M e t a - Modelle

Teil 2:

Meta-Modelle in der Praxis

Vortrag im Arbeitskreis Objektorientierung
14.04.2003

Übersicht

- Vorstellung Person/ Firma
- **Teil 1: Meta-Modelle in Reinkultur**
  Begriffsbestimmung/ Beispiele/ Einsatzgebiete
- **Teil 2: ein bisschen Theorie und viel Praxis**
  die 4-Schichten-Metamodell-Architektur
  Meta-Bestandteile in Anwendungs-Software:
  Vorteile/ Nachteile/ Konsequenzen

Infos zur Person und Firma

Beate Ritterbach

- Methodenberaterung und Software-Entwicklung im Umfeld Objekt-Orientierung/ Java
- seit 1990 freie Beratung/ Entwicklung
- seit 2001: Logimod GmbH

Interessenschwerpunkte

- Techniken zur Entwicklung von **verständlichem** Code
- Agile Prozesse, insbes.Refactoring
- Modellierung
- Programmiersprachen
- zugrundeliegende Paradigmen und Konzepte
- Qualitätskriterien, Messbarkeit, Metriken

Zielsetzung

Erkennen/ Sensibilisieren für Themen wie

- Was sind Meta-Modelle?
- Wofür sind sie einsetzbar?
- Wann bewegt man sich auf der Meta-Ebene?
- Welche Vorteile/ Welche Gefahren birgt es?

Zusammenfassung Teil 1 („Meta-Modelle in Reinkultur“)

Meta-Modelle

- beschreiben die Semantik von Sprachen (= Programmiersprachen, Modellierungssprachen, ggf. Werkzeuge)

Einsatzgebiete:

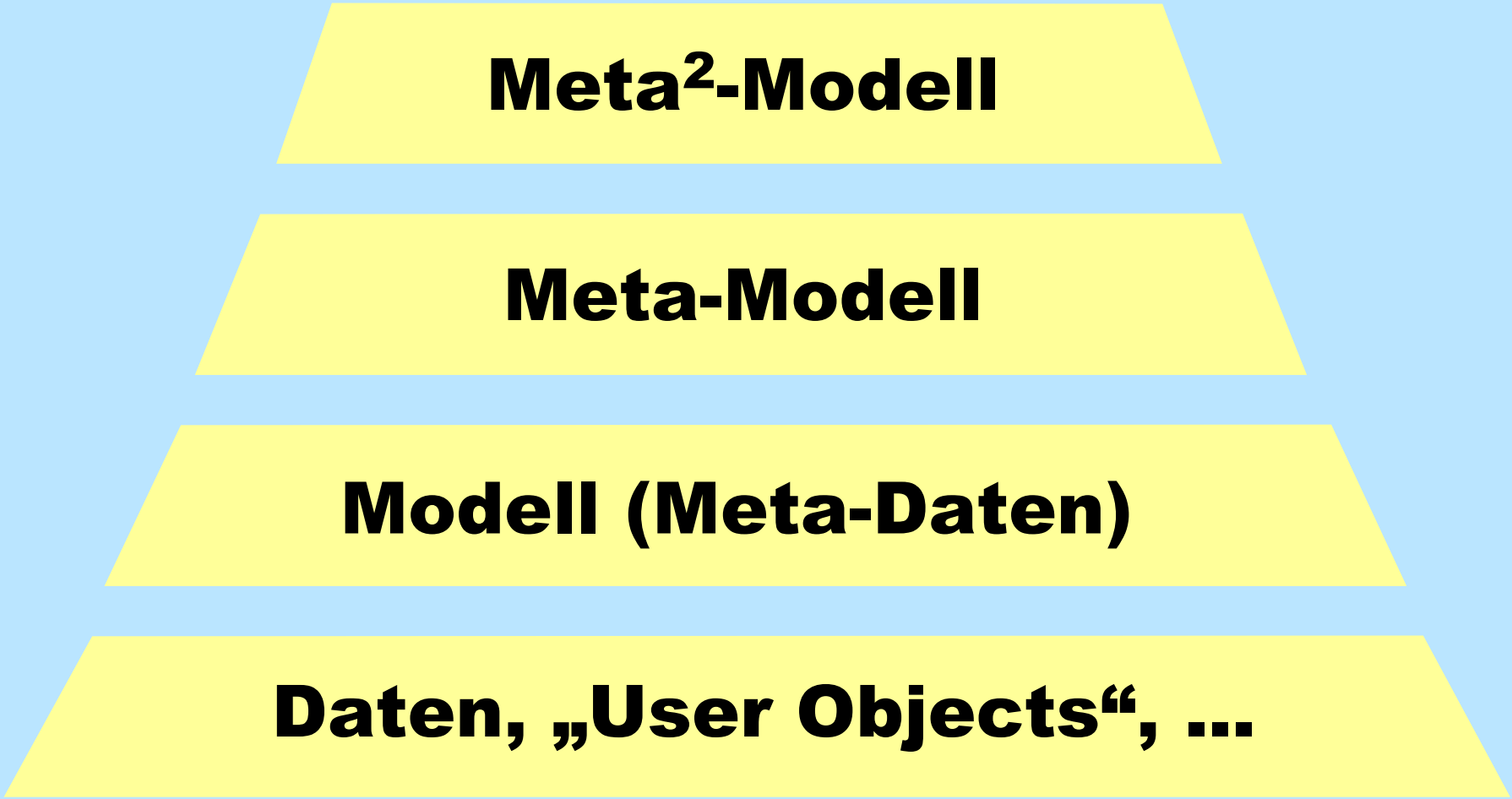
- Entwerfen einer Sprache
- Verstehen / Unterrichten
- Qualitätsbeurteilung/ Vergleiche

Teil 2: Meta-Aspekte in Anwendungssystemen

Gliederung

- Die 4-Schichten Meta-Daten-Architektur
- Mischen der Ebenen - Vorteile/ Nachteile/ Konsequenzen
  - Ebene 1 <-> Ebene 2
  - Ebene 1 <-> Ebene 0
  - Meta-Elemente in Programmiersprachen

4-Schichten Meta-Modell-Architektur



4-Schichten Meta-Daten-Architektur

	Schicht	Definiert	Beispiel-Elemente
3	Meta-Meta-Modell	Sprache zur Spezifizierung von Meta-Modellen	Meta-Klasse, Meta-Attribut, Meta-Operation
2	Meta-Modell	Sprache zur Spezifizierung von Modellen	Klasse, Attribut, Methode, Variable, Assoziation
1	Modell, Meta-Daten	Sprache zur Beschreibung von Informations-Systemen	Vertrag, Window, betrag, setValue(), if (a<b)...
0	Daten, User objects	Ein spezifisches System (information domain)	[Person: name=Rolf, gehalt=32000], 45.1, "hello"

Beziehungen der Schichten

Schicht n ist Instanz von Schicht (n+1), z. B.

- Das UML-Meta-Modell ist eine Instanz des Meta-Meta-Modells
- Ein UML-Modell eines Software-Systems ist eine Instanz des UML-Meta-Modells
- Die Daten bzw. Abläufe im Laufzeitsystem sind eine Instanz des Modells

Datenaustausch

Um Daten auf Ebene n zwischen zwei Systemen auszutauschen, muss das (Teil-)Meta-Modell auf Ebene n+1 übereinstimmen

- Beispiel (Ebene 0/1):
  Datenmigration
- Beispiel (Ebene 1/2):
  Austausch zwischen Modellierungswerkzeugen
  Wechsel der Programmiersprache

Beispiele für Meta-Ansätze

Einige normierte/ kommerzielle Meta-Ansätze:

- MOF (OMG)
  Meta-Meta-Modell zum Erstellen von Meta-Modellen, Untermenge/ Variation der UML
- CDIF, XMI
  Austausch von Daten zwischen CASE-Tools
- XML DTD / XML Schema

Meta-Modell = Ontologie

Trennung der Meta-Ebenen

In den bisherigen Ausführungen waren die Ebenen strikt getrennt:

3 Meta-Meta-Modell
2 Meta-Modell
1 Modell, Programmsourcen (= Meta-Daten)
0 Gegenstandsbereich/ User data

Anwendungs-Software-Entwickler bewegen sich ausschliesslich auf Ebene 1.

Meta-Aspekte in Anwendungssystemen

- Sind die Meta-Ebenen in der Praxis immer deutlich getrennt?
- Ist immer klar, auf welche Ebene eine Information gehört?

- Im folgenden werden einige Anwendungs-Beispiele gezeigt, die Meta-Aspekte beinhalten, und die Konsequenz davon aufgezeigt

Beispiel 1: setValue-Methode

Lösung A

```
void setValue(String name, int value) {
   if (name.equals("height")) { _height = value;}
   if (name.equals("width")) { _width = value;}
   Assert.shouldNeverReachHere();
}
```

Lösung B

```
void setHeight(int arg) { _height = arg;  }
void setWidth(int arg) { _width = arg; }
```

Aus: M.Fowler: Refactoring, Improving the design of existing code

Gegenüberstellung

Lösung A

- **flexibler** – für weitere Attribute verwendbar
- **schmalere Schnittstelle** - nur 1 statt 2 Methoden
- **längerer Code –** Prüfungen, mehr Parameter

Lösung B

- **deutlicher** - Methodenname lässt erkennen, welche Attribute verändert werden
- **sicherer** - nicht vorhandene Attribute nicht ansprechbar (Compile-Fehler statt Laufzeitfehler)

Weitere Fragen

- Wie würde Lösung A ausgesehen, wenn die Attribute von unterschiedlichem Typ wären?
- Wie sieht der Code bei Lösung A aus, wenn nicht alle Attribute des Objektes über die setValue-Methode veränderbar sein sollen?

Ursache für die Schwierigkeiten

- Übergabeparameter „name“ ist ein Meta-Attribut (gehört zu Meta-Klasse „Instanz-Variable“)
- Lösung A hat damit einen Teil des Meta-Modells (Ebene 2) in den Code (Ebene 1) verlagert
- Entsprechend werden Meta-Daten (Attribut-Namen) von Ebene 1 in Ebene 0 verlagert
- Refacoring ("Replace Parameter with Explicit Methods") empfiehlt die Umwandlung von Lösung A in Lösung B !!!

Beispiel 2: Briefschreibe-System

Ausgangspunkt:

Unterschiedliche Systeme (Anbahnung, Vertrag, Inkasso, Schaden, ...) erstellen und versenden Briefe an Kunden, z. B.

- Werbeschreiben,
- Rechnungen,
- Mahnungen,
- Statusinformationen

Anforderungen an Briefschreibung

Brief-Texte und Inhalte sind unterschiedlich, weisen aber auch Gemeinsamkeiten auf, z. B.

- Absender- und Empfänger-Adresse
- Betreff
- Poststrassen-Steuerung, Versand
- Archivierung
- ...

Lösung 1

jedes fachliche System enthält eigene Funktionalität für die Briefschreibung

- Gleiche Funktionalität wird mehrfach entwickelt
- mehrfache Pflege notwendig
- ggf. uneinheitliches Erscheinungsbild

Lösung 2:
Zentrales Briefschreibe-System

Ermöglicht:

a. Definieren von Briefen (genauer: Brief-Schablonen) bestehend aus Texten und Parametern

b. Erstellen von individuellen Briefen auf Basis der Schablonen

Beispiel für Brief-Anwendung

Frau Birgit Meyer
Am Knill 13
23456 Hamburg

Hannover, den 09. 12. 2002

Sehr geehrte Frau Meyer,

hiermit teilen wir Ihnen den Stand Ihrer Lebens-Versicherung 0234987-87 zum 30. 9. 2002 mit:

aktuelles Deckungskapital:	20 845,34 Euro
Gewinnguthaben	2 368,40 Euro
Gesamt-Guthaben	22 456,74 Euro

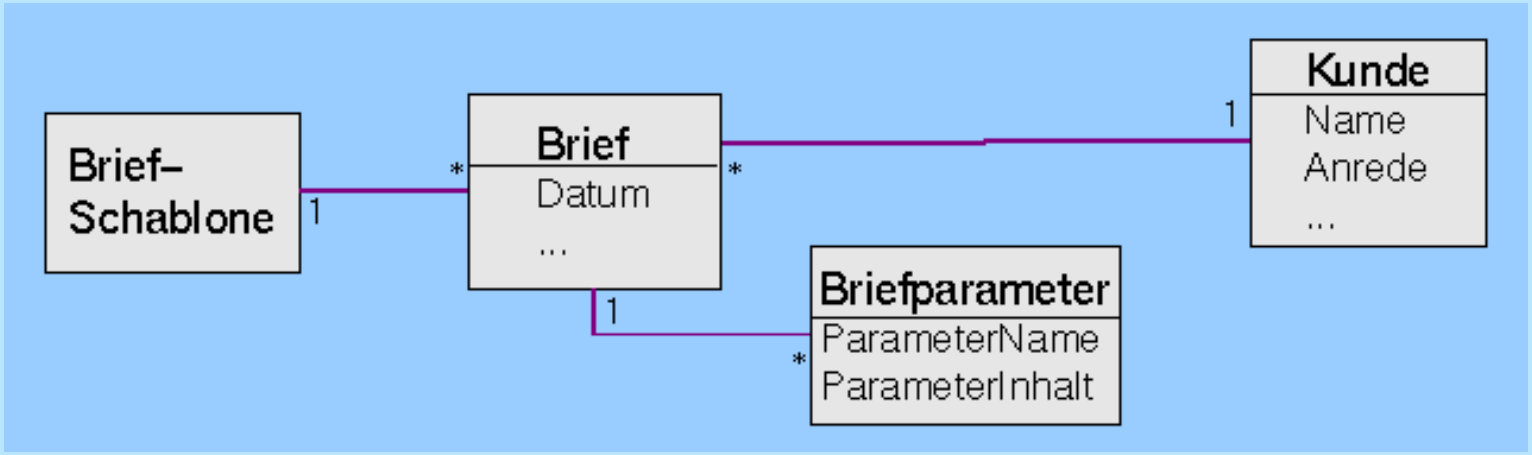
Mit freundlichen Grüssen

Beispiel: Schablone für Statusbriefe

hiermit teilen wir Ihnen den Stand Ihrer Lebens-Versicherung &vsnr zum &statdat mit:

aktuelles Deckungskapital:	&deckKap &waehrung
Gewinnguthaben	&gewinnG &waehrung
Gesamt-Guthaben	&gesamt &waehrung

Parameter als Meta-Komponenten



- Briefschreibung enthält Meta-Bestandteile, d. h. vermischt Ebenen 1 und 2
- Briefschreibung ist kein "reines" Meta-System (wie z. B. ein Modellierungswerkzeug oder ein Compiler)

Bestandteile der Brief-Schreibung

- Schablone: fester Text mit variablen Parametern, für jede Anwendung unterschiedliche Parameter (Art, Anzahl)
- Zum Erstellen eines konkreten Briefes werden die jeweiligen Parameter-Inhalte übergeben
- die flexiblen Parameter befinden sich konzeptionell auf der Meta-Ebene (Ebene 2) !!!
- die Instanzen der Parameter enthalten ihrerseits Beschreibungen von Attributen und werden in jedem Einzelfall instanziiert

Vorteile der Meta-Lösung

- Briefschreibungs-System für unterschiedliche Anwendungs-Situationen einsetzbar
- auch für Briefe, die man bei Entwurf des Brief-schreibungssystems (noch) nicht kennt
- einheitlicher Aufbau aller Briefe
- einheitliche Funktionalität (drucken, versenden, archivieren, ...)
- Wegfall von Mehrfach-Entwicklung/ Pflege

Nachteile der Meta-Lösung

Einsatz von Meta-Bestandteilen in einem Anwendungssystem bringt Kosten und Zusatzaufwände mit sich:

- für die Entwickler des Meta-Systems selbst
- für die Entwickler der benutzenden Anwendungssysteme
- für die Organisation des Projekte und Verteilung der Verantwortlichkeiten

Keine fertige Lösung

Rahmen, mit dem man jeden spezifischen Brief mit (hoffentlich) geringerem Aufwand erstellen kann

Für jeden Einzelfall erforderlich:

- Definieren der anwendungs-spezifischen Briefe (Festlegen Texte, Parameter-Namen, ...)
- Erzeugen der anwendungs-spezifischen Briefe (Festlegen Parameter-Inhalte)

Für beides muss das Briefschreibe-System eine Schnittstelle bieten

Systemsoftware-typische Funktionalität im Meta-System

z. B.

- Prüfen, ob Parameter(name) vorhanden
- Typprüfung der Parameter-Inhalte
- Parsen der Parameter-Inhalte

Führt zu

- Zusatzaufwand für Entwickler des Meta-Systems
- Prüfungen zur Laufzeit statt zur Compile-Zeit
- Verlust von Sicherheit

Verständlichkeit der Meta-Schnittstelle

Notwendige Kenntnisse für Entwickler des Anwendungssystems:

- Programmiersprache des eigenen Systems
- fachliche Anforderungen des eigenen Systems
- (Meta-)Schnittstelle des Briefschreibungs-Systems (keine "übliche" Datenschnittstelle)
- Meta-Daten (= spezifische Brief-Schablone)

I. d. R. ist das Meta-System hausgemacht
-> fehlende /schwer verständliche Dokumentation

Verständlichkeit der fachlichen Logik

Verstehbarkeit/ Nachvollziehbarkeit/ Wartung ist erschwert, weil Spezifikation verteilt ist auf

- Anwendungssystem
- Meta-System
- Meta-Daten

Verantwortlichkeit für die Meta-Daten

Übliche Aufteilung der Verantwortung/ Pflege:
Anwender: Daten (Ebene 0)
Entwickler: Meta-Daten (Ebene 1)

Meta-System: Briefdefinitionen enthalten auch Meta-Daten (z. B. Parameternamen)
- logisch: Ebene 1
- implementierungstechnisch: Ebene 0

-> wer pflegt diese (Meta-)Daten?
Anwender? Entwickler Anwendungssystem?
Entwickler Meta-System?

Funktionalität im Meta-System zu unflexibel

Meta-System legt die verfügbare Funktionalität fest, mit der die einzelnen Briefe erstellt werden können

Was ist mit zusätzlichen Anforderungen, die nicht in diesem Rahmen passen? z. B.

- Grössere Länge für Parameter-Inhalte
- Textabschnitte abhängig von Parameter-Inhalten
- Variables Firmen-Logo
- Briefe nicht nur an Kunden, sondern auch an Mitarbeiter, Investoren, ...

Geteilte Verantwortlichkeit für Entwicklung

- Geringer Einfluss des Anwendungs-Systems auf (Weiter-)Entwicklung des Meta-Systems
- Erhöhter Kommunikationsaufwand zwischen Entwickler Anwendungssystem
  <-> Entwickler Meta-System
- ggf. unterschiedliche Interessenlagen

Aufgabenteilung ist analog Entwickler
<-> Entwicklungs-Software-Hersteller

Fazit: Meta-Bestandteile in Anwendungs-Systemem

Daumenregel für Einsatz von Meta-Bestandteilen in einem "normalen" Anwendungssystem

a. Lass es!

b. Sinnvoll in Ausnahmefällen, d. h. wenn

- Lösung für eine Klasse von gleichartigen Aufgaben benötigt wird
- Flexibilität, aber nur in vorgegebenem Rahmen

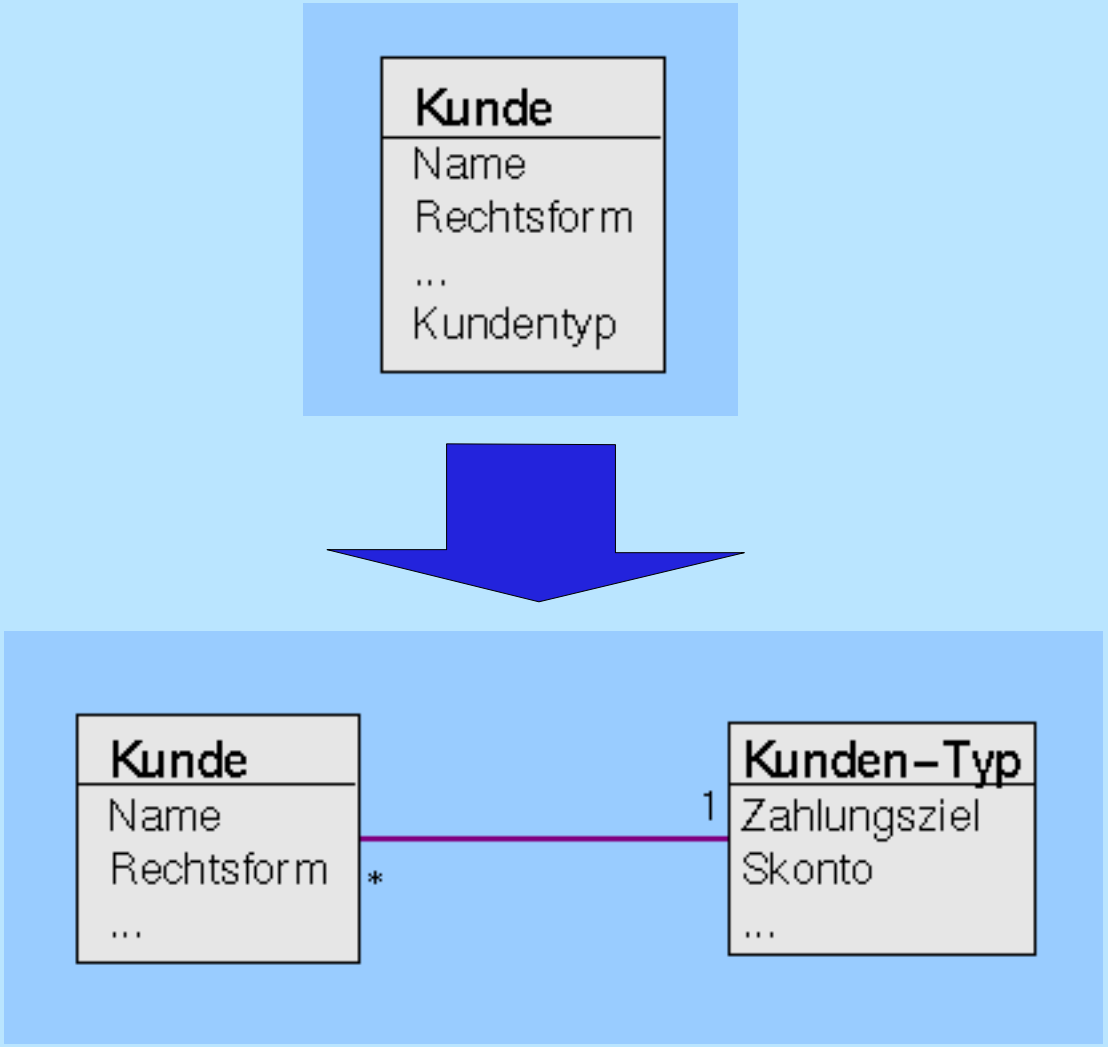
Nutzen abwägen gegen Kosten/ Nachteile !

Vorsicht: kein Metamodell

Achtung!!!

Nicht jede Flexibilisierung eines Modells/
eines Programms basiert auf dem
Einbinden von Meta-Bestandteilen!

Gegenbeispiel: Parametrisierung von Kundentypen



Gegenbeispiel: Parametrisierung von Kundentypen

Beispiel-Code (vorher):

```
...
verzug = heute – rechnung.getErstelldatum();
kunde = rechnung.getKunde();
if (kunde.kundentyp == Kunde.NORMALKUNDE) {
  if (verzug > 21) rechnung.erstelleMahnung() }
else if (kunde.kundentyp == Kunde.GROSSKUNDE) {
  if (verzug > 28) rechnung.erstelleMahnung() }
...
```

Gegenbeispiel: Parametrisierung von Kundentypen

Beispiel-Code (nachher):

```
...
verzug = heute - rechnung.getErstelldatum();
kunde = rechnung.getKunde();
if (verzug > kunde.getKundentyp().getZahlungsziel() )
{
  rechnung.erstelleMahnung()
}
...
```

Gegenbeispiel: Parametrisierung von Kundentypen

- Bei Kundentyp (und seinen Attributen) handelt es sich **nicht** um ein Stück Meta-Modell (Ebene 2), das in den Code (Ebene 1) integriert wurde.
- Hier liegt gewöhnliches Faktorisieren/ Normalisieren vor
- Dadurch können Art und Anzahl der Kundentypen code-unabhängig variiert werden

Kriterien für das Erkennen von Meta-Modell-Bestandteilen

Wann liegt eine Einbettung von Meta-Bestandteilen (Ebene 2) in die Ebene 1 vor??

- wenn die Instanzen der jeweiligen Meta-Daten Struktur oder Verarbeitungsanweisung für weitere Daten festlegen
- wenn die Instanzen der jeweiligen Meta-Daten ihrerseits instanziiert werden (wie die Parameter bei Briefschreibung)

Entgegengesetzte Problematik

- Die Ebene 1 soll normalerweise Aufschluss darüber geben, wie die Daten auf Ebene 0 interpretiert und verarbeitet werden müssen
- Wissen, das auf Ebene 1 nützlich angesiedelt wäre, liegt statt dessen auf Ebene 0
- d. h. bei dieser Gestaltung enthält das Programm/ das Modell keine (vollständige) Information darüber, wie die Daten auf Ebene 0 zu interpretieren sind

Beispiele für Meta-Daten auf Ebene 0

Mitarbeiternummer:

beginnt mit 9 => Mitarbeiter ist ausgeschieden
beginnt mit 1 => Mitarbeiter ist Angesteller
beginnt mit 2 => Mitarbeiter ist extern

Stellen 2-4 der Mitarbeiternummer beinhalten die Abteilung, der der Mitarbeiter zugeordnet ist

Beispiele für Meta-Daten auf Ebene 0

Org. Einheit => bei Endung 00 Direktion, sonst Geschäftstelle

Kunden-Nr => bei Historisierung werden die beiden ersten Stellen um 12 erhöht,
z. B. Satz Nr. 23012345 ist historisierter Vorjahressatz zu Satz Nr. 11001234

Temperatur => 999,9 bedeutet, dass die Temperatur unbekannt bzw. im jeweiligen Kontext nicht sinnvoll anwendbar ist

Gemeinsamkeiten und Konsequenzen

- Meta-Daten sind hier in den Daten selbst "versteckt"
- Damit sind sie auf der Meta-Ebene (Programm/ Modell) nicht explizit erkennbar
- um zu einem funktionsfähigen System zu gelangen, müssen sie trotzdem berücksichtigt werden,
- i. d. R. durch "kryptische" Bedingungen im Programmcode

Indizien f. Meta-Daten auf Ebene 0

- Nummernkreise
- Sprechende Schlüssel
- Abfragen auf fest codierte Nummern im Programm-Code

Daumenregel: Vermeiden !!!

Meta-Bestandteile in Programmiersprachen

Einige Programmiersprachen/ APIs beinhalten Sprachelemente mit Meta-Charakter

=> Konzepte, die dem Programmierer ein Einbinden von Meta-Bestandteilen ermöglichen

Beispiel: Smalltalk

- “alles ist ein Objekt“ - auch Meta-Objekte
- Beispiel: die Klasse Block (BlockClosure)
- Die Instanzen der Klasse Block repräsentieren keine Objekte des Gegenstandsbereiches (wie Number, Set, Person, Window, etc.)
- sondern ein Block ist ein Stück Programmcode, bestehend aus mehreren Anweisungen
- damit ist ein Block ein Objekt der Meta-Ebene!

Smalltalk Block-Objekte

- Ein Block-Objekt kann wie jedes andere Objekt einer Variablen zugewiesen werden.
- Ausführung des Codes im Block wird durch die Botschaft “value“ bewirkt

```
sumBlock := [ sum + (k * k) ].
sum = 0.
k = 2.
sum = sumBlock value.      “ergibt 4“
k = 5.
sum = sumBlock value.      “ergibt 29“
```

Smalltalk Kontrollstrukturen

Meta-Objekte vom Typ Block bewirken:

- Kontrollstrukturen als Sprachelemente verzichtbar
- statt dessen als Methoden bzw. Botschaften

```
"Beispiel Bedingte Anweisung"
x > 0 ifTrue: [x := x + 1] ifFalse: [x := 1 - x ]

"Beispiel Schleife"
x := 0.
[n > 1] whileTrue: [n := n / 2. x := x + 1]
```

Beispiel: eval

LISP: eval (...)

- bedeutet, dass die als Parameter übergebene Liste als Funktionsausführung zu interpretieren und auszuführen ist

gilt analog für JavaScript: eval (...)

- Ermöglicht, einen beliebigen String als JavaScript-Code zu interpretieren und auszuführen

eval ("a = 42 – 17") <=> a = 42 – 17

Beispiel: eval

Das gilt auch für Strings, die z. B. von der Benutzeroberfläche eingelesenen oder im Programmcode zusammengestellt wurden

```
....
string1 = this.form.input1.value;
string2 = this.form.input2.value;
eval (“funct“ + string1 + “(“ + string2 + “)“);
...
```

Beispiel: Java

- die APIs java.lang und java.lang.reflection ermöglichen Zugriff auf Meta-Objekte wie Klassen, Methoden, MethodenParameter, Felder und deren Typ, Modifier, Packages, etc.
- nur Lesezugriff, d. h. der Programmcode kann nicht mittels Reflection-Methoden zur Laufzeit verändert werden.

Fazit: Meta-Bestandteile in Programmiersprachen

- hohe Flexibilität, z. B.
  - Frei definierbare Kontrollstrukturen
  - selbst-modifizierender Code
- geringere Verständlichkeit/ Nachvollziehbarkeit
- geringere Sicherheit
  - Verlagert Prüfungen von Compile-Zeit zu Laufzeit

Zusammenfassung

- Wirklichkeit, Modell, Meta-Modell und Meta^2-Modell bilden eine Architektur mit 4 Schichten
- Es ist möglich, aber in der Regel nachteilig, sich bei der Anwendungs-Entwicklung auf eine „falsche“ Ebene zu begeben
- Insbesondere führt Programmierung auf Ebene 0 bzw. Ebene 2 zu unverständlichem und damit schwer wartbaren Code