MTFLOW – Ein System für workflow-basierte Ausführung von Modelltransformationen

Mohamed Amine Chatti
chatti@informatik.uni-kl.de

19. Mai 2004

Motivation

- **Ziel der Arbeit**
  - Den Ansatz der workflow-basierten Modelltransformationen analysieren und empirisch bewerten
  - MTFLOW (Model Transformation workFLOWs): Eine Prototyp-Implementierung

- **MTFLOW – Überblick**
  - Schnelle Spezifikation von Software-Systemen in einer Produktlinie mit Hilfe der Sprachen UML und OAL
  - Die Spezifikation entsteht durch die aufeinanderfolgende Anwendung von Modelltransformationen, ausgeführt mit Hilfe der Sprache XMI
  - Die Reihenfolge der angewendeten Transformationen wird mit einem Workflow-Modell überwacht

Gliederung

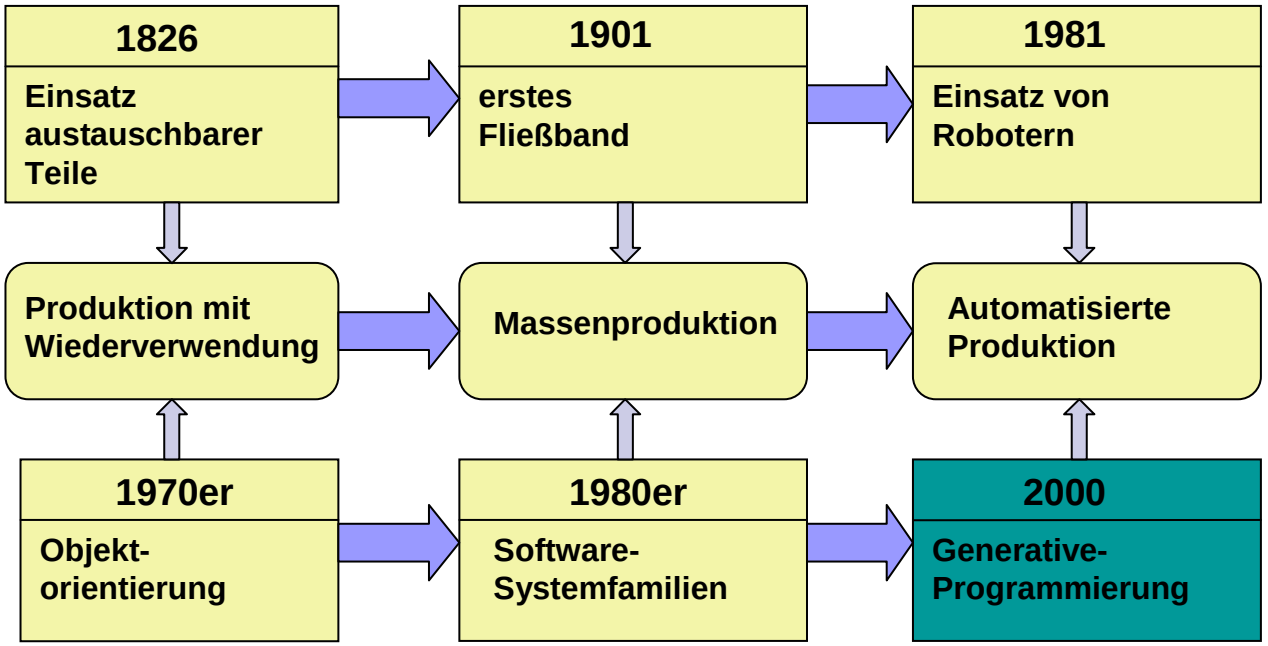
- **Produktlinienansatz und generative Programmierung**
- **Modelltransformationen**
- **Metamodell für MTFLOW-Workflow-Modelle**
- **Versionierungssysteme als Anwendungsbeispiel**
- **Empirische Untersuchungen**
- **Zusammenfassung und Ausblick**

Generative Programmierung

- **Beziehung zu dem Produktlinienansatz bzw. der generativen Programmierung**
  - SW-Produktlinie
  - Analogie Softwareentwicklung-Industrie

Meilensteine der …

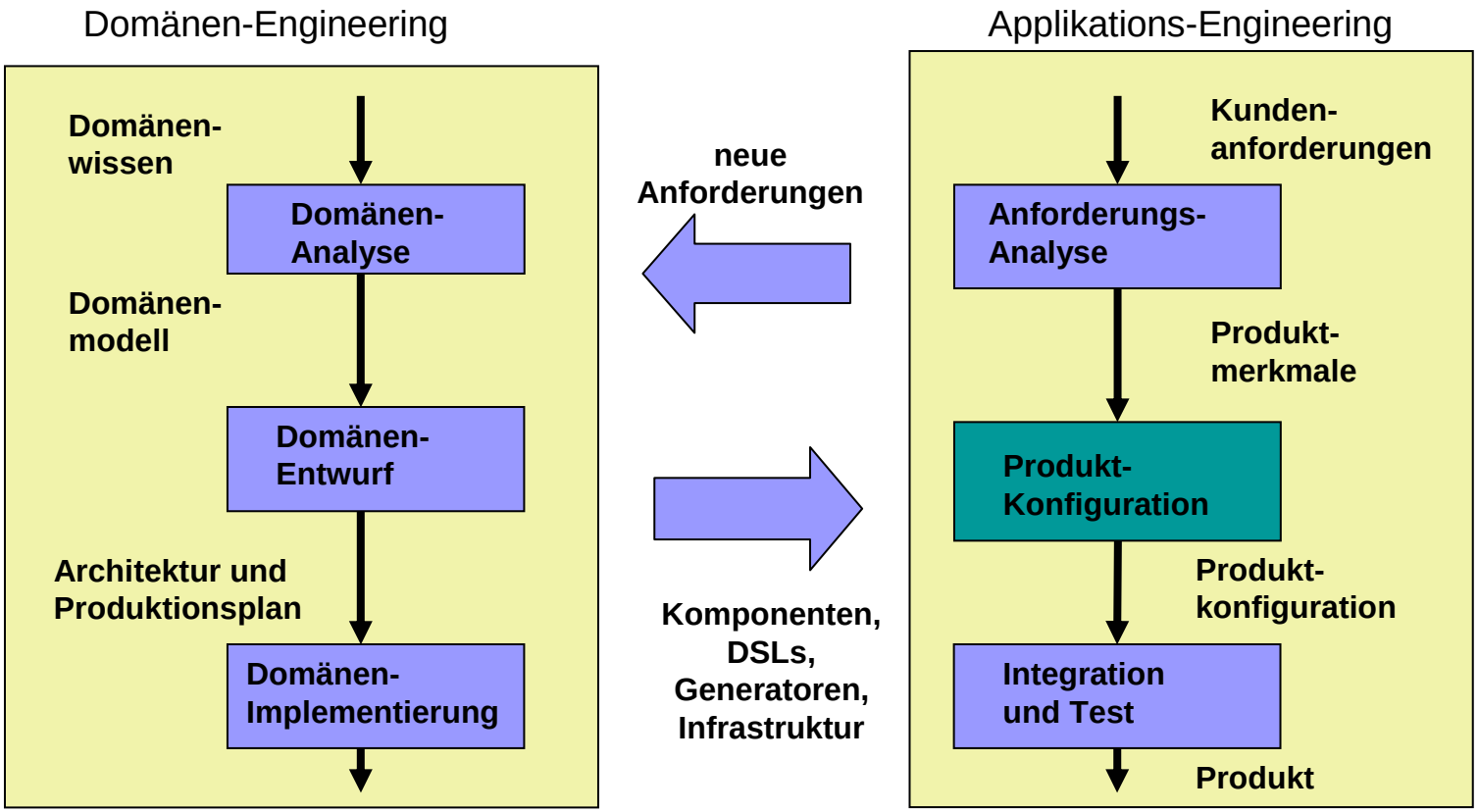
…Industrialisierung



…Software-Entwicklung

Generative Programmierung

- **Ziele der generativen Programmierung**
  - Die Entwicklung von SW-Systemen vereinfachen, verbessern und beschleunigen
  - Automatisierte Produktion der Systeme in einer SW-Systemfamilie
- **Phasen der Produktlinienentwicklung**



Gliederung

- Produktlinienansatz und generative Programmierung
- Modelltransformationen
- Metamodell für MTFLOW-Workflow-Modelle
- Versionierungssysteme als Anwendungsbeispiel
- Empirische Untersuchungen
- Zusammenfassung und Ausblick

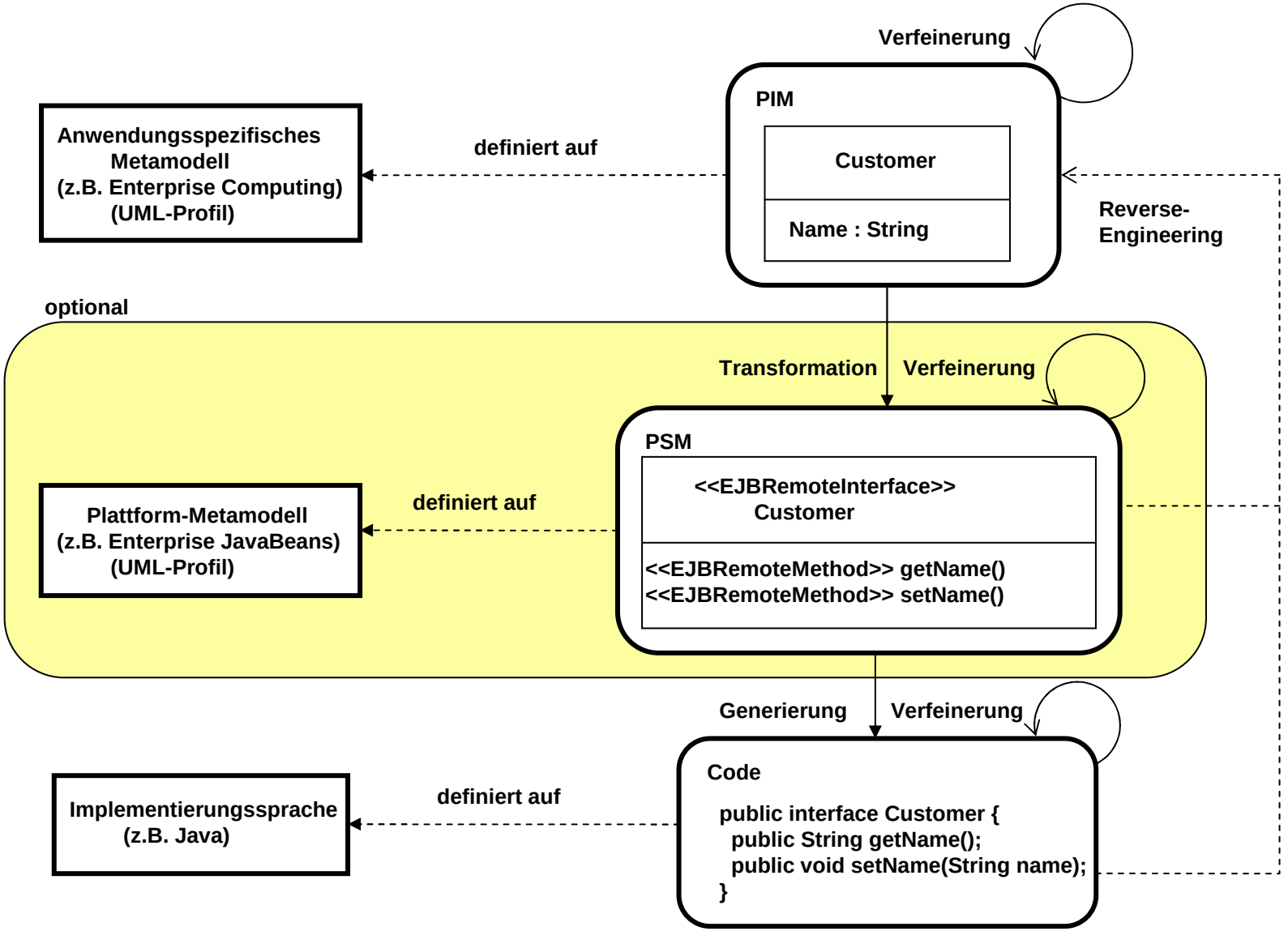
Modelltransformationen in MDA

- **MDA (Model Driven Architecture)**
- Trennung der Definition der Funktionalität eines Systems von der Definition der Implementierung dieser Funktionalität für eine bestimmte Plattform

- Modelle in MDA
  - PIM (Platform-Independent Model)
  - PSM (Platform-Specific Model)

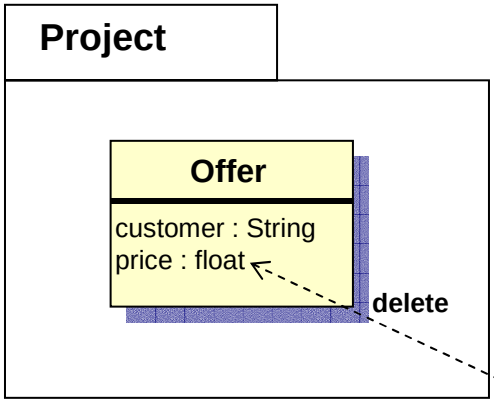
- Abbildung zwischen Modellen
  - PIM zu PIM
  - PIM zu PSM
  - PSM zu PSM
  - PSM zu PIM

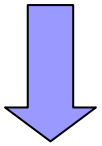
Modelltransformationen in MDA

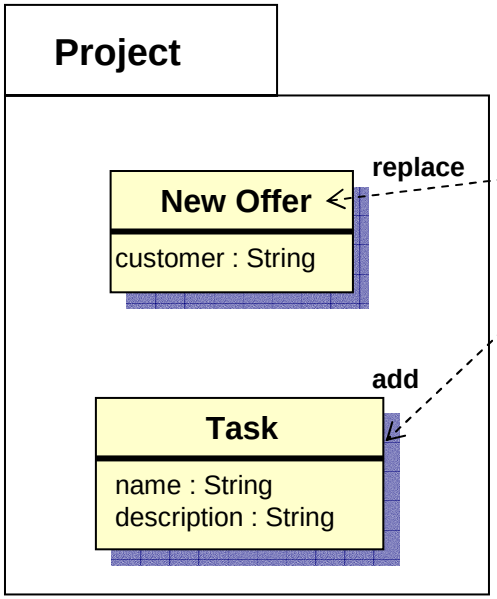


Modelltransformation mit XMI

- **XMI (XML Metadata Interchange)**
  - XMI ist ein textbasiertes Austauschformat für Modelle
  - xmi.id, xmi.uuid
  - idref, href
  - XMI.difference, XMI.add, XMI.delete, XMI.replace
  - Model Merging








```
<XMI.content>
 <UML:Package xmi.id=„p1“ name=„Project“>
  <UML:Class xmi.id=„c1“ name=„Offer“>
   <UML:Classifier.feature>
    <UML:Attribute xmi.id=„a1“ name=„customer“>
     <UML:StructuralFeature.type>
      <UML:DataType xmi.idref=„s“>
      </UML:DataType>
     </UML:StructuralFeature.type>
    </UML:Attribute>
    <UML:Attribute xmi.id=„a2“ name=„price“>
     <UML:StructuralFeature.type>
      <UML:DataType xmi.idref=„f“>
      </UML:DataType>
     </UML:StructuralFeature.type>
    </UML:Attribute>
   </UML:Classifier.feature>
  </UML:Class>
 </UML:Package>
</XMI.content>
```

original.xmi

```
<XMI.content>
 <XMI.difference>
  <XMI.add href=„original.xmi | p1“ >
   <UML:Class xmi.id=„c2“ name=„Task“>
    <UML:Classifier.feature>
     <UML:Attribute xmi.id=„a3“ name=„name“>
      <UML:StructuralFeature.type>
       <UML:DataType xmi.idref=„s“>
       </UML:DataType>
      </UML:StructuralFeature.type>
     </UML:Attribute>
     <UML:Attribute xmi.id=„a4“ name=„description“>
      <UML:StructuralFeature.type>
       <UML:DataType xmi.idref=„s“>
       </UML:DataType>
      </UML:StructuralFeature.type>
     </UML:Attribute>
    </UML:Classifier.feature>
   </UML:Class>
  </XMI.add>
  <XMI.delete href=„original.xmi | a2“/>
  <XMI.replace href=„original.xmi | c1“>
   <UML:Class xmi.id=„c1“ name=„New Offer“/>
  </XMI.replace>
 </XMI.difference>
</XMI.content>
```

difference.xmi

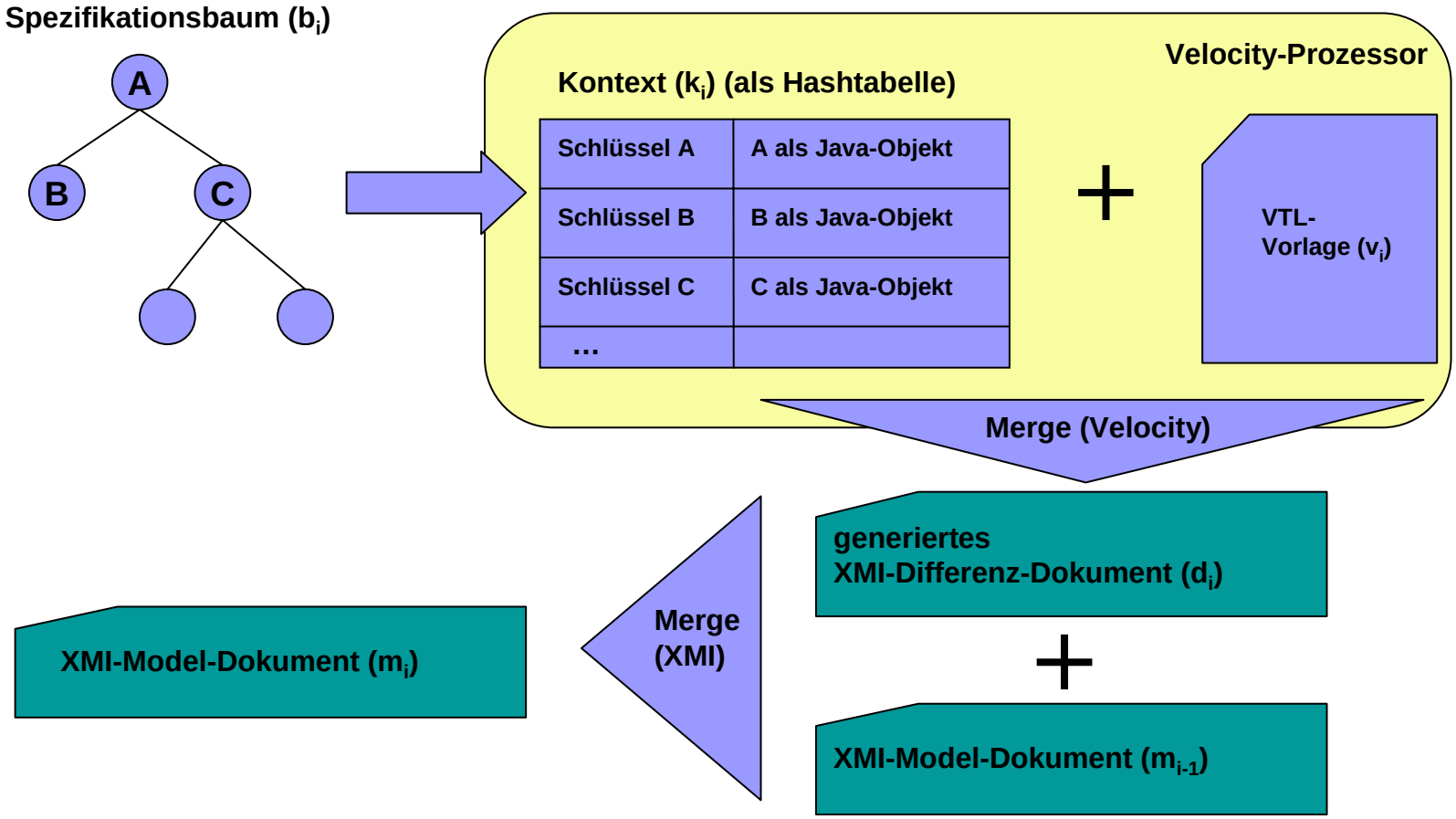
```
<XMI.content>
 <UML:Package xmi.id=„p1“ name=„Project“>
  <UML:Class xmi.id=„c1“ name=„New Offer“>
   <UML:Classifier.feature>
    <UML:Attribute xmi.id=„a1“ name=„customer“>
     <UML:StructuralFeature.type>
      <UML:DataType xmi.idref=„s“>
      </UML:DataType>
     </UML:StructuralFeature.type>
    </UML:Attribute>
   </UML:Classifier.feature>
  </UML:Class>
  <UML:Class xmi.id=„c2“ name=„Task“>
   <UML:Classifier.feature>
    <UML:Attribute xmi.id=„a3“ name=„name“>
     <UML:StructuralFeature.type>
      <UML:DataType xmi.idref=„s“>
      </UML:DataType>
     </UML:StructuralFeature.type>
    </UML:Attribute>
    <UML:Attribute xmi.id=„a4“ name=„description“>
     <UML:StructuralFeature.type>
      <UML:DataType xmi.idref=„s“>
      </UML:DataType>
     </UML:StructuralFeature.type>
    </UML:Attribute>
   </UML:Classifier.feature>
  </UML:Class>
 </UML:Package>
</XMI.content>
```

new.xmi

Modelltransformation mit XMI

- **Erweiterung der XMI-Spezifikation**

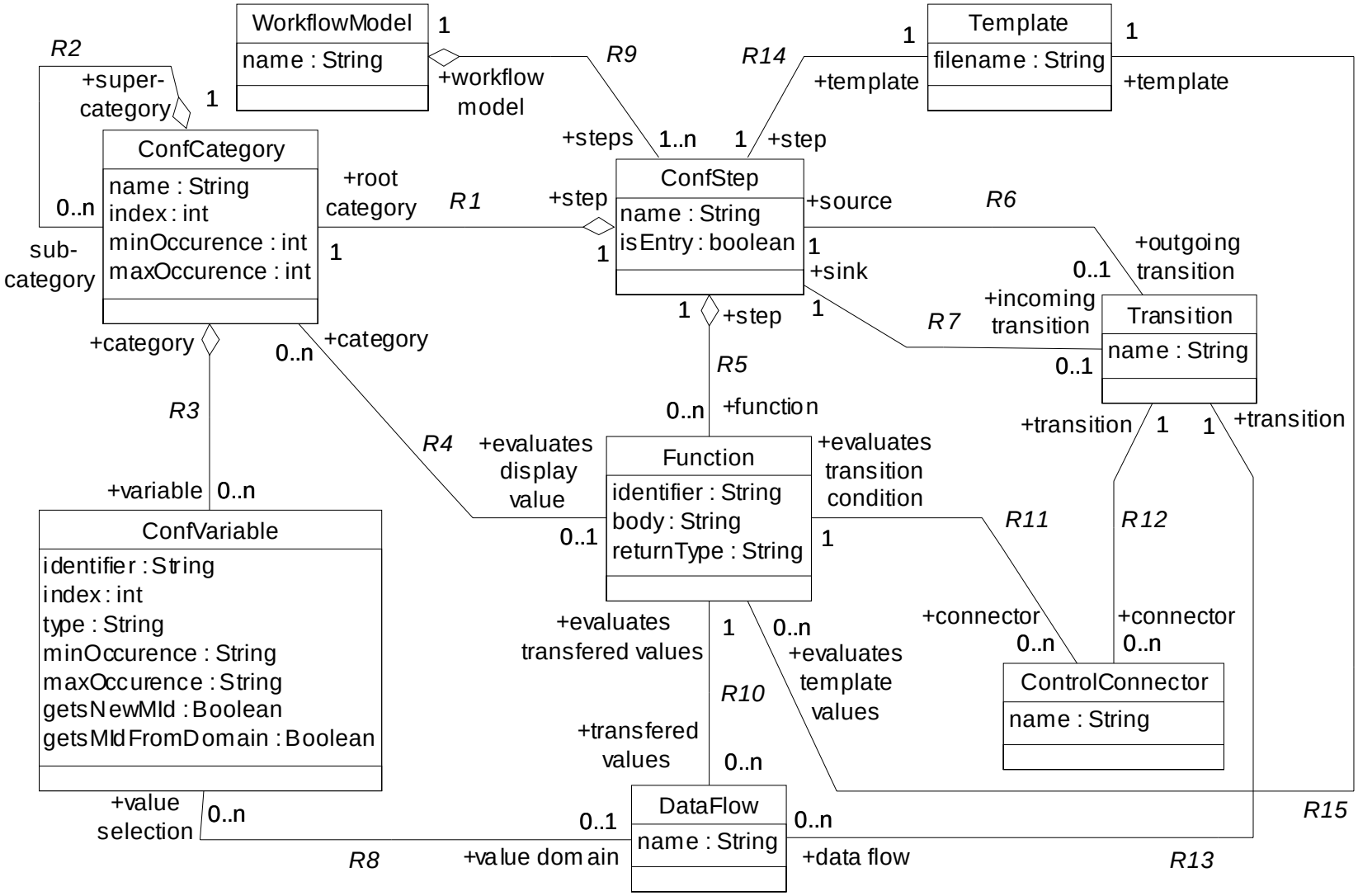
- **Ablauf der Modelltransformation in MTFLOW**



Gliederung

- Produktlinienansatz und generative Programmierung
- Modelltransformationen
- Metamodell für MTFLOW-Workflow-Modelle
- Versionierungssysteme als Anwendungsbeispiel
- Empirische Untersuchungen
- Zusammenfassung und Ausblick

Metamodell für MTFLOW-Workflow-Modelle



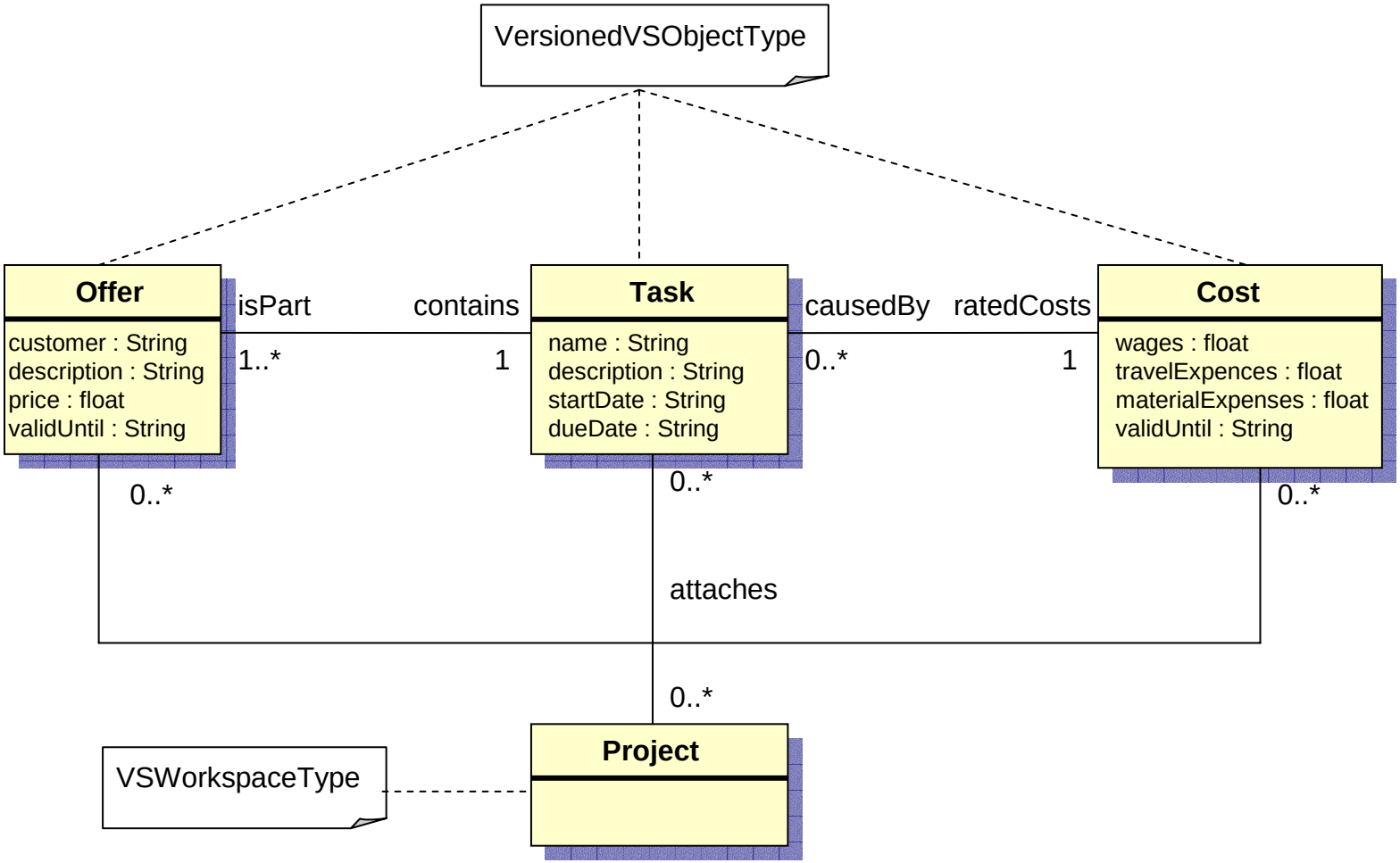
Gliederung

- Produktlinienansatz und generative Programmierung
- Modelltransformationen
- Metamodell für MTFLOW-Workflow-Modelle
- Versionierungssysteme als Anwendungsbeispiel
- Empirische Untersuchungen
- Zusammenfassung und Ausblick

Versionierungssysteme

- **Speicherung versionierter und nicht versionierter Objekte**
- **Attribute**
- **Operationen auf Objekten**
- **Beziehungen zwischen Objekten**
  - Gleitende Beziehungsende (floating relationship ends)
  - Kandidatenmenge (candidate version collection)
  - Zugriffsmöglichkeiten:
    Vorauswahl von Objektversionen (pinning), regelbasierte Auswahl, gesamte CVC
- **Propagierung von Operationen**
- **Arbeitskontexte**
- **Informationsmodell**

Informationsmodell

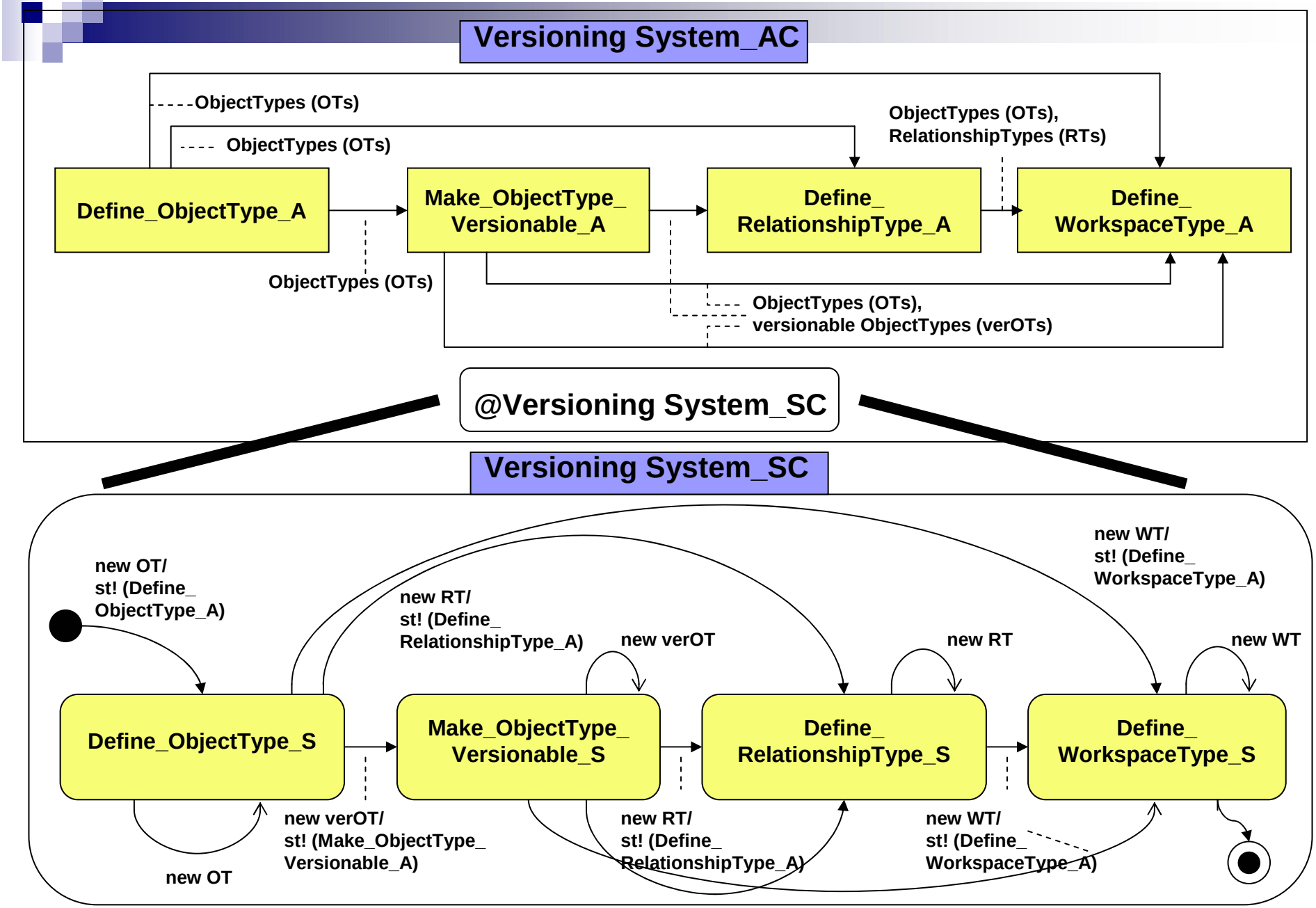


Versionierungssysteme

- **Viele Variationsmöglichkeiten**
  - Angaben über versioniert und unversioniert zu speichernde Objekttypen
  - Beziehungen
  - Arbeitskontexte
  - Propagierung von Operationen über Beziehungen

⟹ **Versionierungssysteme als Produktlinie**

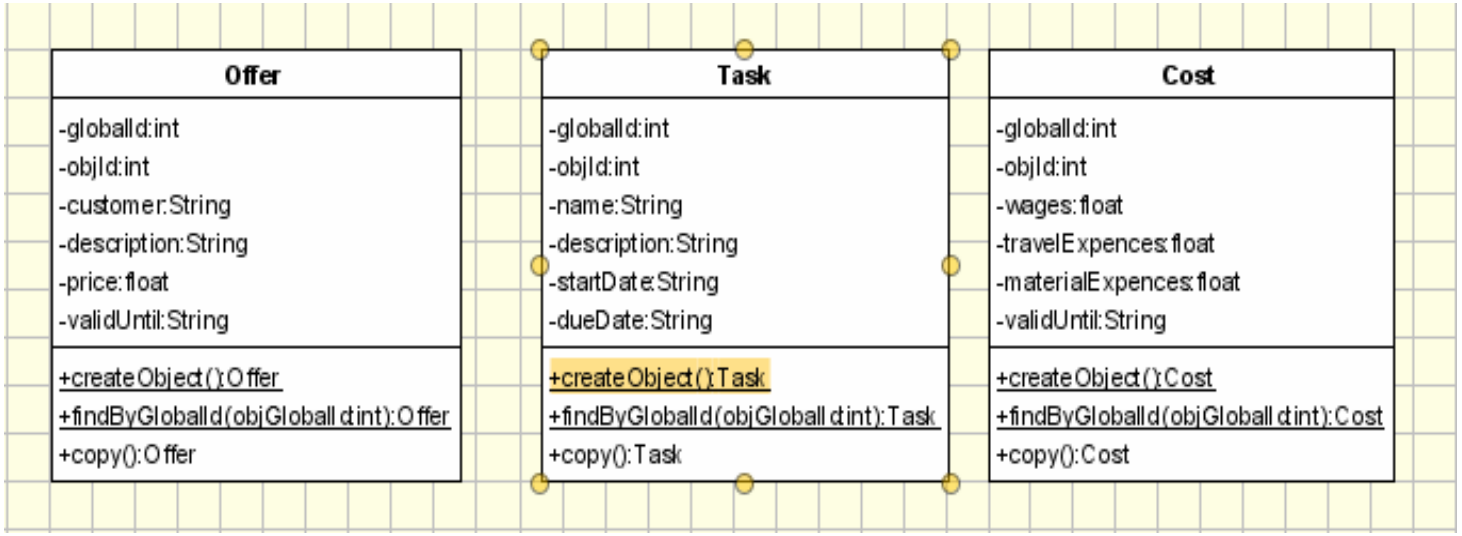
- **Workflow-Modell für Versionierungssysteme**
  - Modellieren von Workflow-Prozessen mit Petri-Netzen oder Statecharts und Activitycharts
  - Schrittweise Verfeinerung der Modelle ist mit Statecharts und Activitycharts möglich


Versioning System_AC
ObjectTypes (OTs)
ObjectTypes (OTs)
ObjectTypes (OTs),
RelationshipTypes (RTs)
Define_ObjectType_A
Make_ObjectType_
Versionable_A
Define_
RelationshipType_A
Define_
WorkspaceType_A
ObjectTypes (OTs)
ObjectTypes (OTs),
versionable ObjectTypes (verOTs)
@Versioning System_SC
Versioning System_SC
new OT/
st! (Define_
ObjectType_A)
new RT/
st! (Define_
RelationshipType_A)
new verOT
new RT
new WT
new WT/
st! (Define_
WorkspaceType_A)
Define_ObjectType_S
Make_ObjectType_
Versionable_S
Define_
RelationshipType_S
Define_
WorkspaceType_S
new verOT/
st! (Make_ObjectType_
Versionable_A)
new OT
new RT/
st! (Define_
RelationshipType_A)
new WT/
st! (Define_
WorkspaceType_A)

Beispiel

Konfigurationsschritt A

Define ObjectType

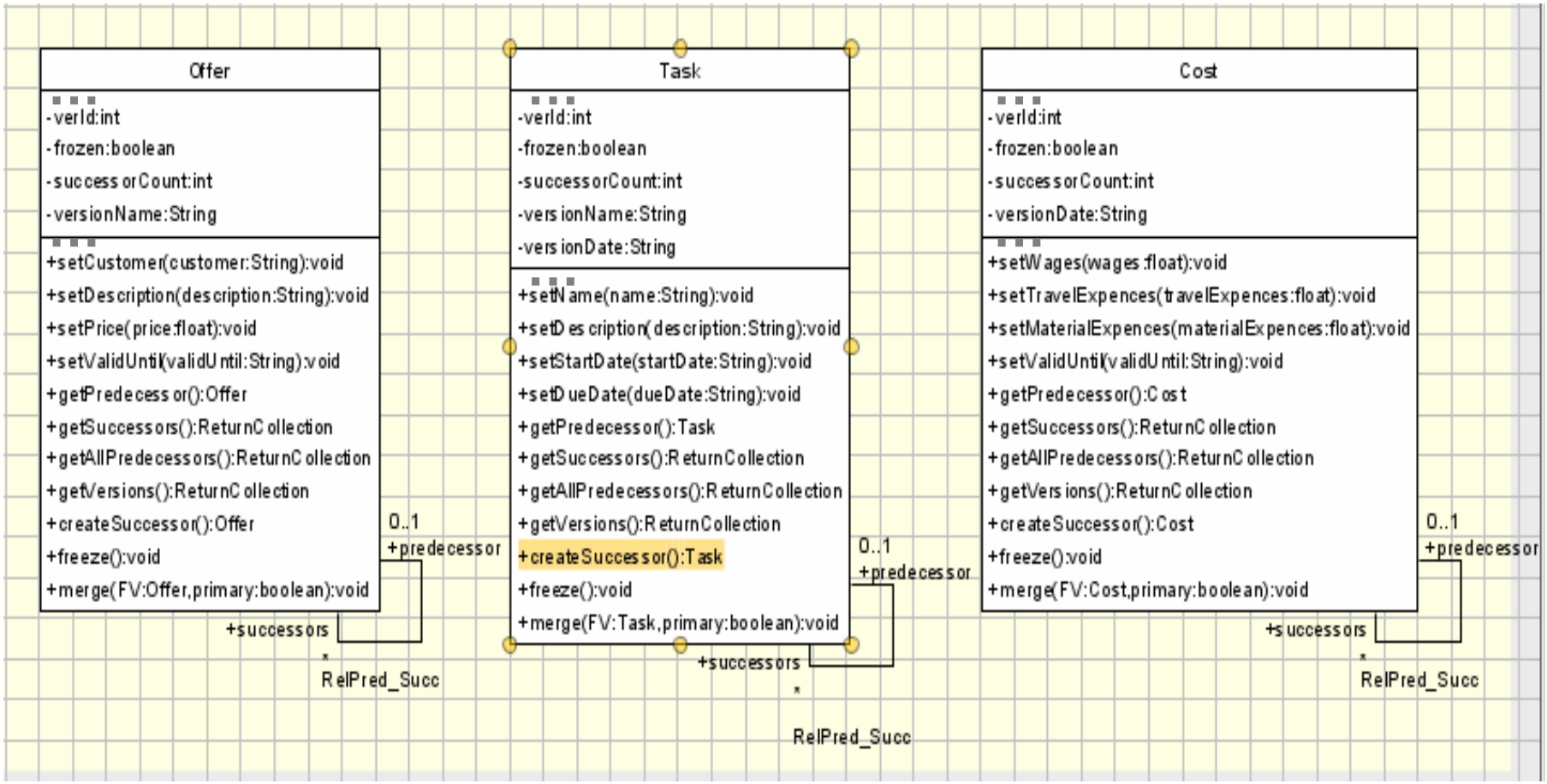



```
public static Task createObject() {          /** lock-end */
    entry//
    //create new instance of Task, set the objId, initialize the attributes
    create object instance newTask of Task;
    select one counter from instances of IdCounter;
    newTask.objId = counter.getNextId();
    newTask.globalId = objId*10000;
    newTask.name = "";
    newTask.description = "";
    newTask.startDate = "";
    newTask.dueDate = "";
    return newTask;
} // end createObject          /** lock-begin */
```

Beispiel

Konfigurationsschritt B

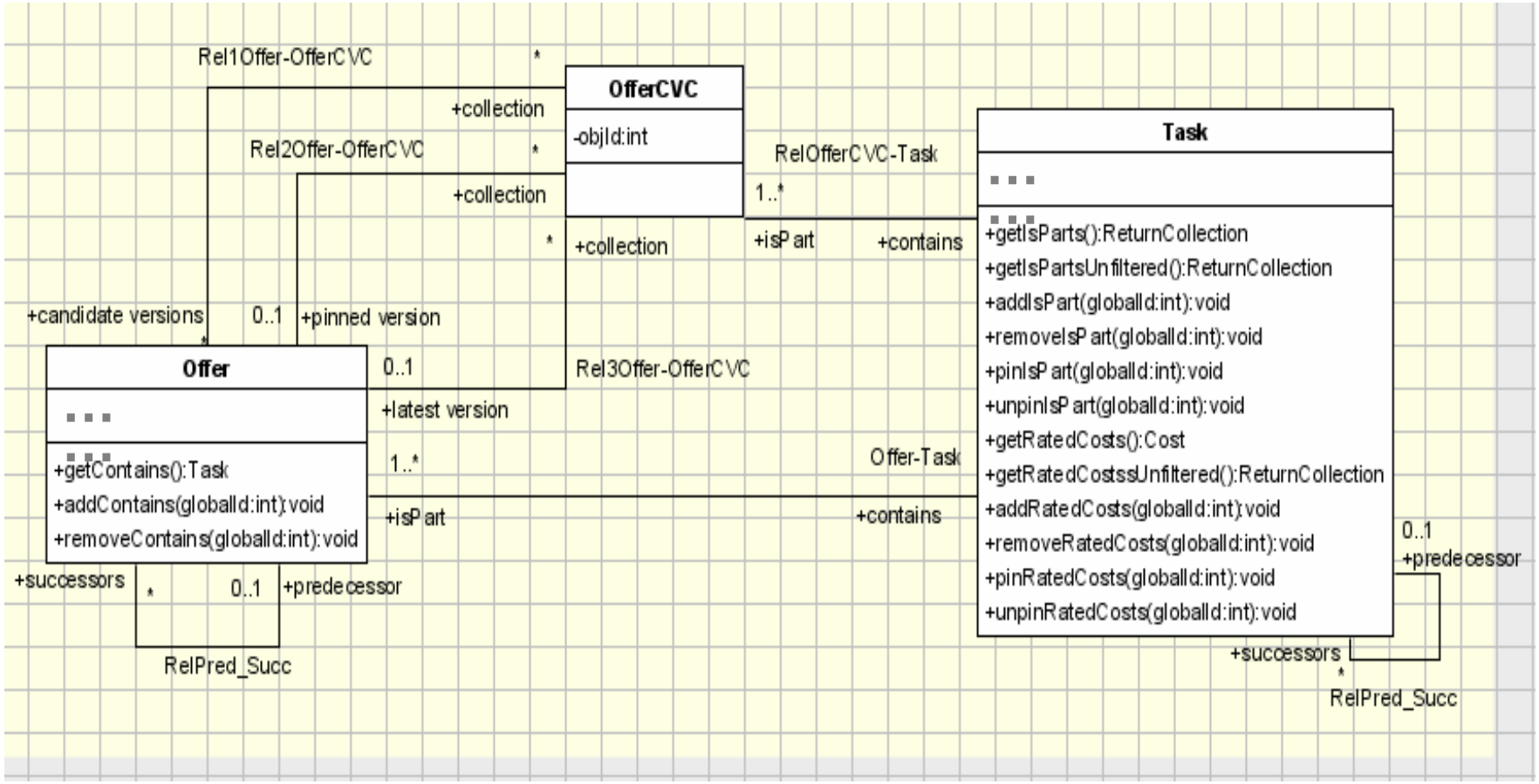
Make ObjectType versionable



Beispiel

Konfigurationsschritt C

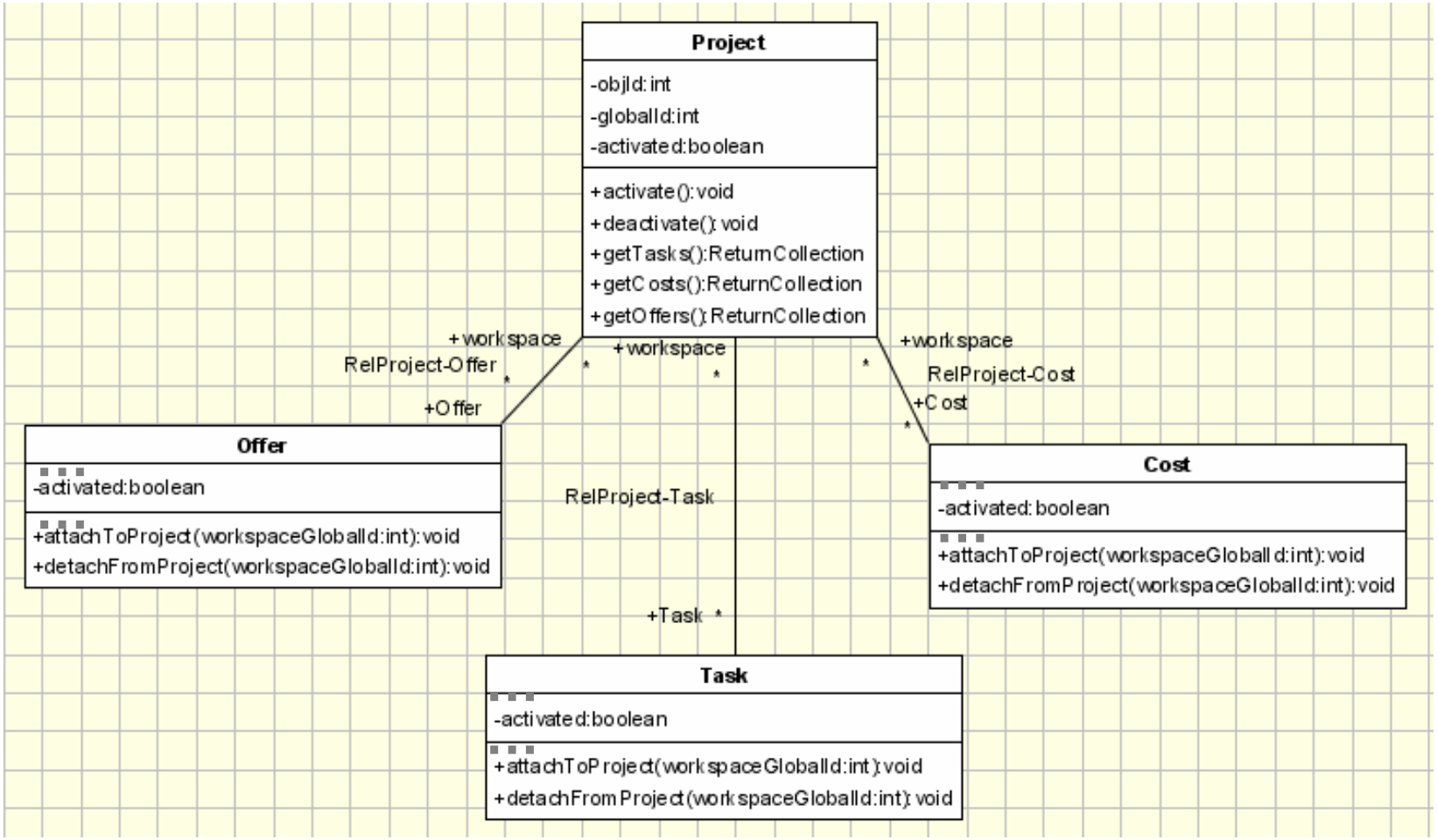
Define RelationshipType



Beispiel

Konfigurationsschritt D

Define WorkspaceType



Gliederung

- Produktlinienansatz und generative Programmierung
- Modelltransformationen
- Metamodell für MTFLOW-Workflow-Modelle
- Versionierungssysteme als Anwendungsbeispiel
- Empirische Untersuchungen
- Zusammenfassung und Ausblick

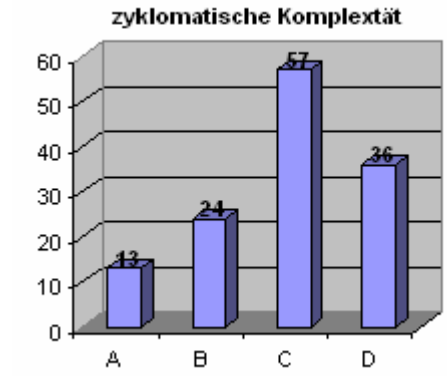
Empirische Untersuchungen

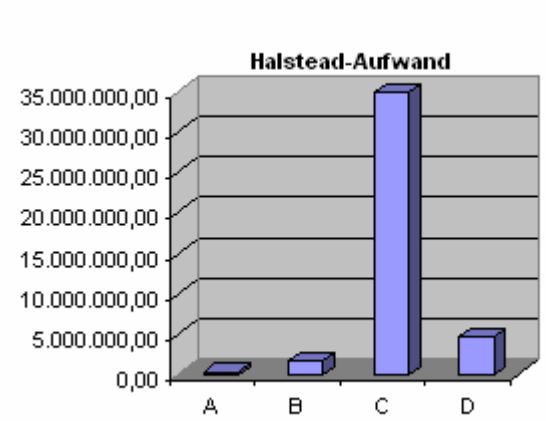
- **durchgeführte Untersuchungen**
  - Analyse der Transformationsvorlagen
  - Analyse der Transformationsergebnisse

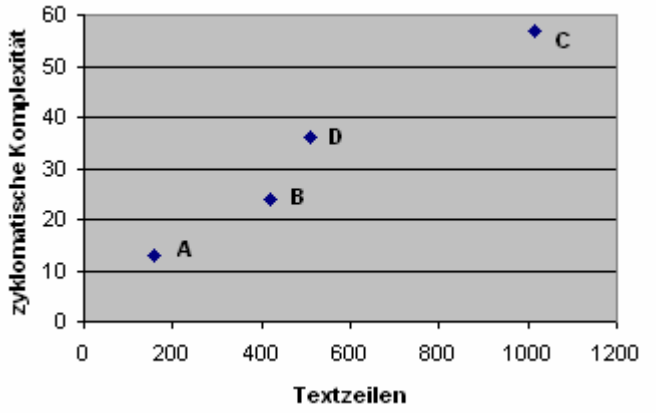
- **verwendete Metriken**
  - Anzahl der Anweisungen
  - zyklomatische Komplexität
  - Halstead-Aufwand
  - MTFLOW-spezifische Metriken

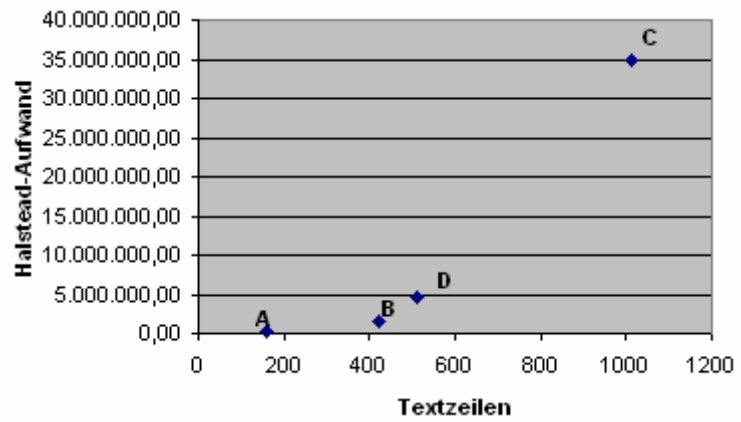
Analyse der Transformationsvorlagen

- Vorlage C hat die höchste Komplexität
- Konstantes Verhältnis zwischen Umfang und Komplexität der Vorlagen
- Anzahl der Anweisungen, Referenzen, Fallunterscheidungen und XMI-Elemente steigen mit der Größe der Vorlagen an



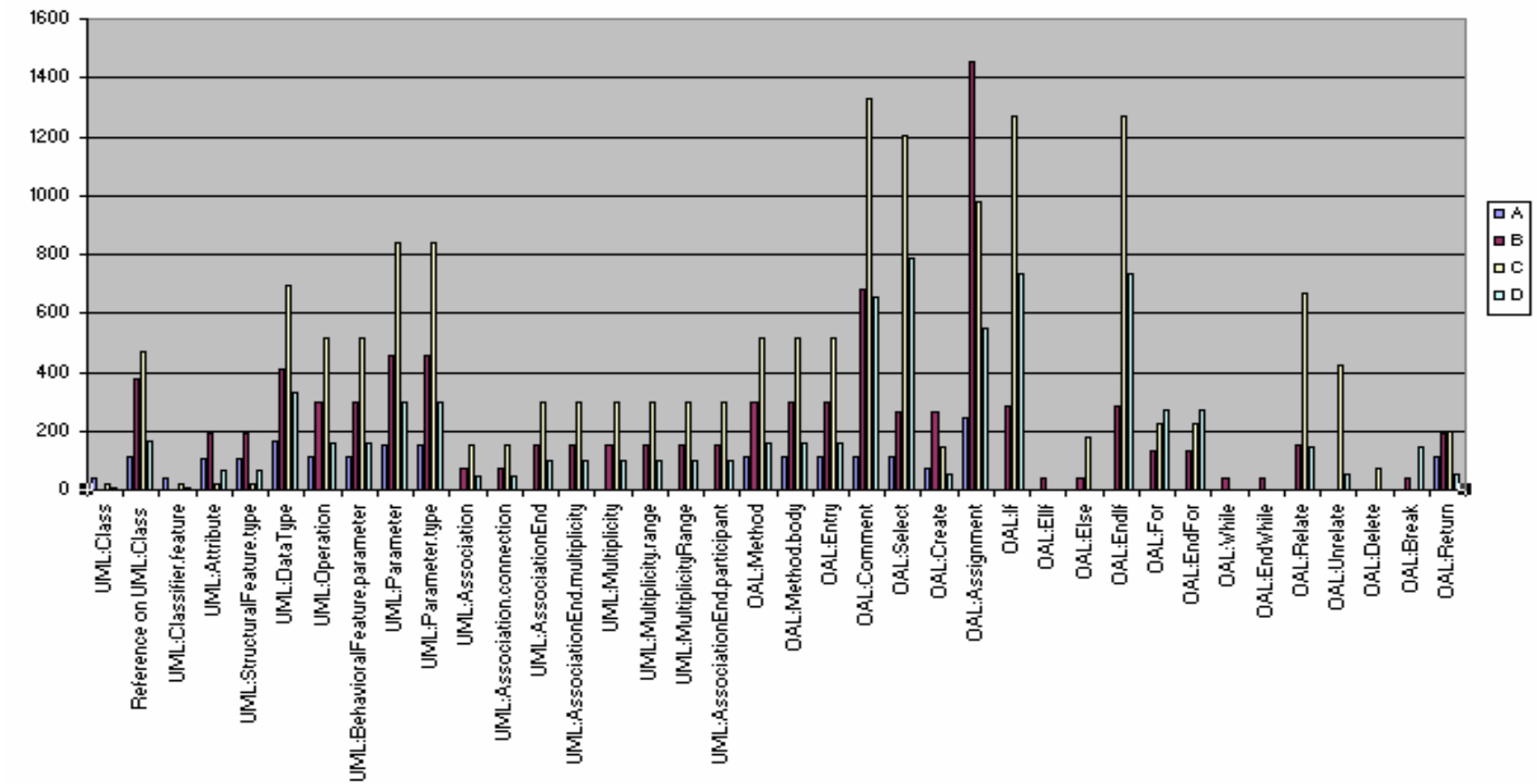






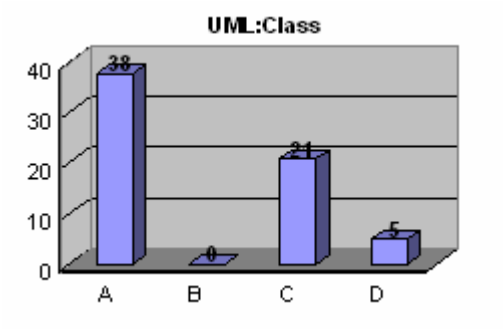
Analyse der Transformationsergebnisse

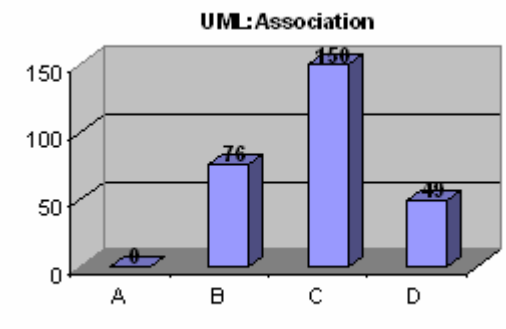
- verschiedene Modelle als Informationsmodelle: UML Core Package, CWM Relational Package, MOF Model Package
- Ermittlung der Anzahl der verschiedenen generierten XMI-Elemente in jedem Konfigurationsschritt

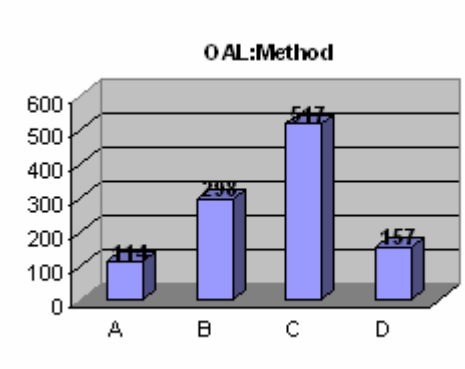


Analyse der Transformationsergebnisse

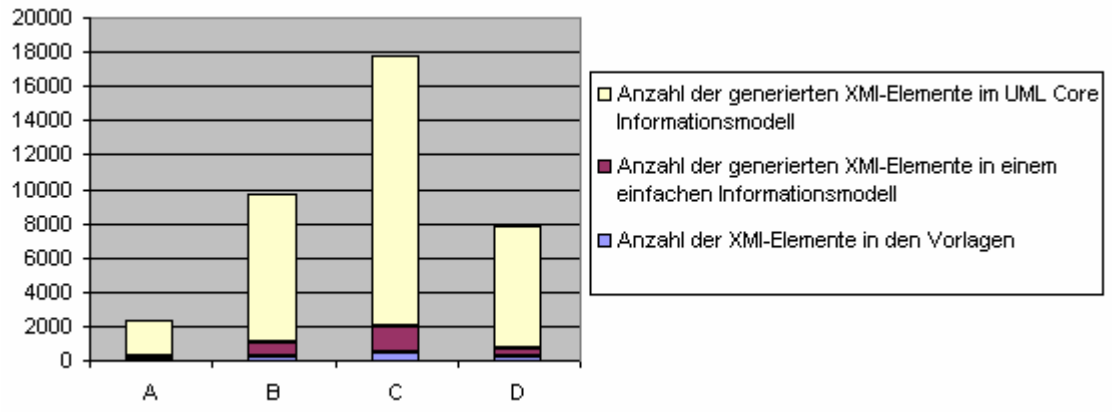
- Anzahl der UML:Class-, UML:Association- und OAL:Method-Elemente





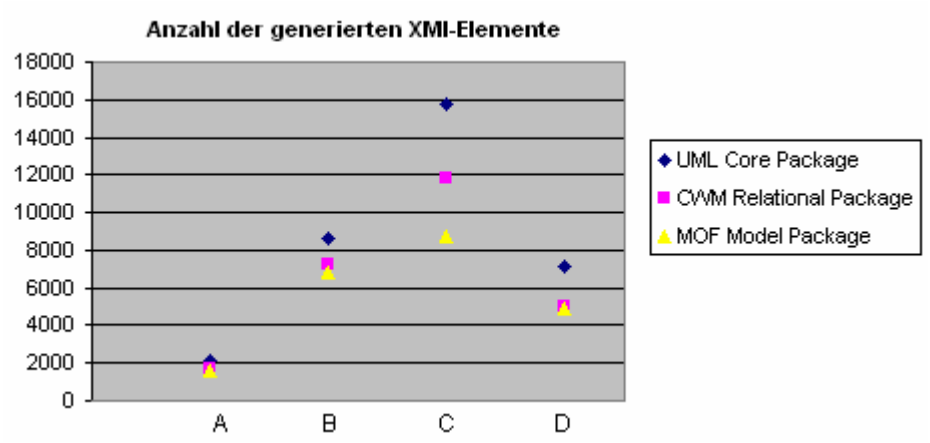


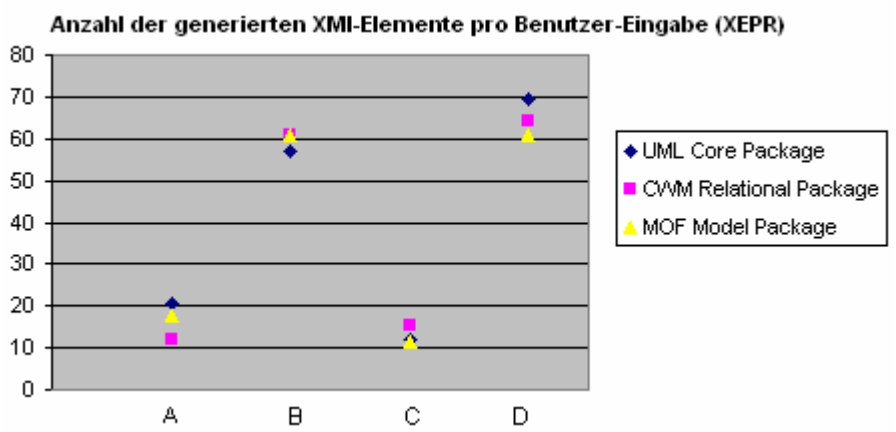
- Anzahl der generierten XMI-Elemente im Vergleich mit der Anzahl der in den Vorlagen enthaltenen XMI-Elemente

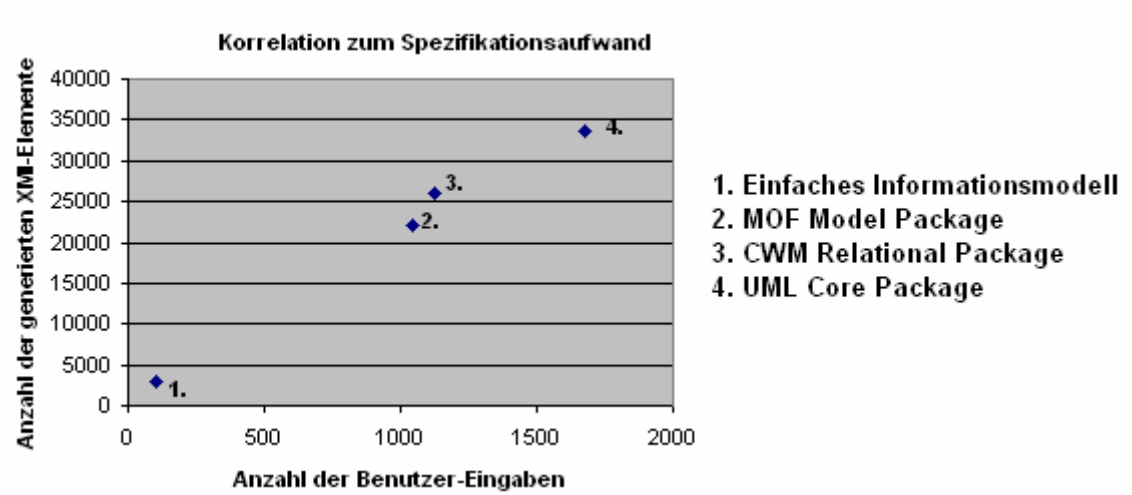


Analyse der Transformationsergebnisse

- Anzahl der generierten XMI-Elemente im Vergleich mit dem Aufwand der Spezifikation eines SW-Systems







Gliederung

- Produktlinienansatz und generative Programmierung
- Modelltransformationen
- Metamodell für MTFLOW-Workflow-Modelle
- Versionierungssysteme als Anwendungsbeispiel
- Empirische Untersuchungen
- Zusammenfassung und Ausblick

Zusammenfassung und Ausblick

- Ein domänenspezifisches Workflow-Modell und eine Menge von domänenspezifischen Vorlagen können bei der Spezifikation eine DSL ersetzen
- OAL als geeignete Sprache zum Modellieren der Semantik
- konstantes Verhältnis zwischen Umfang und Komplexität der Vorlagen
- Vorlage C zu komplex (hoher Halstead-Aufwand)
- Aufteilung der Vorlage C sinnvoll
- Aufwand der Entwicklung der Vorlagen zahlt sich aus
- lineare Abhängigkeit zwischen Anzahl der generierten XMI-Elemente und Aufwand der Spezifikation (Faktor 20)
- Bewertung verschiedener Produktlinien

- **Fragen?**