Schwächen der UML bei komponentenorientierter Spezifikation

Innsbruck, 8. Mai 2008,
Dr. Thomas Tensi, sd&m AG

sd&m

Übersicht

- Einführung
- Komponentenbegriffe der UML und der modularen Sprachen
- Analyse der UML-Komponentenmodellierung
  - Fallbeispiel: Listen mit Cursor
  - Schwächen
    - zu enger Schnittstellenbegriff
    - unvollständige Typalgebra
  - Workarounds
- Zusammenfassung

sd&m AG – software design & management

Geschäftsfelder

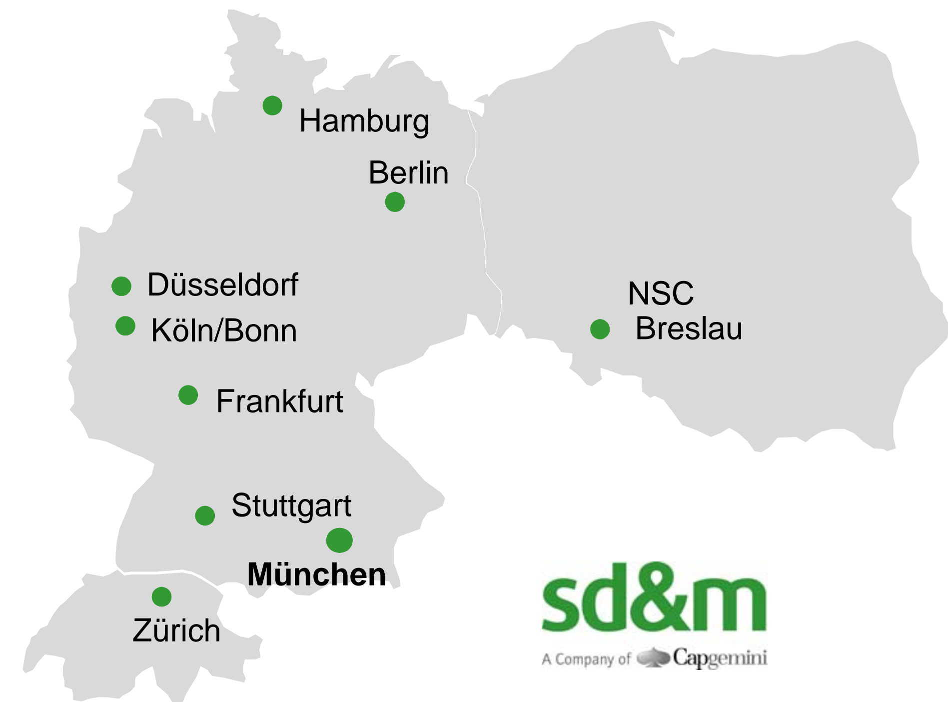
- Entwicklung und Integration maß-geschneiderter Informationssysteme für unternehmenskritische Prozesse
- IT-Beratung mit Umsetzungskompetenz

Kunden

- Namhafte Unternehmen und Organisationen, die durch Einsatz individueller Lösungen Wettbewerbsvorteile erlangen

Kernkompetenz

- Software-Engineering und Projektmanagement



Eckdaten 2007

Mitarbeiter: ca. 1.400
Umsatz: ca. 200 Mio. €

Forschung

Aktionär

Warum ist Komponentenmodellierung in der UML überhaupt interessant?

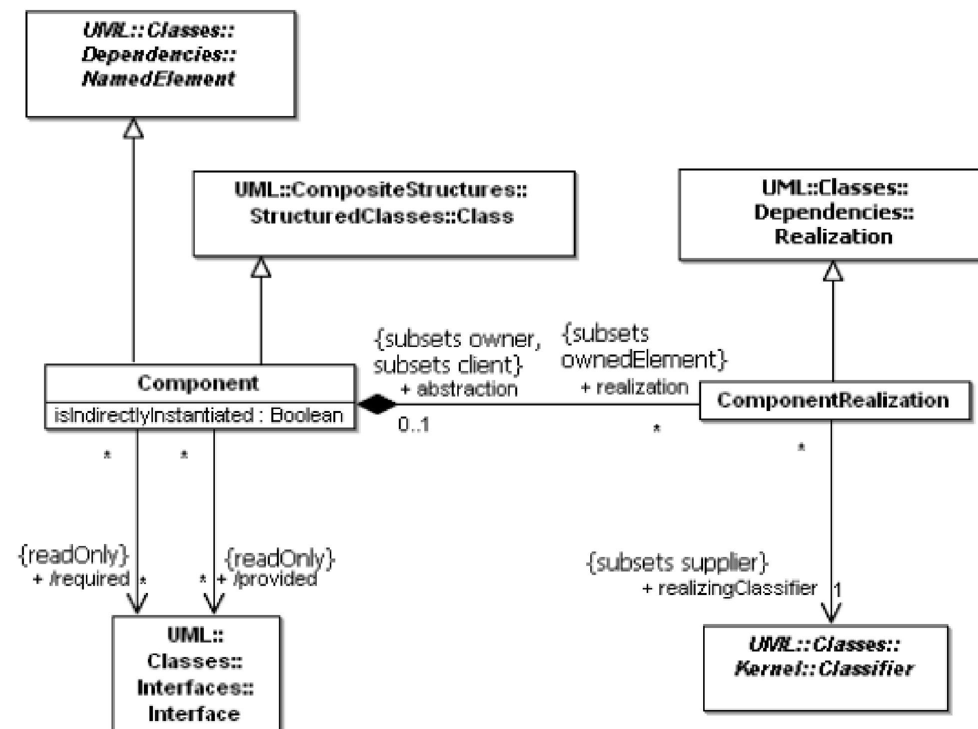
- Die UML ist mittlerweile die etablierte Modellierungssprache für Softwarespezifikation (der "Industriestandard").
  - extrem reichhaltig
  - unterstützt durch viele Werkzeuge
  - leider teilweise unklare Semantik, stark OO-lastig
- Komponentenorientierte Softwarearchitektur gilt derzeit als Stand der Technik ("soA-Hype").
  - wieder einmal!

Vorbemerkung: Komponentenmodellierung ist natürlich möglich in der UML!

- Die Beschreibungsmittel der UML erlauben die Modellierung von Komponentenarchitekturen.
  - mit (formalen) Schnittstellen
  - mit Implementierungsabhängigkeiten
- Aufgrund ihrer starken objektorientierten Ausrichtung und diverser Entwurfsfehler gibt es aber Probleme bei der Komponentenmodellierung.
  - UML ist keine Allzweckmodelliersprache!
  - Die Probleme sind nicht rein akademisch.

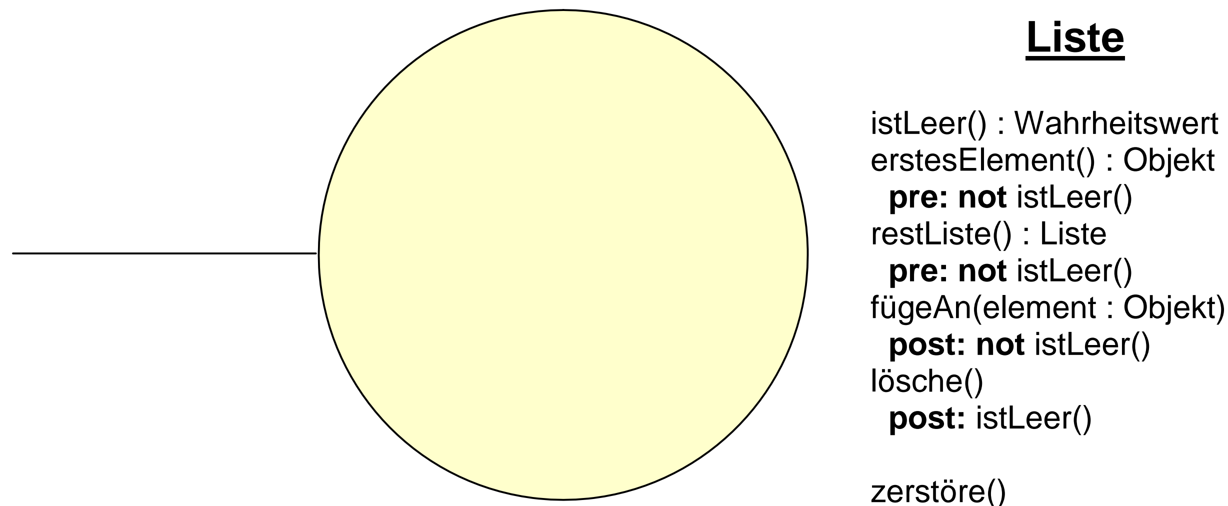
Komponentenbegriff der UML (I)

- Eine Komponente ist eine Klasse.
  - "Eine Komponente ist ein modularer Teil eines Systems, der seinen Inhalt verkapselt und dessen Manifestation in der Umgebung austauschbar ist. Ihr Verhalten ist definiert über angebotene und benötigte Schnittstellen." (UML 2007)



Komponentenbegriff der UML (II)

- Eine Schnittstelle ist ein Namensraum für eine Menge von Operationssignaturen.
  - ggf. ergänzt um Semantikdefinition über Zusicherungen



- Implementierungsbeziehung: zu jeder Schnittstellenoperation muss es eine öffentliche Operation der Komponente oder einer die Komponente implementierenden Klasse geben

Komponentenbegriff in klassischen "modularen" Ansätzen (I)

z.B. in modularen Programmiersprachen wie Ada, Modula-3, ...

- Eine Komponente ("Modul") ist eine Menge von Definitionen für Konstanten, Variablen, Typen und Funktionen.

```
KOMPONENTE Liste
  EXPORTIERT Liste

CONST abc = ...
TYPE T = ...

PROC istLeer (in liste : Liste.T) : Wahrheitswert
BEGIN END


...
PROC zerstöre (inout liste : Liste.T)
BEGIN END
```

Komponentenbegriff in klassischen "modularen" Ansätzen (II)

- Eine Schnittstelle ist ein Namensraum für eine Menge von zusammengehörigen Deklarationen von Konstanten, Variablen, Typen und Funktionen.
  - auch Objektorientierung möglich: über Klassentypen
    - hier aus Vereinfachung nicht benutzt

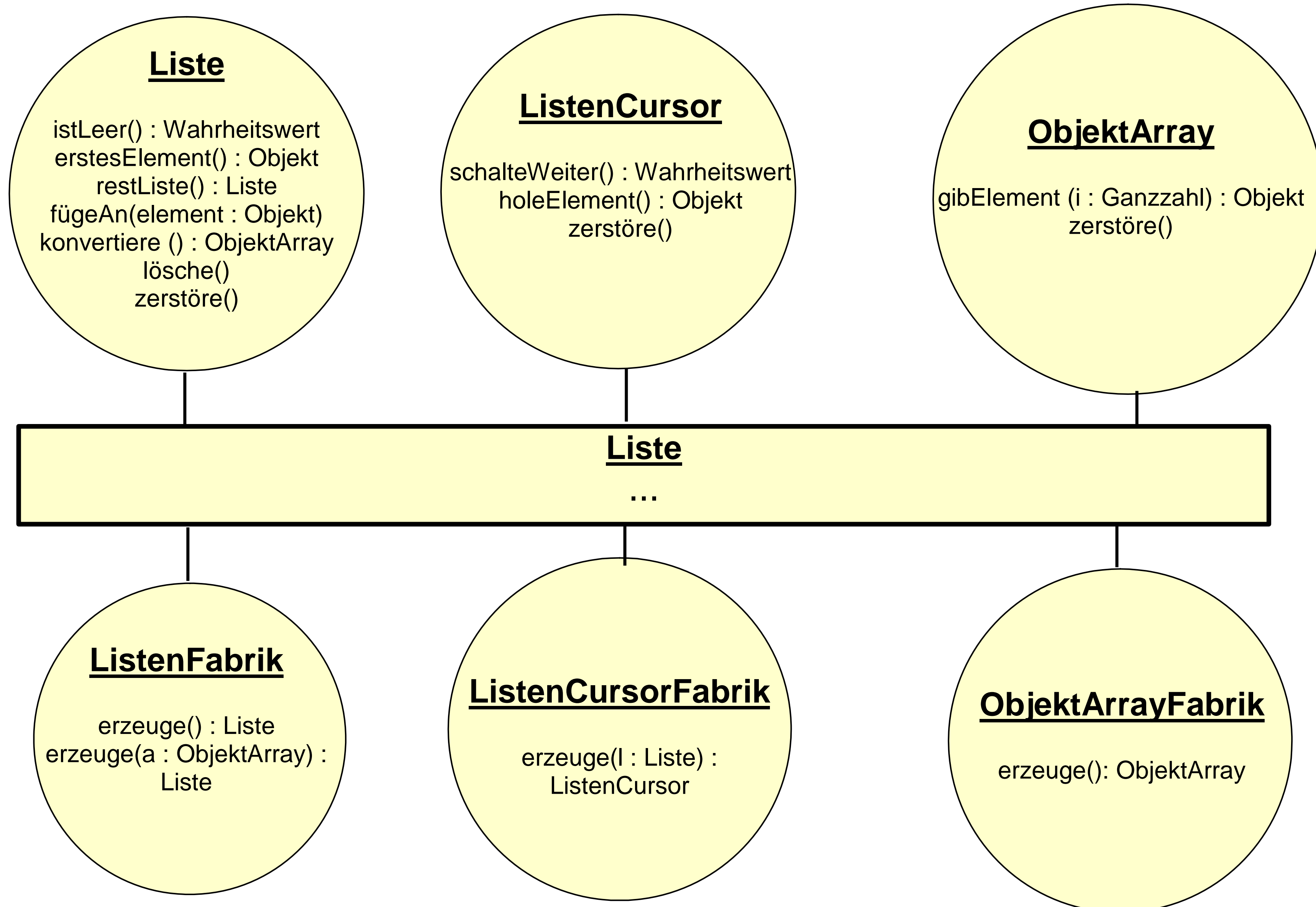
```
SCHNITTSTELLE Liste

CONST xyz = ...
TYPE T = ...

PROC istLeer (in liste : Liste.T) : Wahrheitswert
...
PROC zerstöre (inout liste : Liste.T)
```

- Korrespondenz simpel: jeder Name der Schnittstelle ist definiert durch die Definition mit demselbem Namen in Komponente

Fallbeispiel

- Gewünscht sei eine Komponente für sequentielle Listen von Objekten
  – mit klassischen Listenoperationen (anfügen, Prüfen auf leer, ...)
  – mit Cursor (zum Durchlaufen der Liste)
  – mit Möglichkeit, eine Liste in ein Array oder aus einem Array von Objekten zu konvertieren

Umsetzung des Fallbeispiels in der UML



Schwächen der UML: unpassender Schnittstellen-/Komponentenbegriff (I)

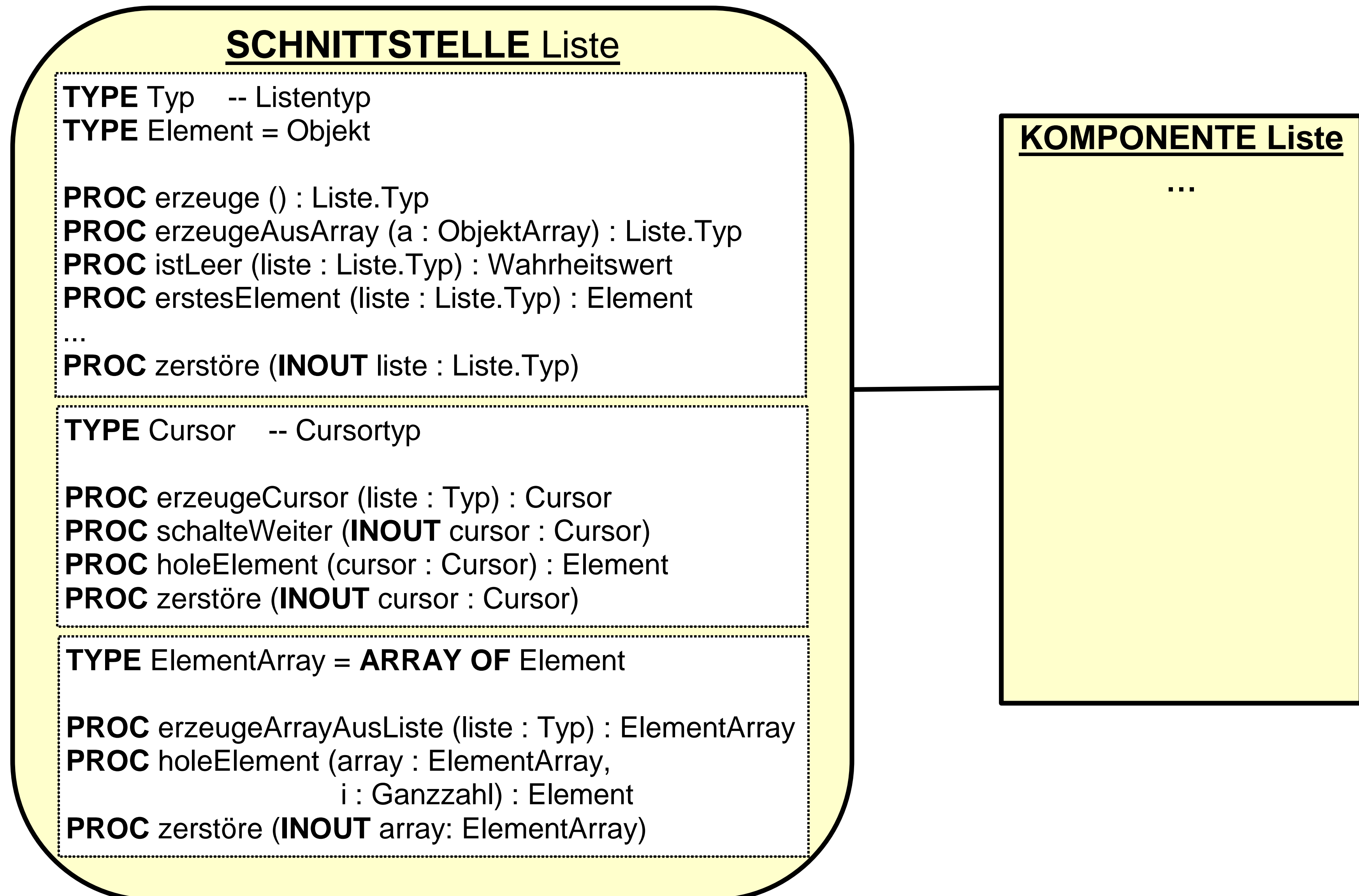
- Schnittstellen sind nicht instanziierbar.
  - Damit kann ein System nicht allein aus den Schnittstellen heraus verstanden werden.
    - auch klassenbezogene ("statische") Operationen werden für den Ablauf benötigt (z.B. Konstruktoren, Suchoperationen...)
    - schmutziger Trick: eigene "Fabrik"-Schnittstelle für statische Methoden
- Rolle von öffentlichen Operationen einer Komponente unklar
  - Wie stehen die öffentlichen Operationen der Komponente in Zusammenhang mit denen ihrer Schnittstellen?
    - Obermenge?
  - Die öffentlichen klassenbezogenen Operationen kommen in Schnittstellen nicht vor. Was ist ihre Rolle?
    - typischerweise sind das Konstruktoren oder instanzübergreifende Operationen

Schwächen der UML: unpassender Schnittstellen-/Komponentenbegriff (II)

- Der Namensraum wird verschmutzt mit Typdefinitionen.
  - Manche Typen sind eigentlich nur im Kontext einer Schnittstelle (eines zentralen Typen) sinnvoll.
    - wer braucht einen "ListenCursor", wenn er keine "Liste" hat?
  - Um das Problem der fehlenden klassenbezogenen Methoden zu umgehen, werden Fabrikschnittstellen eingeführt.
    - das löst aber nicht das Problem, dass die instanzbezogenen Operationen der Schnittstelle eigentlich durch klassenbezogene Operationen der Komponente implementiert werden müssen.

➔ Inflation von - teilweise technischen – Schnittstellen

Umsetzung des Fallbeispiels in modularem Ansatz



Eigenschaften des modularen Komponentenbegriffs

- Eine Schnittstelle ist hier nicht instanziierbar, sondern ein in einem Exemplar vorhandener Namensraum für Deklarationen.
  - klassenbezogene Operationen sind möglich
  - wieviele Instanzen möglich sind, hängt von den Deklarationen ab
    - eine Instanz ➔Funktionen ohne entspr. ersten Parameter
    - viele Instanzen ➔ Typdefinition und Funktionen mit entspr. ersten Parameter (oder Klassentyp)

- Zusammengehörige Typen sind hier in eine syntaktische Einheit zusammengebracht.
  - z.B. "Liste.Typ" "Liste.Cursor" "Liste.Element"
  - Nachteil: Redefinition eines Moduls schwieriger als bei Klassen
    - aber oft unnütz!

Schwächen der UML: unvollständige Typalgebra (I)

- Für Komponenten sind Typumbenennungen nützlich.
  - in unserem Beispiel: `ObjektArray = ARRAY OF Objekt`

- Besonders wichtig sind sie bei generischen Typen:

```
TYPE Mapping = MAP(String, Konto)
TYPE MappingList = LIST(Mapping)
...
PROC pipapo (IN a : MappingList) : MappingList
```

ansonsten müsste an allen Verwendungsstellen der expandierte Typ `LIST(MAP(String, Konto))` verwendet werden

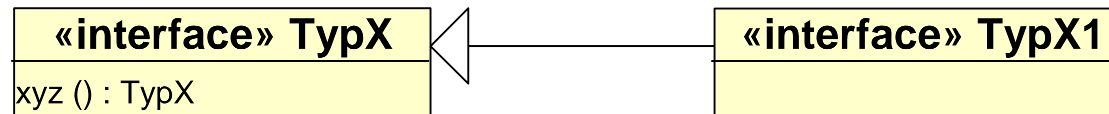
Schwächen der UML: unvollständige Typalgebra (II)

Warum kann die UML keine Typumbenennungen?
Sie kennt doch Vererbung??

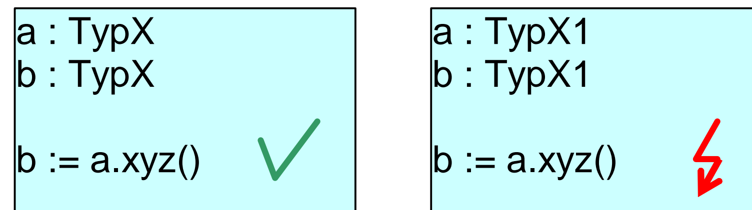
- Die UML kennt nur folgende Typkombinatoren:
  - Strukturierung (Verbund): über Attribute von Klassen
  - Feldbildung (Array): nur auf Attributebene über deren Kardinalitäten
  - Vererbung (mit kontravarianter Redefinition, s.u.)
- Mit Vererbung kann man keine Typumbenennung simulieren
  - siehe Folgefolie

Schwächen der UML: unvollständige Typalgebra (III)

Beispiel: Redefinition von TypX durch TypX1



- Problem: die Operation xyz liefert – angewandt auf ein TypX1-Objekt – ein Objekt vom TypX zurück:



- Eine Redefinition von xyz in TypX1 zu `xyz():TypX1` ist nicht erlaubt, weil in der UML die Signaturen redefinierter Operationen Untertypen seien müssen.
  – sogenannte kontravariante Redefinition

- Das ganze Problem verschärft sich bei generischen Typen!

Schwächen der UML: unvollständige Typalgebra (IV)

- kontravariante Redefinition in der UML:

```
Operation::isConsistentWith (redefinee: RedefinableElement) : Boolean;
    pre: redefinee.isRedefinitionContextValid(self)
    isConsistentWith = (redefinee.oclIsKindOf(Operation) and
    let op: Operation = redefinee.oclAsType(Operation) in
    self.ownedParameter.size() = op.ownedParameter.size() and
    forAll(i | op.ownedParameter[i].type.conformsTo(self.ownedParameter[i].type)))
```

- Typkonformität in der UML:

```
Classifier::conformsTo(other: Classifier): Boolean;
    conformsTo = (self=other) or (self.allParents()->includes(other))
```

Workarounds

Problem	Workaround	Bewertung
nicht instanziierbare Schnittstellen	Ignorieren der UML-Semantik; statische Operationen in Schnittstellen aufnehmen und durch Stereotyp kennzeichnen	umgeht die UML-Semantik; sehr behelfsmäßig; Werkzeugunterstützung zweifelhaft
Verhältnis von öffentlichen Schnittst.-Operationen zu Komponenten-Operationen	Entwurfsrichtlinie: alle öffentlichen Komponentenoperationen müssen in mindestens einer Schnittstelle vorkommen	unproblematisch; zu klären ist, warum Komponenten überhaupt öffentliche Operationen brauchen
Typinflation; keine Zuordnung von Typen zu Schnittstellen	Hilfstypen (wie Listencursor) werden über Abhängigkeits-beziehung der Schnittstelle zugeordnet	Modellprüfer muss kontrollieren, dass diese Zuordnung nicht verletzt wird
unvollständige Typalgebra	nicht prinzipiell lösbar; Verwendung von kovarianter Redefinition für die UML; idealerweise auch Ver-wendung passender Zielsprache (z.B. Eiffel)	umgeht die UML-Semantik; automatische Code-generierung bei kovarianter Redefinition schwierig (wenn Zielsprache es nicht erlaubt)

Zusammenfassung

- Die UML ist nur oberflächlich betrachtet für die Komponentenmodellierung tauglich.
  - Gute Komponentenmodellierung führt in der UML oft zu einer Inflation von Schnittstellen.
- Die klassischen modularen Ansätze der 80er Jahre sind erheblich besser zur Modellierung von Komponentenarchitekturen geeignet.
  - Sie haben insbesondere ein vernünftiges Typmodell.
- Es gibt Techniken, wie man die Schwächen der UML umgeht.
  - Sie sind aber teilweise nicht konform zur UML-Semantik.
  - Ihre Anwendung in einer etablierten UML-Werkzeugumgebung ist problematisch.
- Man sollte daher überlegen, ob man nicht andere Notationen zur Komponentenmodellierung verwendet.
  - z.B. "klassische" Komponentenmodellierung

Menschen machen Projekte.

sd&m

sd&m AG
software design & management AG
Carl-Wery-Str. 42
81739 München
Tel +49-89-63812-0
www.sdm.de