10. Metamodellierung

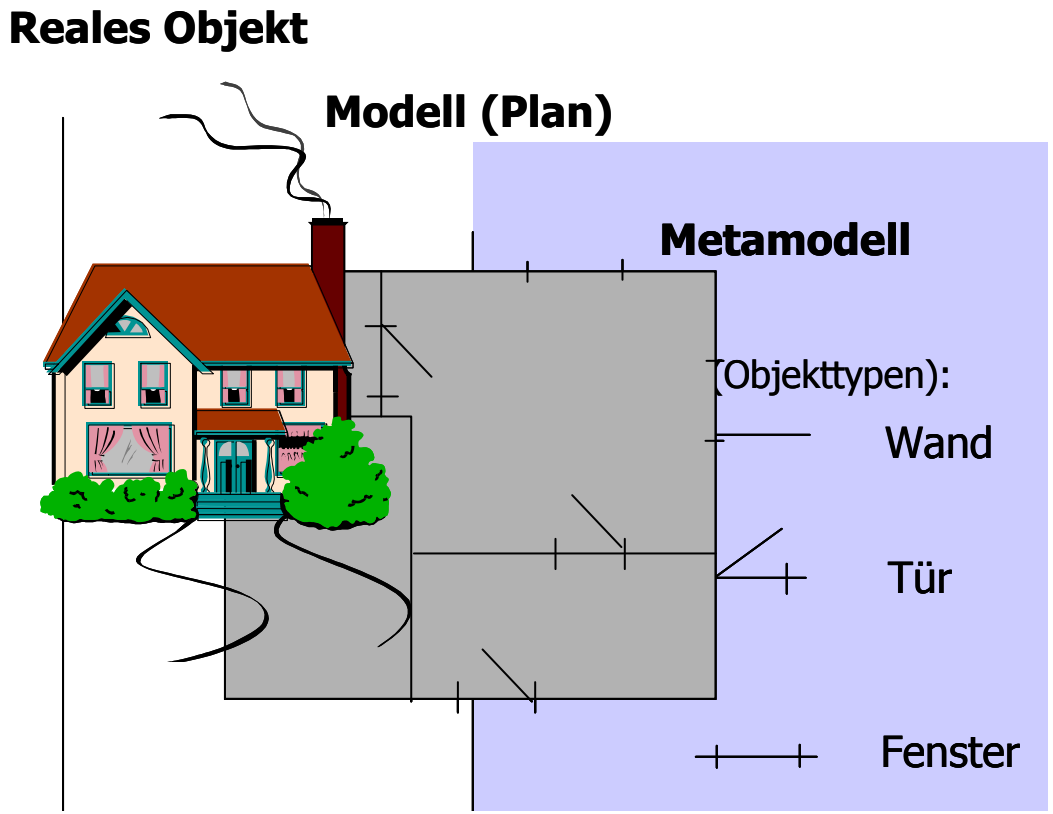
Dr.-Ing. Clemens Reichmann

Clemens.Reichmann@vector.com
Tel. 0721-91430-200

Institut für Technik der Informationsverarbeitung
Fakultät für Elektrotechnik & Informationstechnik
Universität Karlsruhe (TH)

10.1 Metamodell

10.2 Modelltransformation



- Metamodell
  - Das Modell einer Modellierungssprache;
  - es definiert die
    - Objekttypen, die zur Erstellung eines Modells verwendet werden dürfen
    - die Objektattribute
    - ihre Bedeutung
    - die Regeln ihrer Verknüpfung

Ziele der Metamodellierung

- Dokumentation und Verständnis der zu beschreibenden Informationen

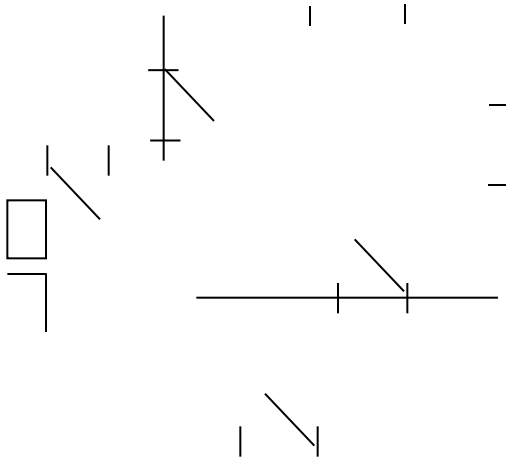
- Definition einer Sprache
  - Formalisierung von Information
  - Granularität der Beschreibung, deren Attribute und Zusammenhänge
  - Was ist erlaubt und was nicht

- Möglichst wenig Businesslogik
- Keine Redundanz
- Grundlage für die konsistente Entwicklung eines Werkzeuges

Analogie: Metamodell der Architektur

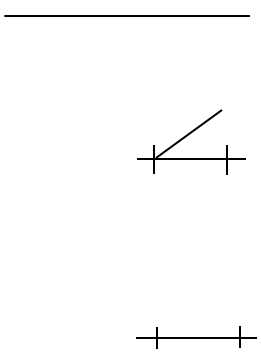
Reales Objekt

Modell (Plan)



Metamodell

Konstrukte (Objekttypen):



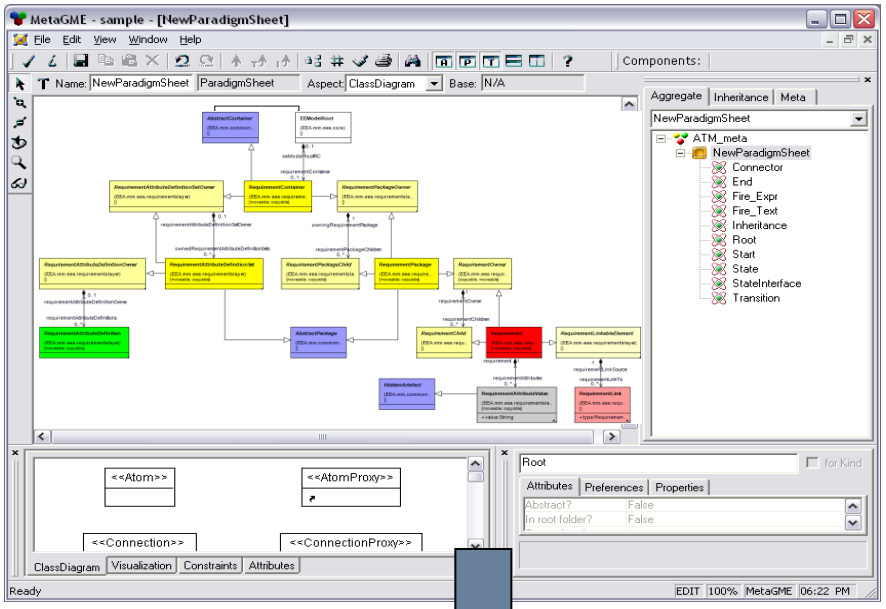
Regeln:

- Eine Tür grenzt links und rechts an eine Wand
- Fenster sind an Aussenwänden

Modell - Metamodell

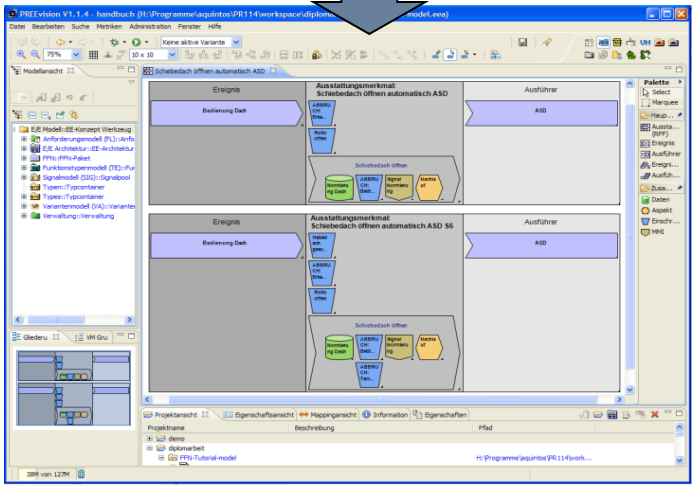
Metamodell
=Sprache
=Symbole

(Eingabe in MMEdit)



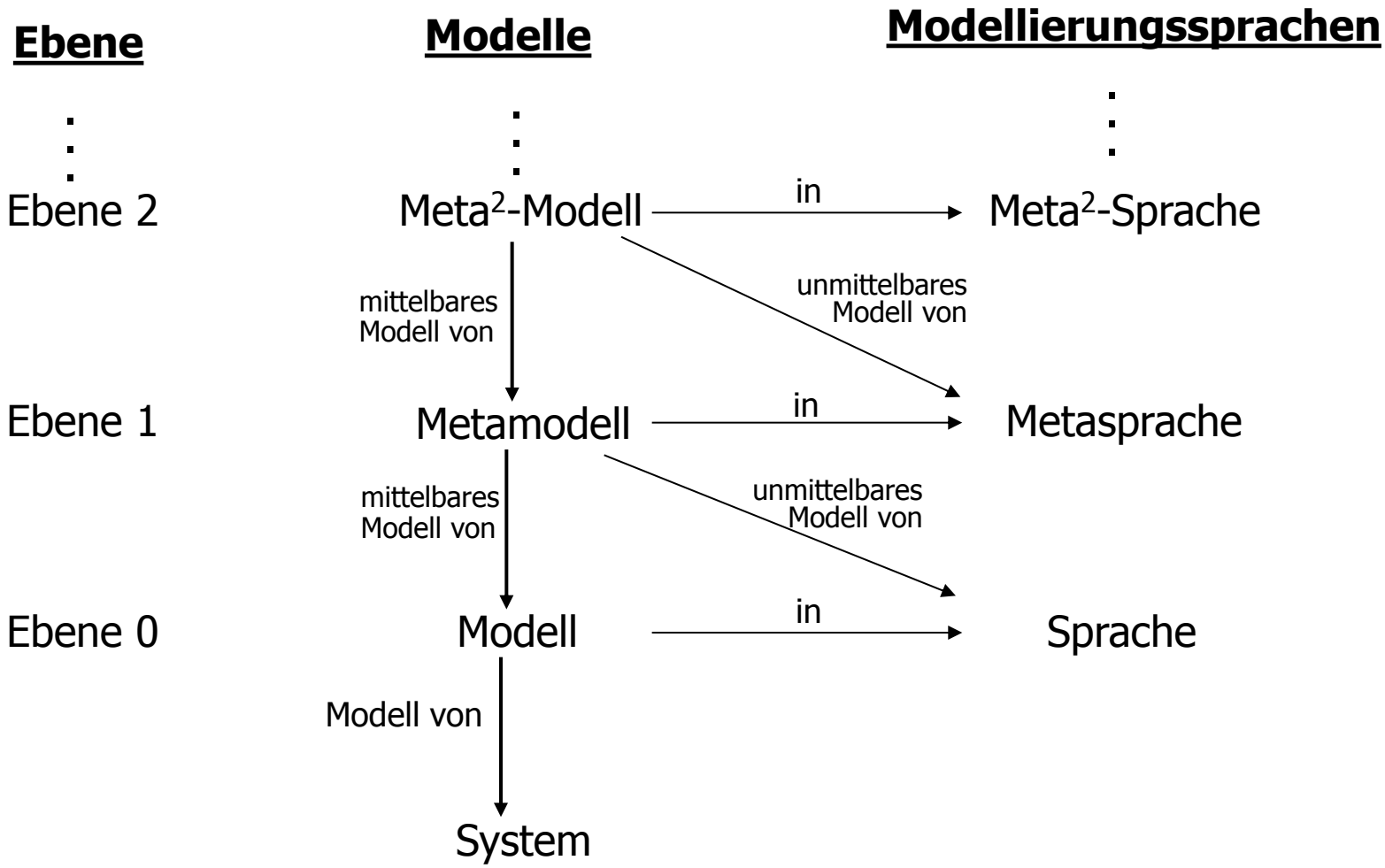
Domänenspezifisches Modell

(Verwendung in PREEvision)



- **Modelle** sind aufgebaut auf der Grundeinheit **Graph**
  = Menge von **Knoten** (Nodes) und **Kanten** (Links)
- **Klassen von Grafen, Knoten, Kanten und ihre Eigenschaften, Beziehungen, Regeln, ..., bestimmen die Entwurfsmethode (das Metamodell)**
- Metamodellierung muss Definition/Anpassung der Graf-, Knoten-, Kanten-Klassen mit ihren Eigenschaften, Beziehungen, Regeln, ... durch den Nutzer zulassen

Sprachbasierter Metamodellbegriff



Meta-Modell Abstraktionsebenen

abstrakt

konkret

M3 Ebene
Meta-Meta-Modell

- MOF (Meta-Object-Facility)

M2 Ebene
Meta-Modell

- UML 2.0
- UML 1.5
- MATLAB Simulink
- Statechart (D. Harel)

M1 Ebene
Modell

- UML Modell
- Simulink Modell
- Statechart Modell

M0 Ebene
Realisierung

- Objekte
- Daten
- Quellcode

Reale Artefakte

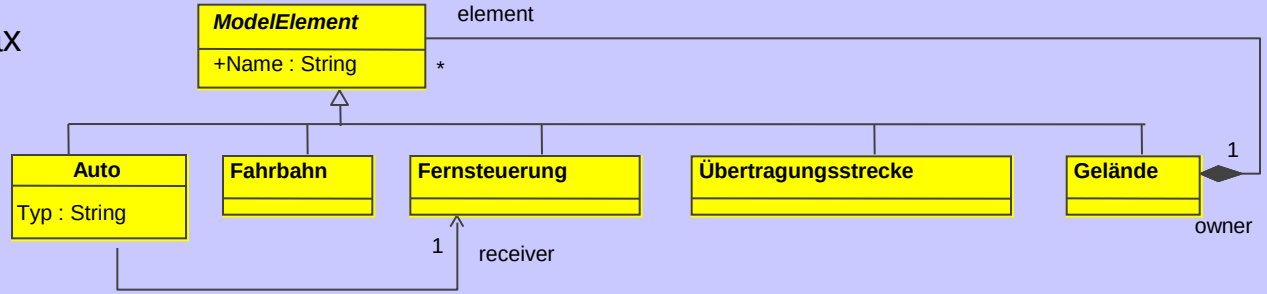
MOF

- MOF ist eng mit dem Metamodellierungsansatz der OMG verknüpft
- Dabei gibt es 4 Ebenen [M3, M2, M1, M0] der Metamodellierung (nicht zu verwechseln mit den Ebenen der MDA)
  - M3: Dies ist die Infrastruktur der Metamodellarchitektur und sie definiert die Sprache zur Spezifikation von Metamodellen (z. B. Metaklasse „Klasse“, Metaklasse „Attribut“, Metaklasse „Operation“). Auf dieser Ebene ist MOF angesiedelt. Metamodelle der UML oder des CWM sind Instanzen des MOF.
  - M2: Ein Metamodell ist eine Instanz des Meta-Metamodells und definiert die Sprache zur Beschreibung der Modelle (z. B. Klasse, Attribut, Operation). Diese Schicht ist das zentrale Element der UML und die Konzepte werden bei der UML-Modellierung verwendet. Zudem werden auf dieser Ebene die Ergänzungen für die unterschiedlichen Plattformprofile spezifiziert.
  - M1: Ein Modell ist eine Instanz des Metamodells und definiert die Sprache zur Beschreibung der Domäne (z.B. Klasse: Person, Operation der Klasse Buch: getName). Zu den Modellen gehören die bekannten UML-Modelle.
  - M0: Ein Objekt ist eine Instanz des Modells und beschreibt die Ausprägungen einer bestimmten Domäne, wie z.B. den Namen einer Instanz der Klasse Person: „Maria“.

Beispiel: Modell, Metamodell

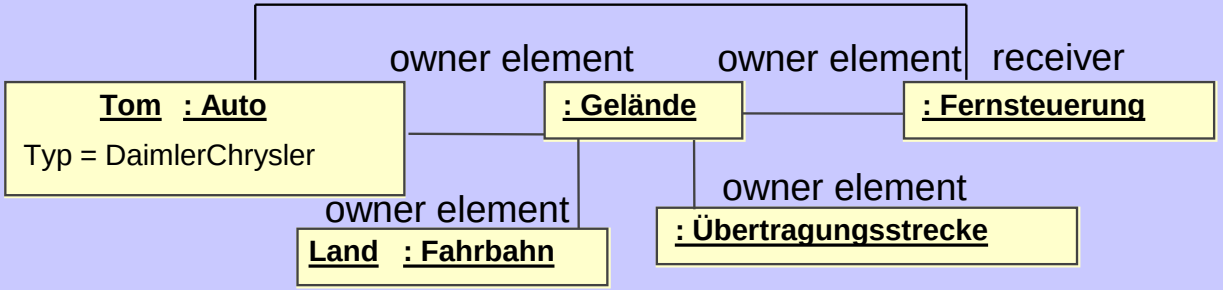
M2 Metamodell

Definition der abstrakten Syntax in MOF Notation



M1 Modell: Abstrakte Syntax

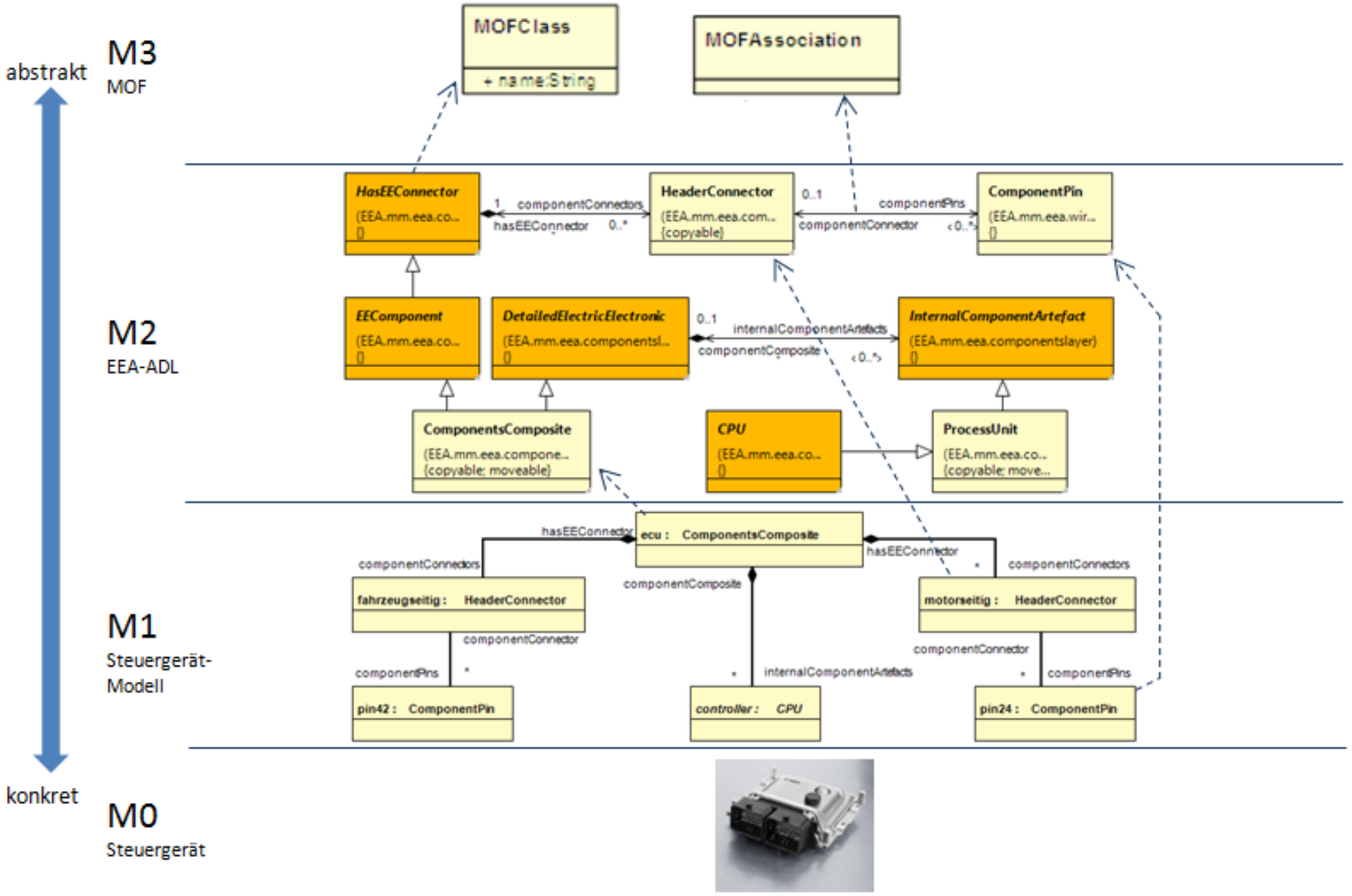
Modell: Daten



M1 Modell: Konkrete Syntax

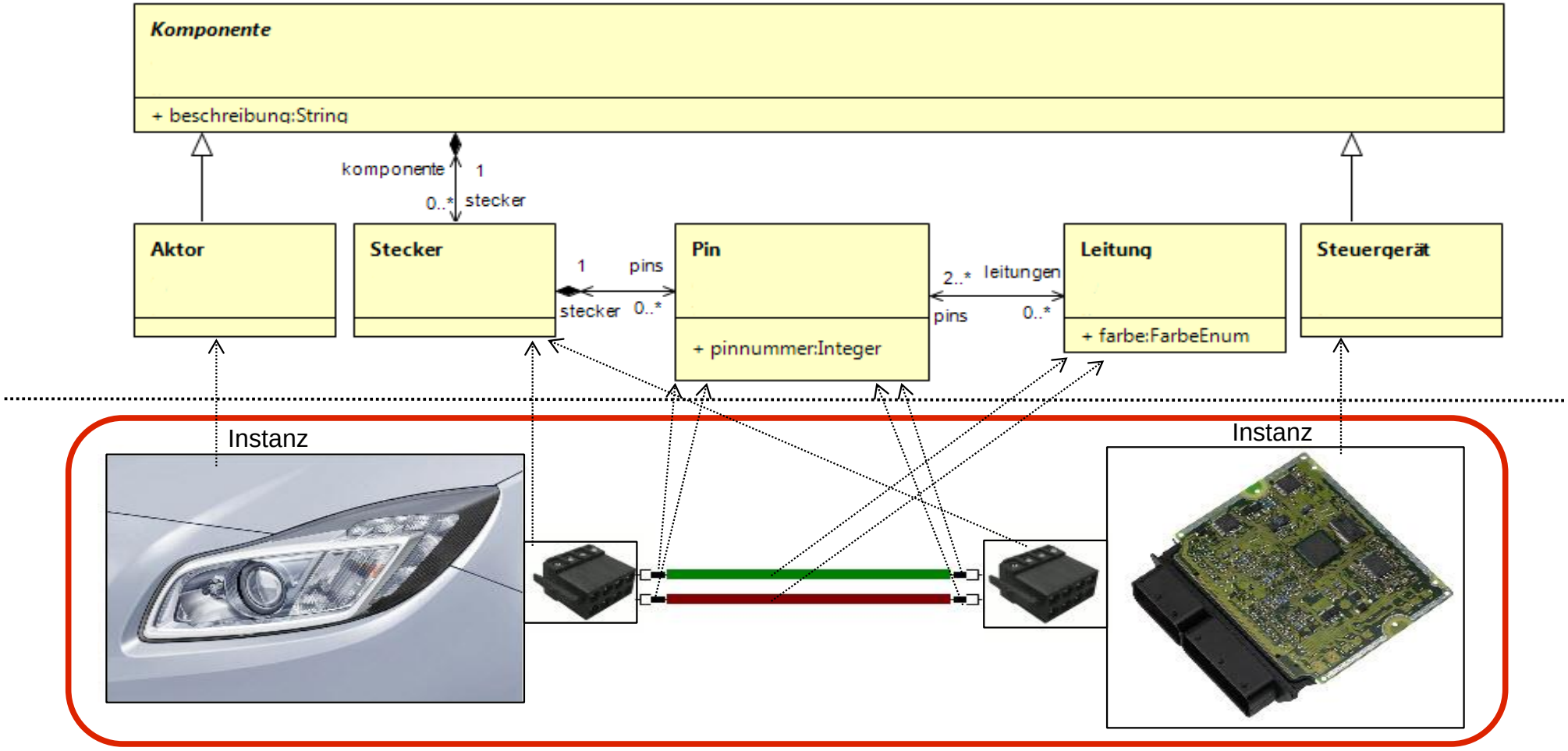
Daten als Symbole

MOF (Meta Object Facility)



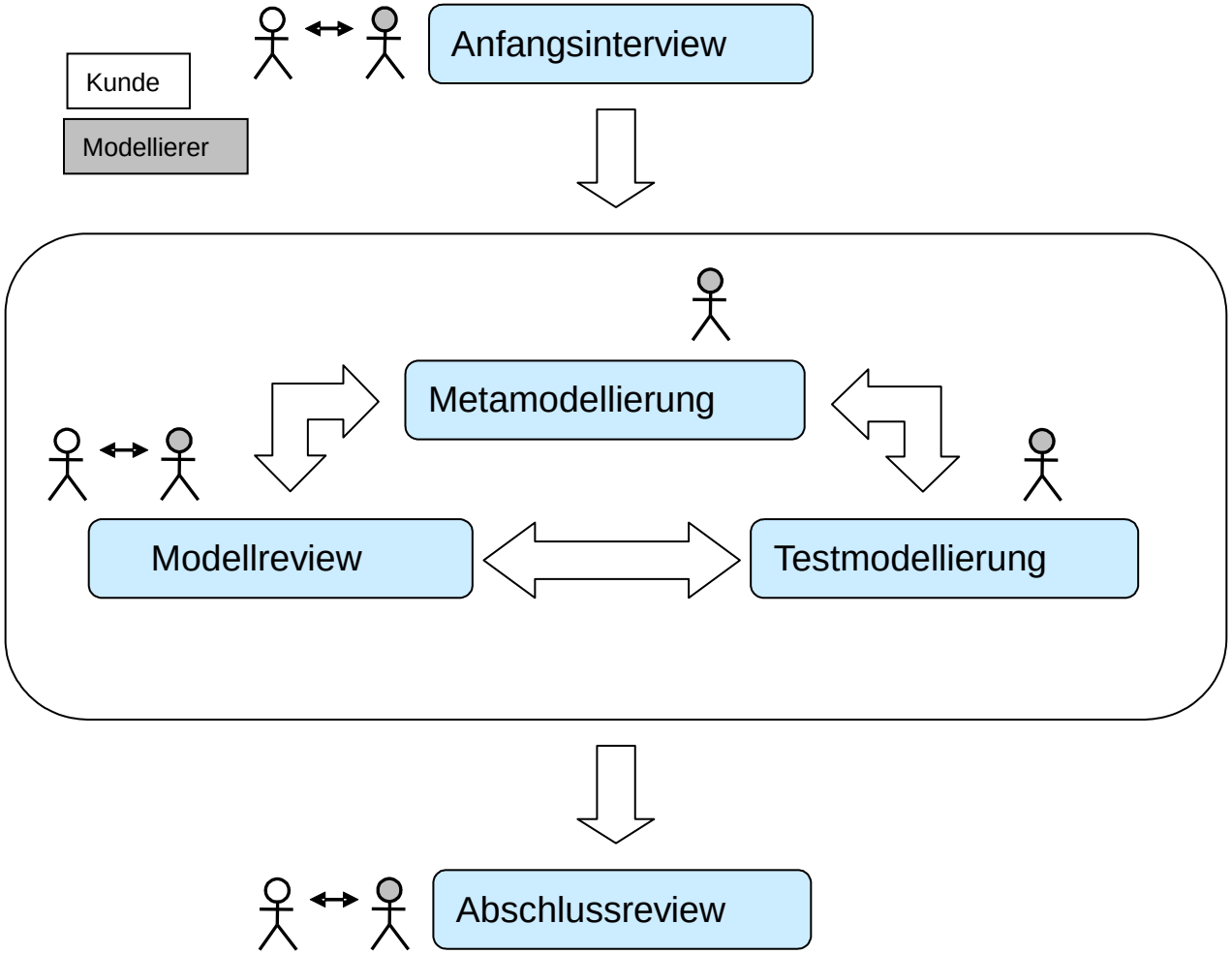
MOF (Meta Object Facility)

M2 Metamodell: MOF-Klassendiagramm

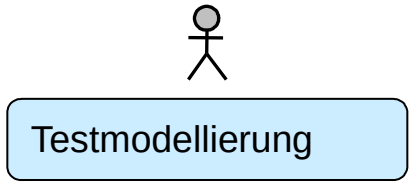


M1 Modell

Vorgehensweise zur Erstellung eines Metamodells



Analyse des Wissensgebietes - Testmodellierung



nicht ausführbare Testmodellierung

- **Objektdiagramme**
- **Zur Überprüfung einzelner, überschaubarer Konzepte**
- **Manueller Prozess**
- **Fehleranfällig**

ausführbare Testmodellierung

- **Generierung und Kompilierung von ausführbarem Code**
  - **Syntaxcheck**
- **Manuelle oder automatisierte Erstellung von Modellen zur Laufzeit**
- **Überprüfung der Persistenzschicht**

Analyse des Wissensgebietes - Testmodellierung

UML - Darstellung des Metamodells

↓ Transformation

MOF - Darstellung des Metamodells

↓ Generierung

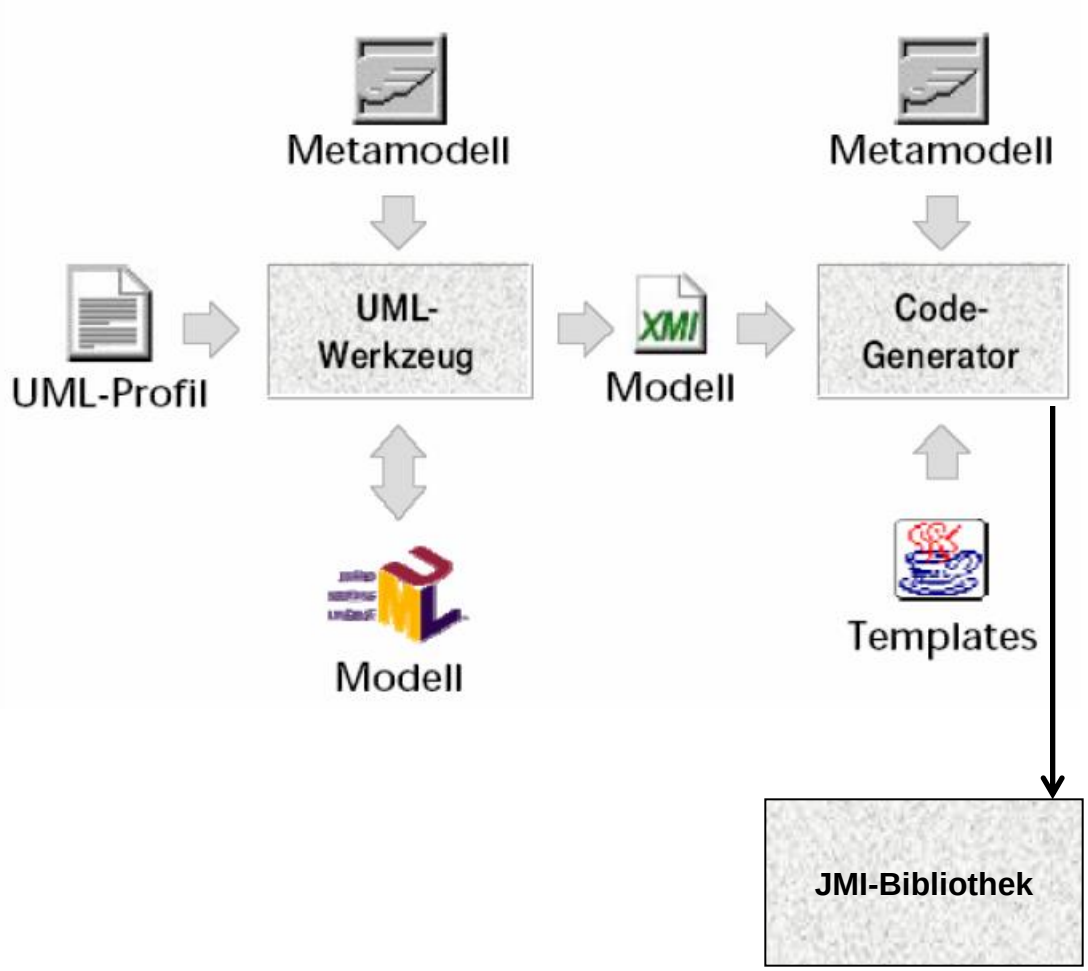
Quellcode des Metamodells

↓ Kompilierung

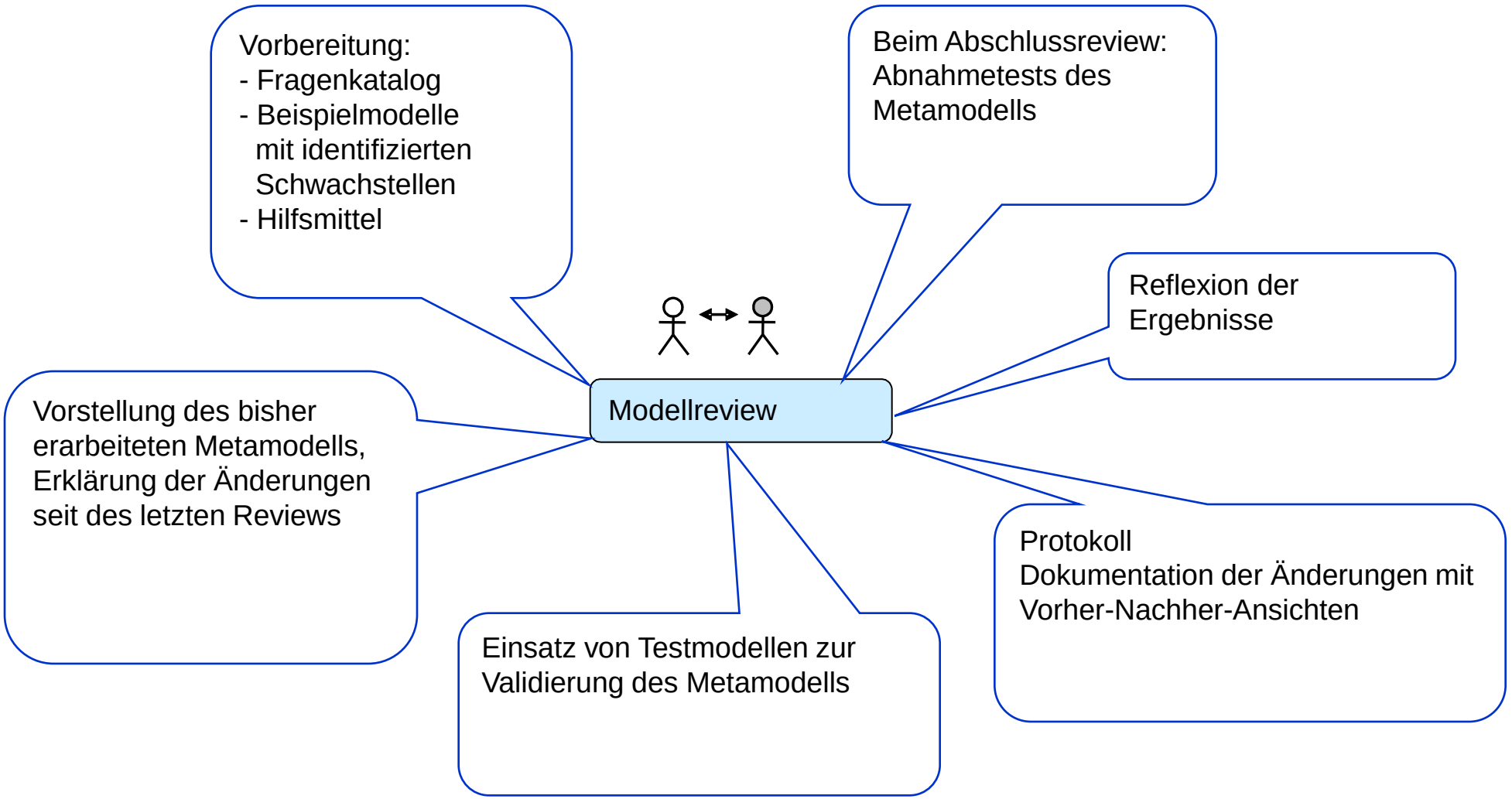
Ausführbare Darstellung des Metamodells

↓ Instanziierung

Testmodell



Analyse des Wissensgebietes - Modellreview

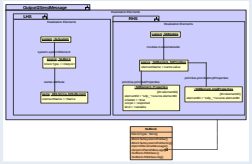


Fragen ?

→ Modell-zu-Modelltransformation (M2M)

Modell zu Modell Transformation

- Tool A:
  z.B. MATLAB/Simulink
  z.B. UML (ARTiSAN)



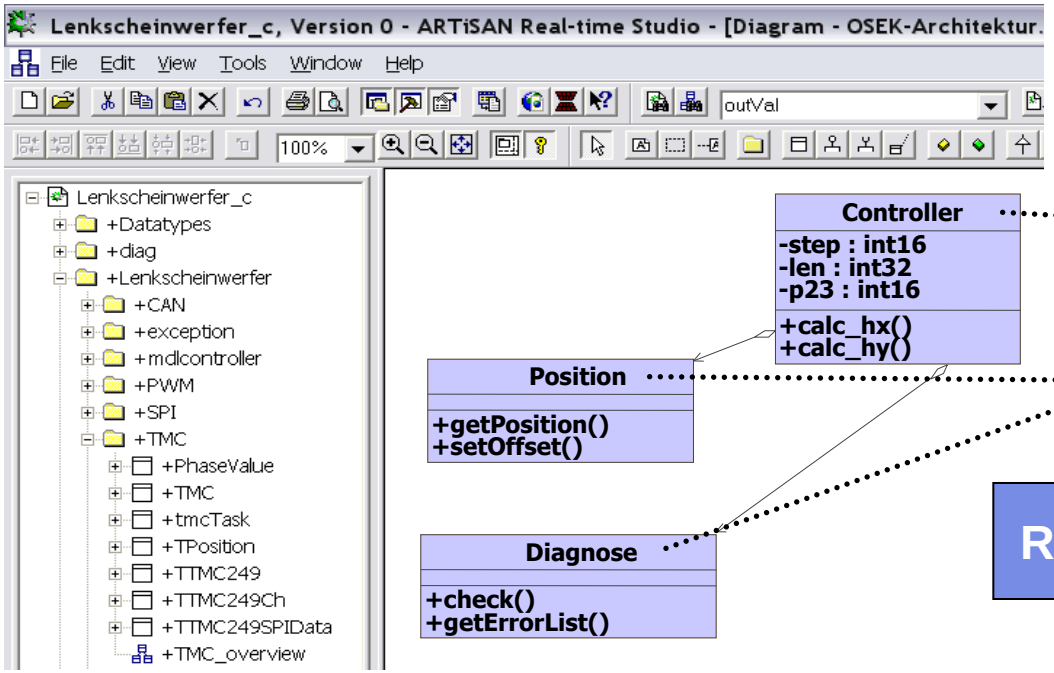
aquintos.M2M

- Tool B:
  z.B. ASCET
  z.B. MATLAB

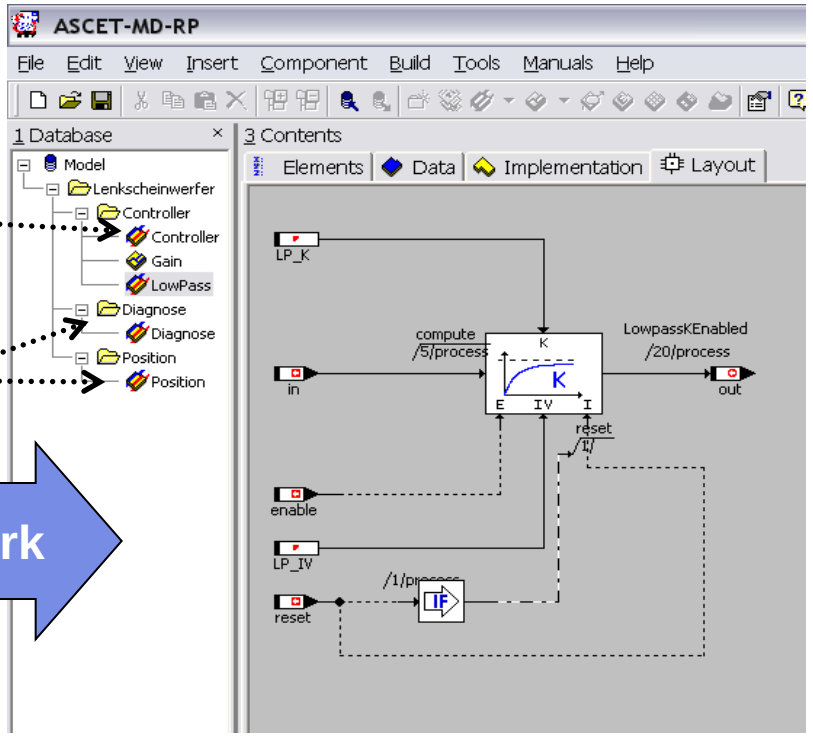
Modell Transformation

UML Klassen Modell
Architektur

Signalfluss Modell
Verhaltensmodellierung

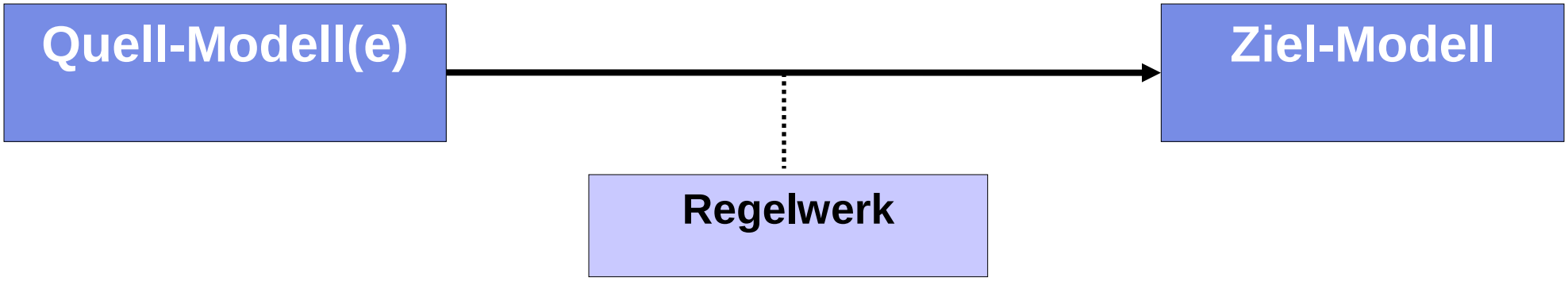


Regelwerk

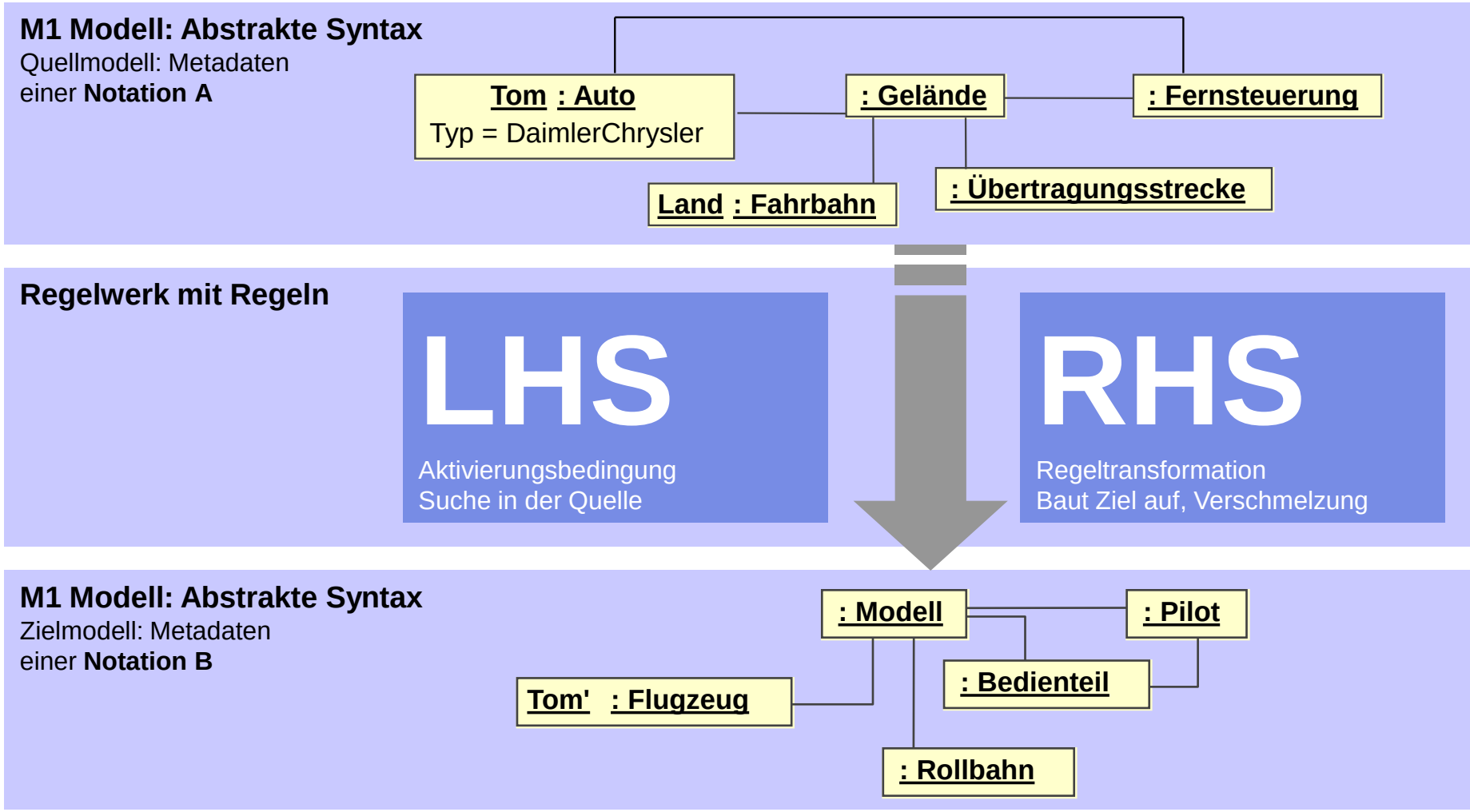


Modell Transformation

- Unter einer **Modell Transformation** versteht man die **Verwandlung** eines Modells in ein anderes. Eine **lokale Veränderung** wird mit einer **Regel** spezifiziert.
- Regeln werden im **Regelwerk** zusammengefasst

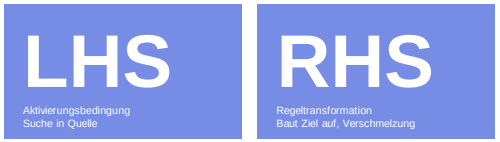


Prinzip der Modell-zu-Modell Transformation



Transformationsregel

- **Eine Regel besteht aus zwei Teilen:**
  - LHS (left hand side): Spezifiziert Suche
    → Aktivierungsbedingung
  - RHS (right hand side): Spezifiziert Zusammenbau
    → Regeltransformation

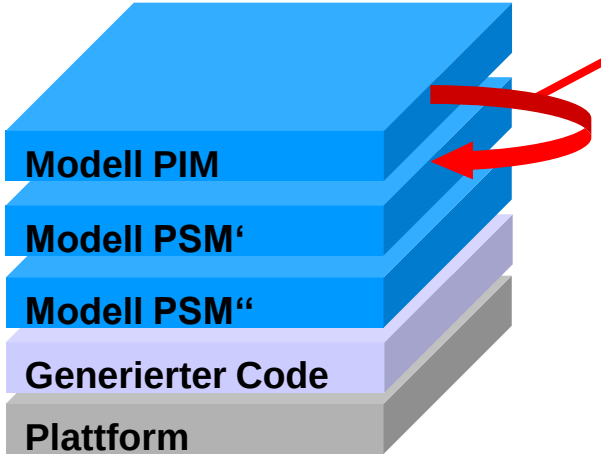


- **Muster**
  - Abstrakte Syntax: Metamodell, detailliert
  - Konkrete Syntax: Textbasiert oder grafisch
- **Metavariable**
  - Nicht typisiert
  - Syntaktisch typisiert (z.B. Ausdruck, Metaklasse)
  - Semantisch typisiert (z.B. Ausdruck ergibt eine Ganzzahl)
- **Logik**
  - Deklarativ (z.B. OCL, Prolog)
  - Imperative (z.B. JAVA, UML Action Language)

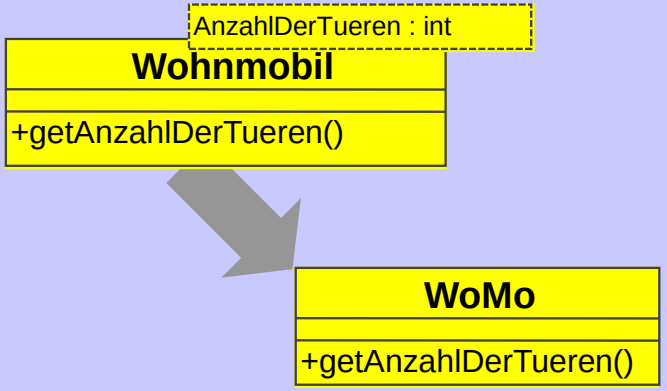
Kategorisierung: Modell Transformation

Modell-zu-Text (Quellcode)
Vom Modell zum Quellcode

Modell-zu-Modell
Intermodell Transformation



M2M: UML Template Mechanismus



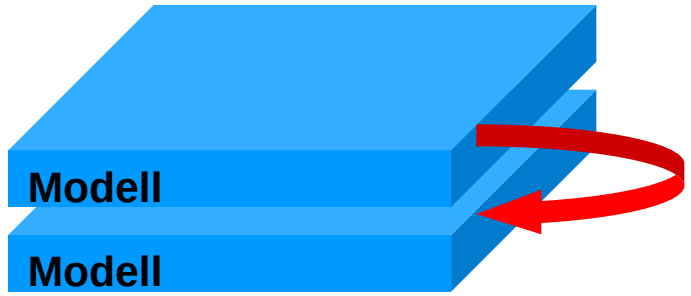
Modell-zu-Text Transformation

```
## --------- Methods ---------
#foreach($method in $class.methods)
   #if($class.kind == "class")
      #if($method.isClassScope() || $method.kind != $method.EXTERN)
         #printMethod($method)
      #end##if
   #end##if
#end##foreach                    Velocity Template Language
```

PIM: platform independent model
PSM: platform specific model

Ansätze: Modell-zu-Modell Transformation

- **Direkte Manipulation**
  Systemnahe Lösung, Entwicklung schwierig
  - **MOLA** (MOdel transformation LAnguage) strukturierter Programmierung und Mustersuche
- **Struktureller Ansatz**
  Zwei Phasen:
  Hierarchie, dann Relationen und Attribute
  - Sodius Scriptor Transformator (Regel Framework)
- **Graphersetzungs Ansatz**
  Typisiert, beschriftet, mit Attributen, sehr komplex
  - **GReAT** (Metamodelling Tool GME)
- **Relationaler Ansatz**
  Verwendung von Relationen und Bedingungen
  - **XMOF** (SUN, Compuware)
- **Hybrider Ansatz**
  - **UMLX** (UML Erweiterung) ATL Atlas Transformation Language
  - **XSLT** (XMI Repräsentation)
  - **M²TOS**

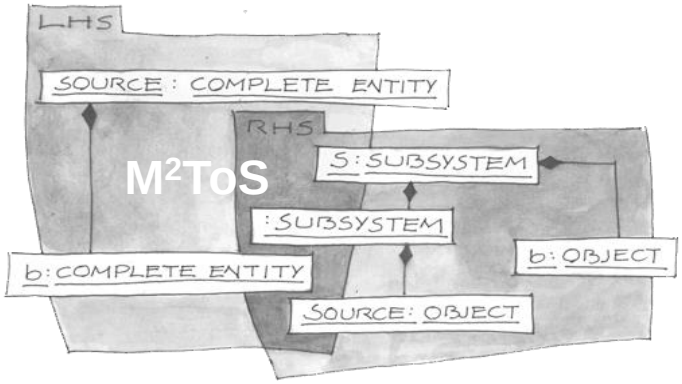


Anwendungsgebiete von Modell-zu-Modell Transformationen

- Modell-**Austausch**
  - Unterschiedliche Notationen
- Modell-**Regelprüfungen**
  - Konsistenzprüfungen
  - Vollständigkeitsprüfungen
  - Modellierungs-Richtlinien (Good Practice) eingehalten
- Modell-**Optimierungen**
  - Z.B. komplexe Blockdiagramm-Strukturen durch Bibliotheksfunktion ersetzten
- Modell-**Refactoring**
  - Funktion verschieben oder austauschen (z.B. Blöcke austauschen)
  - Struktur-Änderungen

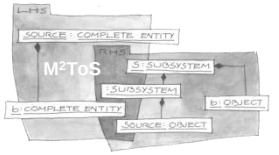
Modell-zu-Modell-Transformations-Sprache (M^2ToS)

- Schwerpunkt
  - Modell-Austausch zwischen CASE Werkzeugen
- Modellbasierte Regelspezifikation
  - Kommunikation
    - **Grafische** Notation
    - **Übersichtliche** Darstellung
    - Auf etablierten Standard aufbauen
      → Unified Modelling Language (UML)
    - Einfache **Wartbarkeit**



  - Komplexe Transformationen
    - **Makrostruktur**
      - Organisation in Klassendiagramm des Quell-Metamodells
    - **Mikrostruktur**
      - Kleine partielle Regeln als Objektdiagramme

Architektur

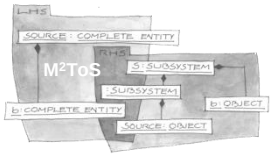


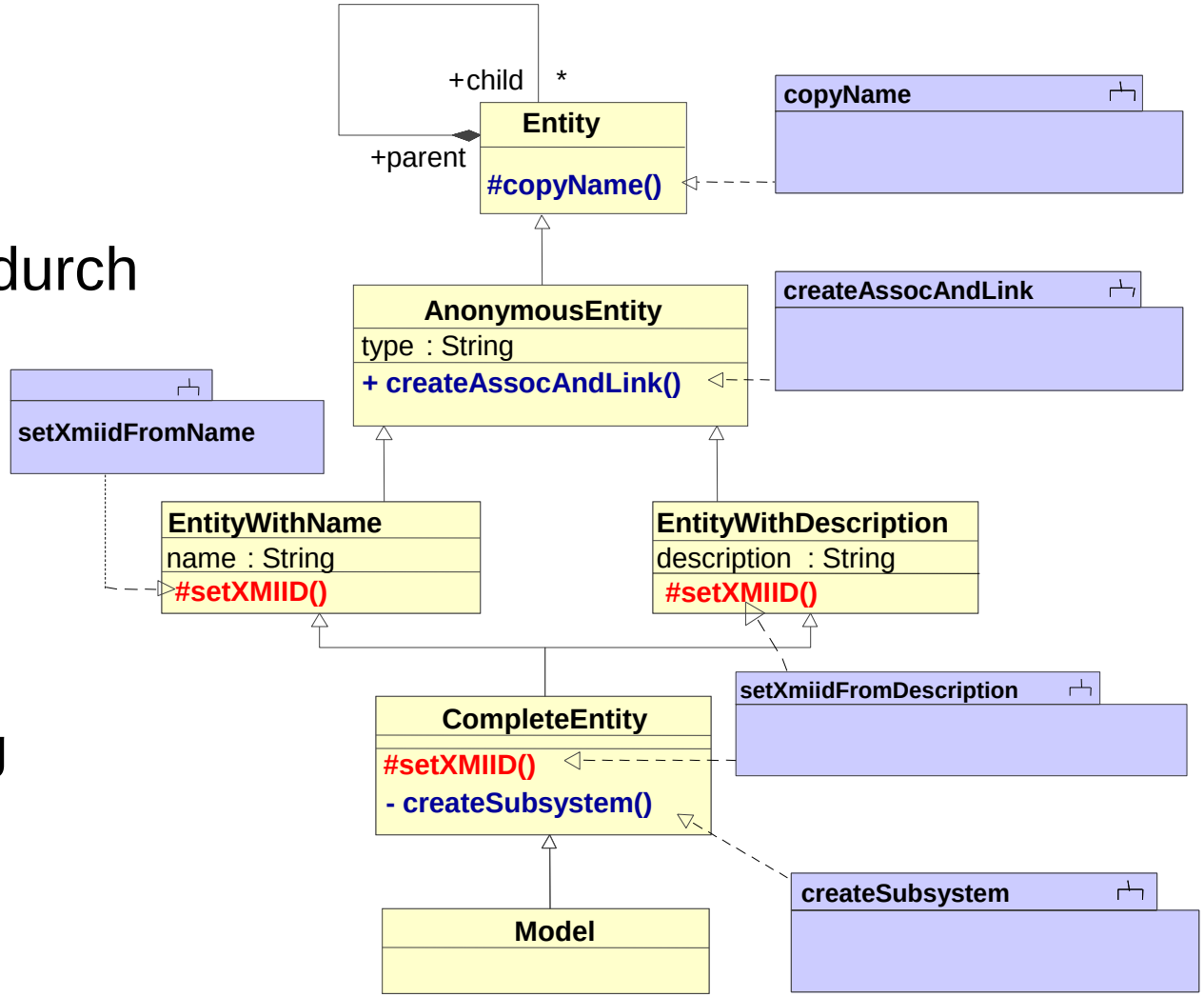
Makrostruktur: Organisation

- Regelorganisation
  - Anhand des Quell-Metamodells
  - Regelzuordnung
  - Gruppierung
- Wiederverwendung durch Vererbung
  - Sichtbarkeiten:

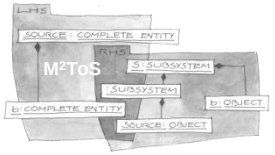
    #Deklaration

    +Öffentliche Regel
    - Lokale Regel
  - Mehrfachvererbung
- Kopiermechanismus





Mikrostruktur: Lokale Transformationsspezifikation



UMLClass_2_Module

LHS

source : Class

modelElement-taggValue

a : TaggedValue

name == RefClass
dataValue

RHS

: SDFolder

{ID=elementName}
elementName =>ASIS

dBase-folder

<<root>>
: DB

folder-component

source : SDModule

folder-component

: SDInstance

{ID=elementName, elementID}
elementName = Utility::CreateClassName(source.name)
elementID = "SDCC_" + source.getUUID()

module-moduleVariable

classComponent-classInstance

a : SDAttribute

elementName = a.dataValue
elementID = "SDCI_" + source.getUUID()

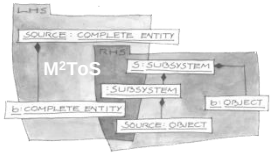
Class

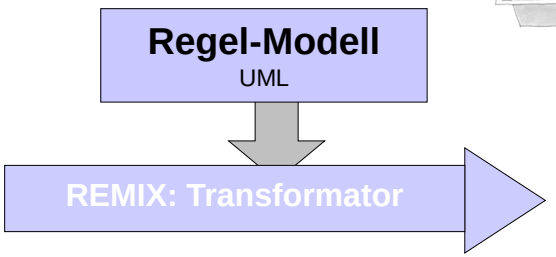
+isActive[1] : Boolean

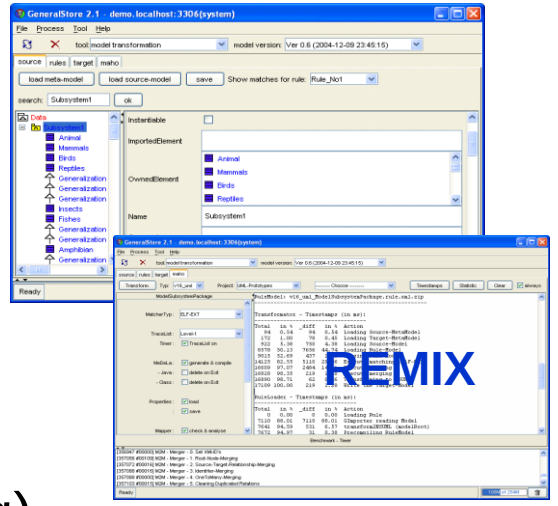
-RG201()
-RG202()
-UMLClass_2_Module()

Modell-zu-Modell Transformator

- Umsetzung der Beschreibungsmittel von M²ToS
- Optimierend: aus Regelmodell
  - Teil generiert
  - Teil interpretiert
  - → sehr gute Performanz
- Hilfe bei der Erarbeitung von Transformationen
  - Regeln getrennt ausführen
    - Mustererkennung (Anzahl der Treffer, welche)
  - Regelmodell Analyse (Modellierungsfehler)
  - Verschmelzungsanalyse (weshalb was)
  - Performanzanalyse (Zeitbedarf in den Phasen)
  - Quellmodell Eingabe
  - Zielmodell Analyse
  - Effektivitäts-Analyse (Treffer zu Verschmelzung)







Fragen ?