

# 22b 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 11-0.6, 17.11.11

- 1) Motivational Example
- 2) Roles in Metalanguages
- 3) Proactive vs. Retroactive Tool Integration
- 4) Role-Based Composition

## 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>

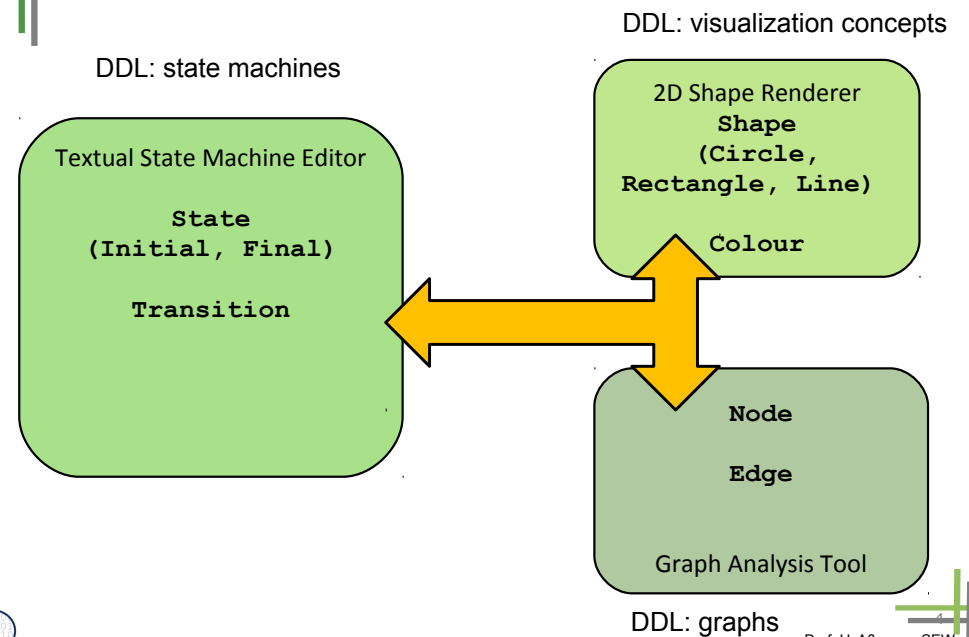
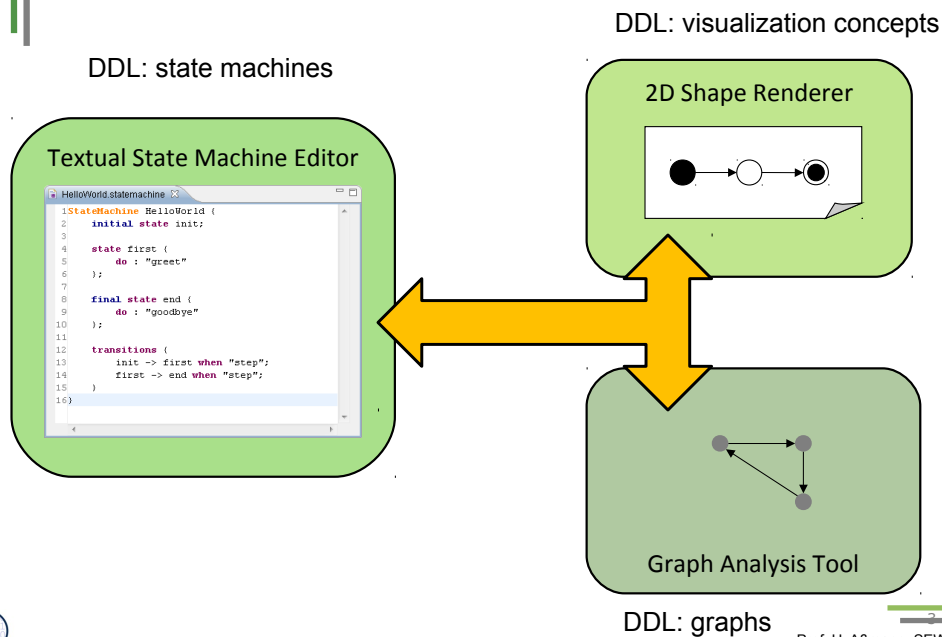


- ▶ <http://www.emftext.org/language/rolecore>

emftext

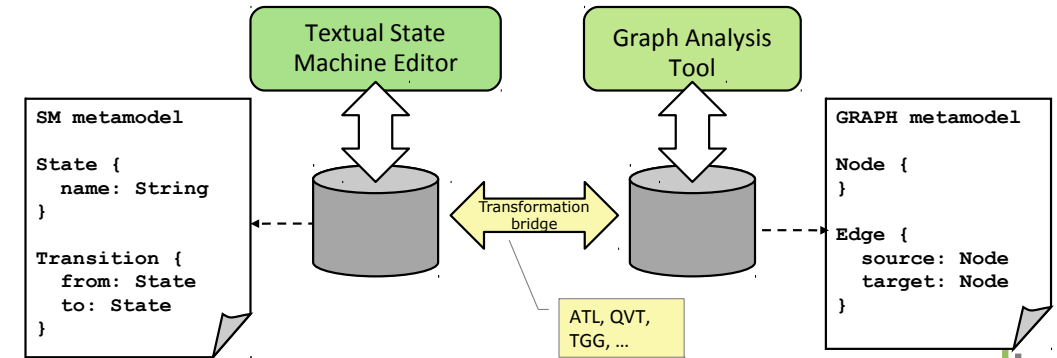
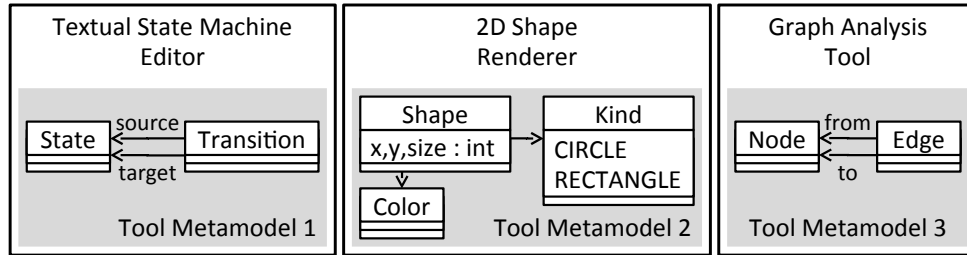
### 22b.1 Motivational Example

### Motivational Example - Language Concepts in Metamodels of the Involved Tools



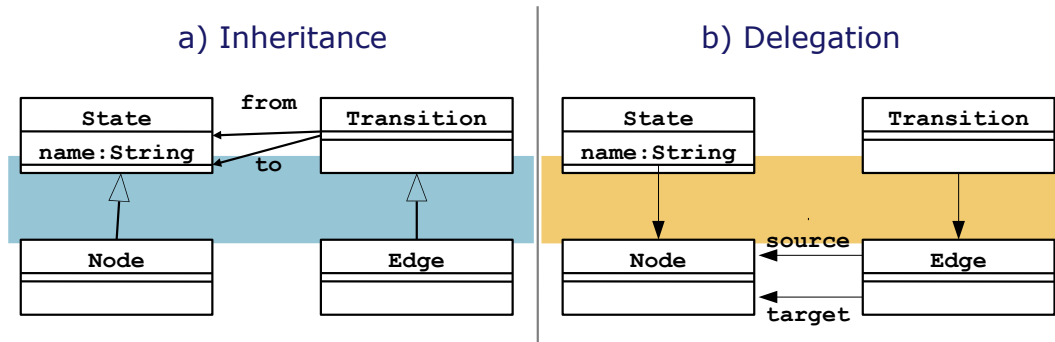
## Retroactive Tool Integration on Repositories

- Tools, metamodels, and repositories already exist
- Use transformations to convert data from one tool to another (transformation bridge, data exchange, Datenverbindung)



## Proactive Tool Integration (Classical)

- Tool, metamodels, and repositories are not fixed yet
- Use metamodel integration to make data from one tool accessible to another (usually by inheritance or delegation)



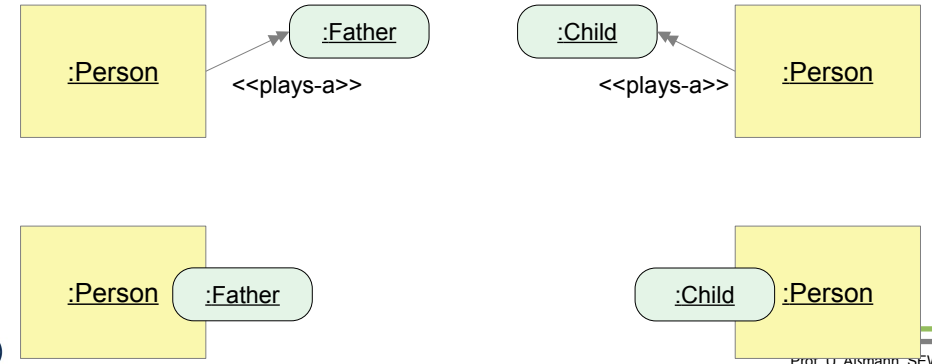
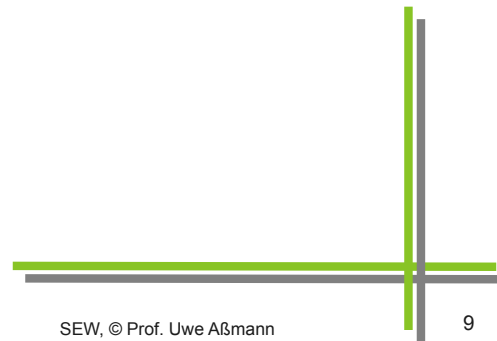
## Proactive vs. Retroactive Tool Integration

	Proactive		Retroactive	
Technique	Inheritance, delegation		Transformation	
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	⊖

## 22b.2 Roles in Metalanguages

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

Databases [Bachmann]  
 Factorization [Steimann]  
 Research in Design Patterns [Reenskaug, Riehle/Gross]



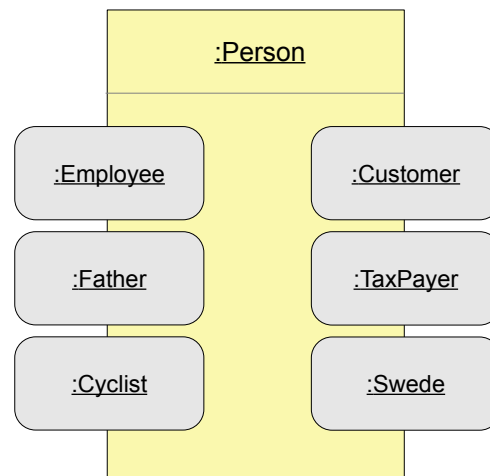
### What are Roles? (Rpt.)

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



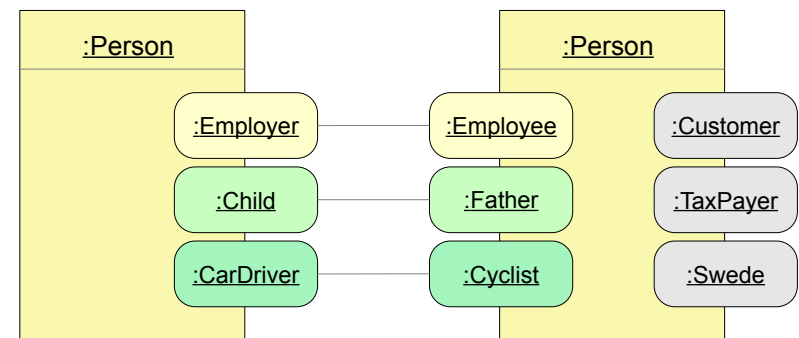
### What are Roles? (Rpt.)

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*



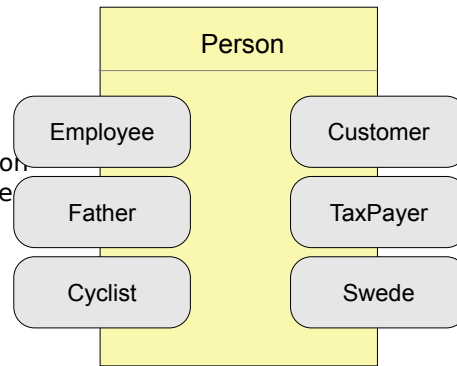
## What are Role Types? (Rpt.)

Role types (abilities) are

- service types
- dynamic types
- collaborative types

Problem:

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

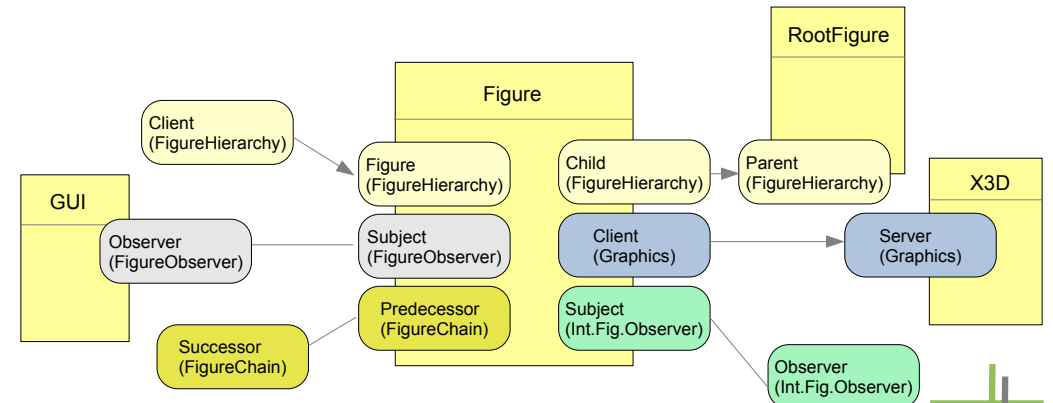


## Collaboration Schemas (Role-Type Model) (Rpt.)

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)

Collaborative Aspects

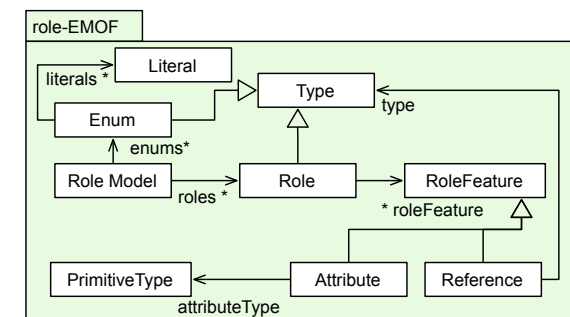
- ObjectTeams [www.objectteams.org](http://www.objectteams.org)
- CaesarJ

Template-based languages

- BETA with the metaprogramming environment Mjölner
- Invasive Software Composition

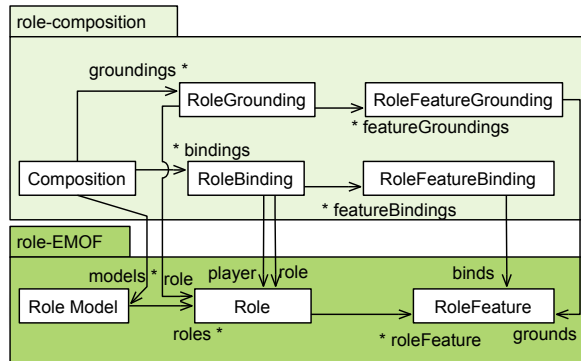
## Roles in a Metalanguage (Metametamodel)

- ▶ Roles can be introduced as modeling concept.
- ▶ Here, an extension of EMOF with roles:



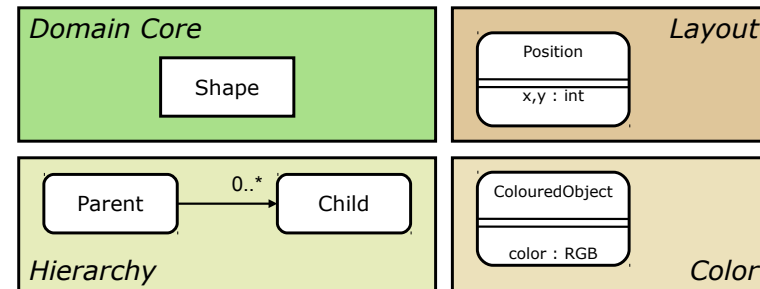
## A Metamodel for Deep Role Composition

- ▶ **Deep roles** are roles playing roles
- ▶ **Flat roles** do not play roles
- ▶ This role composition technique (specified by a role-composition metamodel) allows for deep roles



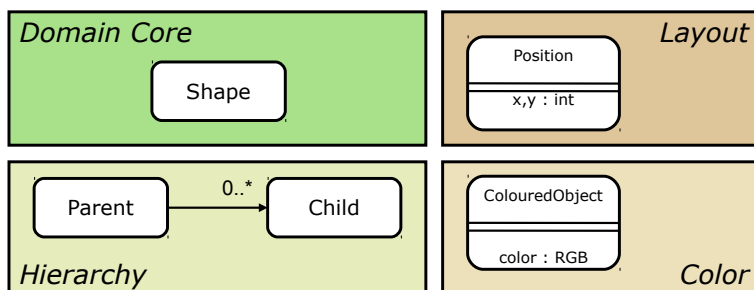
## Example: ShapeRenderer's Metamodel with Roles

- ▶ Roles adhere to a context
- ▶ A context is a specific *concern* (here: colors)
- ▶ Only one natural type, many roles



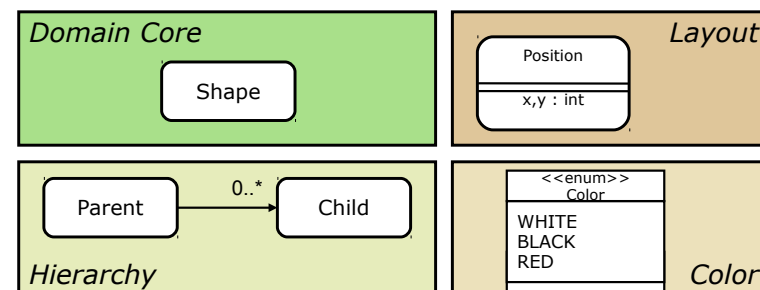
## Example: ShapeRenderer's Metamodel with Deep Roles

- ▶ Because other tools' metamodels might provide the natural types, we first specify all metamodels with deep roles
  - Then, they can be played by the naturals of other tools



## Example: ShapeRenderer's Metamodel with Deep Roles and Enums

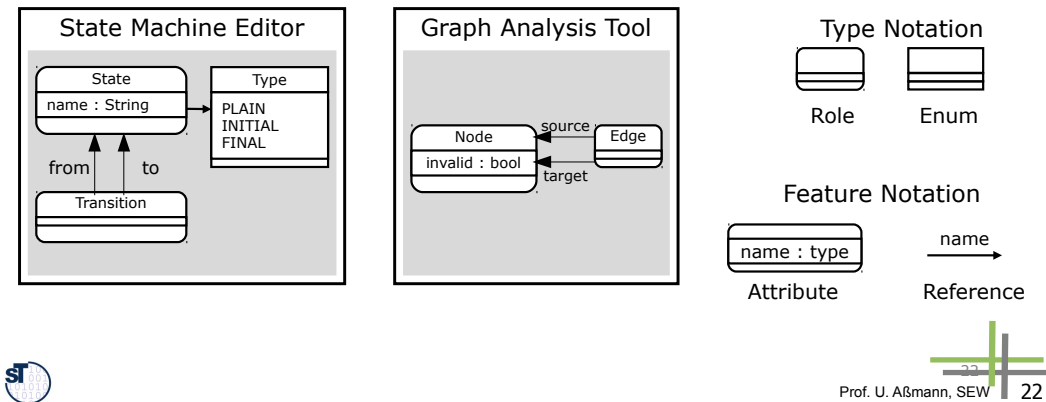
- ▶ Some roles can be represented as enums; then they will become naturals in the implementation



## 22b.3 Proactive Tool Integration with Deep Roles

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

- Specify M2-metamodels also with role types (abilities) not only classes
- At first sight not much different from object-oriented metamodels
- Difference to classical role modeling: Naturals are selected later; first specify everything as deep role; some roles become enums

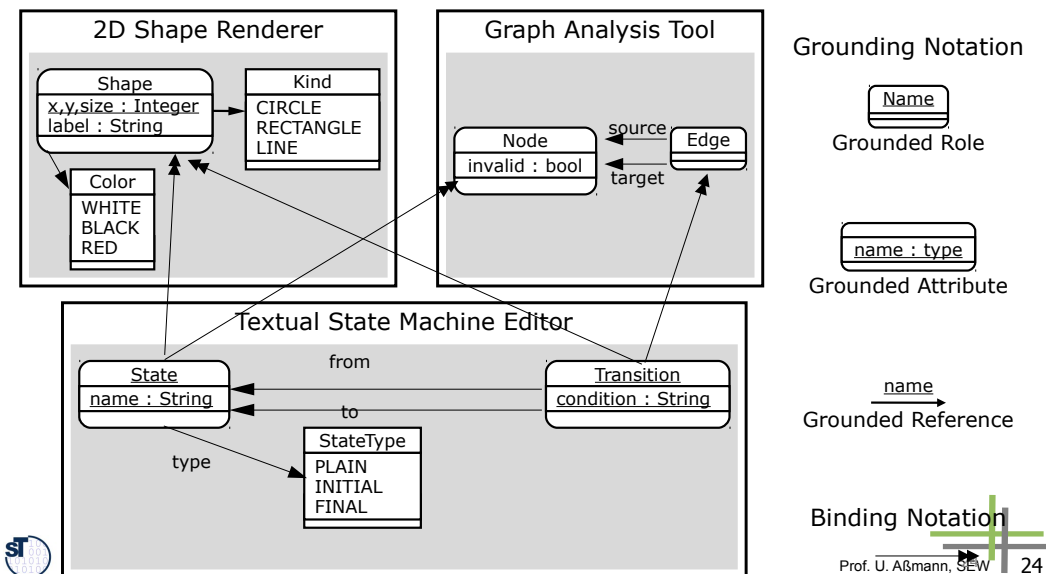


## Tool Integration using Role Bindings (Role Grounding)

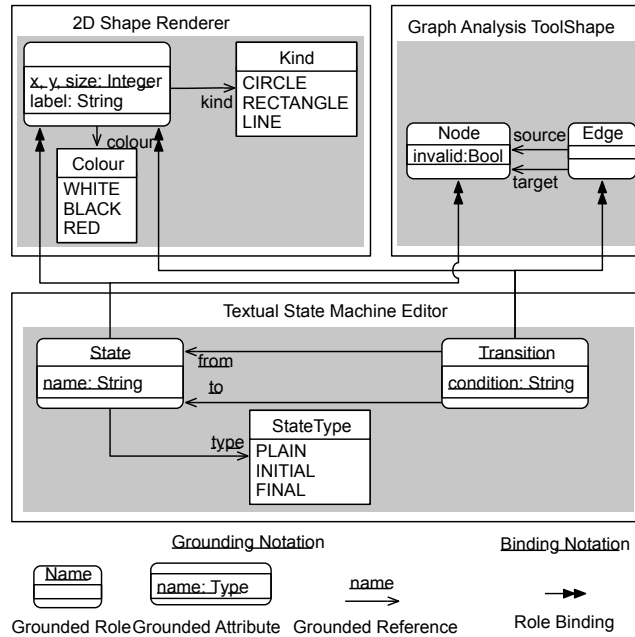
- Role Bindings on the logical level with relationship "plays-a"
  - Connect roles and role players, producing *deep roles*
  - Define how to obtain value of attribute or reference
  - Allow to create views on other classes
- Grounding on the physical level
  - Defines which attributes/classes are represented physically
  - Select natural types
  - 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 binding and role grounding
- We defer the decision "what is a natural"



- ▶ Role binding can be described by a 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
  }
}

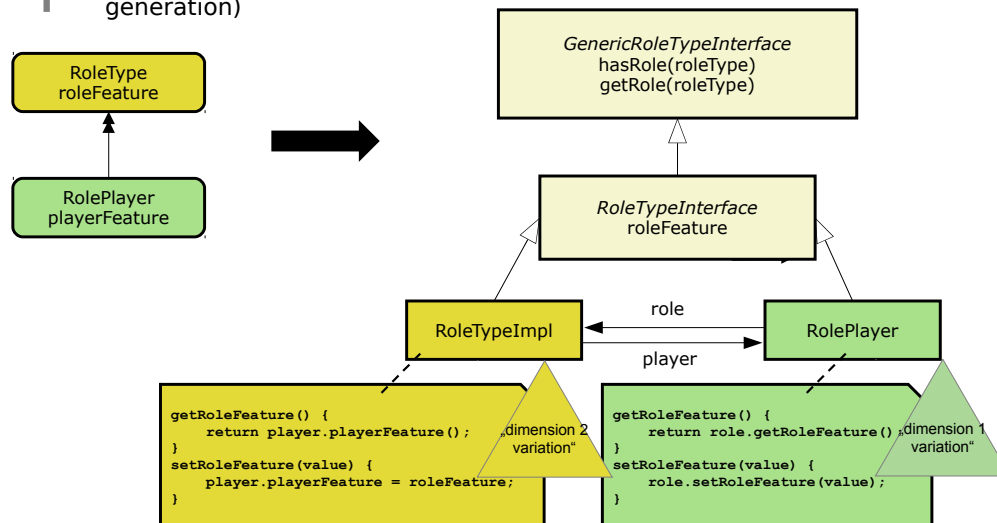
ground State { name, type }
ground Transition { condition, from, to }
    
```

Role Binding Specification

Grounding Specification

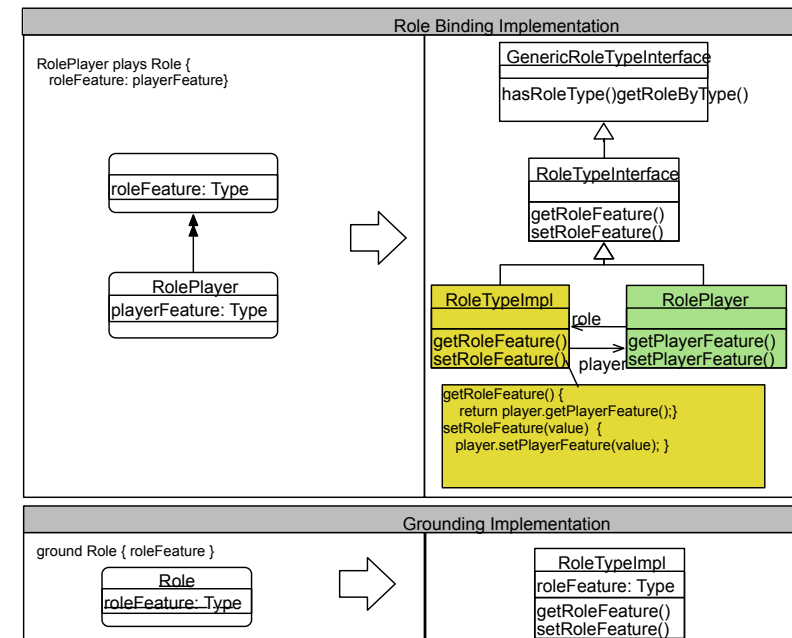
## Role Binding Realisation by e.g., Delegation (Design Pattern Bridge)

- ▶ The constructs of RoleCore can be easily expanded to design patterns (code generation)



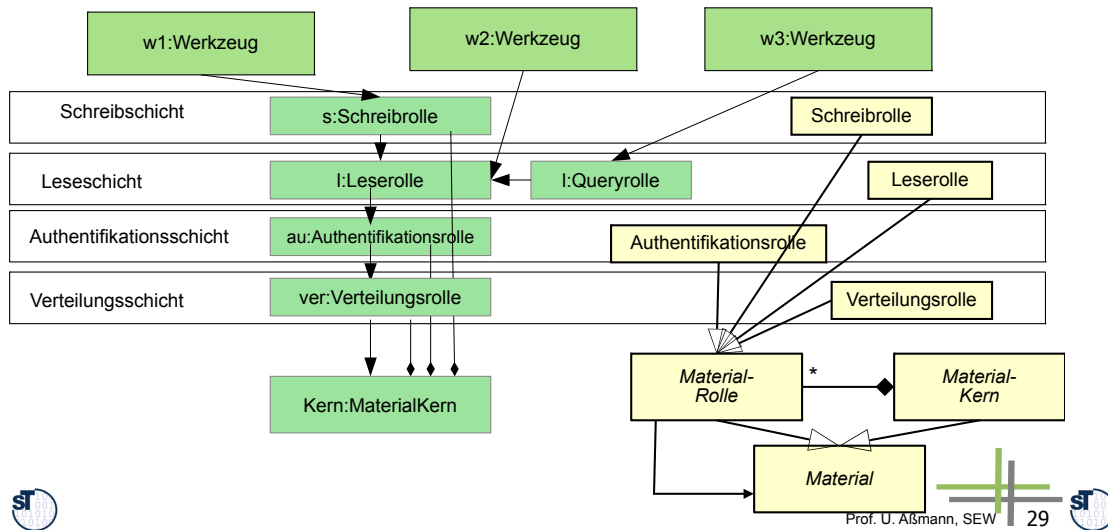
Grounding is straight forward with many design patterns for role implementations

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



## Final Architecture of the Composed Repository

- ▶ When using ROP for binding, the role-access layer architecture for repositories results naturally:



## What Did We Learn?

- ▶ Deep Role Modelling allows for unanticipated tool integration, but needs to be applied at tool 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, a role-based access layering of the repository results naturally.
- ▶ Open Issues
  - Data migration (if grounding evolves)
  - Practical validation required
- ▶ Looking for students!