Metamodelle

Udo Kelter

24.01.2014

Zusammenfassung dieses Lehrmoduls

Metamodelle spielen unter anderem in der modellbasierten Softwareentwicklung und bei der Spezifikation der UML eine große Rolle. Der Begriff Metamodell ähnelt den Begriffen Metadaten und Metasprachen, weist aber einige zusätzliche Komplikationen und erhebliches Potential für Mißverständnisse auf. Ziel dieses Lehrmoduls ist, die genannten Begriffe voneinander abzugrenzen und die Metamodellhierarchie der UML vorzustellen.

Vorausgesetzte Lehrmodule:

empfohlen:
- Modellgetriebene Software-Entwicklung
- Metadaten

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Einleitung

Metamodelle spielen u.a. in der modellbasierten Softwareentwicklung und bei der Spezifikation der UML eine große Rolle. Sehr häufig werden Metamodelle definiert als *Modelle von Modellen* - das ist kurz, knapp, prägnant und falsch, oder zumindest so ungenau, daß es fast immer falsch verstanden wird.

In vielen Dokumenten über die UML wird von "dem" Metamodell eines Modelltyps gesprochen - wie wir sehen werden, gibt es nicht "das eine" Metamodell für einen Modelltyp, sondern mehrere -, und es werden in lockerem Stil `ist_Instanz_von`-Beziehungen von Metamodellen (oder Metamodellelementen) nach Modellen (oder Modellelementen) gezeichnet, wobei eher vage definiert ist, was das wirklich bedeutet.

Die UML-Definition [UML-I] setzt eine Metamodell-Hierarchie ein (s. Bild 1), die auf den ersten Blick identisch zu der Metadaten-Hierarchie in Datenverwaltungssystemen (s. Bild **??** in [MD]) erscheint, bei genauer Betrachtung aber erhebliche Unterschiede aufweist. Diese versteckten Begriffsdifferenzen führen notorisch zu Kommunikationsproblemen, zumal beide Begriffswelten in vielen Fällen parallel verwendet werden müssen.

Kürzel für die Meta-Ebene	dieser Ebene zugeordnete Modelle
M3	Meta-Metamodell
M2	Metamodelle
M1	(Struktur- und Verhaltens-) Modelle
M0	modellierte Realweltentitäten

Abbildung 1: Metamodelle in der UML

Die reine Zuordnung der Metamodelle zu den Ebenen sagt nicht viel aus. Anschaulicher ist eine Beschreibung, was das "Thema", also der Gegenstandsbereich, der einzelnen Ebenen ist, s. hierzu Bild 2.

Metamodelle kann man auch als Metadaten ansehen. Metadaten

Ebene	Gegenstandsbereich (*domain of discourse*) dieser Ebene	Beispiele einer Entität / eines Exemplars
M3	Sprachen / Formalismen, mit denen man Modelltypen definieren kann	die sog. "Infrastruktur" der UML
M2	Typen von Modellen bzw. Modellelementen	ein konkreter Modelltyp, z.B. State Machines wie in der UML 2.0 definiert
M1	(Struktur- und Verhaltens-) Modelle	ein konkretes Datenmodell von Bankkontos, von Kunden, ...
M0	beliebige Realweltentitäten bzw. Daten, die Realweltausschnitte repräsentieren	ein Bankkonto, ein Kunde, eine Lieferung, ...

Abbildung 2: Gegenstandsbereiche der Ebenen in der Metamodellhierarchie der UML

sind Daten über Daten. Für das folgende benötigen wir Grundwissen über Metadaten, und fassen hierzu im nächsten Abschnitt den Inhalt von [MD] zusammen.

2 Meta-Hierarchien

2.1 Metadaten

Metadaten sind Daten über zugehörige Nutzdaten. Entscheidend ist, daß es Nutzdaten gibt und die Metadaten von einem Menschen oder Automaten benutzt werden, um die Nutzdaten verstehen oder verarbeiten zu können. Man kann mehrere Arten von Metadaten unterscheiden:

Instanzbezogene Metadaten sind individuell für jede Nutzdateneinheit. Beispielsweise sind Seitenzahl und Erscheinungsjahr instanzbezogene Metadaten über ein Buch.

Typbezogene Metadaten unterstellen, daß jede Nutzdateneinheit einen Typ hat, und machen

– direkte Angaben über den Typ und damit
– indirekte Angaben über die Nutzdateneinheit dieses Typs.

Beispiele sind die Schemata einer Datenbank oder die Dokumentation einzelner Datenbankfelder. *I.f. behandeln wir ausschließlich typbezogene Metadaten.*

Schemata sind **instantiierbare** typbezogene Metadaten: Unterstellt ist hierbei ein Datenverwaltungssystem (DVS), das mit einem Schema konfiguriert werden kann.

Das Wort "Daten" in Metadaten weist darauf hin, daß es sich hier um die Implementierung von Wissen (oder sogar Informationen, s. [DVS]) handelt. Für diese Implementierung bestehen sehr viele Freiräume, Metadaten können daher in den unterschiedlichsten Speicherungs- und Darstellungsformen auftreten. Der Begriff (Meta-) Daten läßt auch offen, ob die Daten in einer persistenten oder transienten Form vorliegen. Beide Fälle sind möglich. Das gleiche "Dokument" (bzw. das durch die Daten repräsentierte Wissen) kann in persistenter und transienter Repräsentation vorliegen. Beim Einlesen bzw. Sichern von Daten wird zwischen beiden Repräsentationen konvertiert.

2.2 Metasprachebenen in der Linguistik

Die Linguistik unterscheidet analog zu Metadatenebenen Meta*sprach*ebenen:

– Die Objektsprache bildet die Basis. Beispiel: "Der Ball ist rund."
– Die Metasprache enthält Aussagen über die Objektsprache. Beispiel: "Sätze haben meist ein Subjekt und ein Prädikat." Die Aussagen der Metasprache können alle Aspekte der Syntax und Semantik betreffen, u.a. Zeichensätze, Wortbildung, Grammatik usw.
– Die Meta-Metasprache enthält Aussagen über die Metasprache. Beispiel: "Die Metasprache enthält Aussagen über die Objektsprache."

Alle Sprachebenen benutzen die *gleiche Syntax*, die Sprachebene ist nicht syntaktisch erkennbar, sondern ergibt sich nur aus der Bedeutung der Aussage (weitere Beispiele s. [MD]).

2.3 Merkmale semantischer Ebenen

Man erkennt unschwer viele Gemeinsamkeiten von Metadaten- und Metasprachebenen:

- Eine Aussage der Metasprache betrifft die Objektsprache *als ganze* (z.B. Syntax, Grammatik, Semantik usw.), sie betrifft *nicht einzelne Aussagen* in der Objektsprache.

  Gegenstandsbereich der Metasprache ist die Objekt*sprache*, Gegenstandsbereich der Objektsprache ist die reale Welt. Beide Gegenstandsbereiche sind verschieden.

- Analog dazu machen typbezogene Metadaten Aussagen (bzw. repräsentieren Wissen) über *alle Instanzen* des zugehörigen Typs, typischerweise darüber, wie die gespeicherten Nutzdaten strukturiert und zu interpretieren sind.

  Typbezogene Metadaten sind *keine vereinfachten oder gekürzten Darstellungen* der Nutzdaten. Die Gegenstandsbereiche beider Ebenen sind verschieden.

Die Metaebene enthält also immer generelle Aussagen (oder Wissen oder Informationen) darüber, in welcher Form Aussagen (oder Wissen oder Informationen) der nächsttieferen Ebene *sprachlich oder datenmäßig repräsentiert* werden, also über die **Repräsentationsform** der nächsttieferen Ebene.

3 Nutzdaten in Metamodellhierarchien

Die Metamodelle der UML (Meta-Ebene M2) und anderer Modellierungssprachen dienen dazu, Modelle als Klasse von Dokumenten zu spezifizieren, die von Modelleditoren, -Transformatoren und anderen Modellwerkzeugen, die ggf. von verschiedenen Anbietern stammen, verarbeitet werden. D.h. *Modelle der Meta-Ebene M1 nehmen hier die Rolle der Nutzdaten ein.* Modelle der Meta-Ebene M1 entsprechen somit bzgl. ihrer Rolle der Objektsprache bzw. der Ebene der Nutzdaten in der Metadatenhierarchie.

An dieser Stelle muß vor Verwechslungen mit der Metadatenhierarchie gewarnt werden:

- In der Meta*daten*hierarchie sind die Nutzdaten Repräsentationen von Realweltentitäten.
- In der Meta*modell*hierarchie der UML liegen die Nutzdaten dagegen in der Ebene M1.

  Realweltentitäten sind zwar in der UML-Metamodellhierarchie durch die Ebene M0 repräsentiert. Die Ebene M0 dient aber nur dazu, erläutern zu können, was Modelle bedeuten sollen und wie eine Implementierung eines Systems aussehen sollte, das durch ein Modell spezifiziert wird[1]. Realweltentitäten bzw. Daten der Ebene M0 sind also in der UML-Metamodellhierarchie *keine Nutzdaten*.

4 Metamodelle

4.1 Besonderheiten von Metamodellen

Es liegt nahe, ausgehend von der Nutzdatenebene M1 die relativ übersichtlichen Prinzipien zur Bildung von Metaebenen zu nutzen, um weitere Ebenen der Metamodellhierarchie zu bilden. Metamodelle wären dann Modelle, die M1-Modelle modellieren. Bei der Frage, welche Modelle als Metamodelle infrage kommen, stößt man sofort auf einige Probleme, die Metamodelle in mehrerer Hinsicht deutlich komplizierter machen als Metadaten:

1. Metadaten werden nur im Kontext eines konkreten Datenverwaltungssystems betrachtet, z.B. im Laufzeitsystem einer bestimmten Programmiersprache, im Laufzeitkern eines DBMS, Dateisystems o.ä.[2] Die Metadaten müssen genau wie die Nutzdaten den techno-

[1] Dies ist übrigens nur bei wenigen Modelltypen sinnvoll möglich, z.B. bei Klassendiagrammen. Bei vielen Modelltypen ist es eher der Phantasie des Betrachters überlassen zu definieren, was alles eine "Implementierung" des Modells sein könnte.

[2] Natürlich werden auch außerhalb des DVS Repräsentationen der Metadaten benutzt, z.B. eine Repräsentation eines Datenbankschemas in Form von Laufzeitobjekten. Diese Repräsentationen werden aber als thematisch irrelevant angesehen.

logischen Bedingungen dieses Systems genügen. Alle Implementierungsentscheidungen sind bereits gefallen und in den vorliegenden Strukturen manifestiert.

Während (Meta-) Daten also nur in einem festen technologischen Kontext betrachtet werden, existieren Modelle in unterschiedlichen Entwicklungsstufen eines Entwicklungsprozesses.

2. Für persistente Daten betrachtet man nur persistente Metadaten, analog gilt dies für transiente Daten und Metadaten. M.a.W. ist das Persistenzmerkmal auf beiden Ebenen identisch. In den frühen Entwicklungsphasen abstrahieren Modelle dagegen von der Frage, ob die modellierten Daten persistent oder transient existieren.

3. Modelle modellieren nicht nur Daten, sondern auch Verhalten von Systemen. Die Frage stellt sich also, ob Metamodelle (analog zu normalen Modellen) auch das Verhalten von Modellen modellieren sollten bzw. was überhaupt das Verhalten von Modellen ist.

4.2 Modelle in den Entwicklungsphasen

Wir untersuchen zunächst das Problem, daß Modelle, damit auch Metamodelle, in mehreren Entwicklungsphasen auftreten. Wir betrachten hier zunächst nur Datenmodelle.

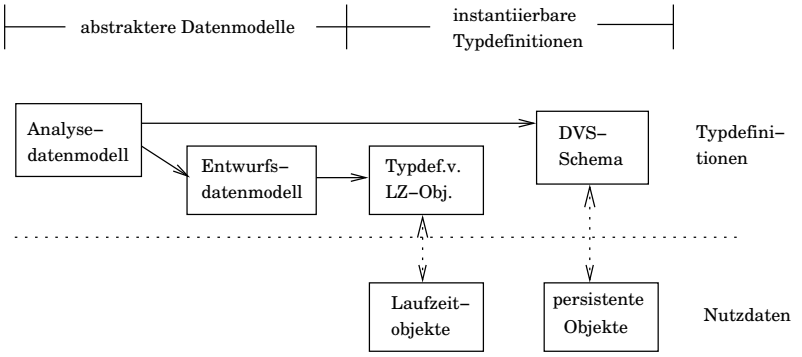

Abbildung 3: Modelle und Typdefinitionen in verschiedenen Phasen

Metamodelle sind insofern ganz normale Modelle, als auch sie ein Mittel zur Entwicklung von Applikationen - hier Modelleditoren, Transformatoren usw. - sind; Nutzdaten sind hier die Modelle. Am Anfang der Entwicklung einer Applikation stehen immer Analysemodelle der Nutzdaten. Die Analysemodelle werden in einem oder mehreren Schritten zu Entwurfsmodellen, Datentypdefinitionen in Programmen oder Schemata von DVS verfeinert (s. Bild 3). Die Typdefinitionen in Programmen bzw. Schemata sind die präzisesten und detailliertesten Entwicklungsdokumente; sie können nach einer Übersetzung bzw. Installation von einem Laufzeitsystem, das transiente oder persistente Instanzen verwalten kann, instantiiert werden.

In Bild 3 ist bewußt offengelassen, ob es sich bei den Nutzdaten um Modelle oder "normale" Nutzdaten handelt; die Strukturen sind in beiden Fallen völlig gleich.

Offensichtlich entstehen im üblichen Phasenmodell der Systementwicklung im Laufe der Entwicklung von Werkzeugen, die Modelle verarbeiten, mehrere Metamodelle, und die eher abstrakt oder eher implementierungsnah bzw. technologiespezifisch sein können. Am Ende der Systementwicklung existieren also *für einen gegebenen Dokument- bzw. Modelltyp i.a. mehrere (Meta-) Modelle*, die abstrakt oder technologiespezifisch sein können.

4.3 Semantische Ebenen vs. Entwicklungsphasen

Wir stehen nun vor der Frage, inwieweit die abstrakteren oder implementierungsnahen Metamodelle eines Modelltyps Modelle voneinander sind und ob diese ggf. ebenfalls in Meta-Ebenen berücksichtigt werden müssen. Hierzu müssen wir zunächst den Begriff Modell genauer definieren.

Modelle. Üblicherweise definiert man ein **Modell** eines Systems O, das auch als das *Original* bezeichnet wird, als ein einfacheres System M, das interessierende Merkmale von O korrekt wiedergibt. Welche Merkmale von O interessieren, hängt ausschließlich von Anwendungskontext ab. M ist kleiner, einfacher, abstrakter, billiger oder schneller

verfügbar als O, bietet jedenfalls gegenüber O einen anwendungsspezifischen Vorteil.

Wenn ein Modell zu einem bereits vorhandenen Original gebildet wird, ist es **deskriptiv** und dient oft dazu, das komplizierte Original zu erklären oder zu dokumentieren. Wenn die Entstehungsreihenfolge umgekehrt ist und wenn das Modell bei der Konstruktion des Originals als "Bauanleitung" benutzt wird, spricht man von einem **präskriptiven** Modell. Die Modelle, die in den frühen Phasen eines Softwareentwicklungsprojekts benutzt werden sind meistens präskriptiv, speziell wenn ein völlig neues System realisiert wird.

Unabhängig von der Reihenfolge der Entstehung kann man, nachdem sowohl Modell und Original vorhanden sind, von einer `ist_Modell_von`-Beziehung vom Modell zum Original sprechen[3].

`ist_Modell_von`-Beziehungen zwischen Entwicklungsdokumenten. Gemäß den vorstehenden Definitionen ist

- ein Analysedatenmodell ein (präskriptives) Modell eines Entwurfsdatenmodells, denn das Analysedatenmodell abstrahiert von diversen Details im Entwurfsdatenmodell,
- ein Entwurfsdatenmodell ein Modell der resultierenden Typdefinitionen von Laufzeitobjekten, die im Quellcode des modellierten Systems notiert sind,
- der Quellcode ein Modell des bei der Übersetzung entstehenden Maschinen- oder Bytecodes bzw. des letztlich entstehenden lauffähigen Systems.

M.a.W. liegt bei jedem Schritt unseres hier unterstellten Entwicklungsprozesses eine `ist_Modell_von`-Beziehung vom abstrakteren zum konkreteren Modell vor, sofern wir den allgemeinen Modellbegriff verwenden.

[3] In der umgekehrten Leserichtung wird diese Beziehung oft als `ist_Instanz_von`-Beziehung bezeichnet.

Konsequenzen. Wenn wir jedes beliebige "Modell eines Modells" als Metamodell bezeichnen, dann folgt aus den vorstehenden Beobachtungen:

1. Ein Analysedatenmodell ist ein Modell eines Modells der Typdefinitionen ist, somit also ein *Metamodell* der Typdefinitionen.
2. Wenn wir das Entwurfsdatenmodell in unserem Entwicklungsprozeß weglassen, ist das Analysedatenmodell nur noch ein *Modell* der Typdefinitionen.
3. Wenn wir den Entwurfsvorgang in zwei Schritte (Grob- und Feinentwurf) aufteilen, wird unser Entwurfsdatenmodell zu einem *Meta-Metamodell* der Typdefinitionen.

Die vorstehenden Beispiele zeigen, daß man *schrittweise Verfeinerungen bzw. Entwicklungsphasen eines Systems nicht sinnvoll als Metamodell-Ebenen auffassen kann.* Die Zahl der Entwicklungsphasen hängt nur vom Entwicklungsprozeß ab und ist insofern völlig willkürlich. Ferner ist diese Anzahl i.d.R. für transiente und persistente Darstellungen derselben Daten bzw. Modelle unterschiedlich.

Alle Modelle, die schrittweisen Verfeinerungen bzw. Entwicklungsphasen eines Systems entsprechen, *modellieren das gleiche System* und liegen daher *semantisch auf der gleichen Ebene*!

Hieraus folgt, daß man bei der Bildung von Metamodell-Ebenen nicht den ganz allgemeinen Modellbegriff benutzen kann, sondern nur einen speziellen, stark eingeschränkten.

4.4 Metamodelle

Nachdem wir die Frage, ob unterschiedlich abstrakte bzw. technologiespezifische Metamodelle zu eigenen Meta-Ebenen führen, mit einem klaren Nein beantwortet haben, bleibt die Hauptaufgabe von Metamodellen darin, Informationen darüber zu liefern, wie Modelle des zugehörigen Typs strukturiert sind und was sie bedeuten. Insb. muß ein Metamodell definieren, welche Modellelemente bei Modellen eines bestimmten Typs auftreten.

Hierbei muß immer eine Repräsentationsform für Modelle festgelegt werden. Üblicherweise wird als Repräsentationsform ein "abstrakter" Graph mit typisierten Knoten und Kanten festgelegt. In vielen Spezifikationen von Modelltypen werden nur noch die Typen der Knoten und Kanten dieses Graphen definiert, daß ein Modell überhaupt als ein Graph repräsentiert wird, wird stillschweigend vorausgesetzt.

Metamodelle sind also *Modelle der Repräsentationsform* von Modellen eines bestimmten Modelltyps, sie sind keine Modelle *des "konzeptuellen Gehalts" von Modellen*, also keine Modelle, die interessierende Eigenschaften eines konkreten modellierten Systems wiedergeben.

Eine wichtige Erkenntnis aus dem vorigen Abschnitt ist, daß die "frühphasigen" Metamodelle die wichtigsten sind: in ihnen sind schon alle inhaltlich relevanten Aspekte dargestellt.

In den konkreteren Metamodellen werden nur noch technische Details der Repräsentationsform ergänzt. Die spätphasigen Metamodelle bzw. zugehörige Modellrepräsentationen beinhalten Details und Implementierungsentscheidungen, die aufgrund der technologischen Plattform, in der die Modelle repräsentiert werden, irgendwie fixiert werden müssen, inhaltlich aber irrelevant sind. Wenn man Modelle maschinell verarbeiten möchte, kommt man aber an spätphasigen, technologiespezifischen Modellrepräsentationen nicht vorbei.

4.5 Verhaltens-Metamodelle

Aufgrund der vorstehenden Überlegungen ist offensichtlich, daß Metamodelle nur Daten modellieren, also nur Datenmodelle sein können, keine Verhaltensmodelle. Modelle von Nutzdaten, also Modelle auf Ebene M1, haben für sich genommen gar kein Verhalten, Verhalten können höchstens Modell-Editoren, -Transformatoren usw. aufweisen. Deren Funktionalität wird bei der Definition der UML oder anderer Modellierungssprachen bewußt nicht definiert, sondern soll von Werkzeuganbietern gestaltet werden.

4.6 Varianten von Modellierungssprachen

Im vorstehenden Abschnitt haben wir gezeigt, daß zu einem einzigen Modelltyp i.a. mehrere, mehr oder weniger implementierungsnahe Metamodelle existieren. Unterschiedliche Metamodelle entstehen auch aus einem anderen Grund: Fallweise soll nur eine Teilmenge der Modellierungssprache unterstützt werden, z.B. um verschiedene Konformitätsstufen zu definieren oder weil bestimmte Modelltransformationen mit bestimmten Modellelementen nicht funktionieren. Gegenüber dem ursprünglichen Metamodell können bei solchen Varianten z.B. einzelne Modellelemente entfallen, oder es gelten schärfere Integritätsbedingungen (z.B. als OCL-Constraints dargestellt).

Bei domänenspezifischen Sprachen kommen umgekehrt ggf. neue Modellelemente hinzu.

Derartige Varianten von Metamodellen entsprechen also einen *anderen Sprachumfang* und haben nichts mit der Phasenbezogenheit von Metamodellen zu tun. Jeder Sprachvariante benötigt eigene abstrakte und implementierungsnahe Metamodelle.

5 Modelle in Entwicklungsphasen – Betrachtung über zwei Ebenen hinweg

Wir hatten oben argumentiert, daß zu *einem* bestimmten Modelltyp auf Ebene M1, z.B. Zustandsdiagrammen, *mehrere Metamodelle* vorhanden sein werden, die durch die Entwicklungsphasen der Werkzeuge für diesen Modelltyp motiviert sind.

Angenommen, wir entwickeln eine Applikation, die irgendwelche M0-Nutzdaten verarbeiten soll. In diesem Entwicklungsprozeß werden natürlich mehrere M1-Modelle unterschiedlichen Typs benutzt werden, wie in Bild 3 dargestellt. Diese Modelle müssen als Dokumente irgendwie verarbeitet werden. Wir unterstellen hier um des Beispiels willen, daß alle Dokumente von Hand editiert und geprüft werden (und nicht durch Übersetzer erzeugt werden). M.a.W. brauchen wir für jeden

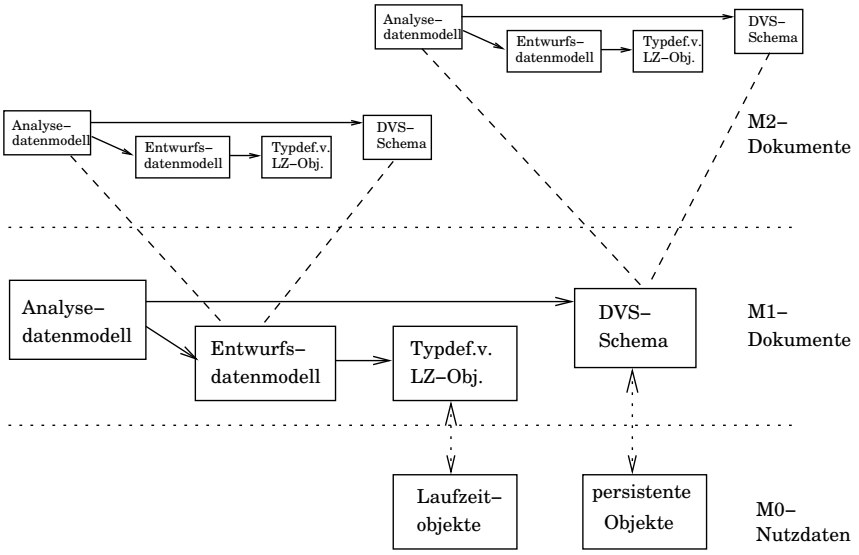

Abbildung 4: Modelle und Typdefinitionen in verschiedenen Phasen

Modelltyp eigene Editoren und andere Werkzeuge, ferner persistente Darstellungen, und dafür Metamodelle.

Alle M1-Modelle modellieren das gleiche System. Dies kann zu der falschen Schlußfolgerung verleiten, daß man alle M1-Modelltypen mit einem einzigen M2-Metamodell spezifizieren könnte. Dies trifft nicht zu, stattdessen braucht man *grundsätzlich für jeden M1-Modelltyp eigene Metamodelle!* In Bild 4 sind als Beispiel für 2 M1-Modelltypen die zugehörigen M2-Metamodelle eingezeichnet. Daß die M2-Metamodelle verschiedener M1-Modelltypen verschieden sein müssen, ergibt sich aus folgenden Beobachtungen:

– In den M1-Modelltypen treten inhaltlich signifikant andere Begriffe auf. Z.B. treten in ER-Modellen und einem relationalen DB-Schema ganz andere Dokument- (oder Modell-) Elemente auf, dementsprechend definieren die zugehörigen M2-Metamodelle andere Typen von Modellelementen.

– In den Metamodellen für die implementierungsnahen M1-Modelle kommen Begrifflichkeiten der jeweiligen Implementierungstechnologie vor, die in implementierungsfernen M1-Modellen gerade fehlen.

Die unterschiedlichen Inhalte in den M1-Modellen verschiedener Entwicklungsphasen können nur in Ausnahmefällen durch Transformatoren ineinander überführt werden und von daher als "äquivalent" angesehen werden; i.a. ist hierzu manuelle Arbeit zu leisten. Dies gilt auch dann, wenn die M1-Modelle "syntaktisch" ziemlich gleich aussehen, z.B. alle Modelle Varianten von Klassendiagrammen mit bestimmten Einschränkungen sind. Daher können fallweise die M2-Metamodelle verschiedener M1-Modelltypen *sehr ähnlich aussehen*, dies sollte man aber als Zufall ansehen.

6 Manifestation und Repräsentationen von Typdefinitionen

In Bild 3 sind nicht unzufällig zwei Ebenen eingezeichnet. Ohne daß es in diesem Bild explizit erwähnt ist, stellt dies den allgemeinen Zusammenhang zwischen Daten und Konzepten auf zwei benachbarten Meta-Ebenen dar. Diese Zusammenhänge werden sehr leicht mißverstanden und sollen daher hier betont werden.

Die untere Ebene weist persistente und transiente Daten derjenigen Ebene, die als Nutzdatenebene betrachtet wird, aus. In diesen Daten "manifestieren" sich die Typdefinitionen dieser Daten insofern, als die Daten gemäß den Typdefinitionen technisch und logisch strukturiert sind. Die Typdefinitionen als solche sind indes auf dieser Ebene normalerweise nicht explizit in irgendeiner Form repräsentiert und auch nicht aus den Nutzdaten rekonstruierbar!

Dies ist besonders offensichtlich, wenn man konventionelle Programmiersprachen wie z.B. C, und strikt getrennte Entwicklungs- und operative Produktivumgebungen auf verschiedenen Rechnern unterstellt. Ein Quelltext z.B. in C würde auf dem Entwicklungsrechner compiliert werden, das ausführbare binäre Programm würde man an 100 Anwender verschicken, die den Quelltext nicht kennen und mit

einer Kopie des binären Programms auf ihrem operativen Rechner arbeiten. Daten der Ebene M0 existieren nur auf den operativen Rechnern der Anwender, die Quelltexte und andere Daten der Ebene M1 nur auf den Entwicklungsrechnern.

Bei objektorientierten Programmiersprachen, die Reflexion unterstützen, und generischen Programmiertechniken, wie sie z.B. in EMF/Ecore verwendet werden, wird die strikte Trennung zwischen Entwicklungs- und operative Produktivumgebung weitgehend aufgehoben.

Reflexion. Die eigentlich klare Trennung der semantischen Ebenen wird durch Reflexion in Java und vergleichbare Konzepte in anderen Programmiersprachen oder Datenverwaltungssystemen vernebelt: Man kann hier zu den vorhandenen Objekten Daten erzeugen, die deren programmiersprachlichen Typ beschreiben. In Java kann z.B. mit der Operation `getClass()` zu einem beliebigen Objekt `X` ein Objekt vom Typ `Class` erzeugt werden, das den Typ von `X` beschreibt. Nach Ausführung von `Class T = X.getClass();` hat man also zwei Objekte `X` und `T`, von denen eines Nutzdaten darstellt und das andere Metadaten zu diesen Nutzdaten, also eine Semantikebene höher liegt. Im Gegensatz zu einer Sprache wie C ist hier auch in der Produktivumgebung eine komplette Repräsentation der Datentypen vorhanden. Die Informationen über die Typen der Nutzdaten werden bei der Übersetzung der Quelltexte generiert und passend codiert in den Bindemodulen (z.B. class-Dateien) oder ausführbaren Programmen gespeichert.

Das vorstehende Beispiel der Objekte `X` und `T` zeigt ferner folgendes: Wenn sich zwei Objekte im gleichen Hauptspeicher (oder in der gleichen Datenbank) befinden, *impliziert dies keineswegs, daß sie auf der gleichen semantischen Ebene liegen.* Die semantische Ebene eines Objekts hängt nur von seiner inhaltlichen Bedeutung ab.

Interpreter-Architekturen. Interpreter-Architekturen sind ein verbreiteter Architekturstil, der insb. bei adaptiver bzw. variantenreicher Software zum Einsatz kommt. Hierbei werden mehr oder weniger große Teile der Applikationslogik nicht mehr “hart” im Quelltext im-

plementiert, sondern in einer speziellen Steuersprache, die man abhängig vom Kontext als domänenspezifische Sprache (DSL), “Ressource”, Steuerdaten o.ä. bezeichnet. Der Interpreter für diese Steuerdaten wird oft als *virtuelle Maschine* bezeichnet.

Modelle (bzw. Metamodelle) werden in manchen Applikationen unmittelbar als Steuerdaten verwendet, Beispiele folgen im nächsten Abschnitt.

7 Mehrebenen-Applikationen und die Zuordnung von Applikationen zu Metaebenen

Auf allen Ebenen in Bild 4 kommen Daten (bzw. Dokumente) vor, und zwar nicht nur die Modelle, sondern speziell auf Ebene M1 auch Quellprogramme, Datenbankschemata und viele weitere Entwicklungsdokumente. Diese Daten bzw. Modelle werden natürlich von Applikationen gelesen und erzeugt.

Normalerweise arbeiten Applikationen nur auf einer Ebene, z.B. könnte ein in C geschriebenes Buchungssystem auf Ebene M0 Buchungsdaten erfassen und auflisten. Ebenfalls nur auf einer Ebene, nämlich M1, arbeitet ein (komplett in C geschriebener) Transformator, der z.B. Klassendiagramme in entsprechende Programmrümpfe übersetzt, oder ein Modelltransformator, der Modelle transformiert.

Es gibt auch Applikationen, die gleichzeitig Daten verarbeiten, die zu *verschiedenen Ebenen* gehören. Beispiele:

1. Das einfachste Beispiel ist ein Datenbankmanagementsystem (DBMS). Wenn man es auf einem Rechner installiert hat, dann muß man zunächst Schemata eingeben; diese werden intern gespeichert. Danach kann man Nutzdaten entsprechend den Schemata eingeben. Das DBMS arbeiten also gleichzeitig mit Daten der Ebenen M0 und M1.
2. Ähnlich wie ein DBMS arbeiten XML-Prozessoren: diese lesen eine XML-Datei mit Nutzdaten und ggf. eine weitere Datei mit einer Dokumenttypdefinition oder einem XML-Schema ein. Im Unterschied

zu einem DBMS werden die Schemata jedesmal neu eingelesen, dies ist aber unerheblich. Die XML-Prozessoren arbeiten also auf den Ebenen M0 und M1.

3. "Generische" Modelleditoren und andere Werkzeuge verarbeiten Modelle unterschiedlicher Typen. Beispielsweise sind die meisten UML-Diagrammtypen Graphen, die sich nur in Details unterscheiden. Die generischen Modelleditoren realisieren daher viele Funktionen, die für alle Modelltypen einheitlich sind. Die Besonderheiten, die bei einem konkreten Diagrammtyp zu beachten sind, werden durch "Steuerdaten" vorgeben.

   Zentraler Teil der Steuerdaten ist ein Metamodell des konkreten Diagrammtyps, deswegen werden sie auch als Meta-CASE (-Werkzeuge) bezeichnet. Hinzu kommen i.d.R. Angaben zum Layout, z.B. ob Graphknoten als Kästen oder Ovale darzustellen sind. Bei jeder Ausführung liest der generische Modelleditor also neben dem Modell das zugehörige Metamodell und die weiteren Steuerdaten[4] ein und erzeugt am Ende einer Sitzung ggf. ein neues (gespeichertes) Modell. Die generischen Editoren und andere generische Werkzeuge arbeiten also gleichzeitig mit Daten der Ebenen M1 und M2. Die M2-Daten werden hierbei im Gegensatz zu den M1-Daten nur gelesen und nicht verändert.

Die diversen üblichen Ebenenzuordnungen ("ein Modelltransformator arbeitet auf Ebene M1", "generische Modelleditoren arbeiten auf den Ebenen M1 und M2") sind zwar anschaulich, aber als allgemeine Behauptung falsch! *Typischerweise* verarbeiten diese Systeme Daten, die man den genannten Ebenen zuordnet. Niemand ist aber gehindert, diese Systeme zur Verarbeitung von Daten einzusetzen, die anderen Ebenen zuzuordnen sind. Den oben erwähnten generischen Grapheditor, den wir eigentlich zum Editieren von UML-Modellen einsetzen wollten, können wir genausogut einsetzen, um

[4] Die weiteren Steuerdaten werden übrigens oft auch als Metamodelle bezeichnet, weil sie ja "neben" den anderen Metamodellen stehen. Dies ist aber begrifflich falsch und irreführend, weil sie kein Original abbilden, also eine grundsätzliche Bedingung für den Modellbegriff nicht erfüllen.

– einen Familienstammbaum zu erstellen, das sind offenbar Nutzdaten der Ebene M0, oder
– die Metamodelle für die UML-Modelle zu erstellen, also offenbar für Nutzdaten der Ebene M2.

Wir kommen zur überraschenden Erkenntnis, daß (Ein- oder Mehrebenen-) *Applikationen per se überhaupt keiner Ebene fest zugeordnet sind*. Auf welcher Ebene sie im Einzelfall arbeiten, hängt nur von den Eingabedaten ab. Bei Mehrebenen-Applikationen bezeichnet man die Daten der unteren Ebene als Nutzdaten; diese können erzeugt, verändert und gelöscht werden. Die Daten der nächsthöheren Metaebene werden i.d.R. nur gelesen und wirken praktisch als Steuerungs- bzw. Konfigurationsparameter.

Literatur

[UML-I] Unified Modeling Language: Infrastructure, Version 2.4.1; OMG, Doc. formal/2011-08-05; 2011

[UML-S] Unified Modeling Language: Superstructure, Version 2.4.1; OMG, Doc. formal/2011-08-06; 2011

[MD] Kelter, U.: Lehrmodul “Metadaten”; 2008

[DVS] Kelter, U.: Lehrmodul “Datenverwaltungssysteme”; 2005