Projekt Erdbebenfrühwarnung
im WiSe 2010/11

Entwicklung verteilter eingebetteter Systeme

Prof. Dr. Joachim Fischer
Dipl.-Inf. Ingmar Eveslage
Dipl.-Inf. Frank Kühnlenz

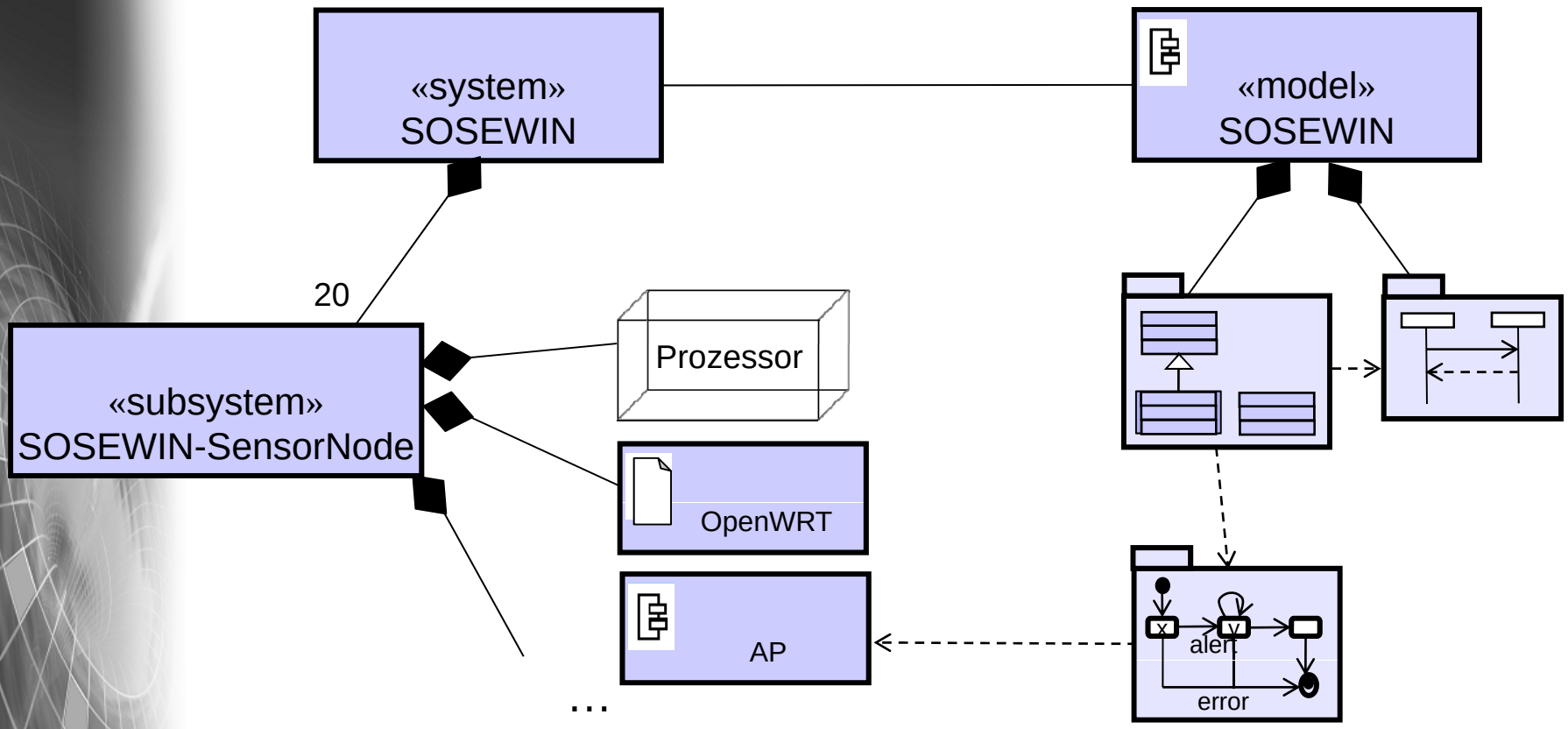
fischer|eveslage|kuehnlenz@informatik.hu-berlin.de

4. UML-Überblick

1. Historie von UML
2. Modellierungselemente von UML im Überblick
3. UML-Diagrammarten
4. Diagrammrepräsentationen in UML
5. Zum UML-Standard
6. Beispiel: UML-Klassendiagramm

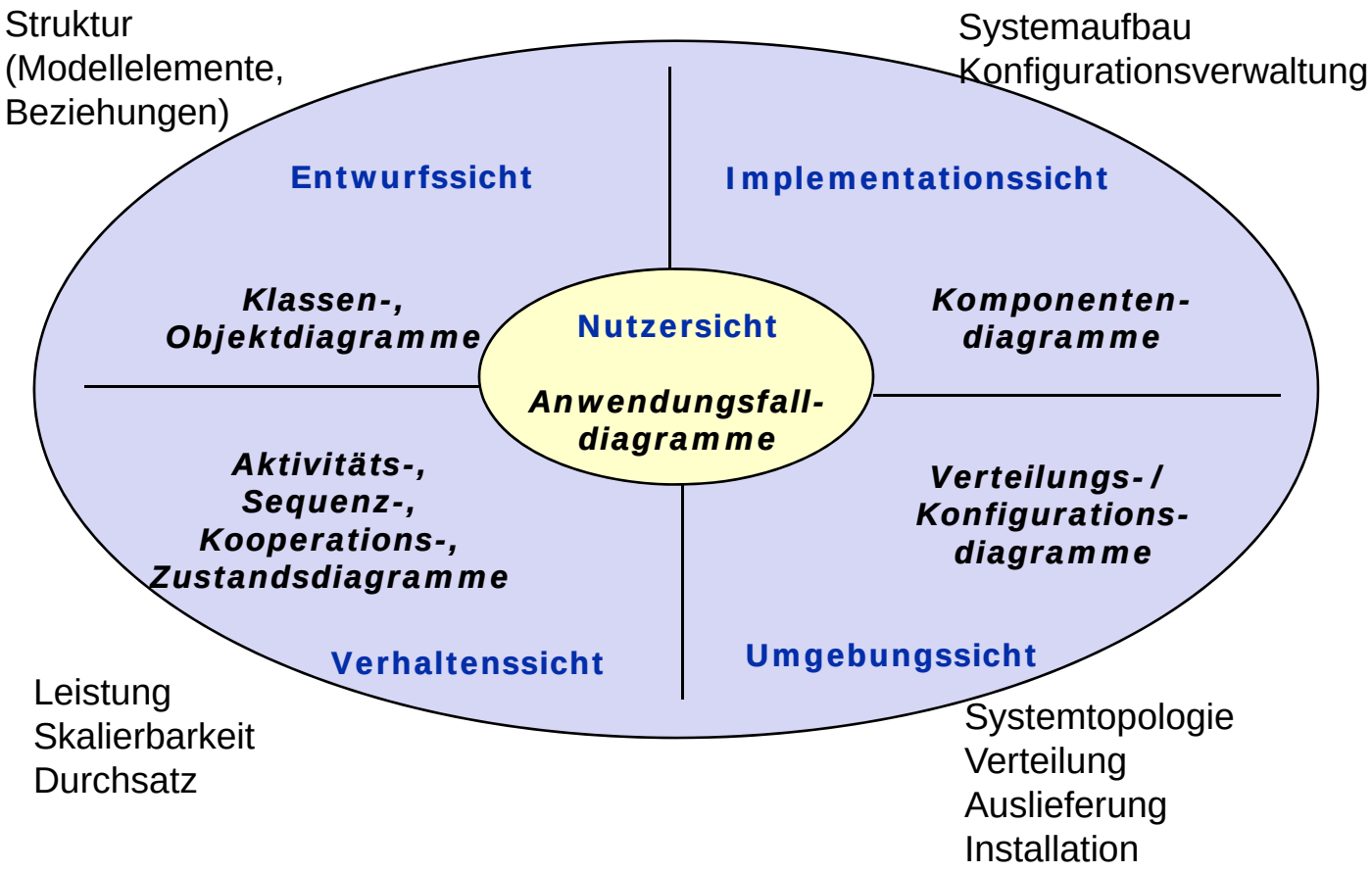
Erste Anwendung

instanziierbar



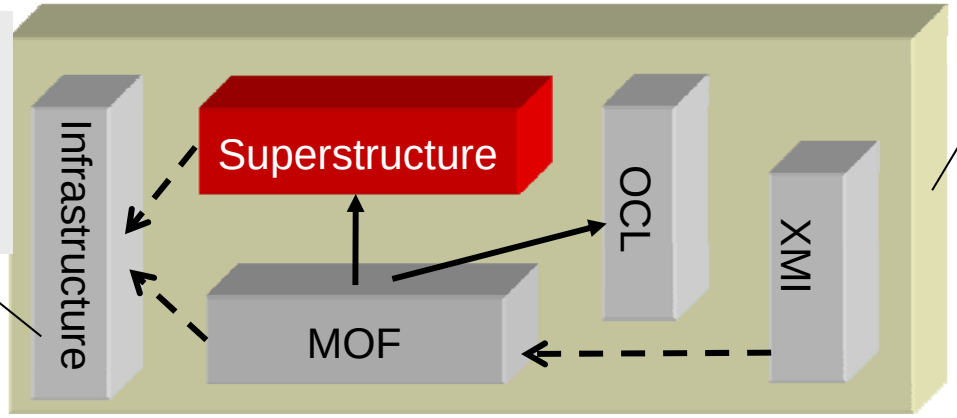
Systemarchitekturmodell eines Systems

- … unter Verwendung von UML

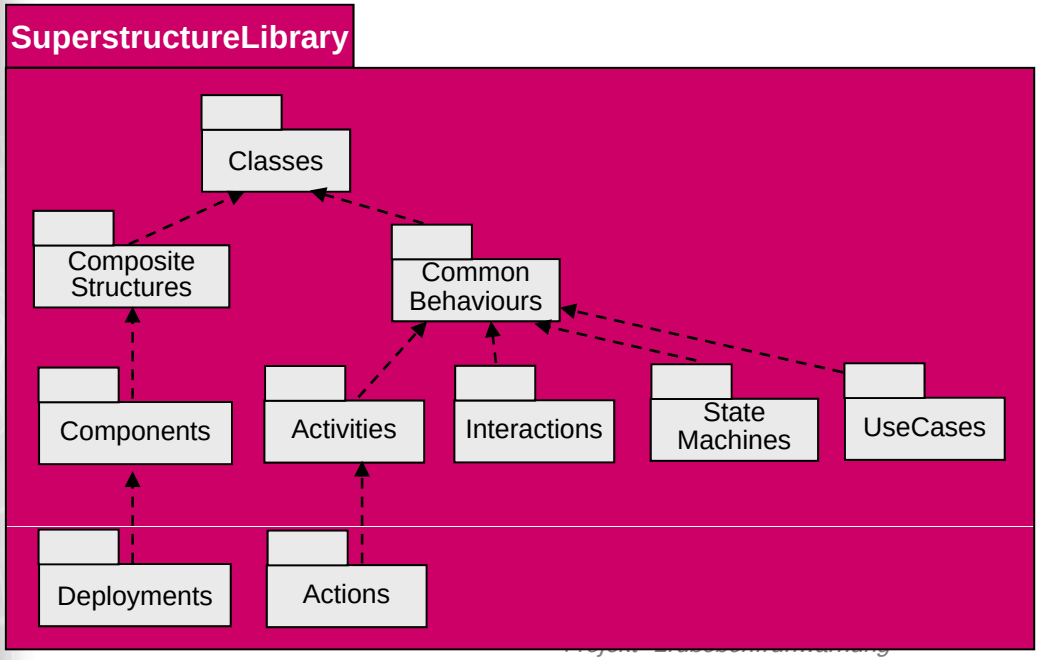


Struktur des Standards

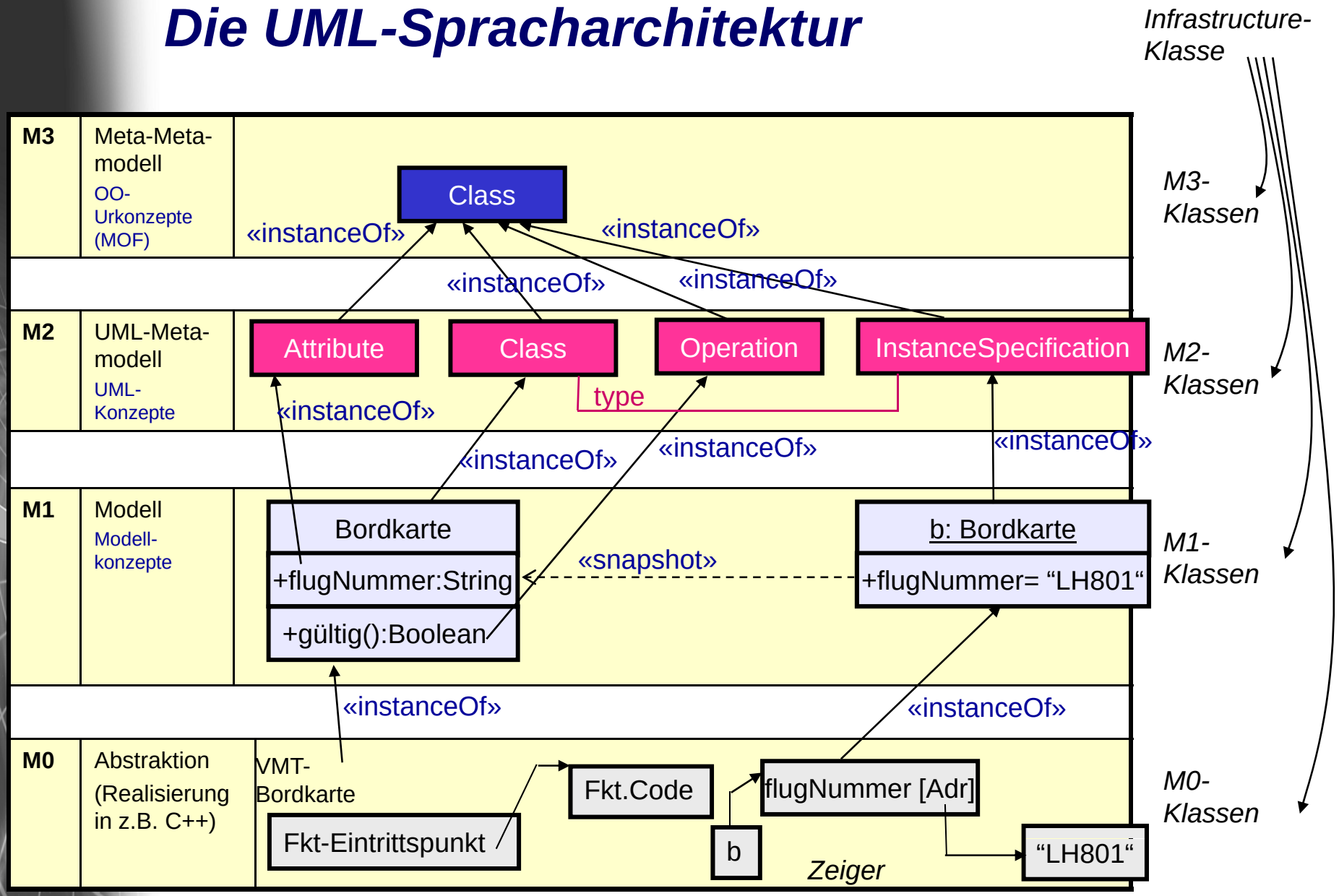
grundlegende Sprachkonstrukte in abstrakter Syntax (Klasse, Assoziation, …)



- Modellierungs-konzepte
- Sprachdefinitions-konzepte
- Transformations-konzepte



Die UML-Spracharchitektur



UML-Modifikationen (Aufbau eigener Sprachen)

- im Prinzip kann unter Verwendung von UML ein eigenes Metamodell (für eine DSL) aufgebaut werden

Wege, UML zu erweitern

a) direkte Erweiterung des UML-Metamodells:
Einführung weiterer Metaklassen, die von UML-Metaklassen erben

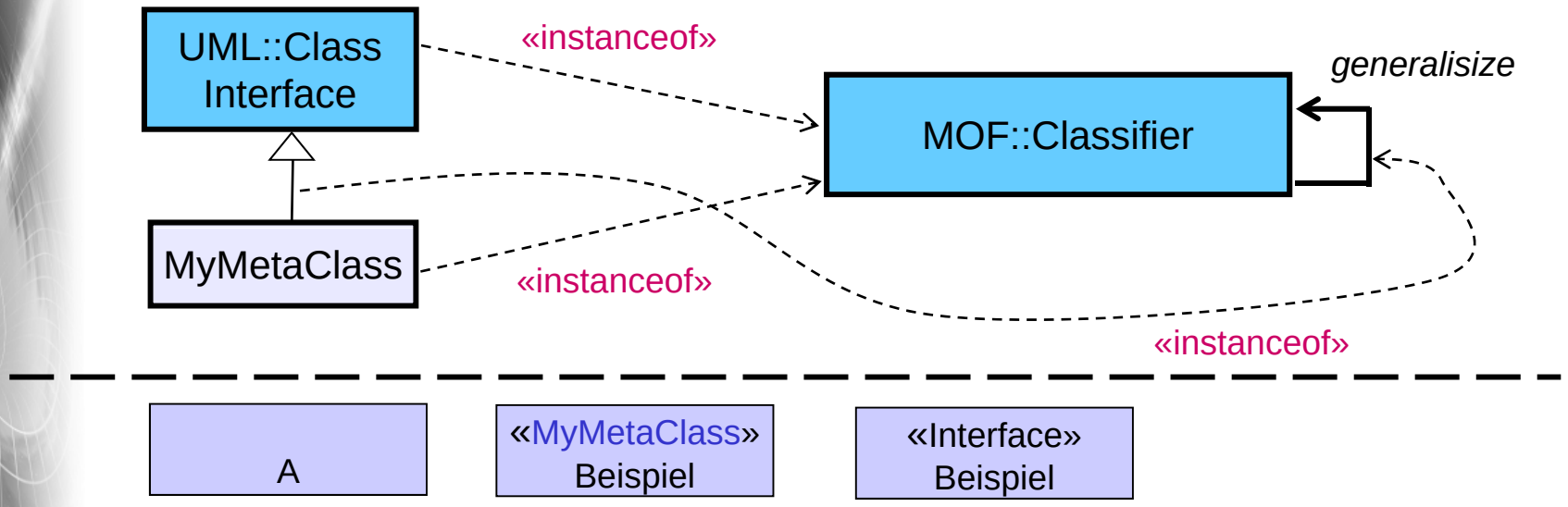
b) Profil-Mechanismus

Achtung:
- beide Ansätze erlauben es aber **nicht**, Teile der UML auszublenden
- das komplette UML-Metamodell steht im Hintergrund, das nur mit OCL in der Instanziierung eingeschränkt werden kann

a) direkte Erweiterung des UML-Metamodells

- **Resultat**: ein erweitertes UML-Metamodell
  führt zu einer erweiterten UML-Sprache (UML*)

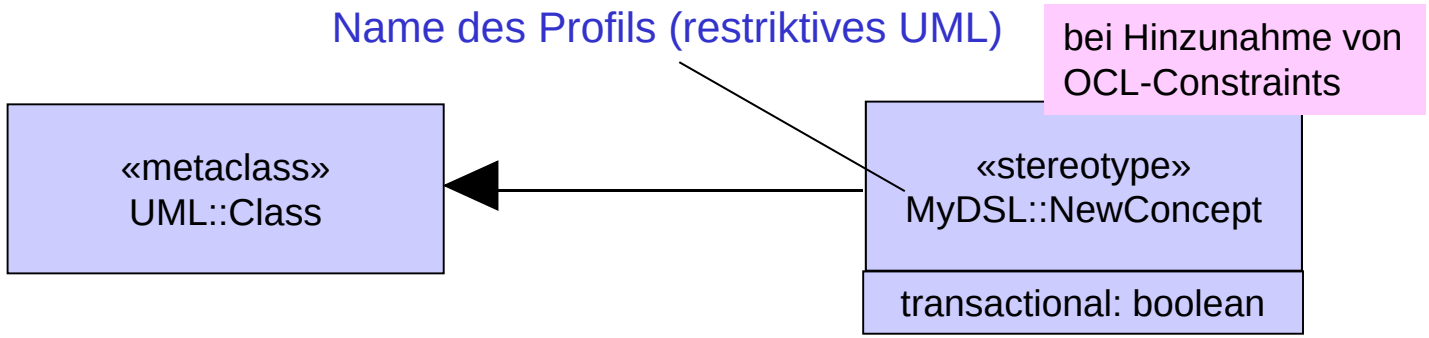
  in einem UML*-Modell lässt sich nun MyMetaClass benutzen



Achtung: Ansatz funktioniert für jede MOF-basierte Sprache (z.B. UML, …)
aber ohne Tool-Unterstützung (da neue Sprache!!!)

b) Profile und Stereotype

- Definition einer Stereotype als Extension (nur für UML erlaubt) im Kontext eines Profils



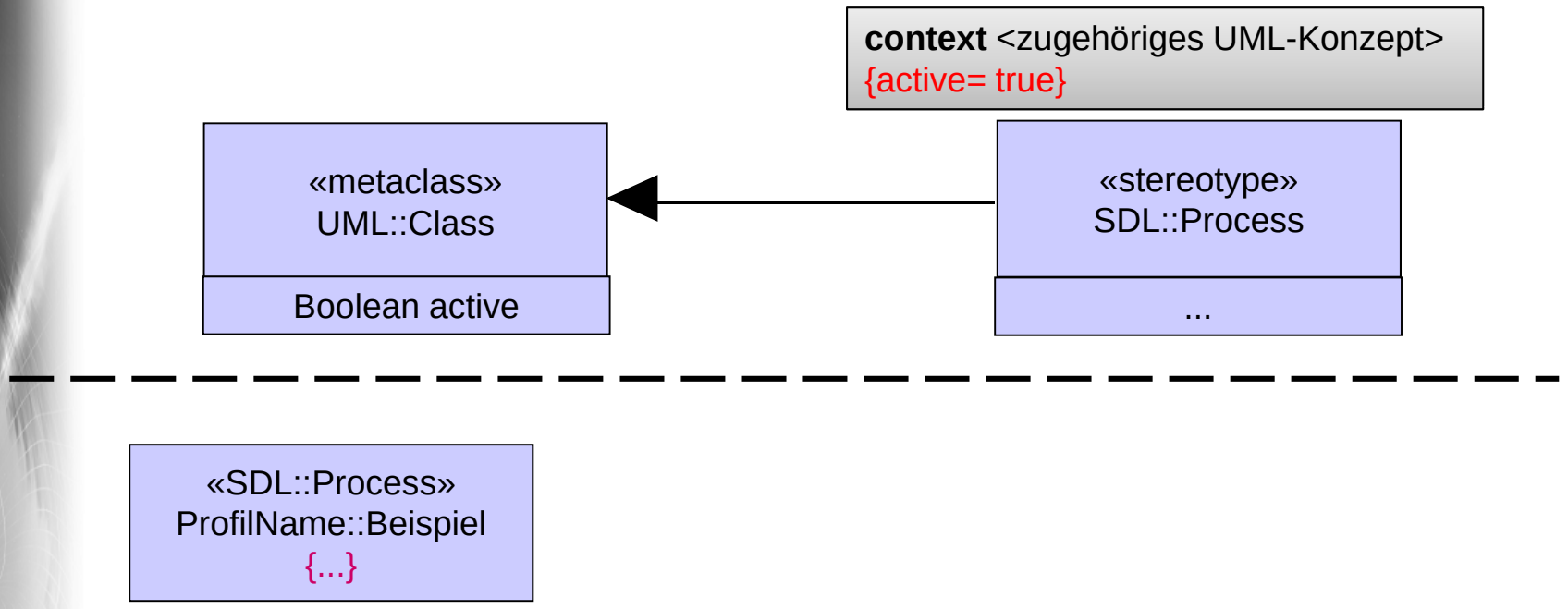
«MyDSL::NewConcept»
ProfilName::Beispiel
{transactional= true}

Einschränkung in der Erweiterung: Reduktion auf Attribute (*Tagged Values*)

Vorteil:
kann Erweiterung mit UML-Tool vornehmen und Profil über spezielle Pakete benutzen
bleibendes Problem: Grafik für spezielle Symbole

b) Profile und Stereotype

- Beispiel: SDL als UML-Profil



Vorteil:
kann UML-Tool zur Beschreibung und Analyse von SDL-Modellen benutzen (falls SDL als Einschränkung des UML-Metamodells beschreibbar ist)

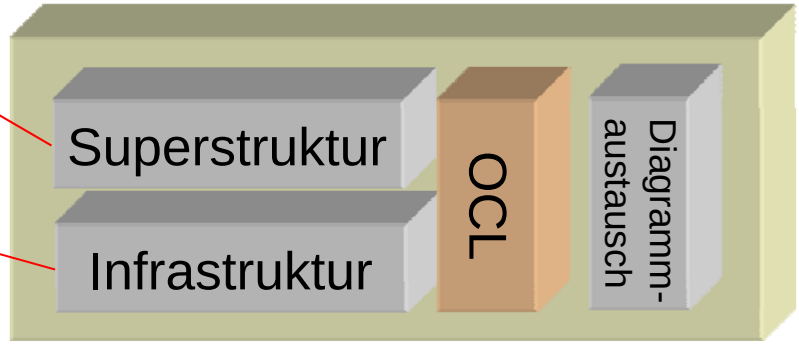
4. UML-Überblick

1. Historie von UML
2. Modellierungselemente von UML im Überblick
3. UML-Diagrammarten
4. Diagrammrepräsentationen in UML
5. Zum UML- und OCL-Standard
6. Beispiel: UML-Klassendiagramm

Einordnung von OCL im UML-Standard

UML-Vollsprache (Meta-Modell)

UML-Kernsprache (Meta-Modell)



- OCL ist eine textuelle Sprache
- Empfehlung der OMG zur Formulierung von
  1. Randbedingungen/Einschränkungen
  2. Anfragen
  3. Aktionen
  4. Navigationen

  } in Form von Annotationen
- OCL basiert auf
  - Prädikatenlogik (1.Stufe) und
  - Mengentheorie in einer (weitgehend) intuitiven Syntax
- prinzipiell ist aber jede andere Sprache als Alternative in UML zulässig (inkl. natürliche Sprachen)

OCL- Einsätzfälle

- Definition von
  - Invarianten für Klassen u. Typen
  - Invarianten für Stereo-Typen
  - Vor- und Nachbedingungen für Operationen
  - Wächterbedingungen
  - Ziele (Zielmengen) von Nachrichten und Aktionen
  - Constraints von Operationen
  - Ableitungsregeln von Attributen
- Navigationsbeschreibung (im Objektmodell)
  - Anfragesprache

Präzisierung von UML-Modellen

- ab UML 1.3: Built-In-Nutzung zur
  - Definition von *Well-Formedness Rules* für
    - Invarianten und
    - Metaklassen

  in der Abstrakten Syntax der Sprache

Präzisierung der UML-Sprach-definition

OCL als funktionale Spezifikationssprache

- OCL wurde ursprünglich als eigene Technik entwickelt
  - inzwischen Teil des OMG-Standards für UML
- OCL als Spezifikationssprache erlaubt, UML-Modelle
  - näher zu erläutern, durch Bedingungen einzuschränken
  - Regeln für Ausprägungen und Ausführung zu formulieren u. deren Konsistenz überprüfbar zu machen
- OCL ist eine funktionale Sprache zur Bildung von Ausdrücken über
  - sowohl OCL- als auch
  - UML-Daten

  mit Kunst/Tricks der Zusammenführung
- OCL-Ausdrücke …
  - lassen **keine** Seiteneffekte zu (Konsequenz: auch nur Query-UML-Operationen erlaubt)
  - sind Prädikate, logische Bedingungen/Vergleiche
  - werden einzeln gebildet und ausgewertet und nicht als Gesamtprogramm, (auch nicht die, die nur für ein Diagramm gelten sollen)

Ausdrucksberechnung

- Operanden
  - Werte von Container- und Standard-Typen
  - Objekte eines Typs
- Ausführungsreihenfolge
  - von links nach rechts
- Ergebnis
  - nach ausgeführter Berechnung ist ein Wert oder Objekt eines Typs
- Bestimmung von Nachfolgern des resultierenden Objektes mittels
  - Attribut oder Member-Funktion oder
  - Navigation über bestehende Assoziationen

Mächtigkeit von OCL-Ausdrücken

Anwendung vordefinierter Operationen (Standardbibliothek)

- insbesondere für Kollektionen
  - Teilmengen- und Projektionsbildungen,
  - Zusammenfassungen,
  - All- und Existenz-Quantoren für Iterationen

- mit Möglichkeit einer
  - kombinierten und
  - rekursiven Anwendung

dazu mehr

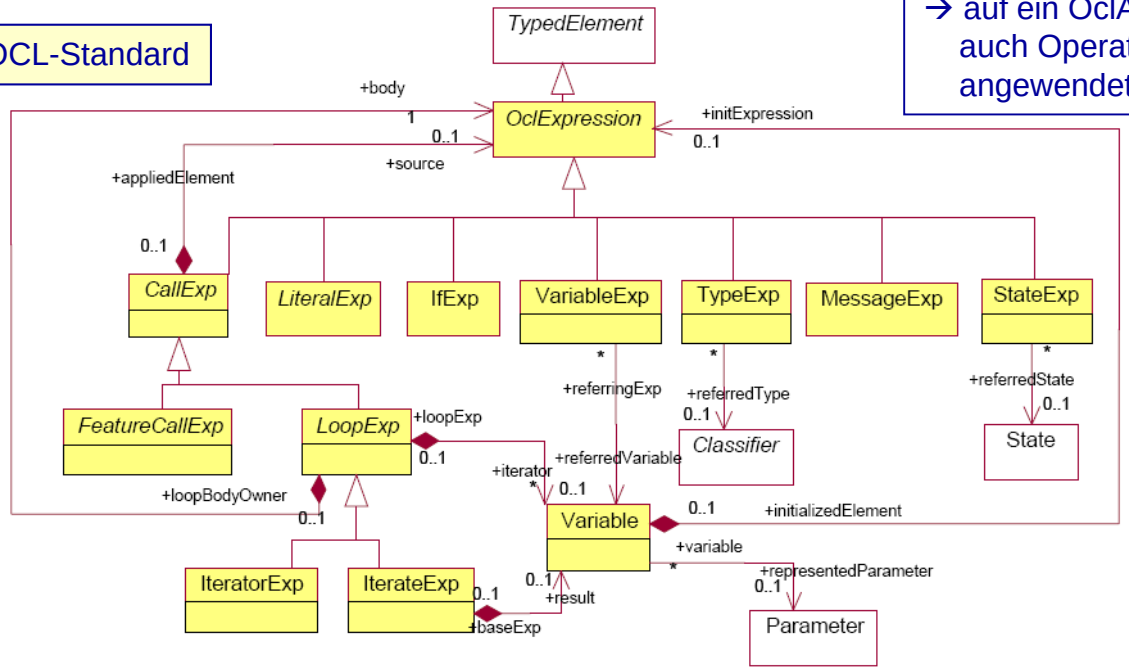
Zusammenfassung: OCL-UML-Zusammenhang (1)

- OCL-Ausdrücke sind getypt ⟷ OCL ist eine getypte Sprache
- wichtige Eigenschaft der Typkonformität
  - Typ legt fest, welche Operationen der zugeordneten Werte/Objekte erlaubt sind
  - Typhierarchie legt fest, welche polymorphen Ersetzungen erlaubt sind und welche nicht

Zusammenhang zu UML:
jeder UML-Classifier entspricht dem OCL-Typ OclAny (Supertyp)
→ OclAny-Operationen sind anwendbar auf alle entspr. UML-Objekte

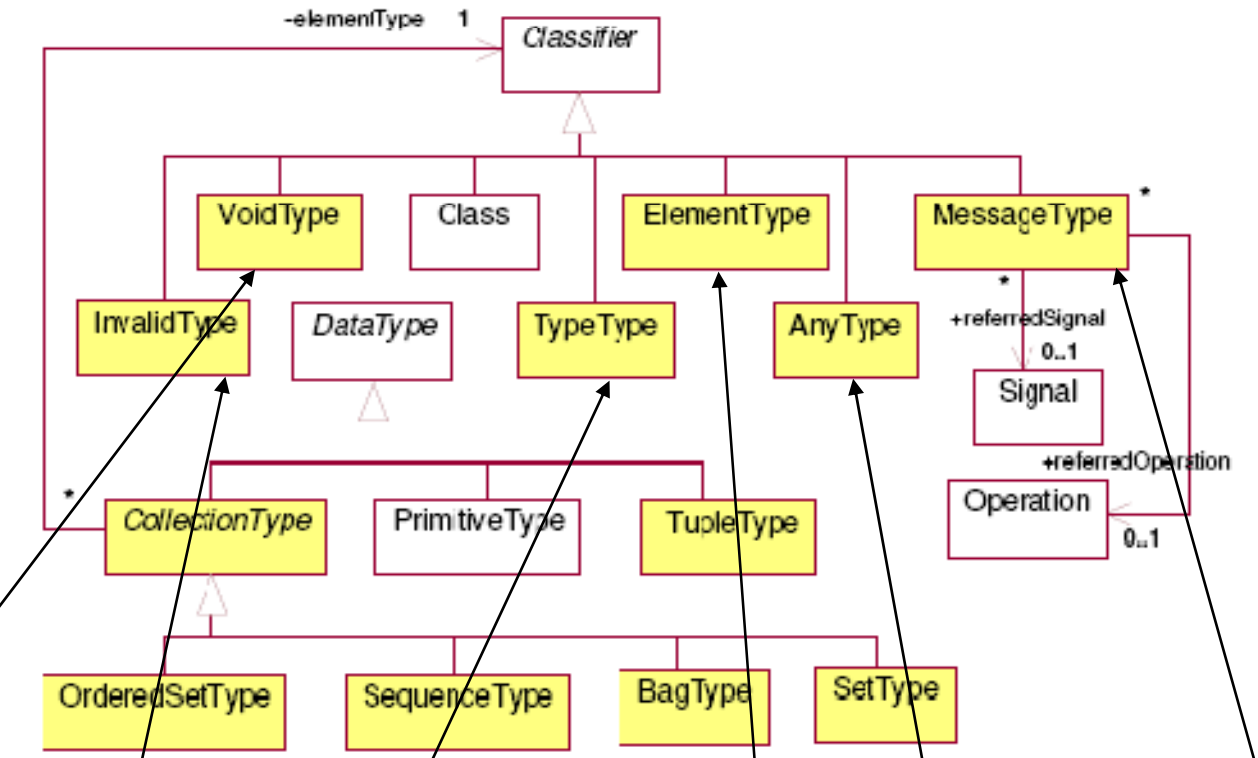
Paradoxon
Die Metaklasse von OclAny (AnyType) ist aber eine Spezialisierung von UML-Classifier (also entgegengesetzt!)
→ auf ein OclAny-Ausdruck dürfen nun auch Operationen von Classifier angewendet werden

aus: OCL-Standard



Zusammenfassung: OCL-UML-Zusammenhang (2)

OCL-Standard



instanceOf

OclVoid

konform zu allen Typen

OclInvalid

konform zu allen Typen
nur eine Instanz: **null**

OclType

liefert Eigenschaften eines UML-Typs in Form von Strings (interpretierbar in OCL)

OclElement

Template-Parameter

OclAny

konform zu OCL-Typen

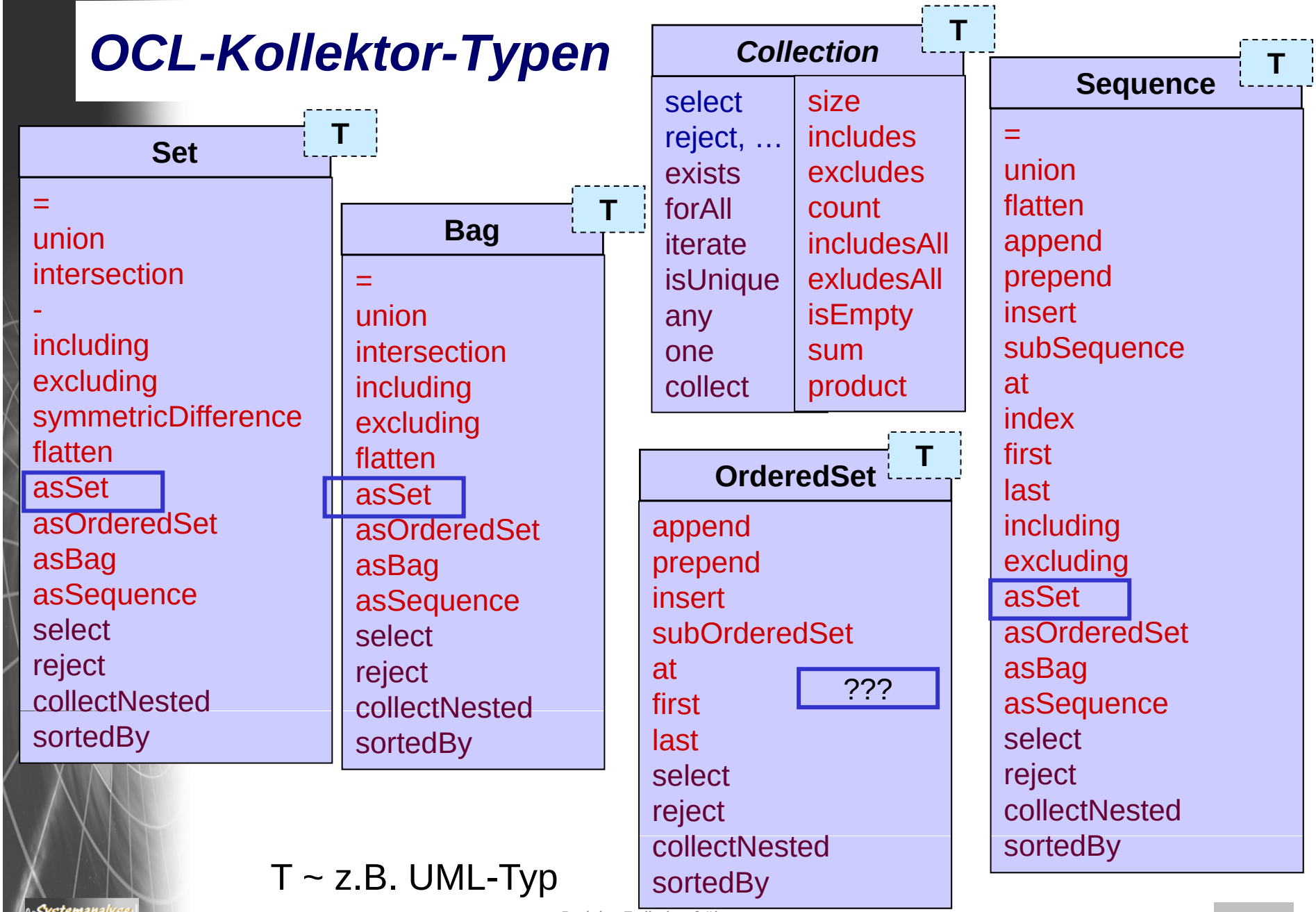
ohne Collection und UML-Typen (bietet allg. Operationen)

OclMessage

liefert Eigenschaften von UML-Signalen und Operationen in Form von Strings (interpretierbar in OCL)

UML-Standard

OCL-Kollektor-Typen

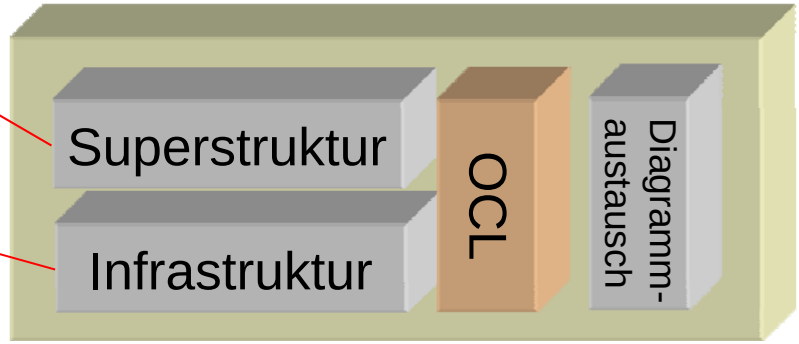


T ~ z.B. UML-Typ

UML-Kernsprache

UML-Vollsprache
(UML-Meta-Modell)

UML-Kernsprache



Kernsprache (mit konkreter Syntax) definiert

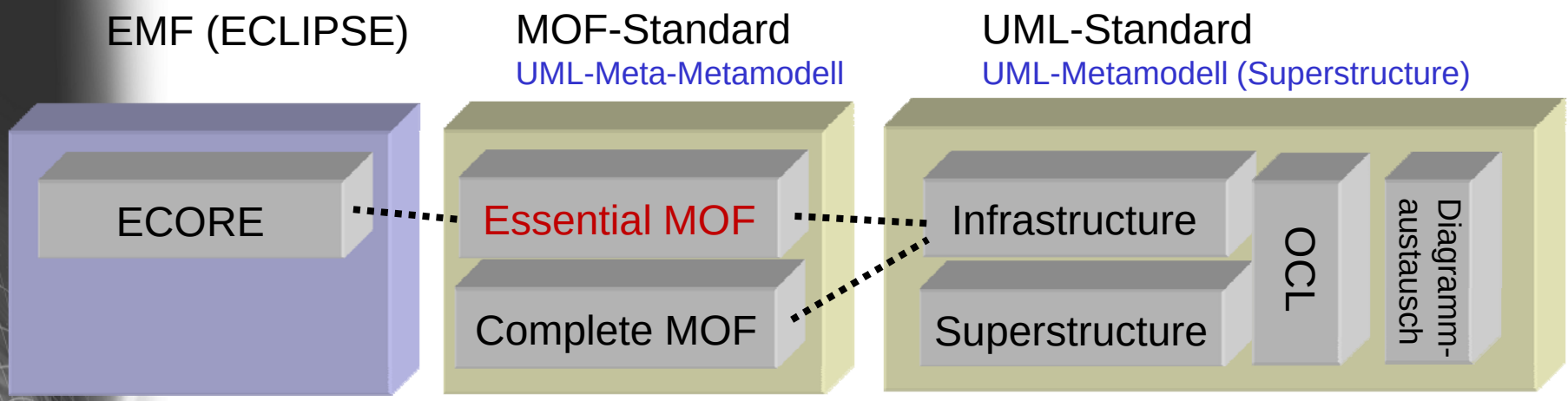
- Class als (Klasse)
- Generalisierung (Vererbung)
- Assoziationen mit Rollen, Multiplizitäten
- Verbindung von UML-Classifier mit OCL-Typ-Konzept

Zusammenhang zu UML:
jeder UML-Classifier entspricht einem OCL-Typ

OclAny als abstrakter Basistyp

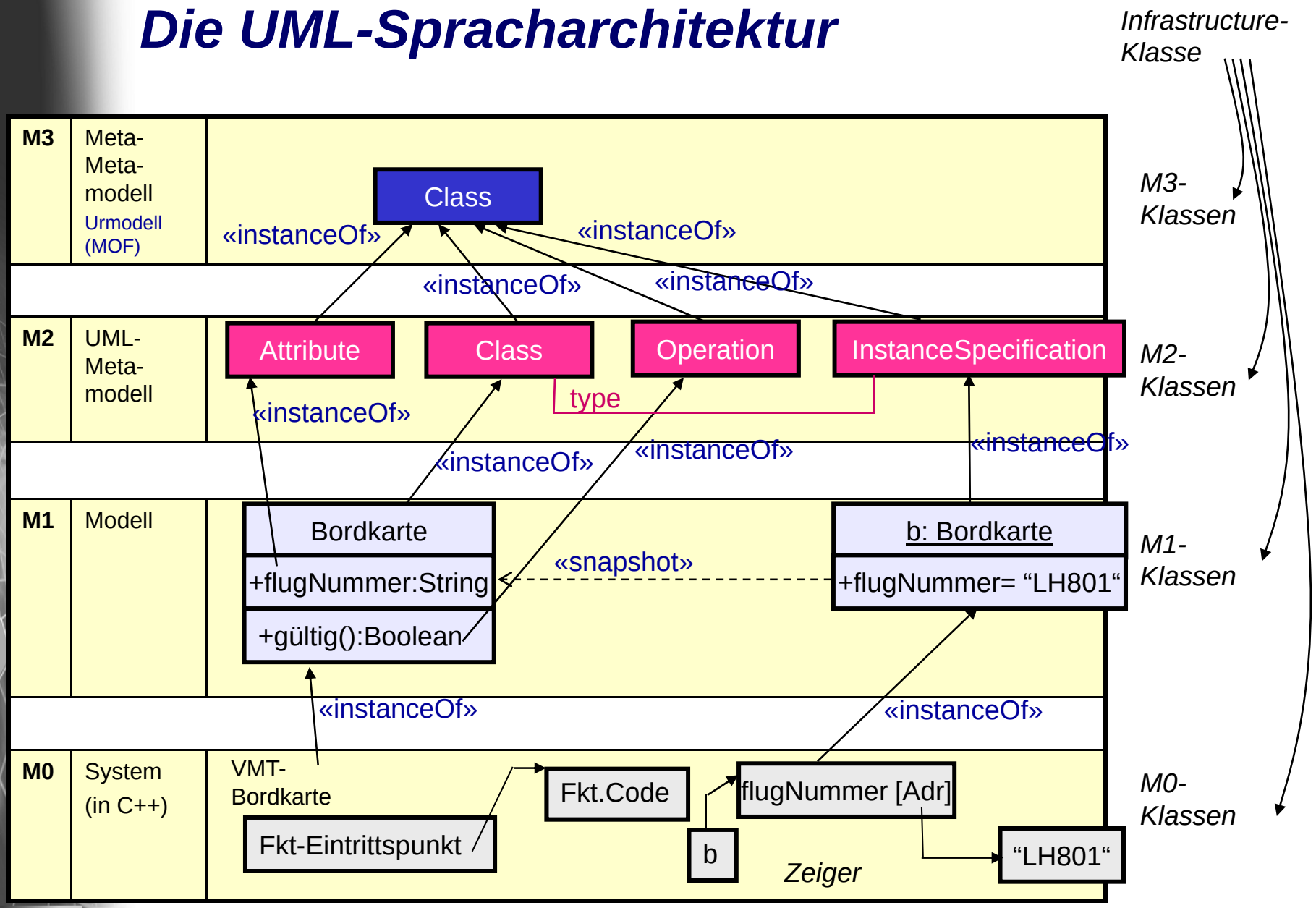
Mit der Kernsprache ist man in der Lage,
die Sprache UML in Form eines Metamodells zu definieren

Aufbau und Einordnung der UML-Sprachdefinition



- Allgemeine Vorteile der Bausteinstruktur
  - Erhöhung der Modularität
  - Parallelisierung der Arbeit an den Sprachkonzepten (mit gewissen Integrationsaufwand)
  - weitere Unterstrukturen (Pakete) zum Abgleich mit anderen Standards
- Metamodell-basierte UML-Sprachdefinition
  - MOF-orientiert (4-Ebenen, objektorientiert: Klasse mit Attributen und Operationen, Referenz, Assoziation, Paket)
- Meta-Metamodelle (für UML, OCL, CWM, …) ist MOF
  - MOF hat keine Notation, bedient sich deshalb einer UML-Kernsprache (Infrastructure)
  - ECORE ist das Meta-Metmodell von EMF und identisch mit EMOF (Essential MOF)
  - Infrastructure (interpretiert auf M2-Ebene) ist u.a. das Meta-Metamodell von UML

Die UML-Spracharchitektur



Modell, Metamodell, Metamodellhierarchie

Definition:

- Ein Modell (model) ist eine Sammlung von Objekten und Links (Modellelemente), die ein reales oder abstraktes System beschreiben.
- Ein Metamodell (metamodel) ist ein Modell, das zur Beschreibung anderer Modelle verwendet wird. Die Modelle, die durch das Metamodell beschrieben werden, sind ihre Instanzen und das Metamodell ist ihr Typ.
- Eine Metamodellhierarchie ist ein Baum von Modellen, die über InstanzVon-Beziehungen verknüpft sind.
- Eine Modellschicht (model layer) beschreibt alle (Meta-)Modelle mit dem gleichen Abstand zu dem Wurzel-Metamodell in der Metamodell-Hierarchie. Jede Schicht wird mit Namen und Nummer bezeichnet.

Bezeichnung M mit Nummern 0..3 wird von **MOF** vorgegeben und in UML verwendet (M0 wird durch Modellierungsframeworks nicht unterstützt)

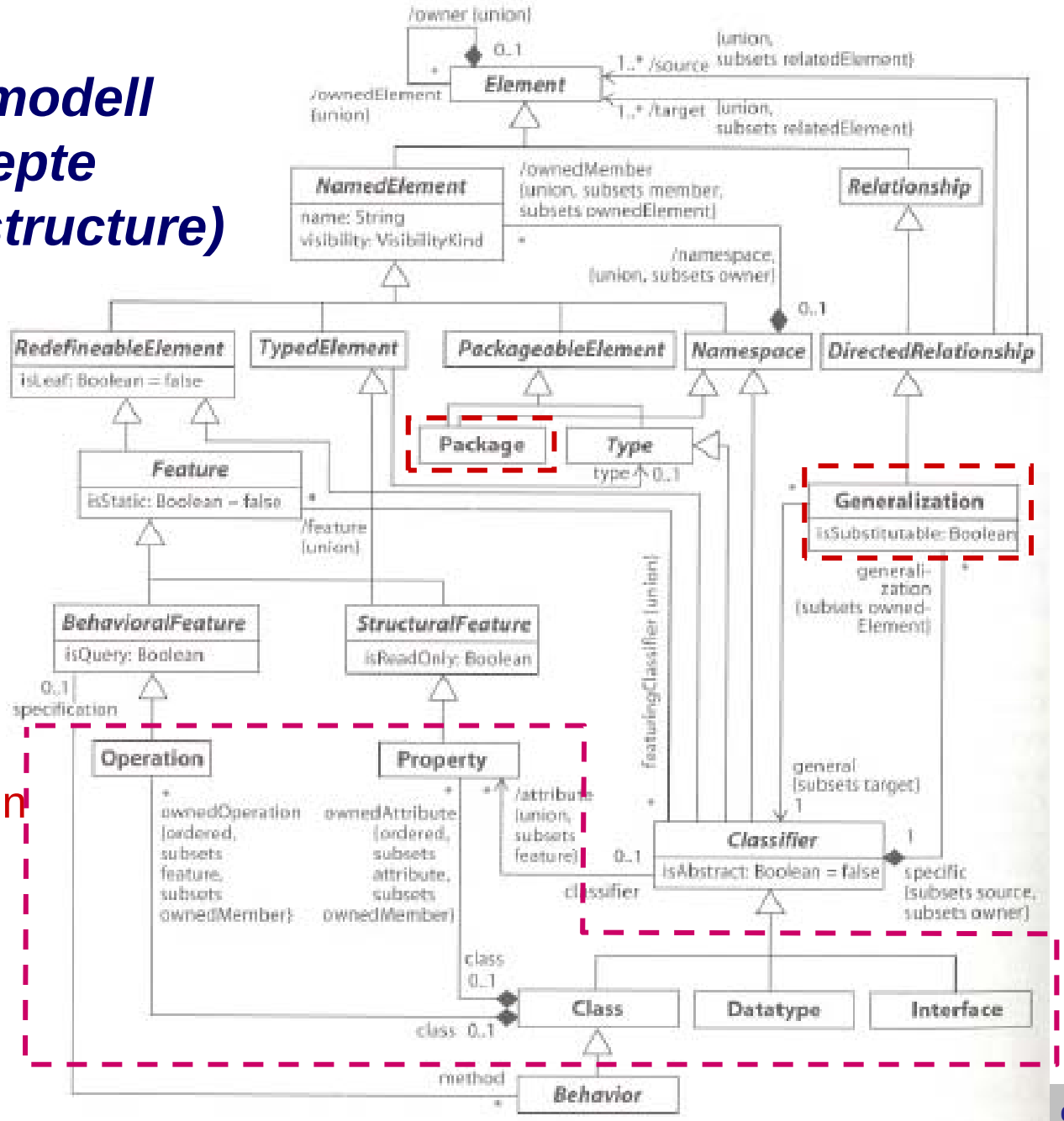
MOF-Instanziierungsemantik

Eine M1-Element-Instanz ist ein Wert, dessen Typ durch ein M2-Element-Instanz einer M3-Klasse beschrieben ist

.

UML-Metamodell (Kernkonzepte der Superstructure)

Konkrete Metaklassen

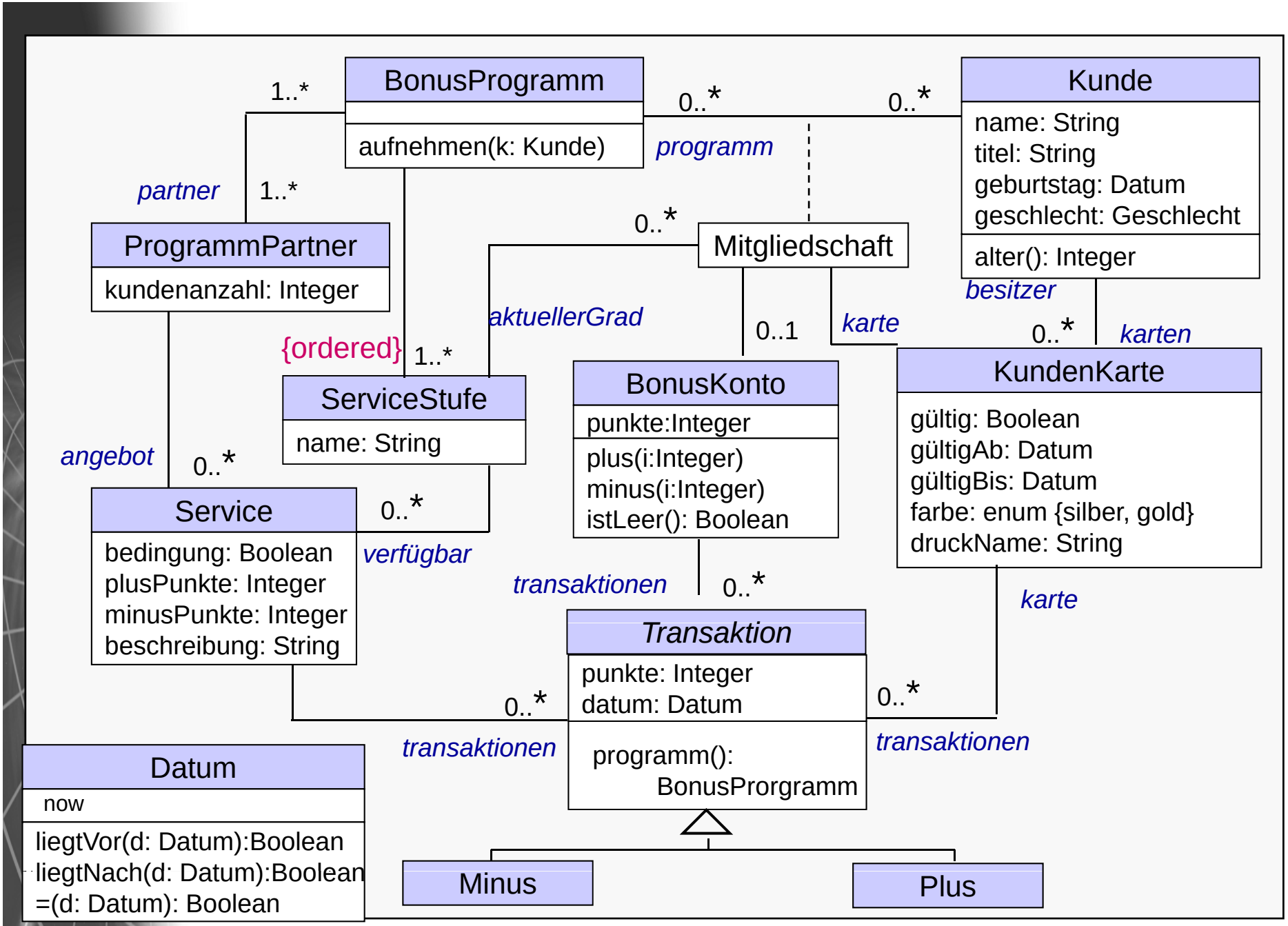


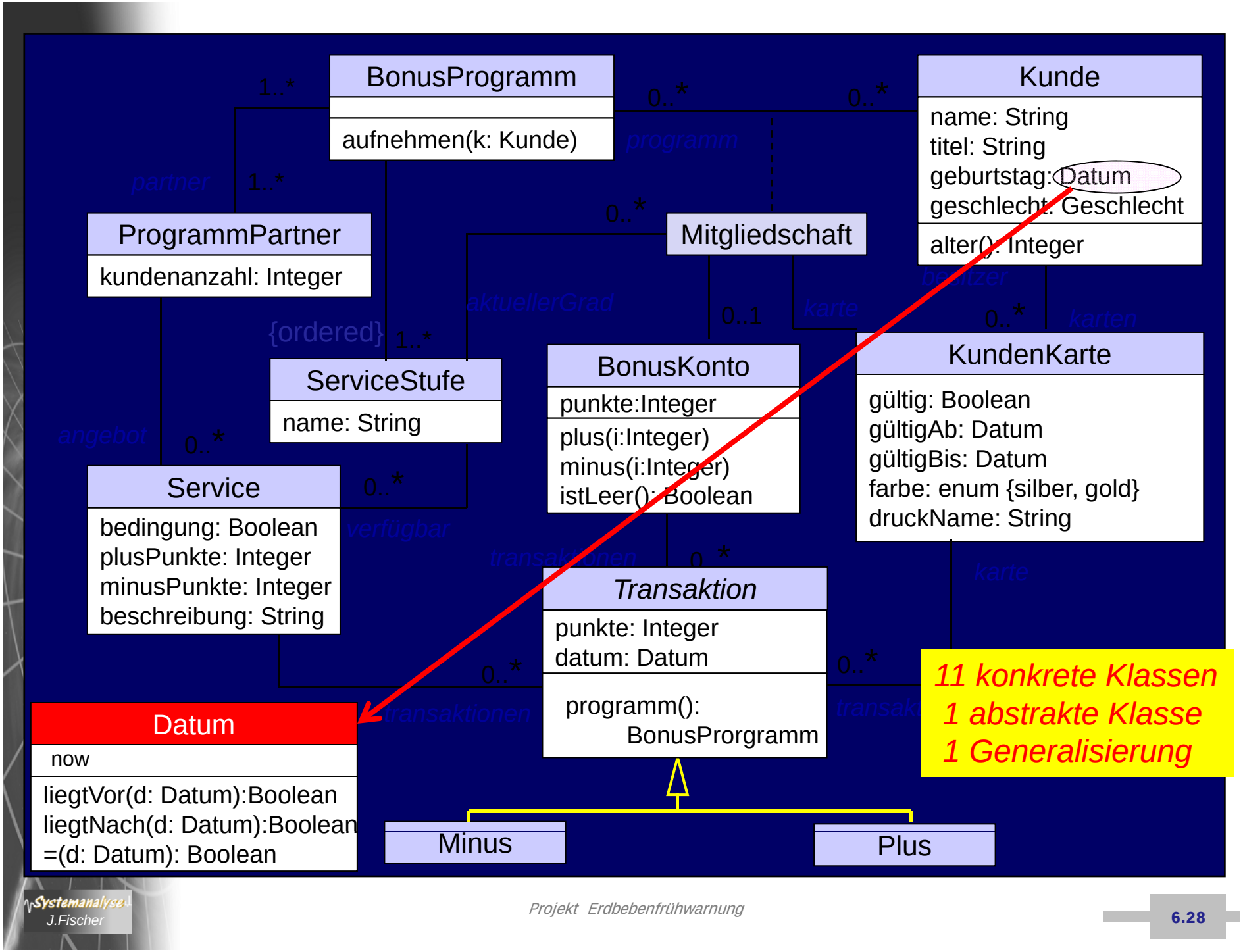
Allgemeine Vorteile von Metamodellen

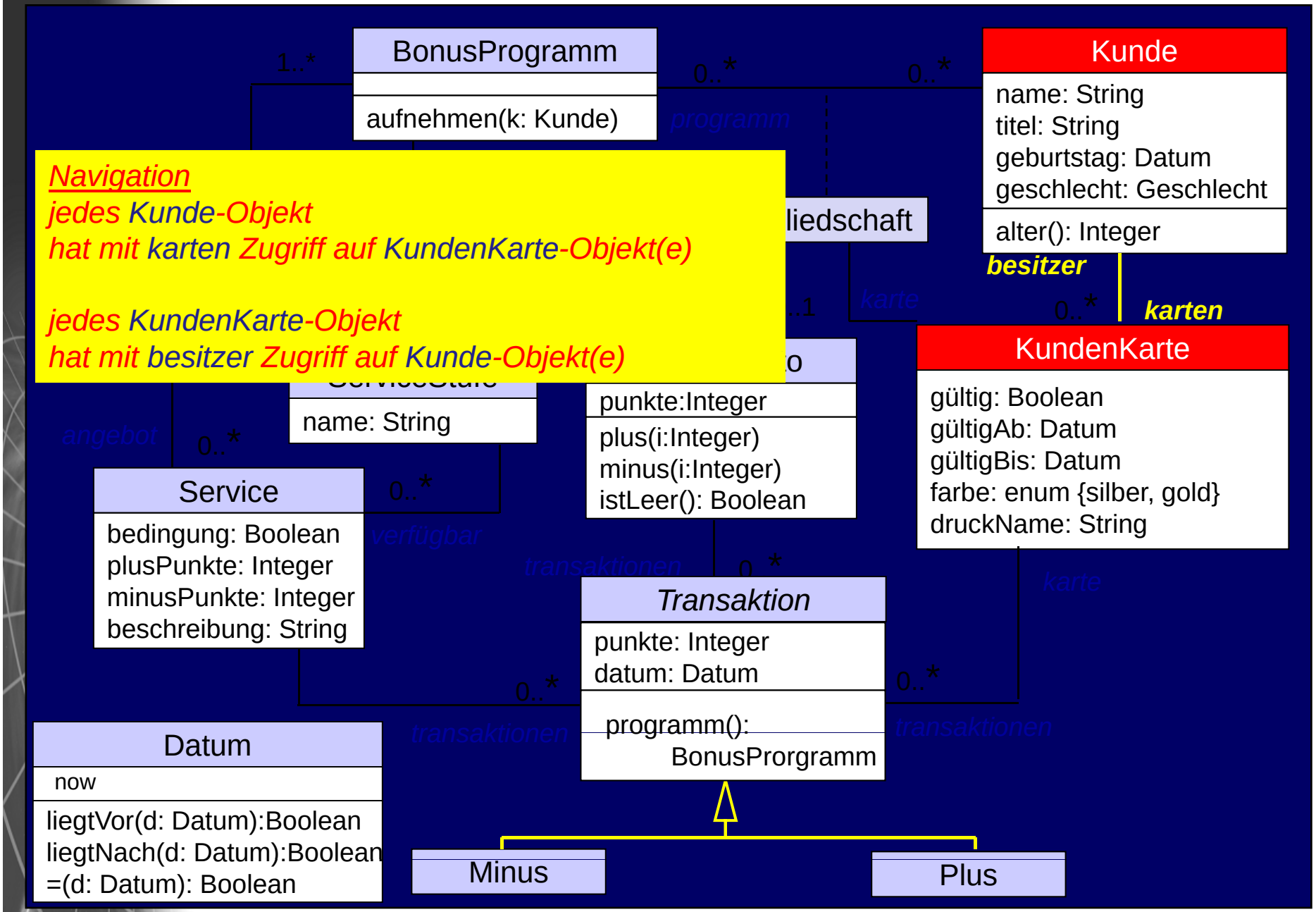
- prägnanter, präziser als natürliche Sprache (entsprechend der abstrakten Syntax textueller Sprachen)
- Basis zur Überprüfung der semantischen Korrektheit
- Grundlage zur Verwaltung von Modellen in Repositories
  - diagrammübergreifend
  - lebenszyklusübergreifend
  - zielsprachenunabhängig
  - werkzeugunabhängig
- einheitliches Austauschformat
  - da alle auszutauschenden Modelle Instanzen des einen Metamodells sind
- objektorientierte Metamodelle können nachträglich leicht erweitert werden
  - durch Spezialisierung und Re-definition
  - UML macht davon Gebrauch (Kern→Vollsprache→Profile→DSLs)

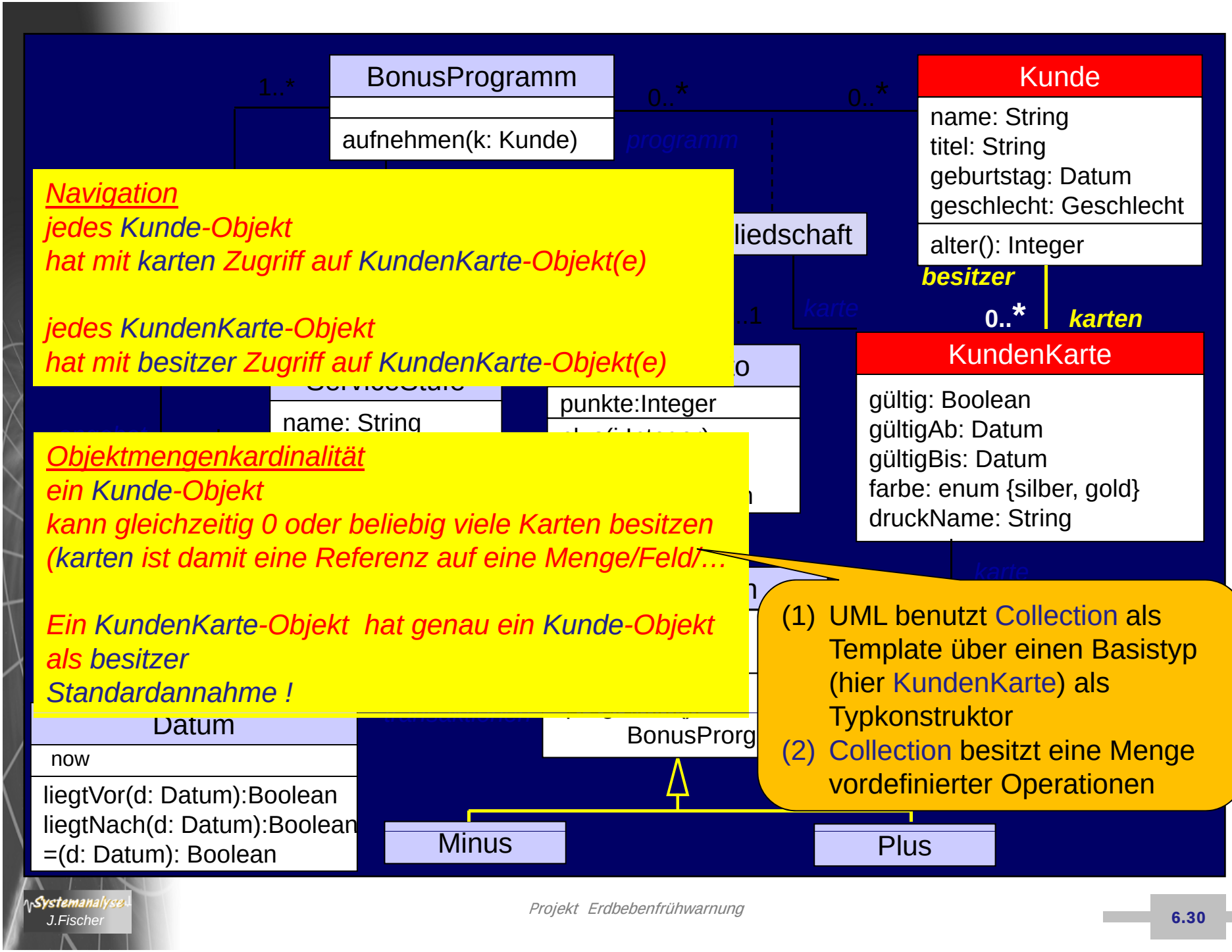
4. UML-Überblick

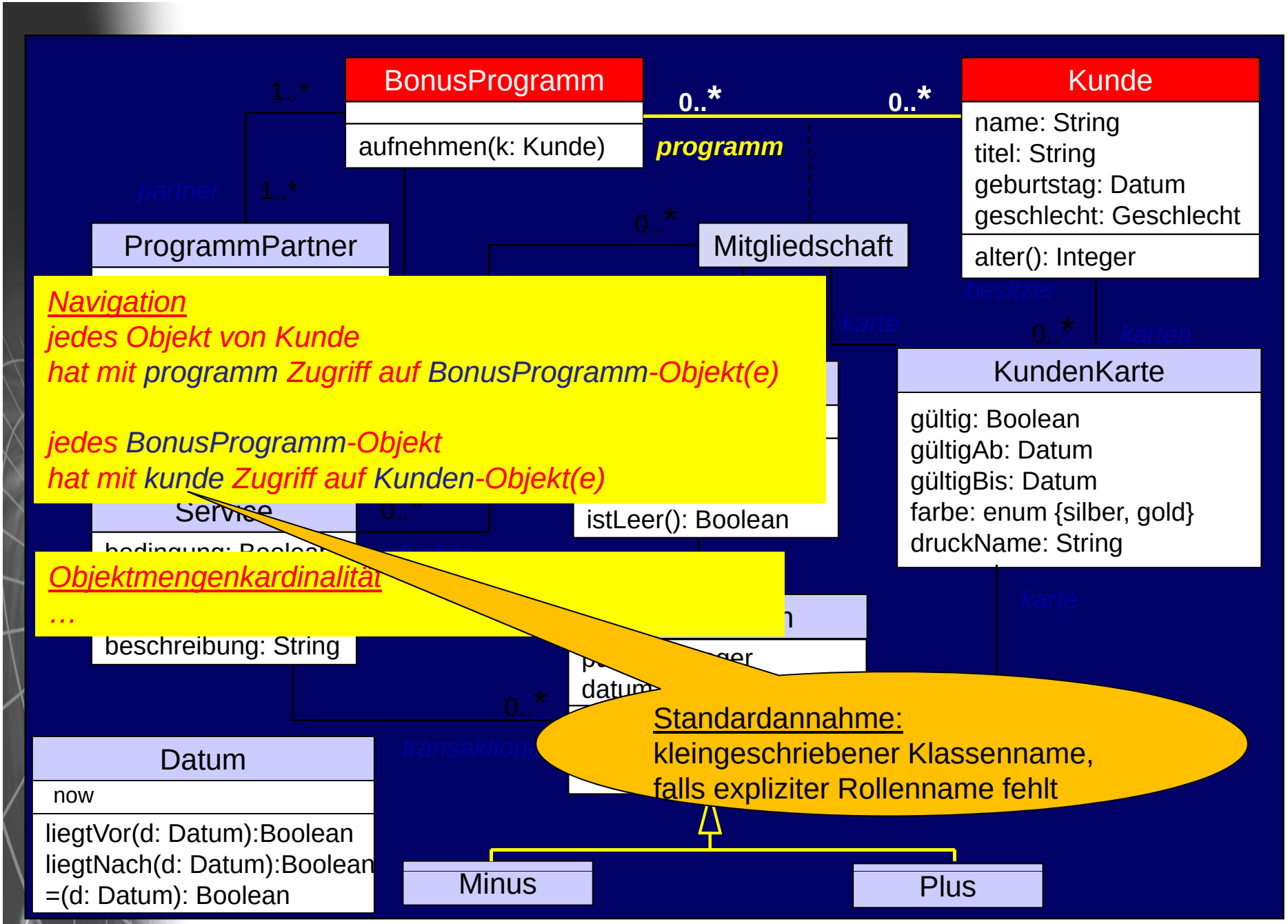
1. Historie von UML
2. Modellierungselemente von UML im Überblick
3. UML-Diagrammarten
4. Diagrammrepräsentationen in UML
5. Zum UML- und OCL-Standard
6. Beispiel: UML-Klassendiagramm

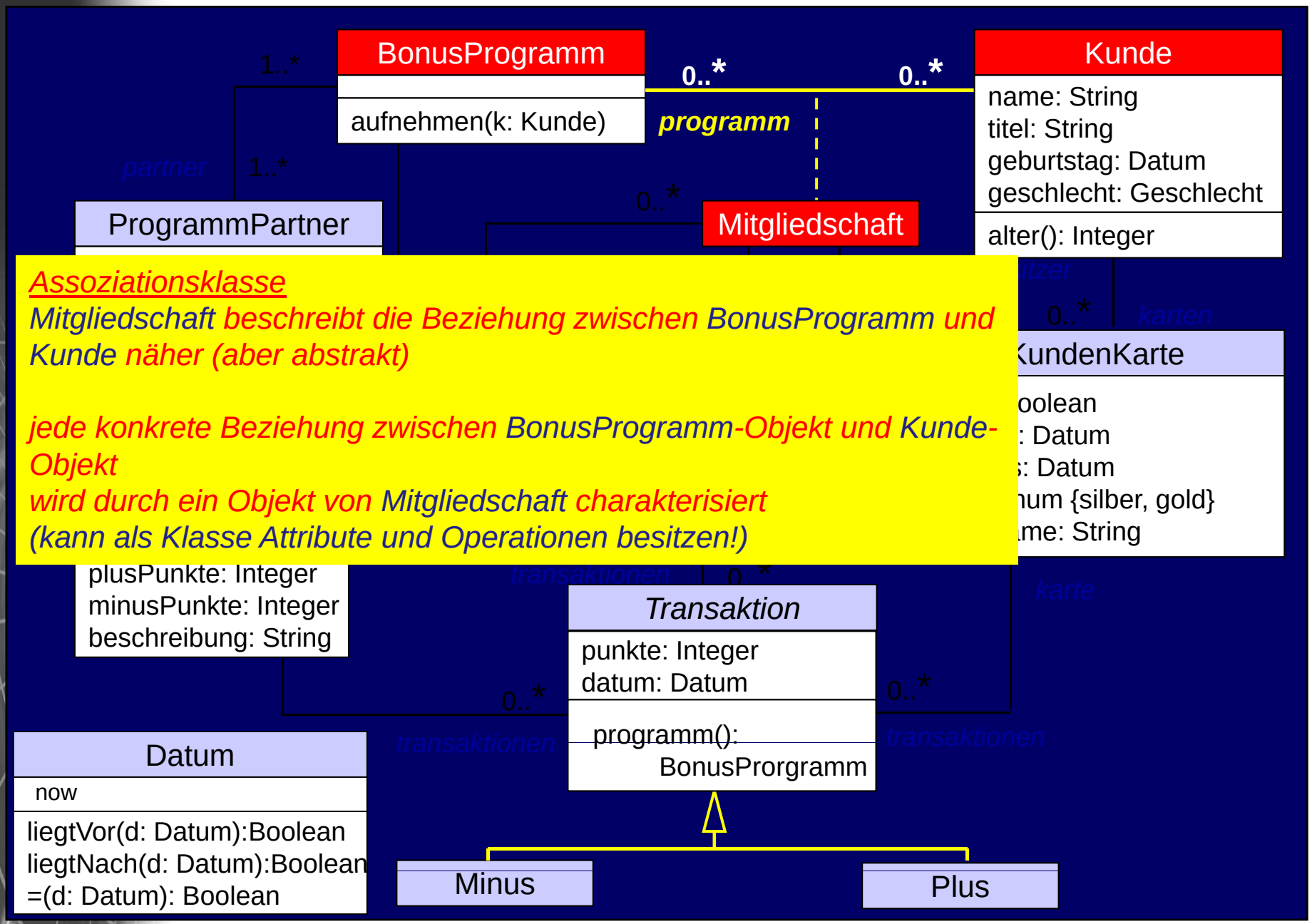

BonusProgramm
aufnehmen(k: Kunde)
Kunde
name: String
titel: String
geburtstag: Datum
geschlecht: Geschlecht
alter(): Integer
1..*
0..*
0..*
programm
partner
1..*
ProgrammPartner
kundenanzahl: Integer
0..*
Mitgliedschaft
besitzer
aktuellerGrad
{ordered}
1..*
0..1
karte
0..*
karten
ServiceStufe
name: String
BonusKonto
punkte:Integer
plus(i:Integer)
minus(i:Integer)
istLeer(): Boolean
KundenKarte
gültig: Boolean
gültigAb: Datum
gültigBis: Datum
farbe: enum {silber, gold}
druckName: String
angebot
0..*
Service
bedingung: Boolean
plusPunkte: Integer
minusPunkte: Integer
beschreibung: String
0..*
verfügbar
transaktionen
0..*
karte
Transaktion
punkte: Integer
datum: Datum
programm():
BonusProrgramm
0..*
transaktionen
0..*
transaktionen
Datum
now
liegtVor(d: Datum):Boolean
liegtNach(d: Datum):Boolean
=(d: Datum): Boolean
Minus
Plus

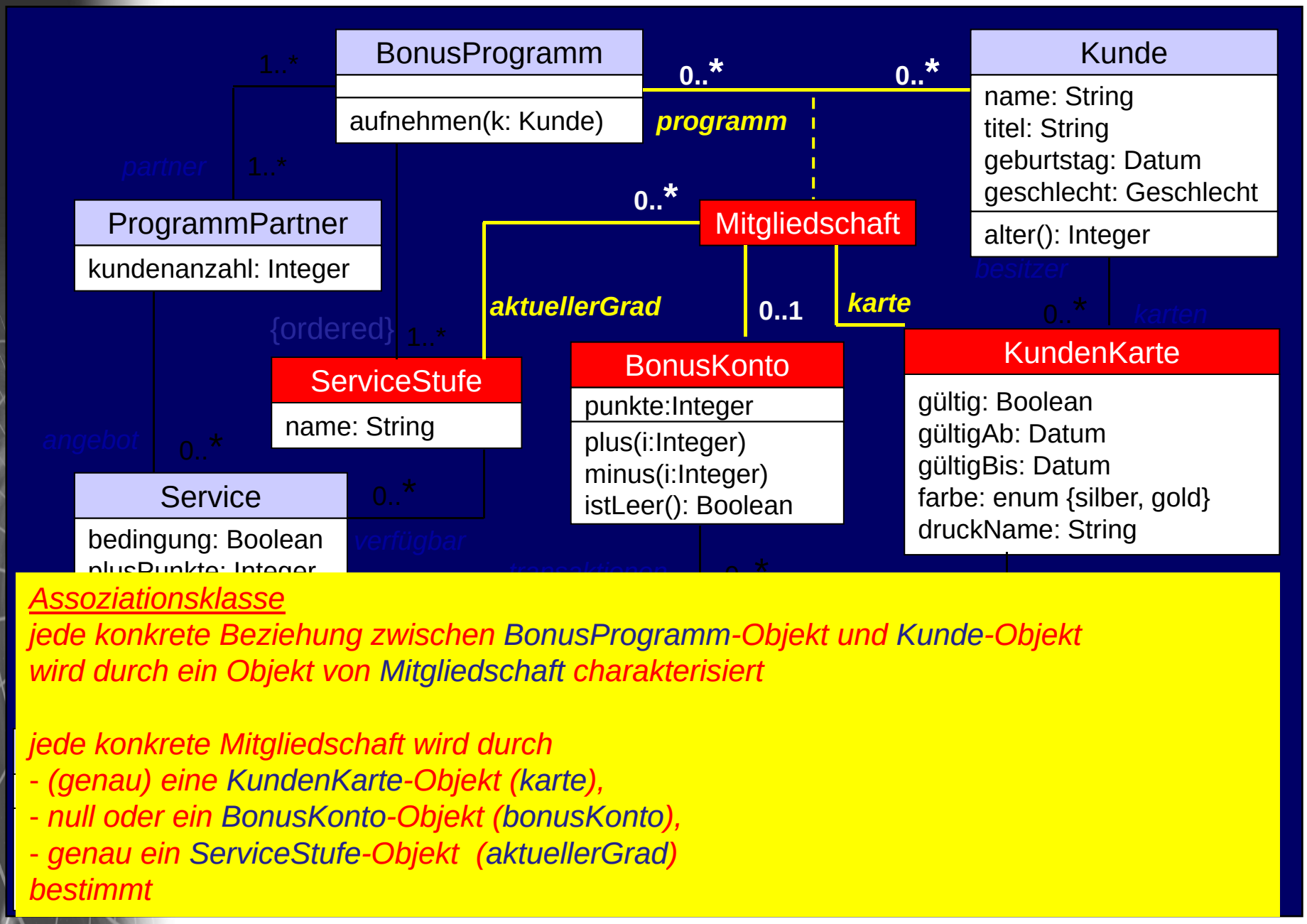

BonusProgramm
aufnehmen(k: Kunde)
Kunde
name: String
titel: String
geburtstag:Datum
geschlecht: Geschlecht
alter(): Integer
1..*
0..*
0..*
programm
partner
1..*
ProgrammPartner
kundenanzahl: Integer
0..*
Mitgliedschaft
besitzer
aktuellerGrad
{ordered}
1..*
0..1
karte
0..*
karten
ServiceStufe
name: String
BonusKonto
punkte:Integer
plus(i:Integer)
minus(i:Integer)
istLeer(): Boolean
KundenKarte
gültig: Boolean
gültigAb: Datum
gültigBis: Datum
farbe: enum {silber, gold}
druckName: String
angebot
0..*
0..*
Service
bedingung: Boolean
plusPunkte: Integer
minusPunkte: Integer
beschreibung: String
verfügbar
transaktionen
0..*
karte
Transaktion
punkte: Integer
datum: Datum
programm():
BonusProrgramm
0..*
0..*
transaktionen
transak
11 konkrete Klassen
1 abstrakte Klasse
1 Generalisierung
Datum
now
liegtVor(d: Datum):Boolean
liegtNach(d: Datum):Boolean
=(d: Datum): Boolean
Minus
Plus


BonusProgramm
aufnehmen(k: Kunde)
1..*
0..*
0..*
programm
Kunde
name: String
titel: String
geburtstag: Datum
geschlecht: Geschlecht
alter(): Integer
besitzer
karten
0..*
Navigation
jedes Kunde-Objekt
hat mit karten Zugriff auf KundenKarte-Objekt(e)
jedes KundenKarte-Objekt
hat mit besitzer Zugriff auf Kunde-Objekt(e)
liedschaft
karte
..1
KundenKarte
gültig: Boolean
gültigAb: Datum
gültigBis: Datum
farbe: enum {silber, gold}
druckName: String
name: String
punkte:Integer
plus(i:Integer)
minus(i:Integer)
istLeer(): Boolean
angebot
0..*
Service
bedingung: Boolean
plusPunkte: Integer
minusPunkte: Integer
beschreibung: String
verfügbar
transaktionen
0..*
Transaktion
punkte: Integer
datum: Datum
programm():
BonusProrgramm
karte
0..*
transaktionen
Datum
now
liegtVor(d: Datum):Boolean
liegtNach(d: Datum):Boolean
=(d: Datum): Boolean
Minus
Plus


BonusProgramm
aufnehmen(k: Kunde)
1..*
0..*
0..*
programm
Kunde
name: String
titel: String
geburtstag: Datum
geschlecht: Geschlecht
alter(): Integer
besitzer
0..*
karten
karte
KundenKarte
gültig: Boolean
gültigAb: Datum
gültigBis: Datum
farbe: enum {silber, gold}
druckName: String
Navigation
jedes Kunde-Objekt
hat mit karten Zugriff auf KundenKarte-Objekt(e)
jedes KundenKarte-Objekt
hat mit besitzer Zugriff auf KundenKarte-Objekt(e)
Objektmengenkardinalität
ein Kunde-Objekt
kann gleichzeitig 0 oder beliebig viele Karten besitzen
(karten ist damit eine Referenz auf eine Menge/Feld/…
Ein KundenKarte-Objekt hat genau ein Kunde-Objekt
als besitzer
Standardannahme !
punkte:Integer
name: String
Datum
now
liegtVor(d: Datum):Boolean
liegtNach(d: Datum):Boolean
=(d: Datum): Boolean
BonusProrg
Minus
Plus
(1) UML benutzt Collection als Template über einen Basistyp (hier KundenKarte) als Typkonstruktor
(2) Collection besitzt eine Menge vordefinierter Operationen


BonusProgramm
aufnehmen(k: Kunde)
1..*
1..*
partner
0..*
0..*
programm
Kunde
name: String
titel: String
geburtstag: Datum
geschlecht: Geschlecht
alter(): Integer
ProgrammPartner
Mitgliedschaft
0..*
karte
besitzer
karten
0..*
KundenKarte
gültig: Boolean
gültigAb: Datum
gültigBis: Datum
farbe: enum {silber, gold}
druckName: String
Navigation
jedes Objekt von Kunde
hat mit programm Zugriff auf BonusProgramm-Objekt(e)
jedes BonusProgramm-Objekt
hat mit kunde Zugriff auf Kunden-Objekt(e)
istLeer(): Boolean
Objektmengenkardinalität
...
beschreibung: String
karte
0..*
transaktionen
Standardannahme:
kleingeschriebener Klassenname,
falls expliziter Rollenname fehlt
Datum
now
liegtVor(d: Datum):Boolean
liegtNach(d: Datum):Boolean
=(d: Datum): Boolean
Minus
Plus


BonusProgramm
aufnehmen(k: Kunde)
1..*
0..*
0..*
programm
Kunde
name: String
titel: String
geburtstag: Datum
geschlecht: Geschlecht
alter(): Integer
partner
1..*
0..*
ProgrammPartner
Mitgliedschaft
Assoziationsklasse
Mitgliedschaft beschreibt die Beziehung zwischen BonusProgramm und Kunde näher (aber abstrakt)
jede konkrete Beziehung zwischen BonusProgramm-Objekt und Kunde-Objekt
wird durch ein Objekt von Mitgliedschaft charakterisiert
(kann als Klasse Attribute und Operationen besitzen!)
0..*
karten
KundenKarte
plusPunkte: Integer
minusPunkte: Integer
beschreibung: String
transaktionen
0..*
karte
Transaktion
punkte: Integer
datum: Datum
programm(): BonusProrgramm
0..*
0..*
transaktionen
transaktionen
Datum
now
liegtVor(d: Datum):Boolean
liegtNach(d: Datum):Boolean
=(d: Datum): Boolean
Minus
Plus



Assoziationsklasse

jede konkrete Beziehung zwischen BonusProgramm-Objekt und Kunde-Objekt wird durch ein Objekt von Mitgliedschaft charakterisiert

jede konkrete Mitgliedschaft wird durch
- (genau) eine KundenKarte-Objekt (karte),
- null oder ein BonusKonto-Objekt (bonusKonto),
- genau ein ServiceStufe-Objekt (aktuellerGrad)
bestimmt