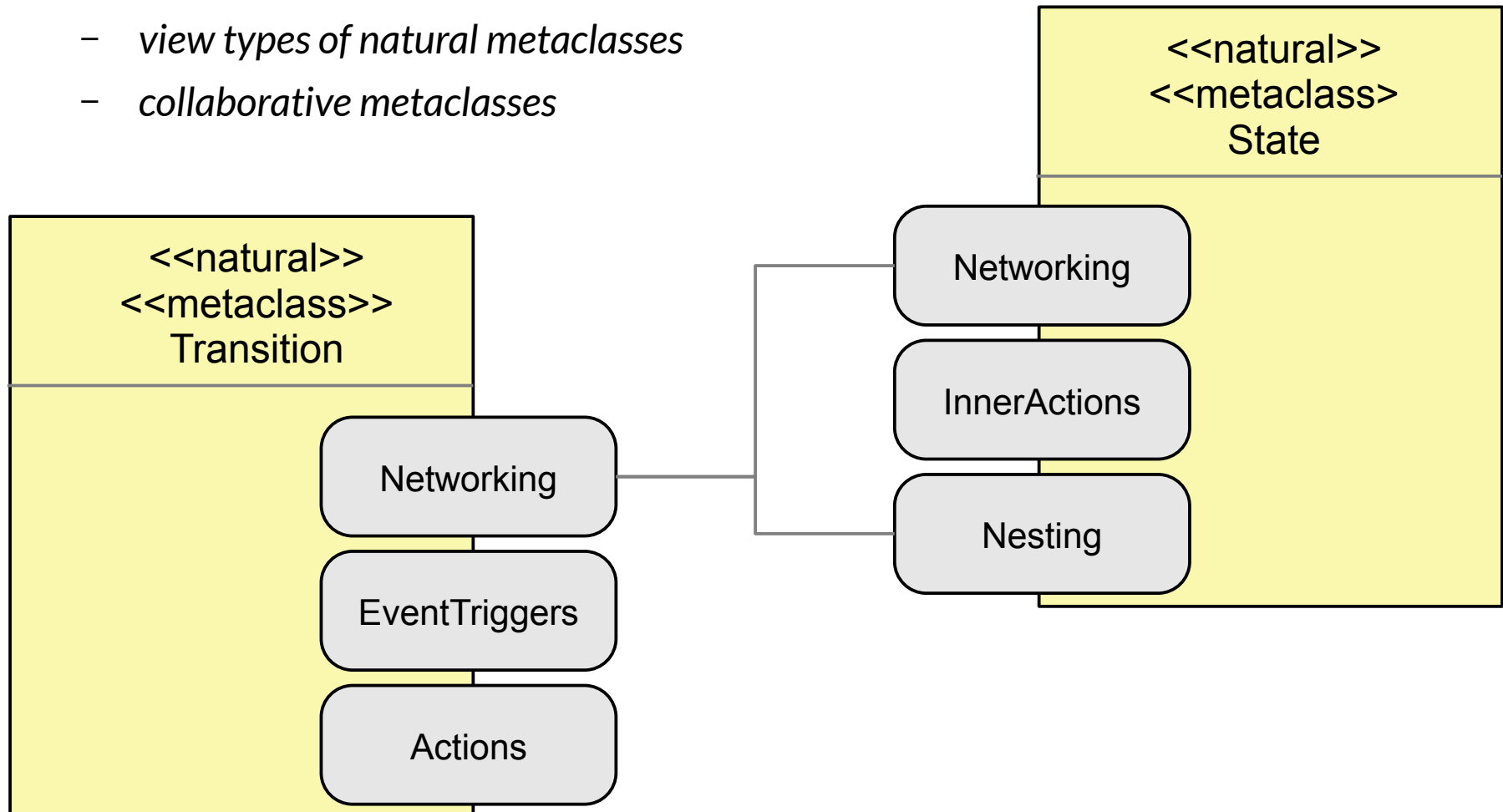
## Template-based languages

- BETA with the metaprogramming environment Mjölner
- Invasive Software Composition

# What are Role Metaclasses (on M2)?

Role metaclasses are

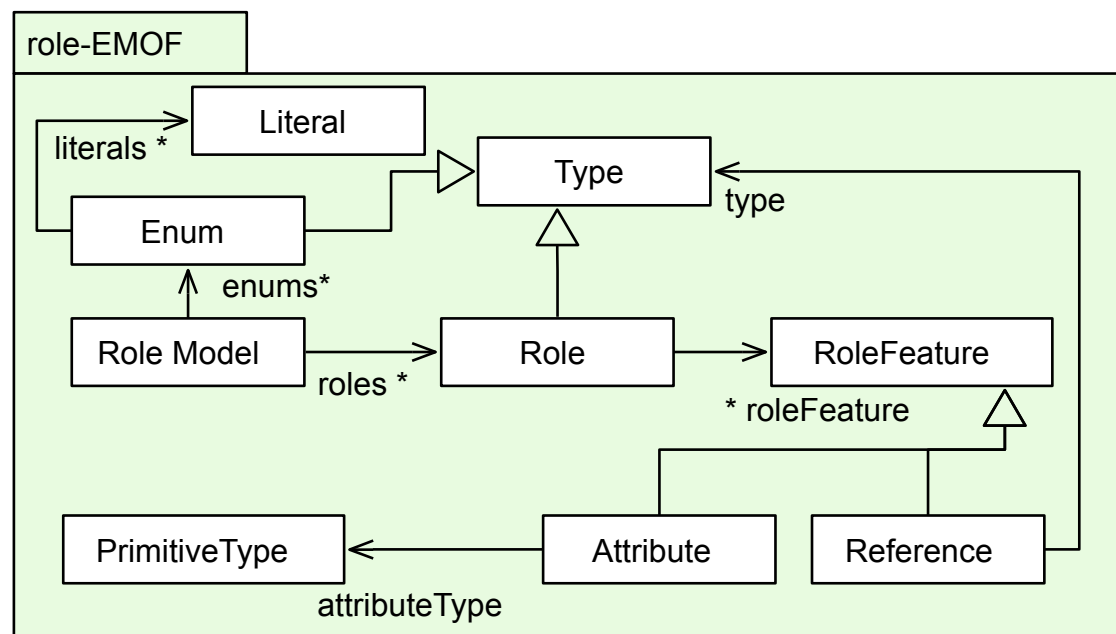
- *service types of natural metaclasses*
- *view types of natural metaclasses*
- *collaborative metaclasses*



# 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:

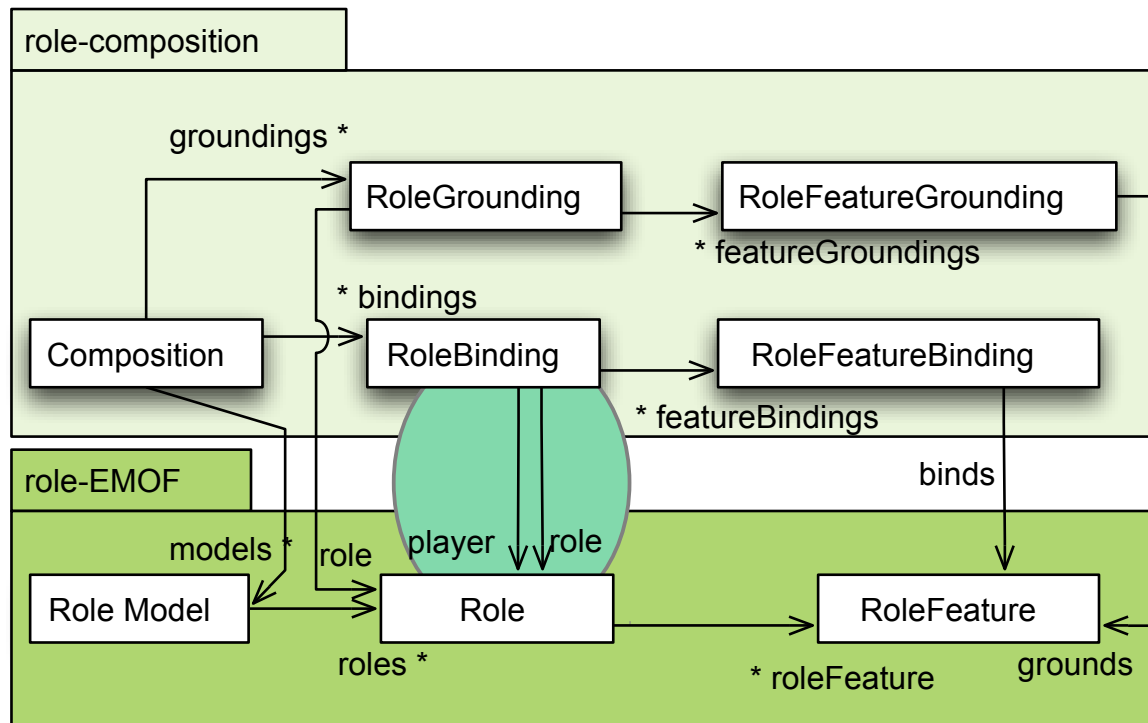
**M3**



# “Deep-Role-EMOF”, a Metamodel for Deep Role Composition

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

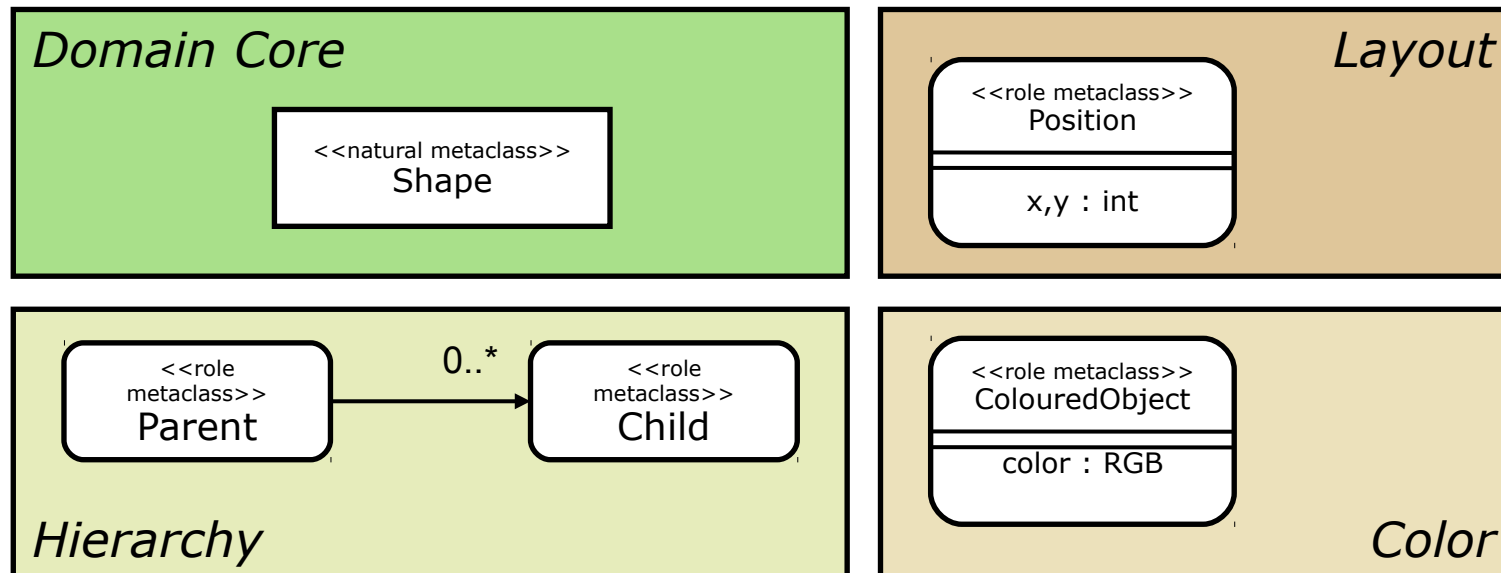
M3





# Example: The Material Metamodel of Tool ShapeRenderer with Roles

- ▶ Roles of Material Metaclasses adhere to a context
  - A context is a specific *concern* (here: colors)
- ▶ Only one natural metaclass, many role metaclasses



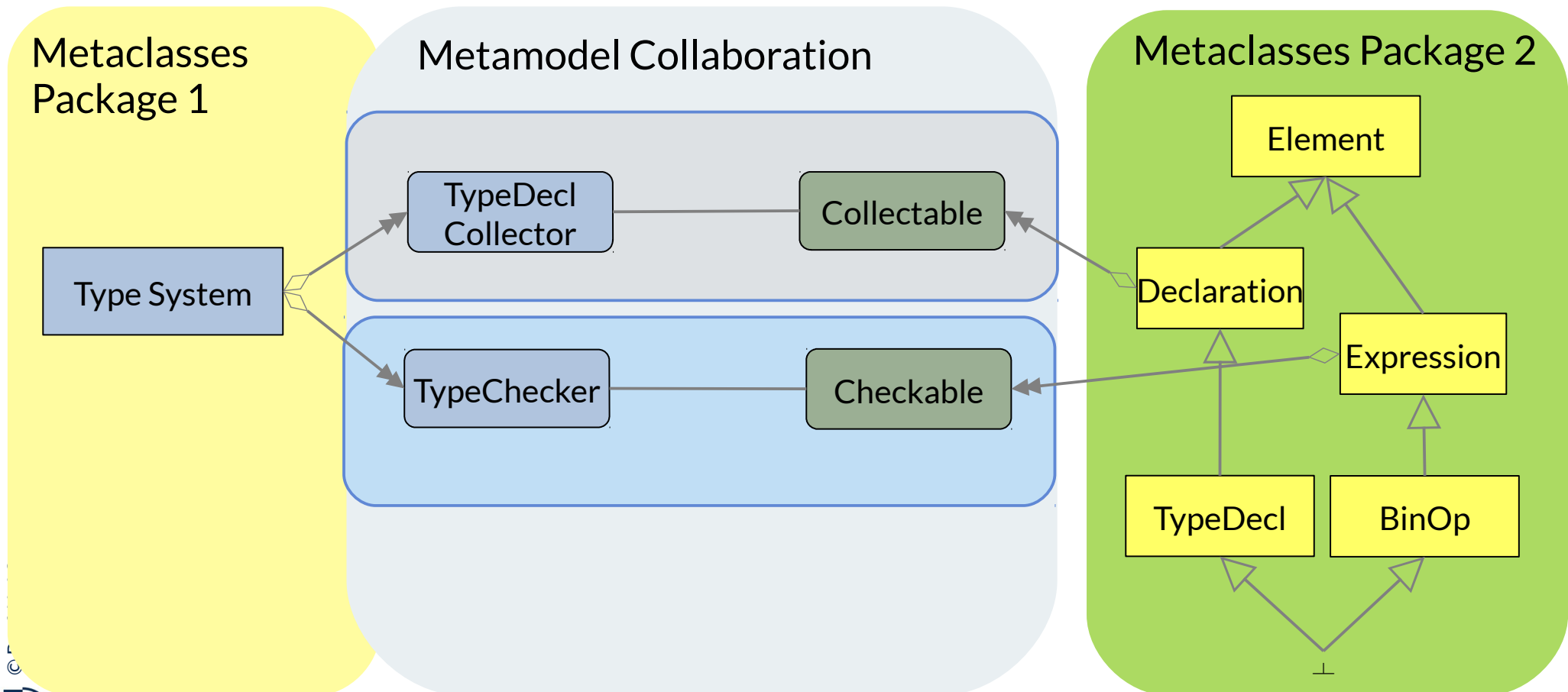


# 31.3. Role-Based Language Composition



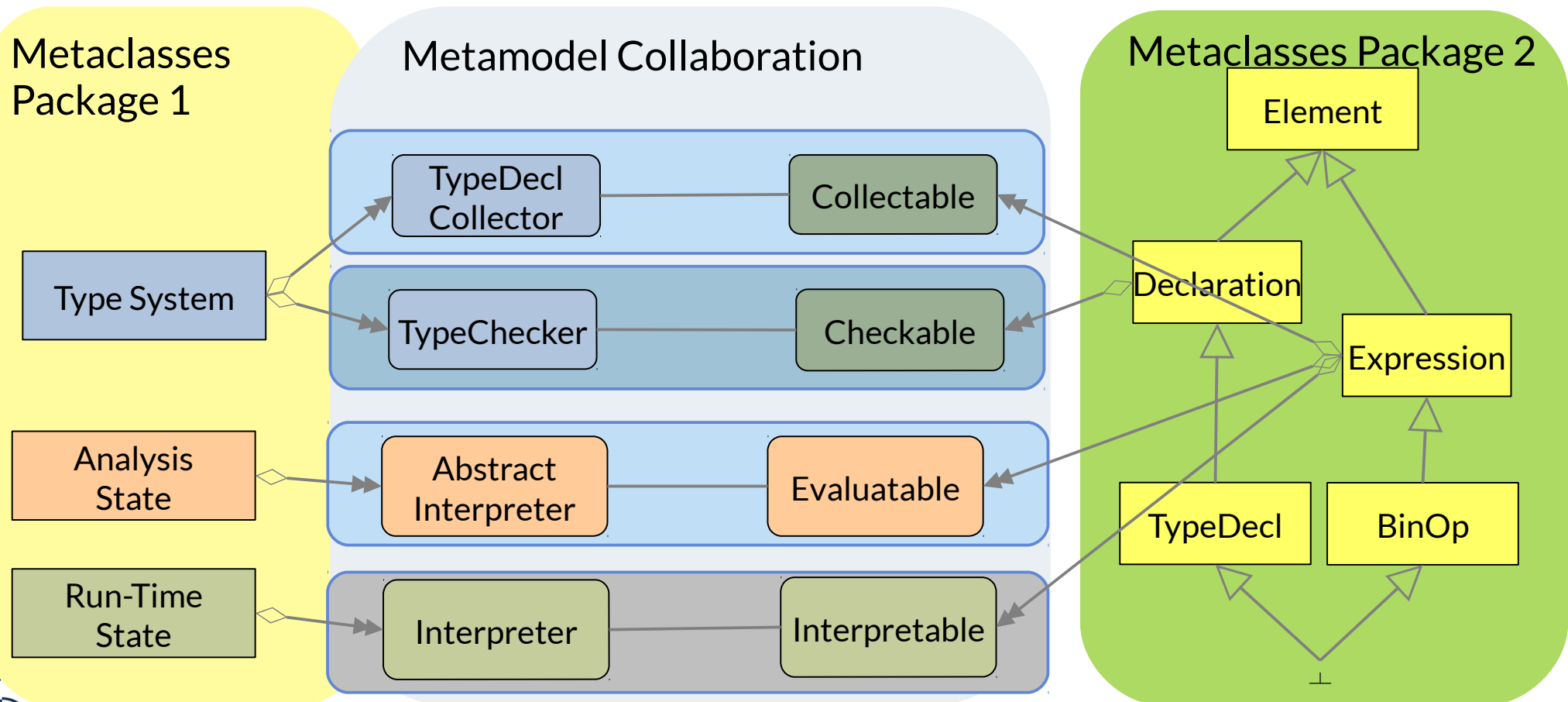
# Good News: Role-Based Extension of Metaclass Hierarchies is Simple

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



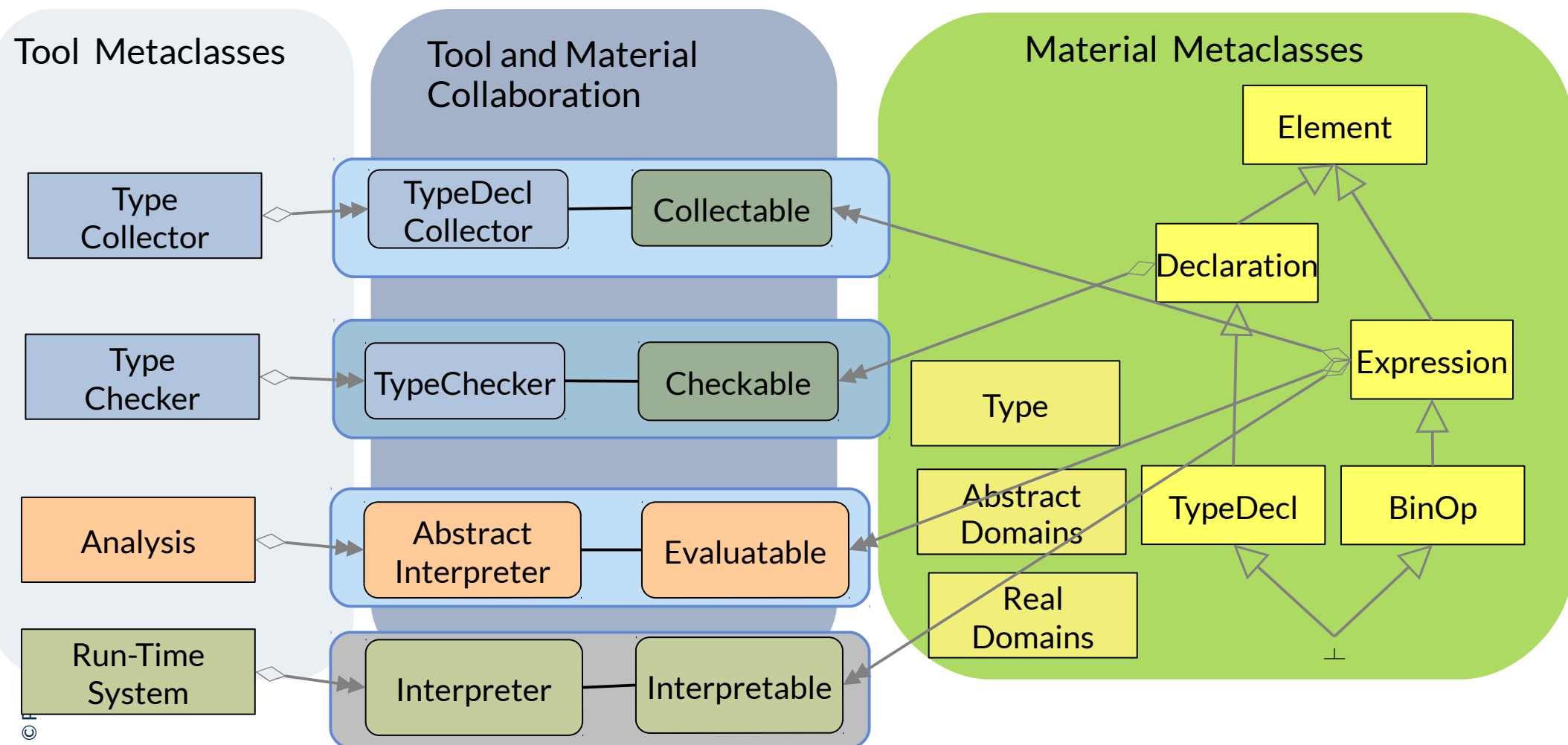
# Good News: Role-Based Extension of Metaclass Hierarchies with New Metaclass Collaborations is Simple

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



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

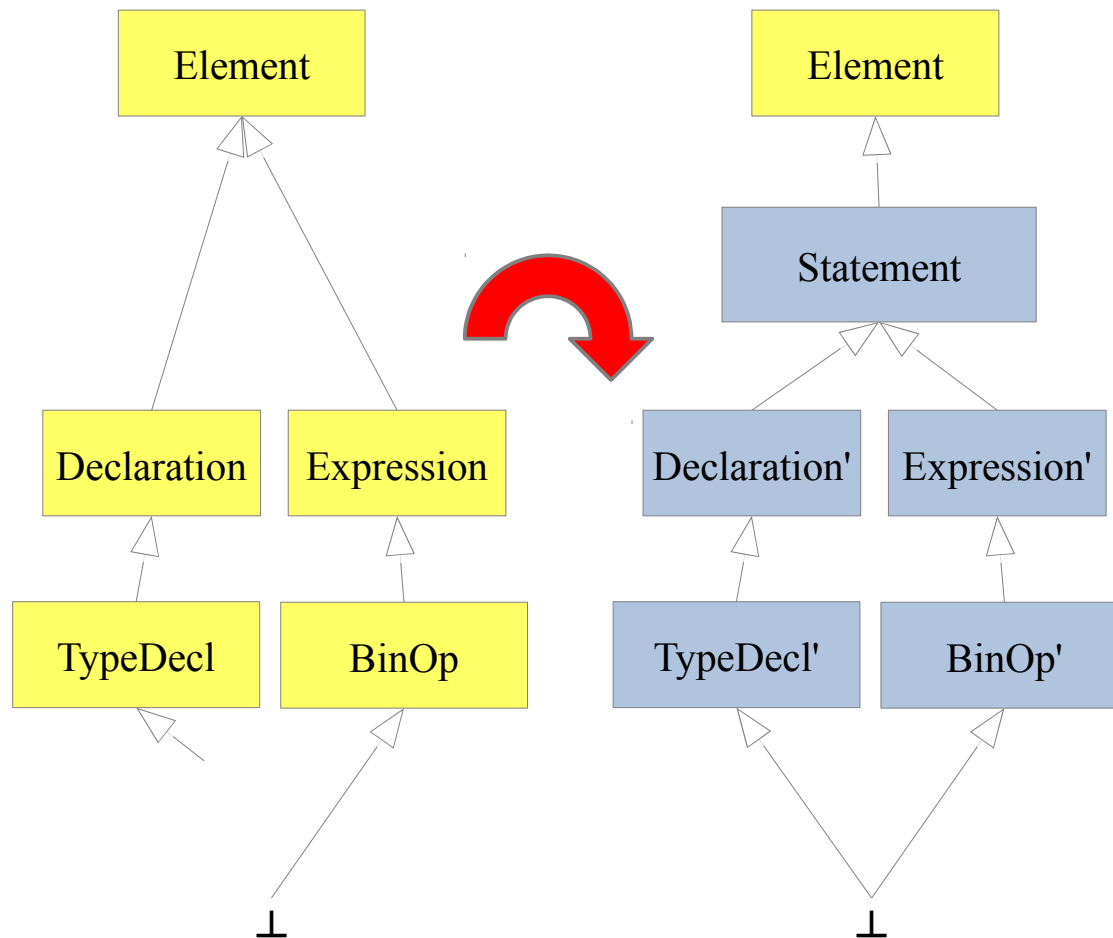
- Given a metaclass hierarchy, metaclass roles can be added in new tool-material collaborations



# Bad News: Superimposition of Entity Natural Superclasses Stays HARD

Identity of all derived subclasses changes

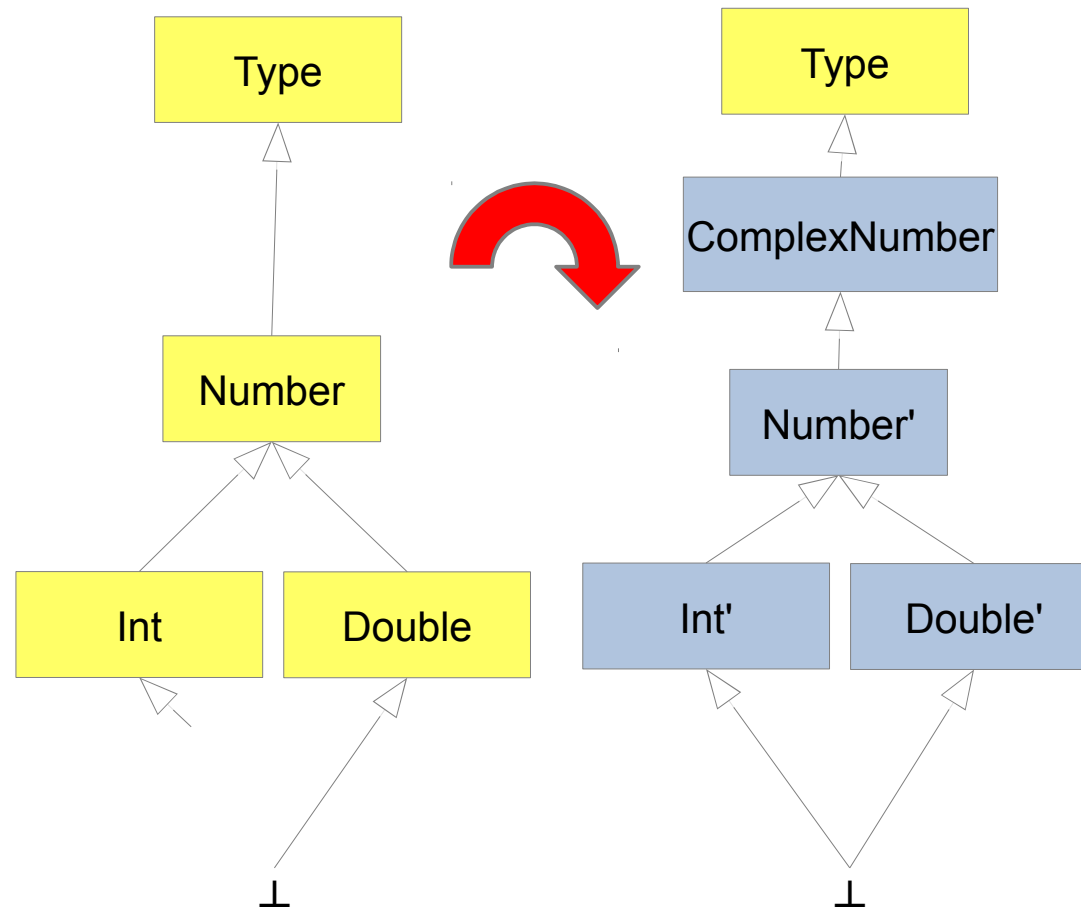
- Declaration --> Declaration' under-a Statement
- Expression --> Expression' under-a Statement



# Example: Complex Numbers

Superimposing a new concept ComplexNumber to a type hierarchy is an extension of the entity (natural) concepts of a language

- Due to identity change, type rules for all Numbers have to be changed [van Wyk, JLE application]



## 31.4 Rolecore: a DSL for Metamodel Integration with Deep Role Metaclasses

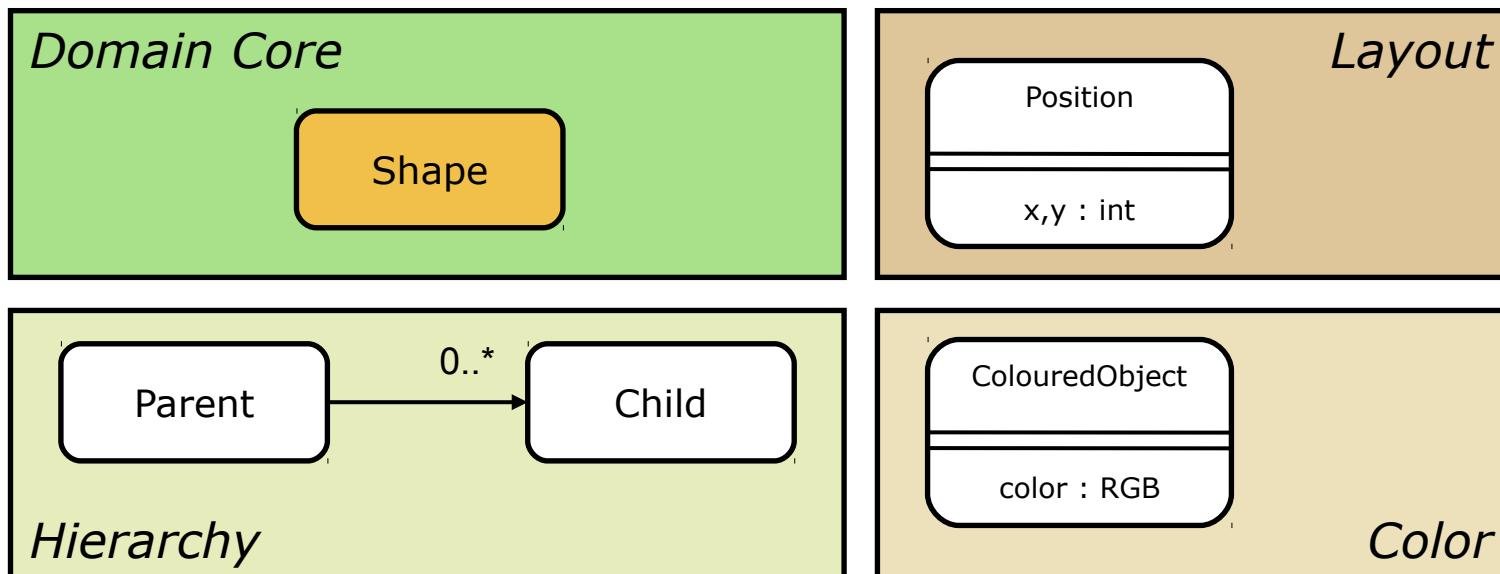
- ▶ Rolecore is a domain-specific language (DSL), with a preprocessor generating Java (standard language)
  - Employs Role-Object Pattern for roles in the generated code
  - Maximal runtime flexibility, but slow
- ▶ Developed by Christian Wende and Mirko Seifert





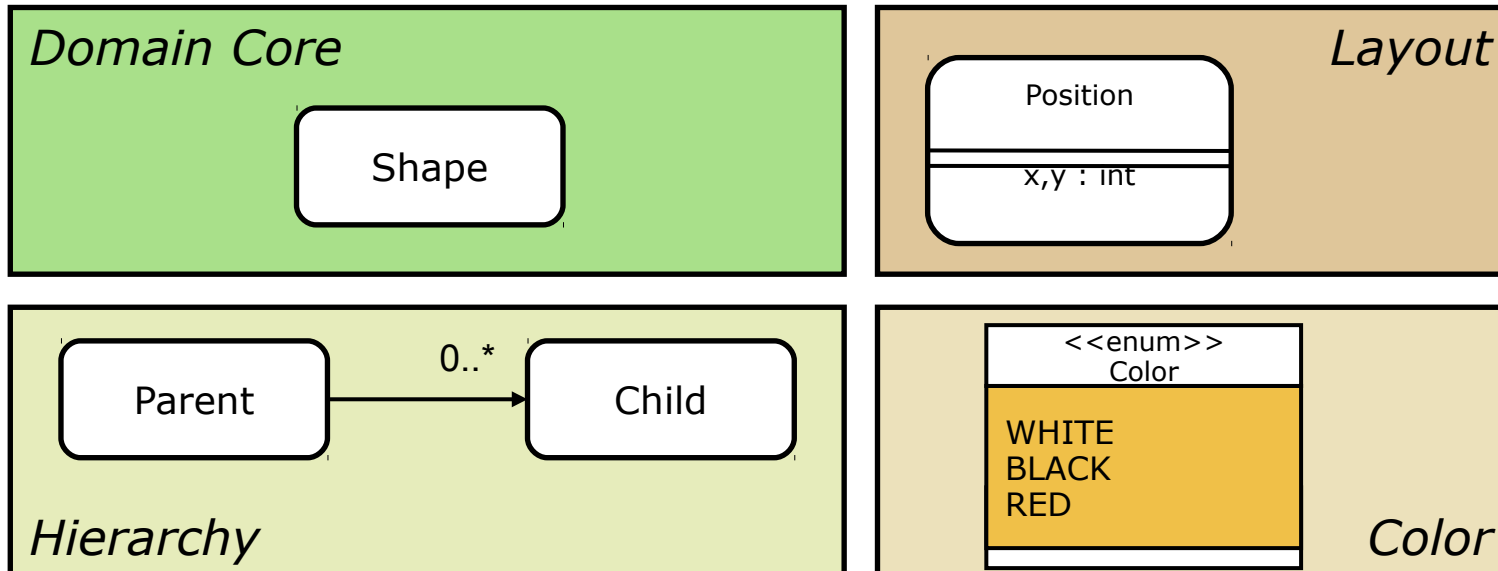
# Example: ShapeRenderer's Metamodel with Deep Roles

- ▶ The Rolecore-DSL is a textual DSL for 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: Tool ShapeRenderer's Material Metamodel in RoleCore, with Deep Roles and Enums

- ▶ In RoleCore, some roles can be represented as enums if their attributes have finite value ranges
- ▶ Then they will become natural classes in the implementation



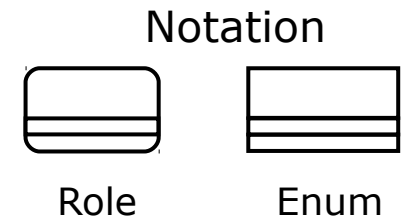
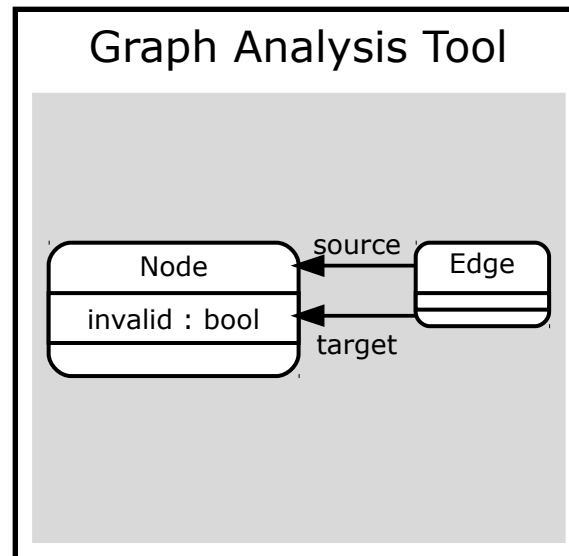
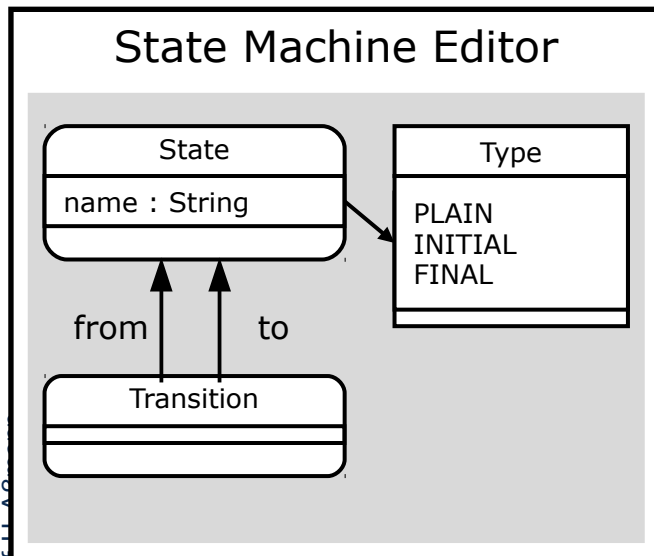


## 31.4.1 Proactive Material Integration with Deep Roles in RoleCore




# 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:
  - 1) First specify everything as deep role
  - 2) Select those roles which should become enums
  - 3) Naturals are selected last

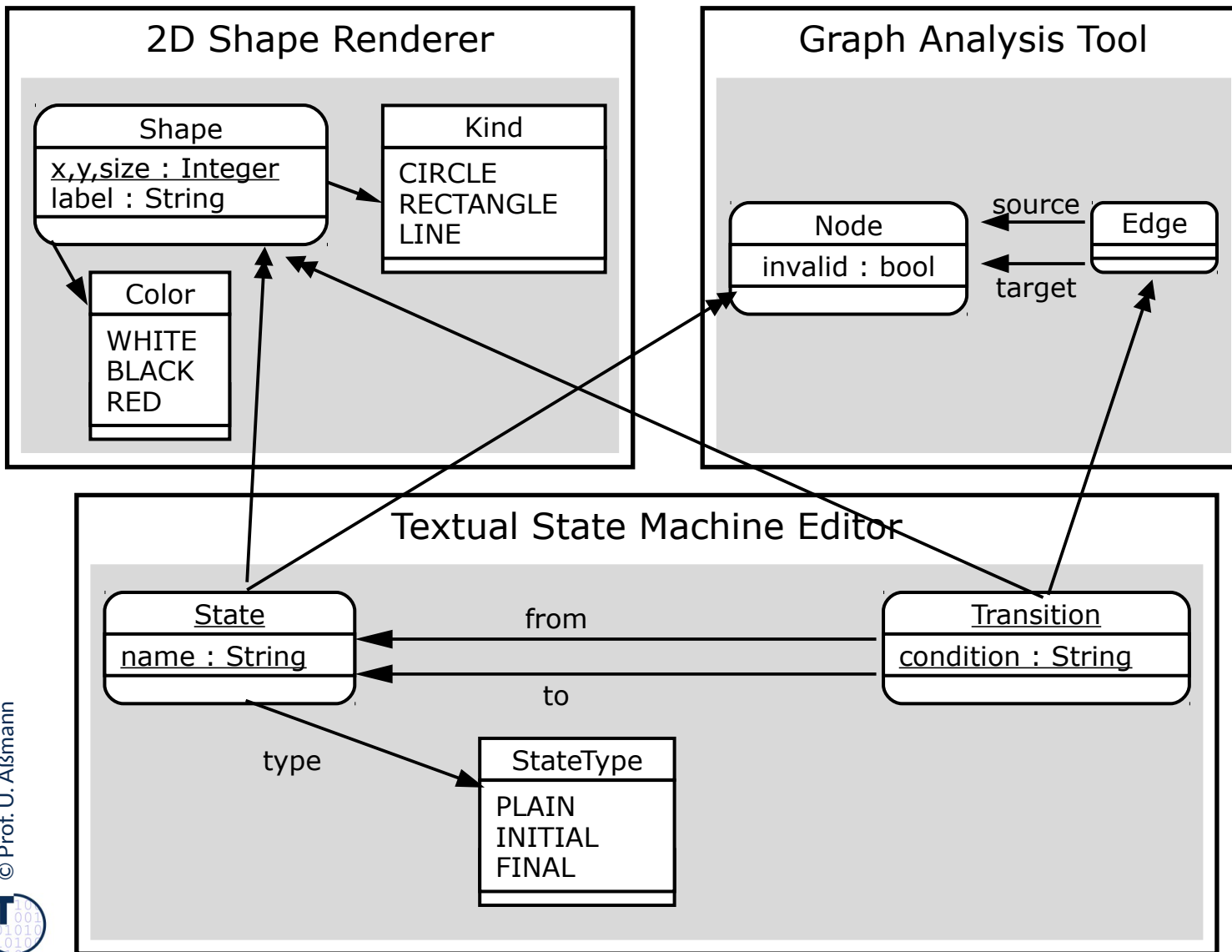


# For Tool Integration, the RoleCore Approach Uses Role Bindings and Role Grounding

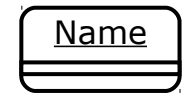
- ▶ **Role Bindings** on the logical level 
  - with relationship “plays-a”
  - Connect roles and role players, producing deep roles
  - Define how to obtain value of attribute or reference
  - Allow to create views on other classes
  
- ▶ **Role Grounding** on the physical level
  - Defines which attributes/classes 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 Binding and role grounding

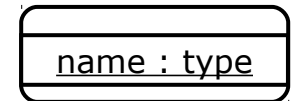
- ▶ Composition by *deep role binding* and *role grounding*
- ▶ We defer the decision “what is a natural” to later



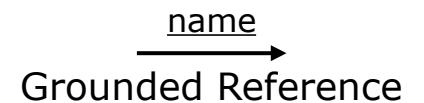
## Grounding Notation



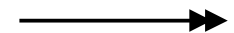
Grounded Role



Grounded Attribute



## Binding Notation



## 31.4.2 Grounding: Mapping Role Metaclasses to Programming Languages



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

- ▶ Role binding (playing) and role grounding can be described by the RoleCore DSL.

```
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  
  }  
}
```

Role Binding  
Specification

Link mapping  
from one role to the  
other

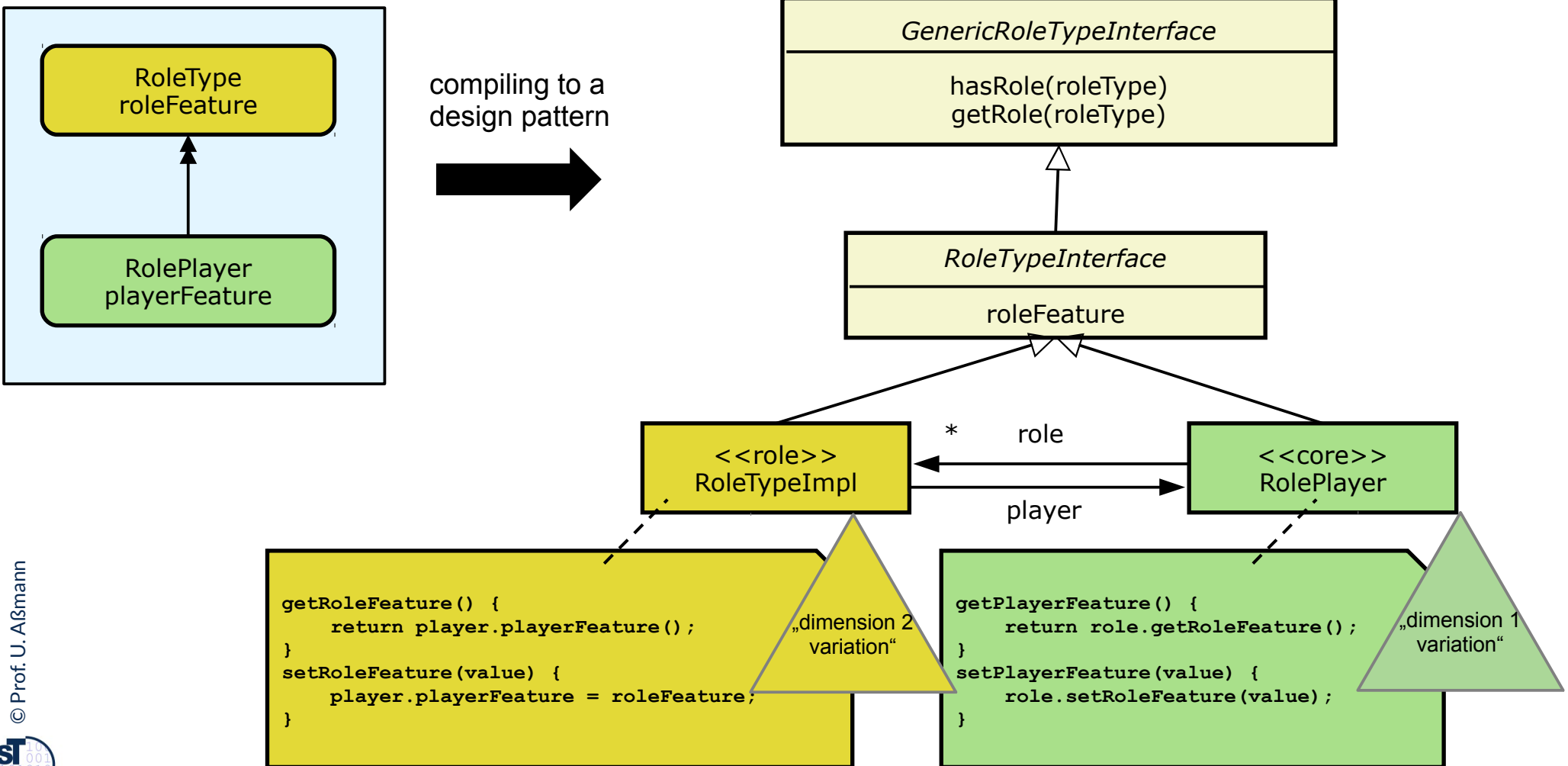
```
ground State { name, type }  
ground Transition { condition, from, to }  
}
```

Grounding  
Specification

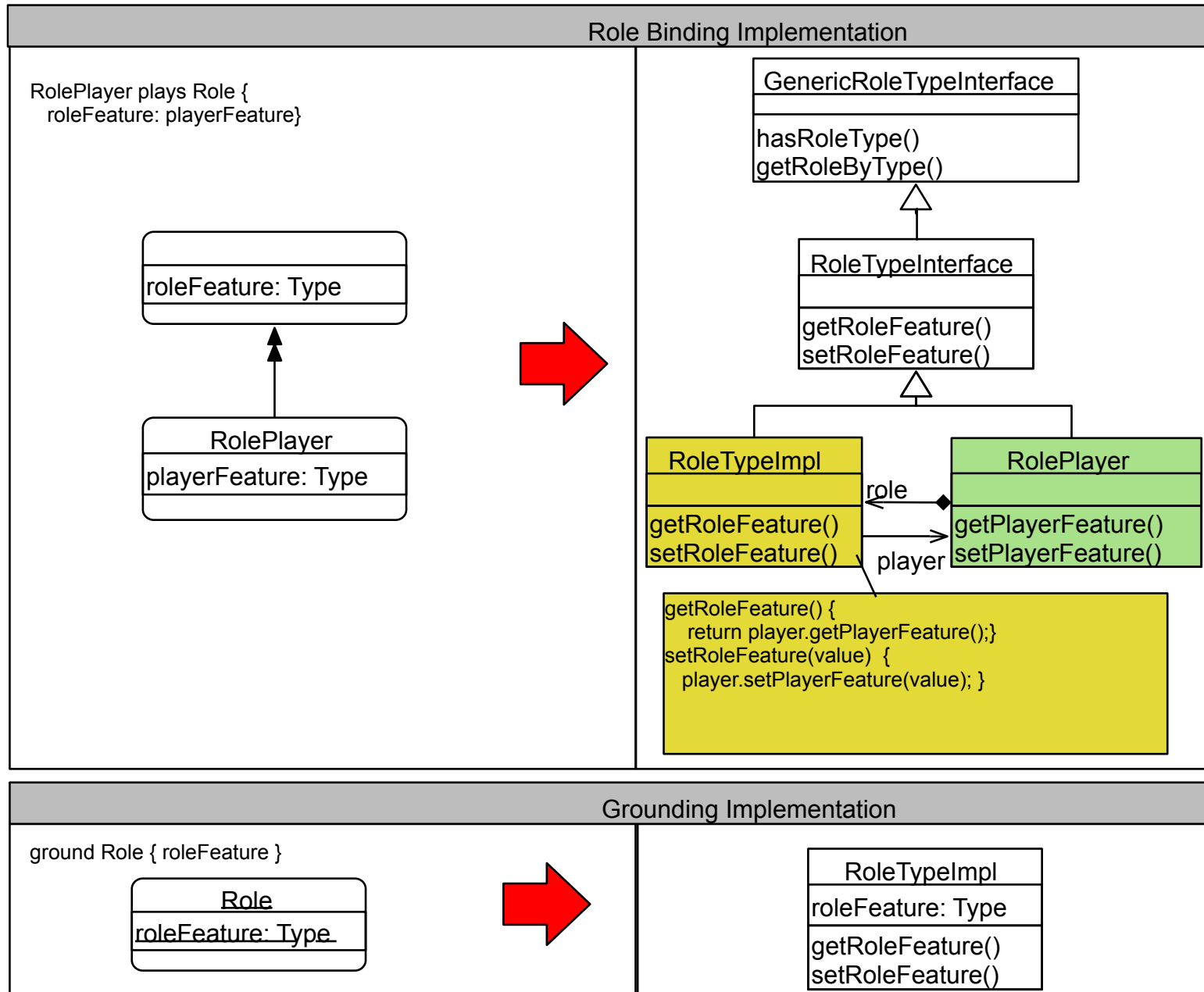


# Role Binding Realisation by e.g., Delegation (Design Pattern “Generative Role Object Pattern”)

- ▶ 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

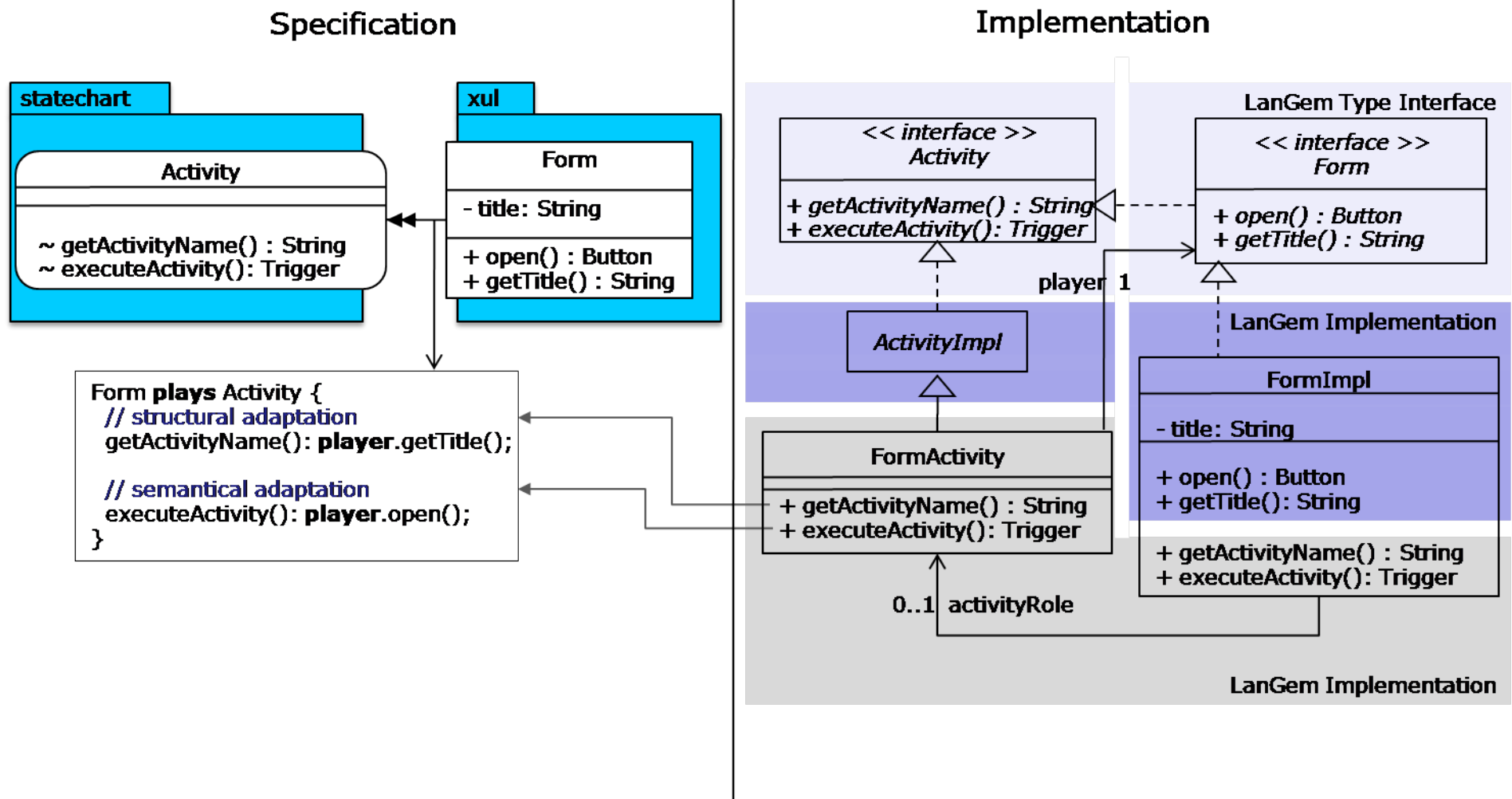


# Role Binding Implementation with Generative Role Object Pattern (ROP)



# Implementation Pattern for Role Superimposition

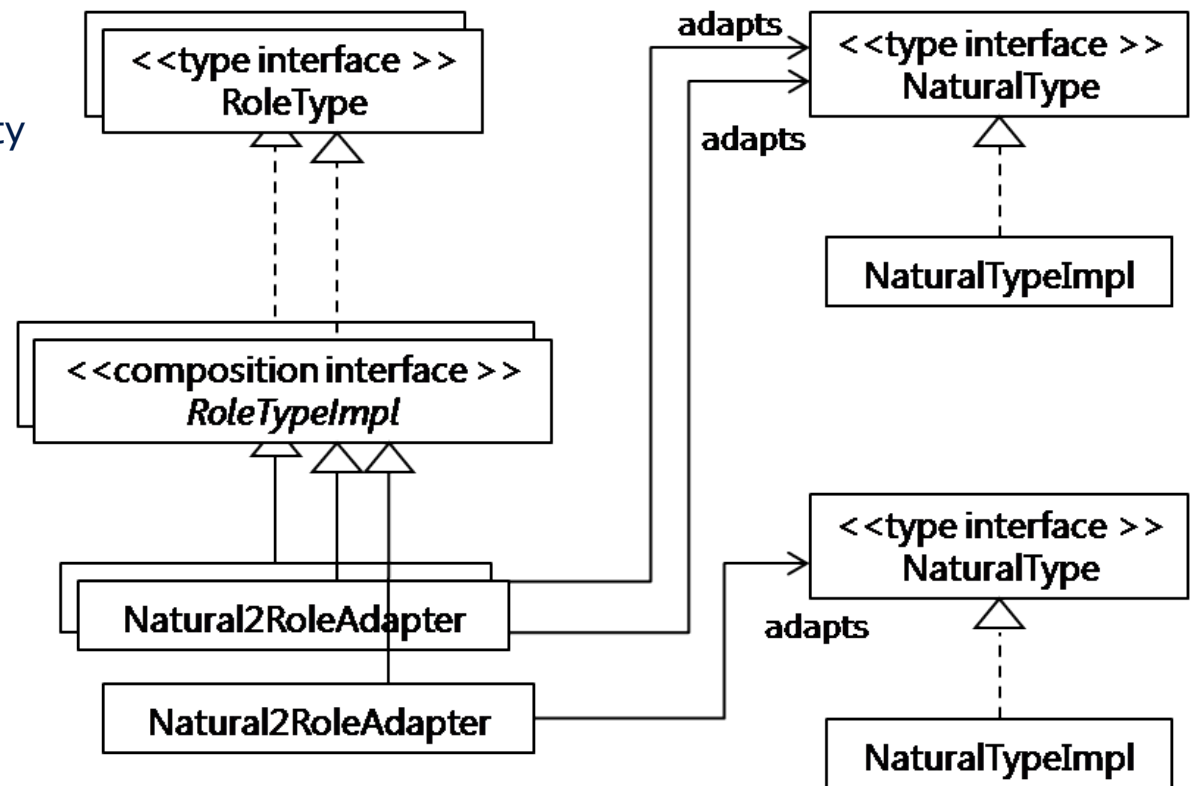
## Generative Role Implementation Pattern



# Implementation Pattern for Role Superimposition

## Characteristics of the Implementation Pattern

- Preserves the type interface of the composed LanGems and, thus, the compatibility with LanGem syntax and semantics
- Encapsulates implementation of individual LanGem
- Adapts type interface of role player to contract (composition interface) of the role
- Allows for binding multiple roles to one natural to superimpose multiple LanGems
- Allows for binding multiple naturals to the same role to provide variability in LanGem composition





## 31.5. Role-Based Language Composition with LanGems



**FeatureMapper**

Mapping Features to Models



**LanGems**

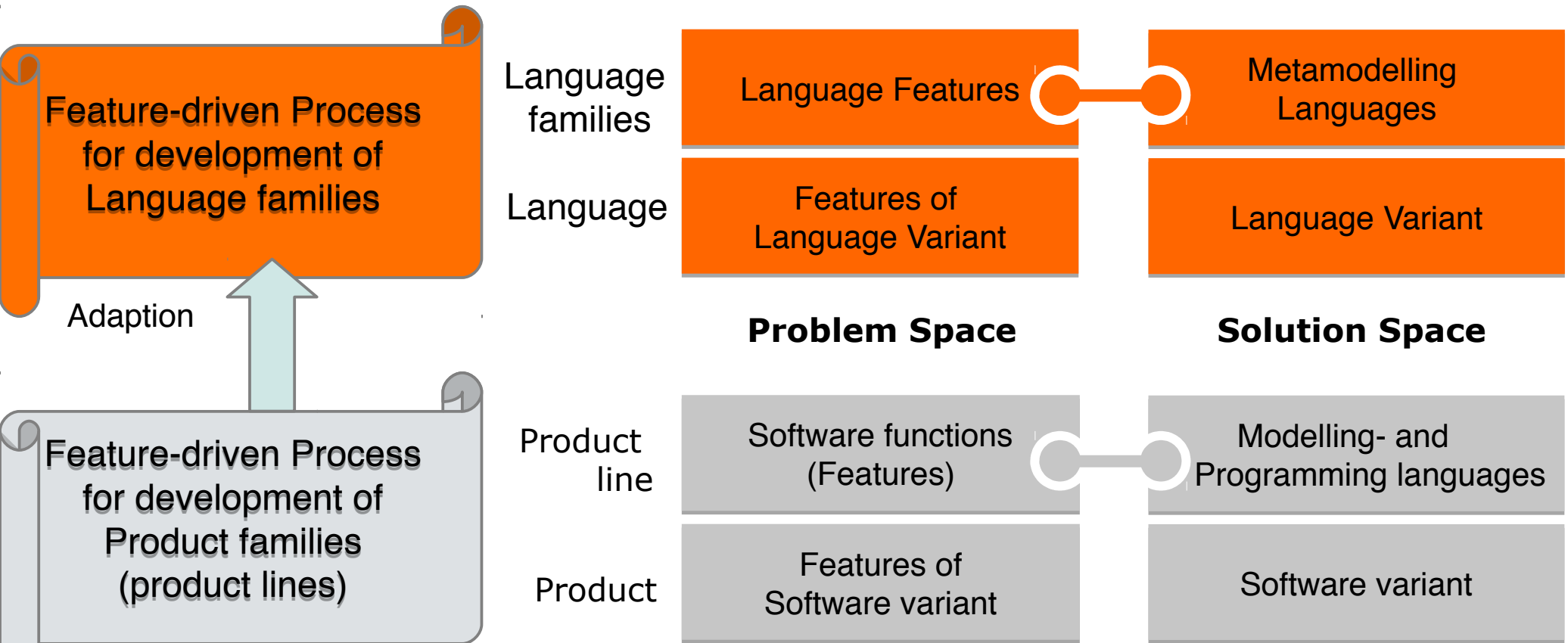


DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

# Feature-Driven Development of Language Families

## PhD Christian Wende (2012)

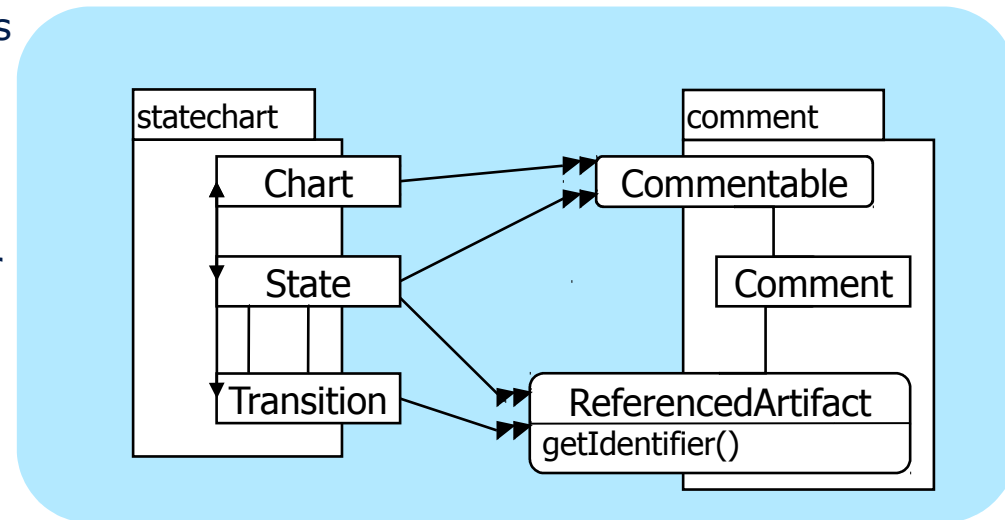
### Systematic variability management for language families with feature models



# 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>

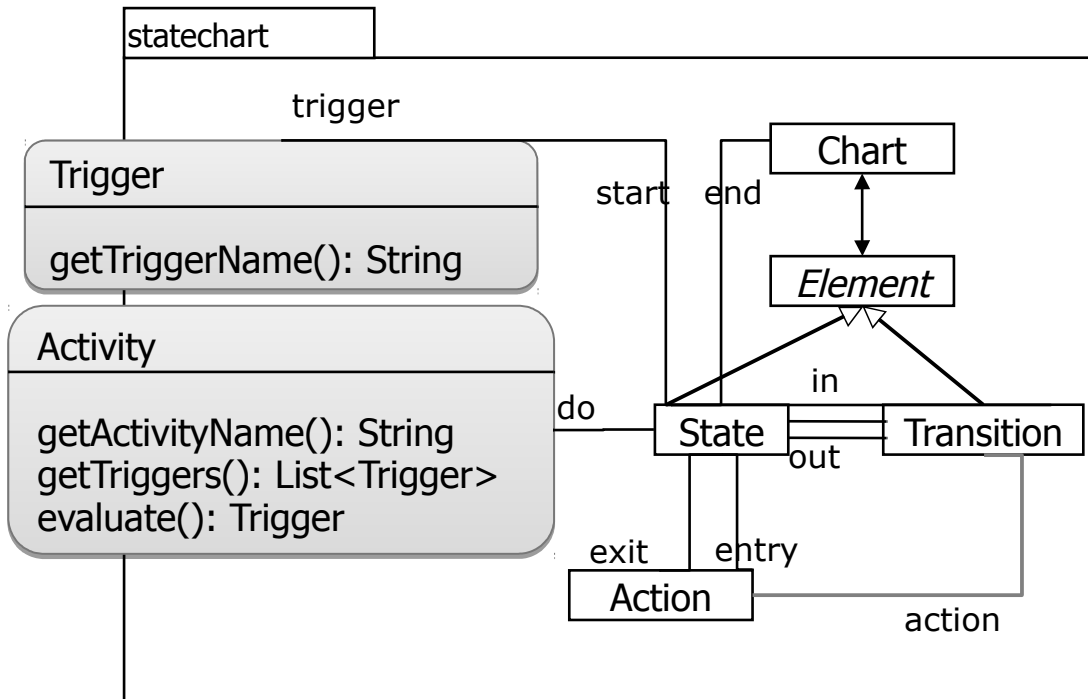


# Example: Statecharts and Forms

## Language Component: statechart

## concrete Syntax (offered by EMFText)

### Abstract Syntax



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

    from init to data
      when login do {}

    state data {...}
    ...
  }
```

### Semantics:

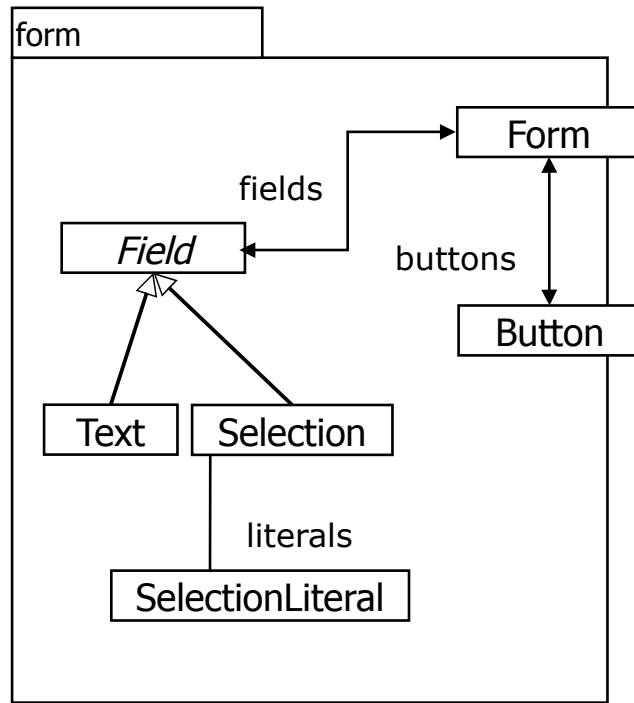
Operational Semantik written in Java based on the interfaces of abstract syntax



# Example (2): Forms

## Language Component: form

### Abstract Syntax

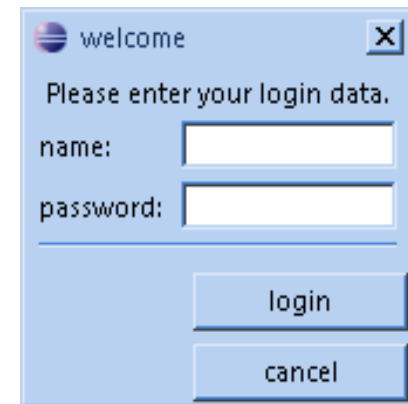


### concrete Syntax (with EMFText)

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

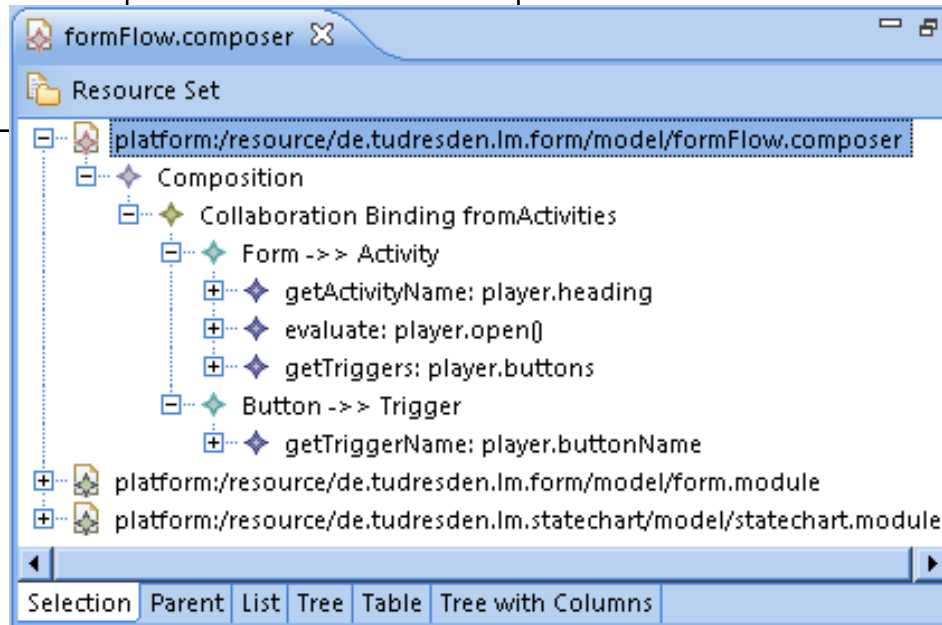
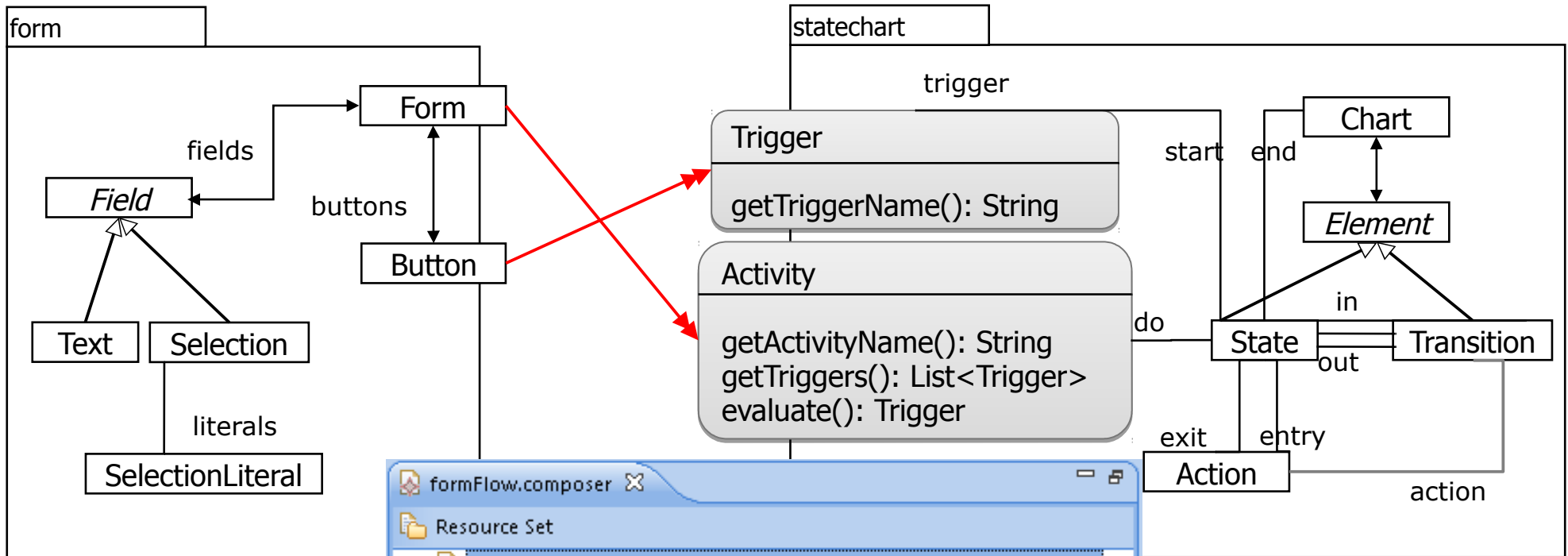
### Semantics:

Operational Semantics with Java



# Example (3) Composition of Statechart and Form

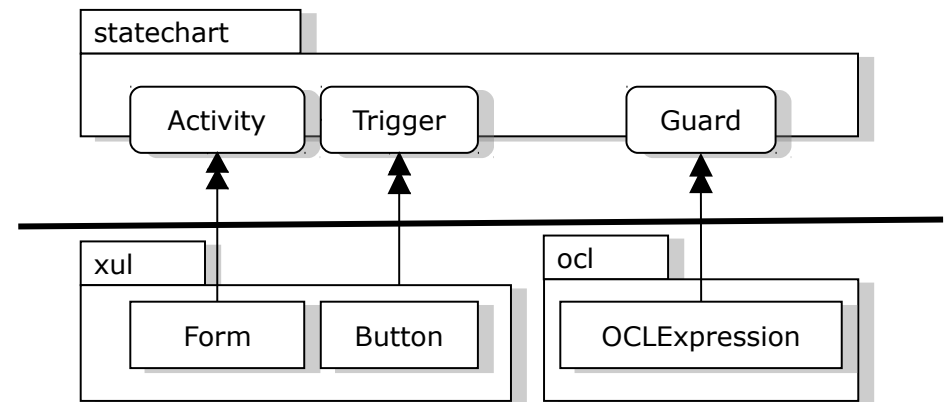
## Language Composition: $\text{formFlow} := \text{form} \text{-->>} \text{statechart}$



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 structural and semantic adaptation of the role player w.r.t the role contract

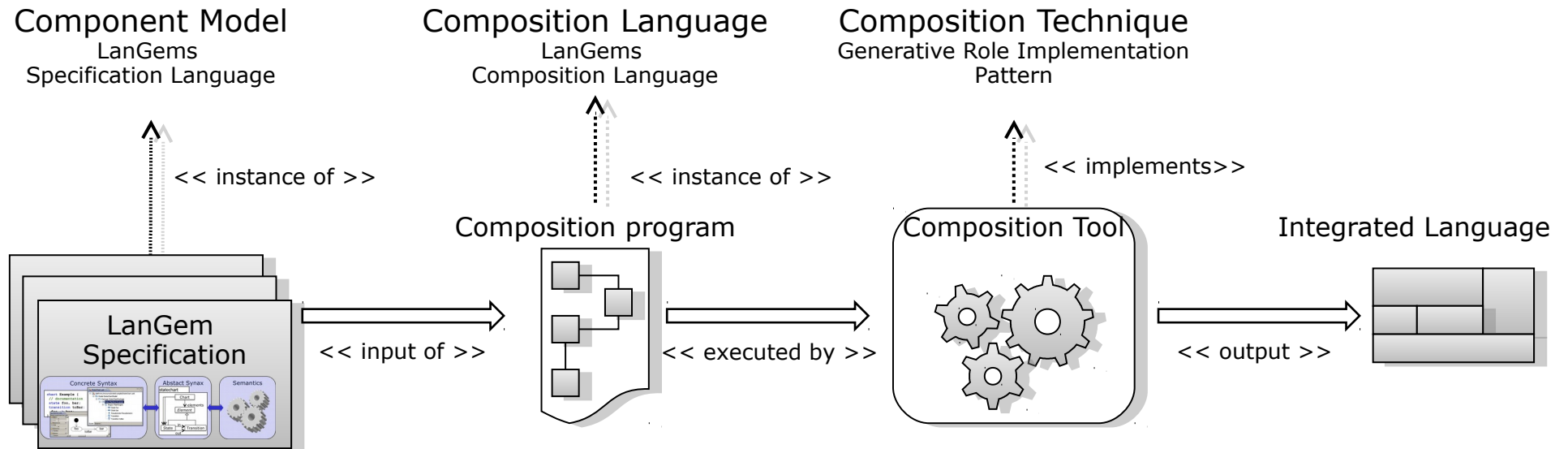


```
Form plays Activity {
    getActivityName(): player.getTitle();
    executeActivity(): player.open();
}

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

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

# LanGems Composition System



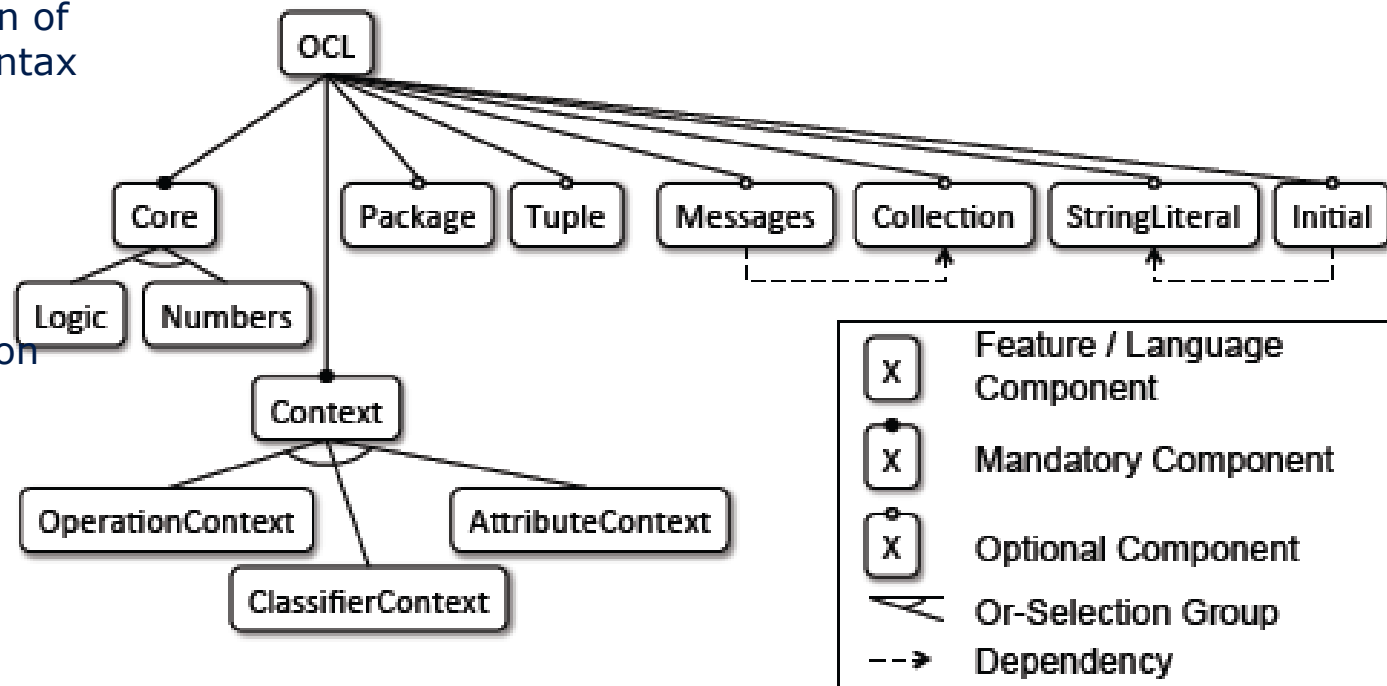
# Case Study: Modularisation of OCL

## 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



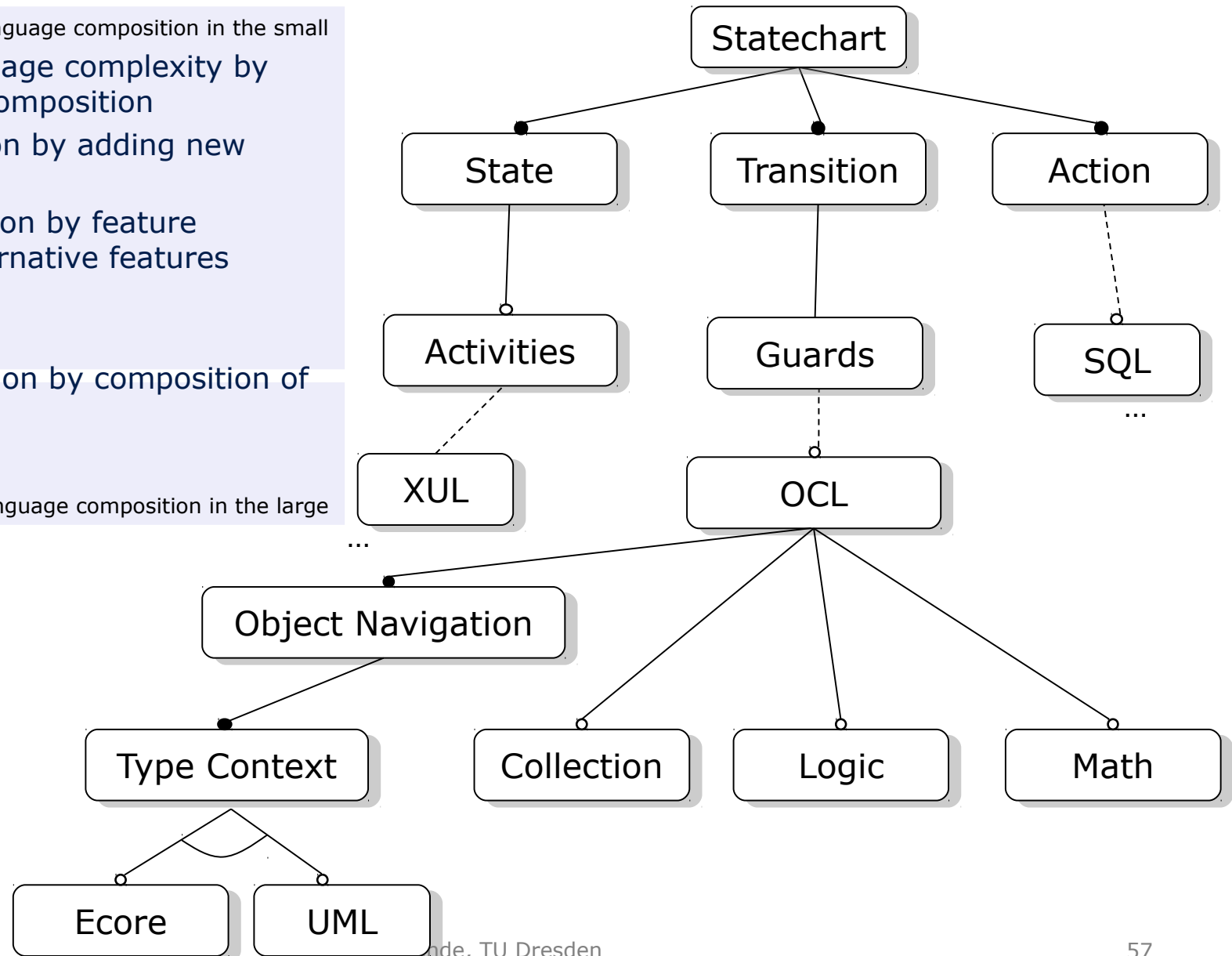
# Case Study Statecharts: The Problem Space Perspective

## 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

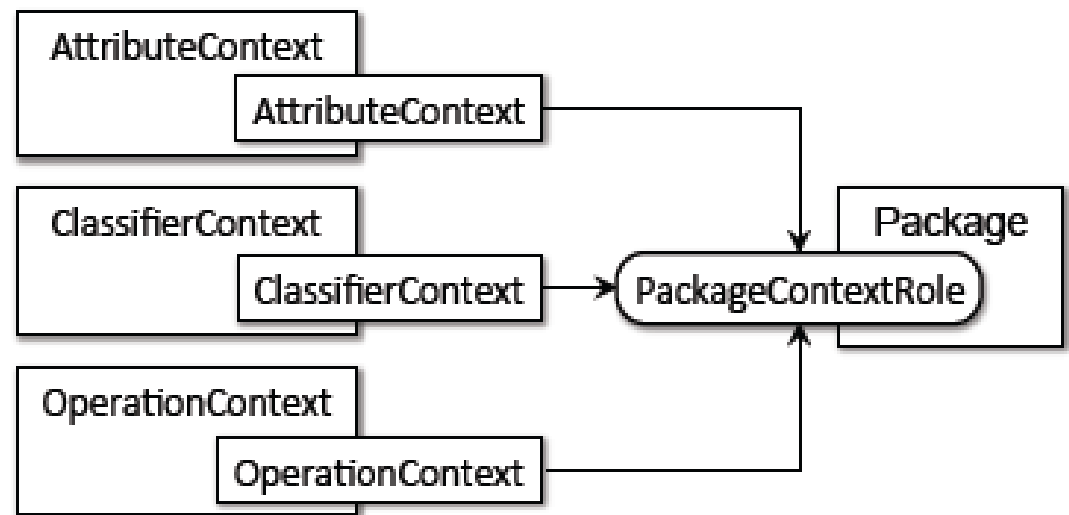
language composition in the large



# 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://Db.inf.tu-dresden.de/rosi)



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

