TU Dresden
Fakultät Informatik
Lehrstuhl Softwaretechnologie

Zwischenbericht zum Thema

XMI für prozedurale Programmstrukturen und Transformation in UML

Hochschullehrer : Prof. Dr. rer. nat. habil. H. Hußmann
Betreuerin: Dr. B. Demuth
Referent: Axel Großmann

Inhalt

- Aufgabenstellung
- Motivation
- Prozedurales Metamodell
  - Anforderungen
  - Erster Entwurf
  - Nutzung der UML
- Erfahrungen mit XMI
  - XMI Toolkit
  - allgemeine Probleme

Aufgabenstellung

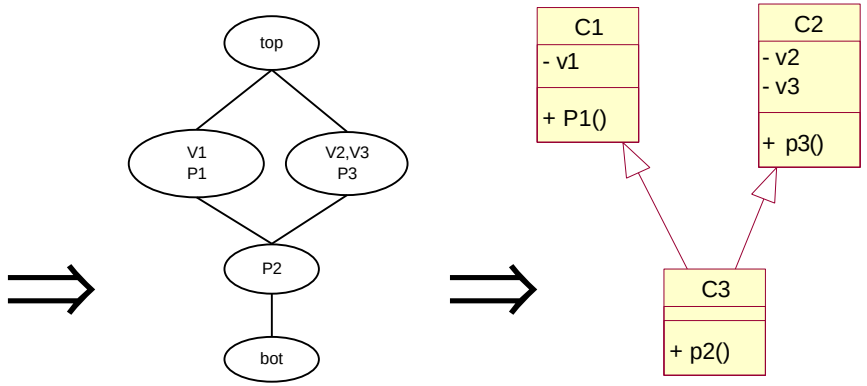
- Entwurf eines Metamodelles für prozedurale Programmstrukturen
- Nutzung von XMI für den Modellaustausch
  - Generierung eines entsprechenden DTD
- Implementierung eines Algorithmus zur Objektidentifikation in prozeduralem Code
  - Transformation prozeduraler in objektorientierte Strukturen
  - Darstellung der Ergebnisse in UML/XMI

Motivation

- Echtes Round-Trip-Engineering
  - –Reverse Engineering nicht nur für objektorientierte Sprachen
  - –Gemeinsame Nutzung verschiedener CASE- und CARE-Tools durch XMI

Bsp.: Objektidentifikation durch *Formale Begriffsanalyse*

Variable Prozedur	V1	V2	V3
P1	x		
P2	x	x	x
P3		x	x



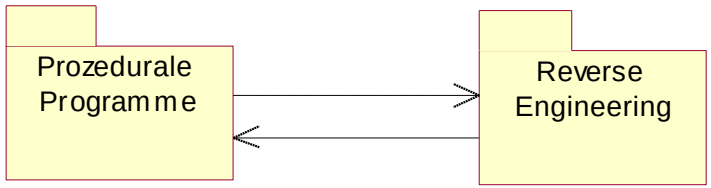
Prozedurales Metamodell Anforderungen

werden bestimmt durch:

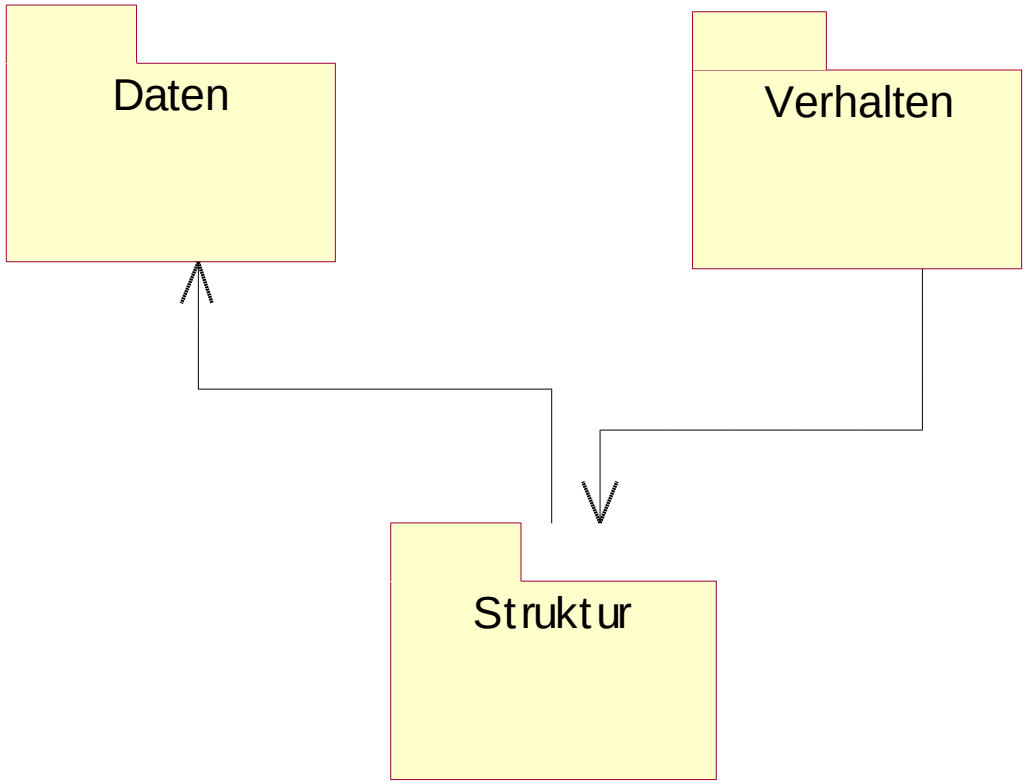
- Eigenschaften prozeduraler Programme
  - Struktur (Programm, Prozedur, Variable)
  - Verhalten (Aufruf, Definition, Verwendung)
- Nutzungsszenarien
  - Systembewertung (Metriken, Anomalien)
  - Visualisierung (Redokumantation)
  - Transformation (Fehlerbeseitigung, Refactoring)
  - Reengineering (Neuentwurf)

Notwendig: Beschränkung auf ein Szenario $\Rightarrow$ Objektidentifikation

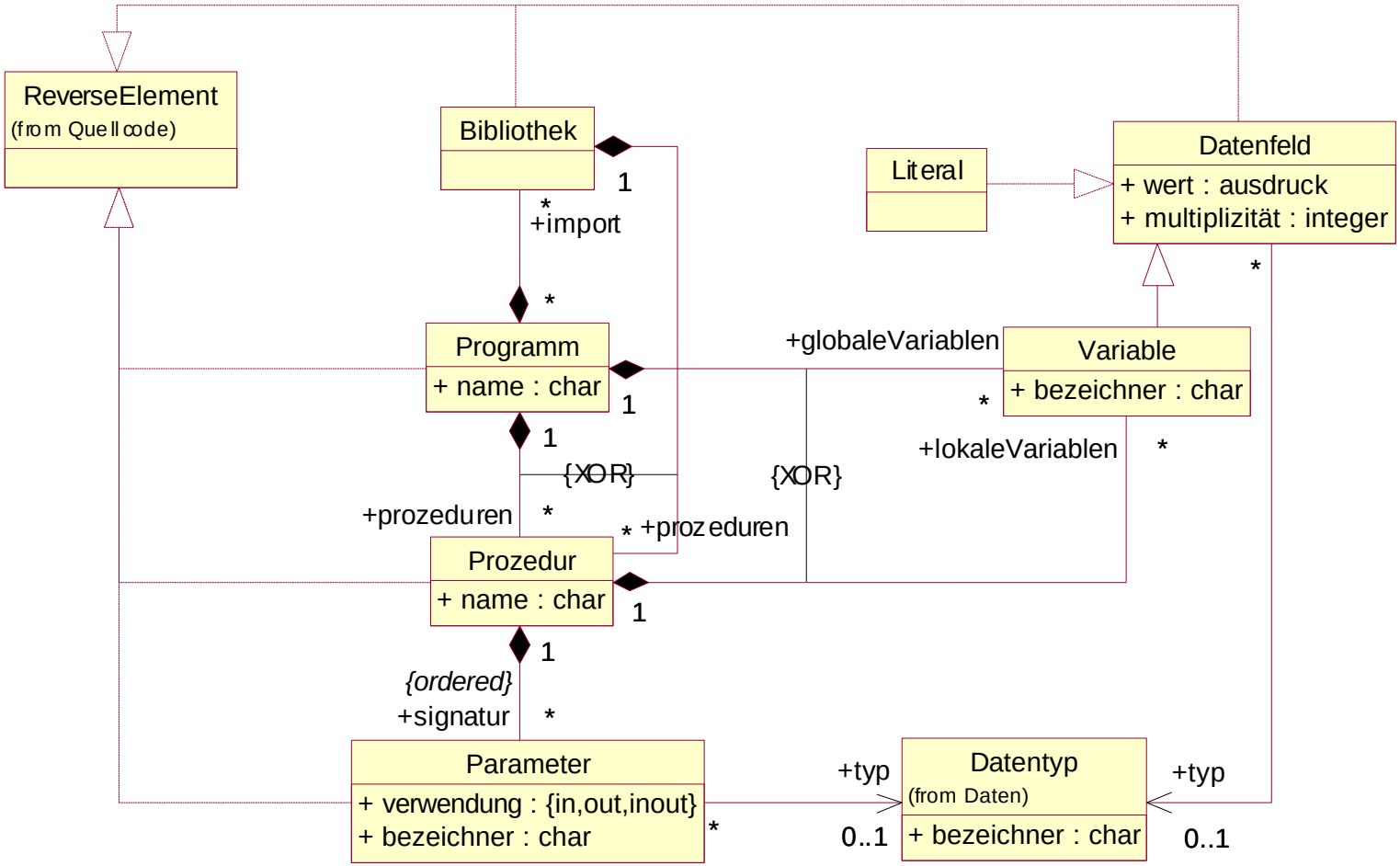
Erster Entwurf



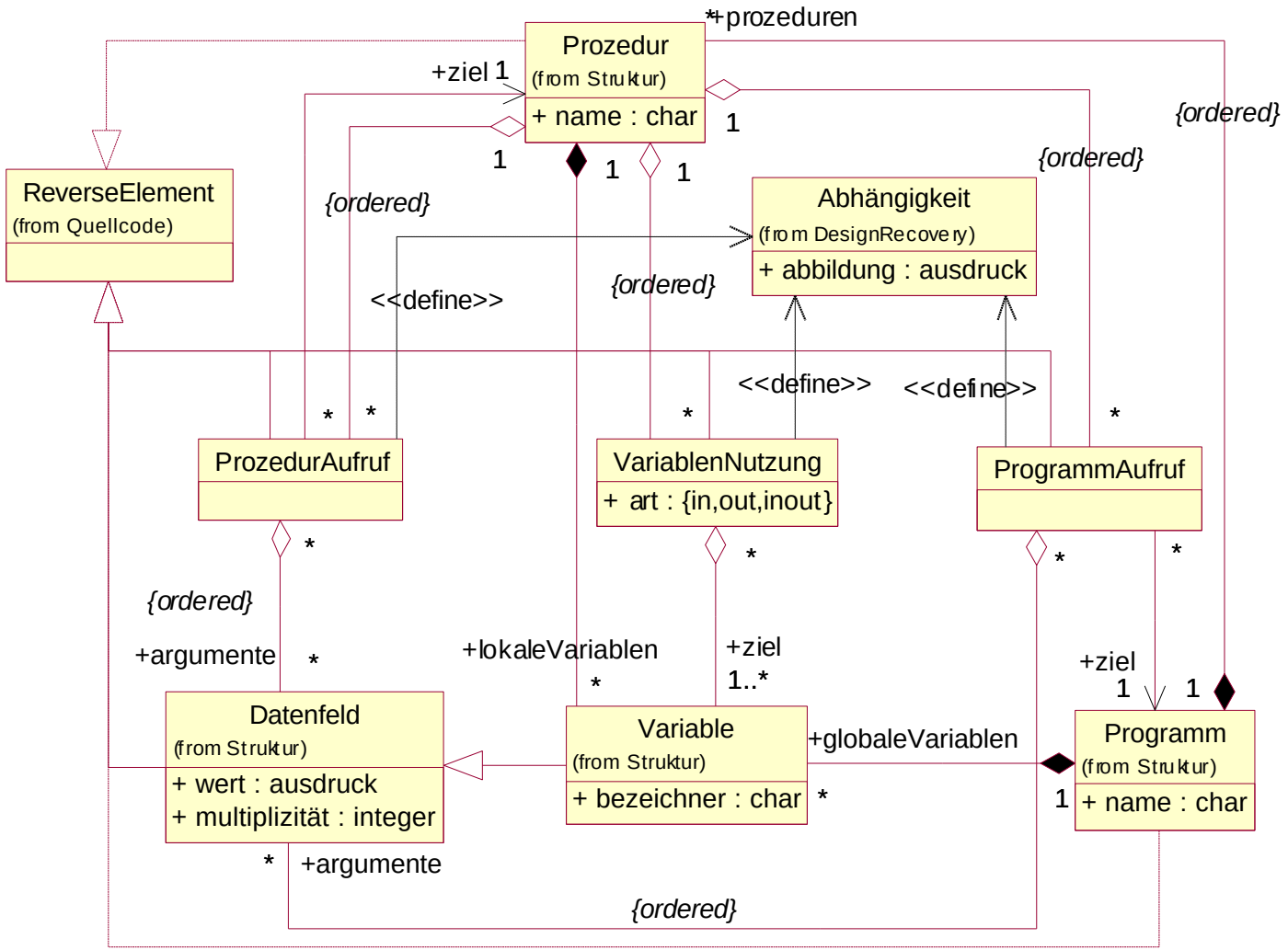
Erster Entwurf Package Prozedurale Programme



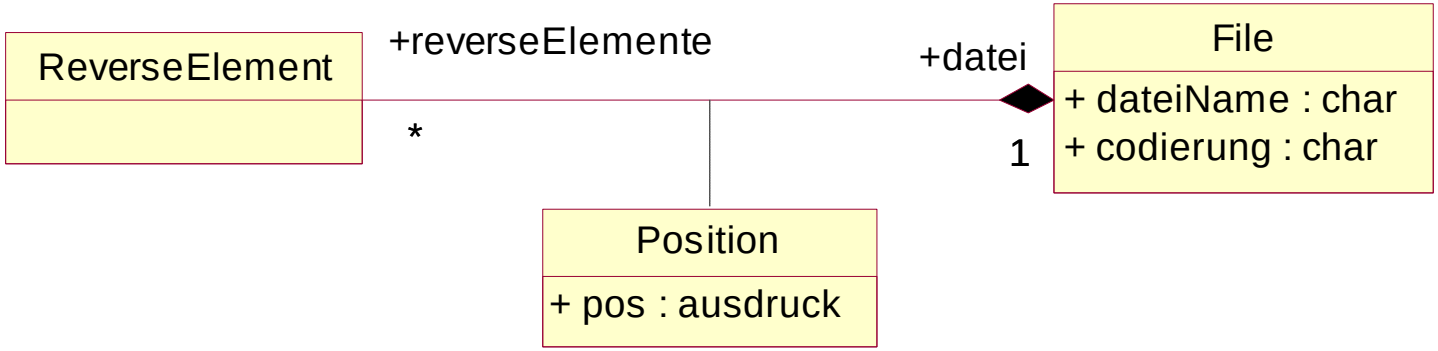
Erster Entwurf Package Struktur



Erster Entwurf Package Verhalten



Erster Entwurf Package Quellcode



Nutzung der UML

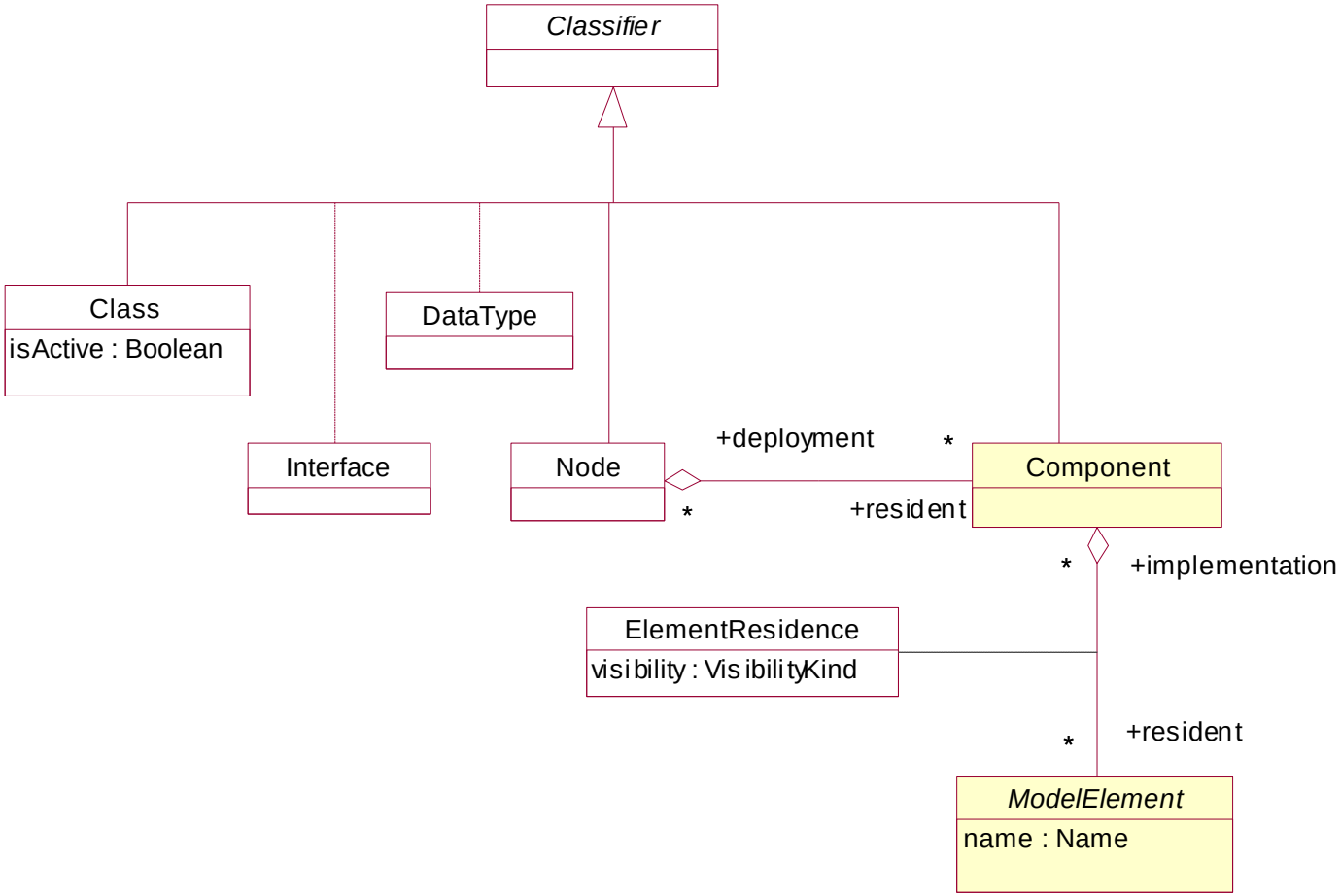
- Allgemein akzeptierter Standard
- XMI-DTD standardisiert
- Große Anzahl von Tools
- Implementation eines Systems darstellbar
- Erweiterbarkeit ohne Änderung des Metamodelles
- Round-Trip-fähig durch Integration von Analyse, Design und Implementation in einem Metamodell

⇒Identifizierung geeigneter Modellelemente

⇒Definition der Erweiterungen auf Metamodellebene

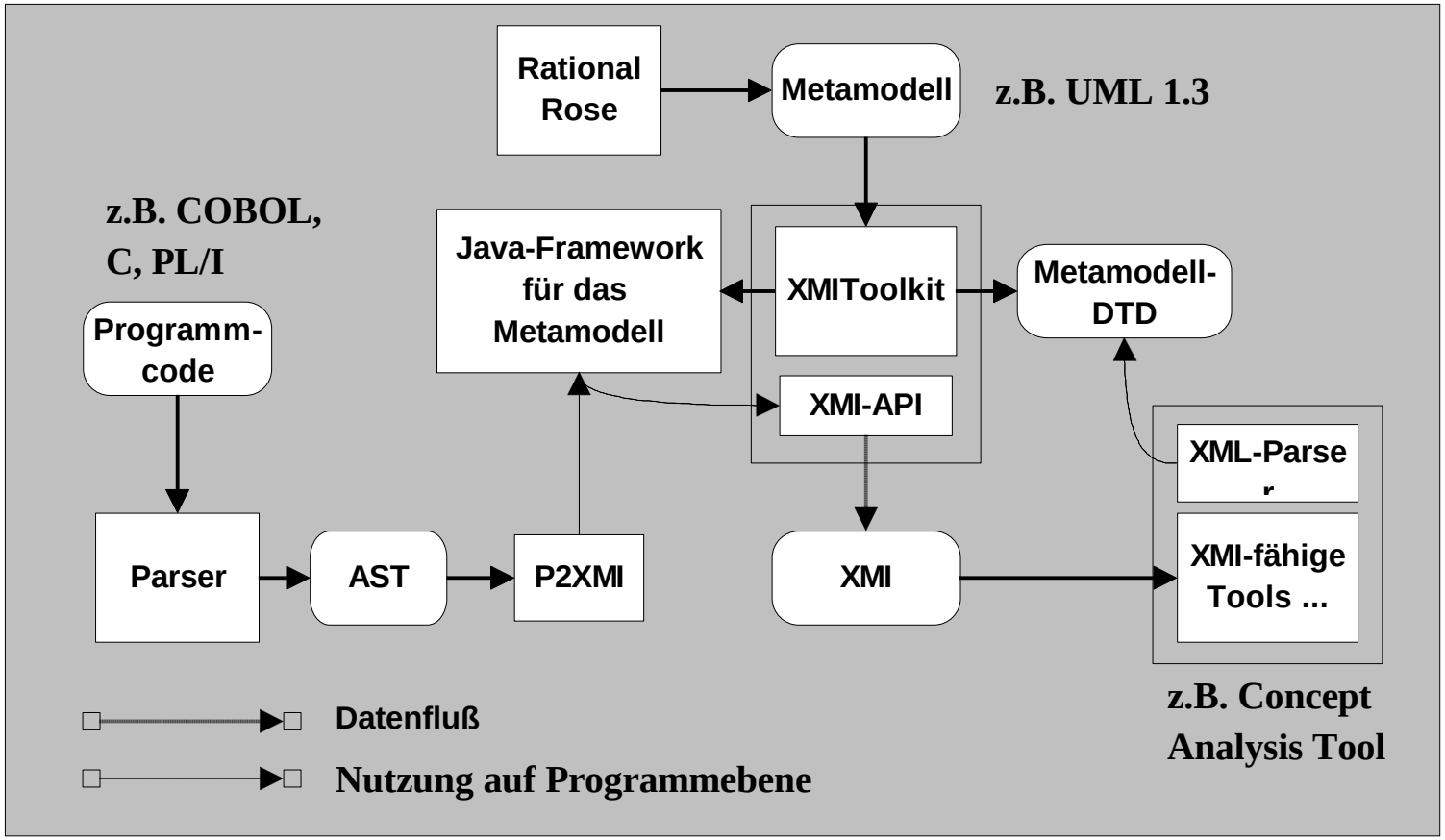
⇒Vorschläge für erweiterte Notation

Nutzung der UML Metamodell für Component



Erfahrungen mit XMI

Beispiel: Werkzeugkette mit dem XMIToolkit



Erfahrungen mit XMI XMIToolkit von IBM

Eigenschaften:

- Rose Modell ⇔ UML1.1/XMI
- Rose Modell oder UML1.1/XMI ⇔ Java
- UML-Modell (Rose od. UML1.1/XMI)⇒XMI-DTD
- Java-API zum Lesen und Schreiben von XMI-Dateien

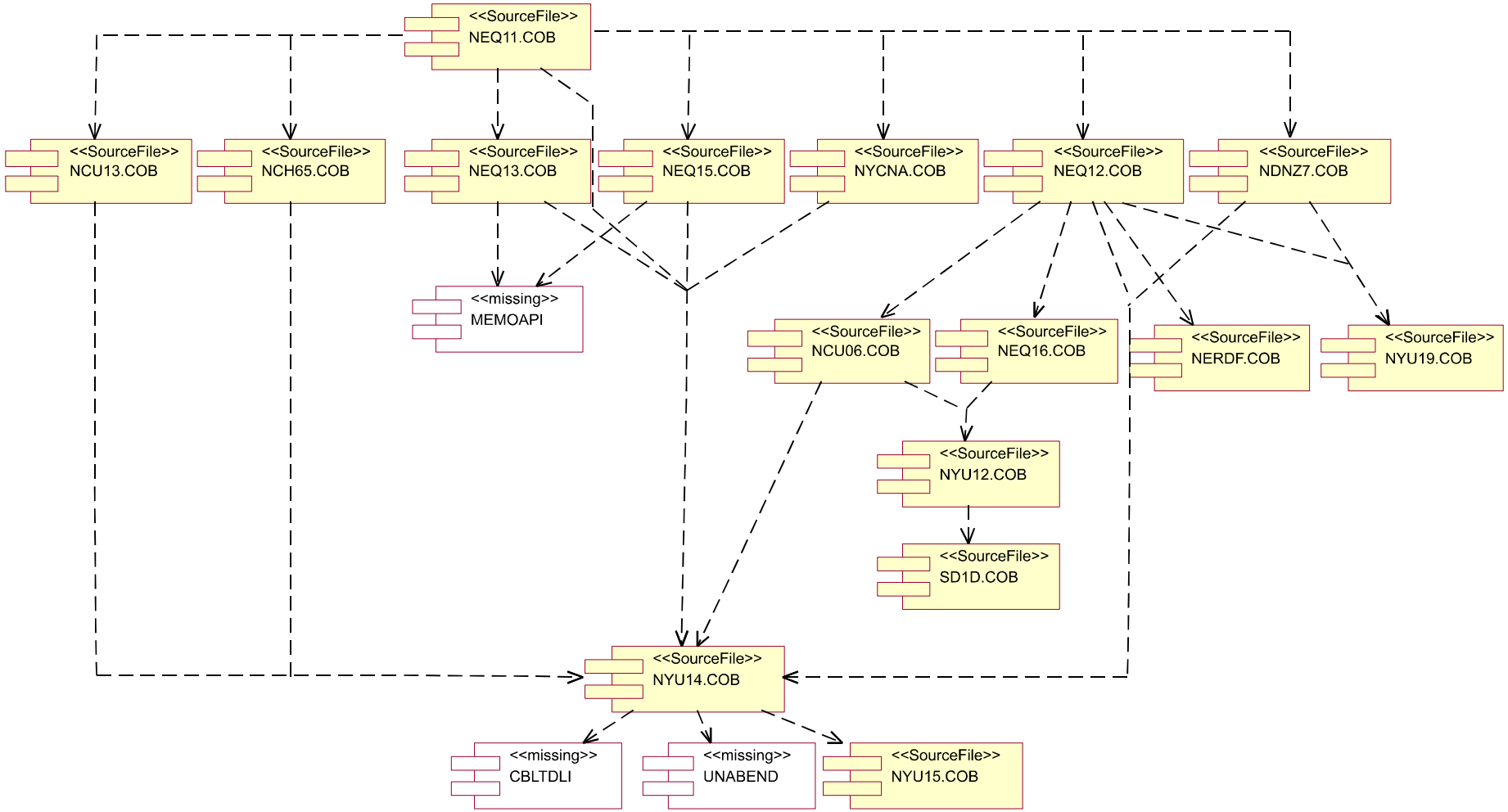
Probleme:

- nur ein Teil der UML 1.1 wird erkannt
- API kann kein DTD lesen und verarbeiten
- Java-Framework-Generator ignoriert Packages und Mehrfachvererbung

Allgemeine Probleme

- Informationsverlust bei:
  - Konvertierung in internes Format eines Tools
  - Unvollständige UML-Unterstützung
- Zweifelhafter Nutzen der XMI.Extension
  - nicht Round-Trip-fähig
- Kein CASE-Tool mit vollständiger UML-Unterstützung
  - implementationsnahe Diagramme fehlen
  - Semantik verändert
  - Notation eingeschränkt

Beispiel: Systemübersicht als Component Diagram



Beispiel: Java-Klasse für Element „Programm“

```
import com.ibm.xmi.framework.*;
import java.util.*;
public class Prozedurale Programme.Struktur.Programm extends Reverse
                                    Engineering.Quellcode.ReverseElement {
    public Programm() {
        super("Prozedurale Programme.Struktur.Programm");
    }
    public Programm(String name) {
        super("Prozedurale Programme.Struktur.Programm");
        setName(name);
    }
    public String getName() {
        return getXMIValue("Prozedurale Programme.Struktur.Programm.name");
    }
    public void setName(String value) {
        setPropertyValue("Prozedurale Programme.Struktur.Programm.name", value);
    }
    public Vector getProzeduren() {
       return getLinkedObjects("Prozedurale Programme.Struktur.Programm.prozeduren");
    } ...
```