Metamodellierung als Ansatz zur Lösung der Modellaustauschproblematik im Umfeld objektorientierter Modellierungssprachen

Der OMG-Standard *eXtensible Markup Language Metadata Interchange* (XMI)

Mario C. Jeckle*

Zusammenfassung

With the unanimous voting of the OMG in September 1998 the Unified Modeling Language (UML) became an international industry standard. In the meantime the language is widely accepted and applied successful in industrial and research projects. But the UML is not a language in the classical sense, it is only a graphical notation which is not supplemented by a textual notation. Such a textual format is actually needed to exchange UML based models between heterogeneous CASE and modeling tools. To bridge the boundaries of tools in order to enable open transfer of models is the aim of the XML Metadata Interchange Format - in short XMI - which is an upcoming OMG standard.

1 Einführung

Die Komplexität moderner Systeme erfordert mannigfaltige Implementierungssprachen innerhalb vielfältigster Entwicklungsumgebungen auf gemischten Plattformen einzusetzen und im Projektablauf zu koordinieren.
Diese Heterogenitäten sind nicht ohne Produktivitätsverlust behebbar. Als Ausweg erlegt man sich oftmals selbst gewählte Beschränkungungen auf. Dies äußert sich in der Fokussierung auf eine Implementierungssprache oder Plattform, geht letztlich aber immer mit einem Verlust an Flexibilität im Entwicklungsprozeß, und auch des entstehenden Gesamtsystems, einher (die Konzentration auf eine weit verbreitete Plattform eines Herstellers und die resultierende Abhängigkeit sei hier nur als bekanntes Beispiel aus der Praxis angeführt).

Ziel der Entwicklung der UML war die Bewältigung der, durch den Methodenstreit der 80er Jahre eindrucksvoll verkörperten, Heterogenität im Umfeld der objektorientierten Modellierungssprachen.

*DaimlerChrysler Research Ulm

Durch die Vereinigung einiger Vorgängernotationen (im wesentlichen Boochs Wolken ([Boo94]), der *Object Modeling Technique* (Abk. OMT) nach Rumbaugh et al. ([RBP+91]) und Jacobsons *Object Oriented Software Engineering*-Methode (Abk. OOSE) ([Jac94]) — besser bekannt als *Use-Case*-Ansatz — ist eine Darstellungsmöglichkeit verschiedener Problemaspekte objektorientierter Systeme (z.B. statische Klassenstrukturen, Interaktion und Verteilung) entstanden. Jedoch ist die UML momentan keine Sprache, die sich zur Beschreibung aller Problembereiche gleichermaßen eignet; die Darstellung harter Realzeitbedingungen und Workflowmodellierung seien hier nur als zwei der bekannteren Beispiele angeführt. Wohl auch deshalb ist es der UML bisher (noch) nicht gelungen andere Modellierungs- und Spezifikationssprachen vollständig zu ersetzen.
Häufig finden in konkreten Projekten mehrere Werkzeuge und Notationen, je nach Problemstellung, Anwendung. Zwar bildet die UML einen geschlossenen de facto Standard, jedoch ist dieser bisher nicht von allen Werkzeugherstellern vollständig umgesetzt worden.

Sicher existieren über objektorientierte Systementwicklungsprozesse hinaus auch andere, beispielsweise die klassischen strukturierten Methoden. Jedoch die Intention des im folgenden vorgestellten Formats ist (zunächst) „lediglich“ die Behebung der Heterogenität im Bereich objektorientierter Modellierungssprachen und deren Werkzeuge.

Der vorliegende Beitrag soll die theoretischen Hintergründe des neuen methodenneutralen Transferformates beleuchten. Besonderer Wert wird auf die Unbefangenheit in Hinsicht auf bestehende Implementierungen gelegt. Deshalb wird auf die Vorstellung und Besprechung konkreter Werkzeuge verzichtet.

Ferner definiert die UML, die ihre Wurzeln in der (versuchten) Entwicklung einer *Unified Method* hat, bis heute keinen Entwicklungsprozeß. Nichtsdestoweniger ergeben sich aus dem Einsatz beliebiger „zeitgemäßer“ Entwicklungsprozesse Auswirkungen auf den Informationsfluß im Projekt. Als zeitgemäß soll hierbei der Aspekt *inkrementeller* und *iterativer* Entwicklungszyklen gegenüber traditionelleren, auf dem Wasserfallmodell beruhenden, Ansätzen angesehen werden. Diese Entwicklungsprozesse bedingen das mehrfache Durchlaufen gleicher Entwicklungsphasen, wobei Ergebnisse aus vorangegangenen Iterationsläufen in die aktuelle Phase übernommen werden (siehe [HJK+99]).

Aufgrund der Festlegung existierender CASE-Tools auf bestimmte Entwufsphasen, ist ein Modellaustausch zwischen den Iterationsphasen oftmals gleichbedeutend mit der werkzeuggrenzenüberschreitenden Übernahme des entstandenen Designs aus vorangegangenen Entwicklungsphasen.

Zusammenfassend läßt sich festhalten: Den geschilderten Heterogenitätsszenarien ist die gewünschte zeitnahe Informationsweitergabe, konkret die Übermittlung der entstehenden Modelle, innerhalb des Projekts gemein. Als Stolpersteine in der Praxis offenbaren sich der gleichzeitige und gemeinsame Einsatz verschiedener Modellierungssprachen und heterogener Werkzeuge im Systementwicklungsprozeß.

Offenkundig wäre eine homogene Landschaft aus genau einem mächtigen Werkzeug, welches sich für alle Entwicklungsstadien, von den frühen Analysen bis hin zur Implementierung, gleichermaßen eignet, auf Basis einer leistungsfähigen lingua franca als Notation, die im Wortsinne die „besten“ (etwa intuitivsten, ausdrucksstärksten,...) Ansätze vermengt, wünschenswert.
Doch steckt in der diskutierten, oft als Hemmschuh verstandenen, Heterogenität auch ein mitunter wettbewerbsdifferenzierender Freiheitsgrad, der die Kombination verschiedenster Werkzeuge und Sprachen im Projekt erlaubt.

Das OMG *XML Metadata Interchange Format* (Abk. XMI) hat sich die Überbrückung der beiden dargestellten Heterogenitätsdimensionen, nämlich Werkzeug- und objektorientierte Modellierungssprachenlandschaft, zum Ziel gesetzt.

2 Standards

Der von der OMG akzeptierte Vorschlag eines Transferformates für objektorientierte Modelle, das *Stream-based Model Interchange Format* (Abk. SMIF) setzt auf der *eXtensible Markup Language* (Abk. XML) (siehe [Wor98]) des W3-Konsortiums (W3C) auf. Konsequenterweise wird die Benennung entsprechend einer der im XML-Umfeld üblichen Konventionen fortgeführt, und das Austauschformat als neues Mitglied der XML-Sprachfamilie mit eXtensible Markup Language Metadata Interchange — kurz XMI — bezeichnet. Eine andere Benennungsvariante wäre die Betonung der Markup-up-Komponente im Namen und die drei-buchstabige Abkürzung auf „ML“ enden zu lassen; etwa IML für Interchange Markup Language.

Während SMIF im Aufruf zur Normungsvorschlagseinreichung (engl. Request For Proposals) die Problemstellung bezeichnet, ist XMI die letztlich verabschiedete Lösungstechnologie. SMIF und XMI sind im Prinzip synonym; im folgenden wird jedoch der bekanntere Begriff XMI bevorzugt.

Durch die Verwendung von XML, einer Sprache des W3C, zur Festlegung eines neuen OMG-Standards werden die beiden Welten, einerseits die Standards des Web und andererseits die der Objekttechnologie, „verheiratet“.
Zum Begriff „Standard“ ist noch anzumerken, daß es sich bei allen OMG-Standards, wie UML, dem später noch zu erwähnenden MOF und damit auch XMI, um internationale Industriestandards handelt, also um Normen jenseits der staatlichen Normungsorgane — wie DIN oder ANSI, mit den zugehörigen internationalen Gremien, die in der International Standardization Organization (ISO) organisiert sind. [1]

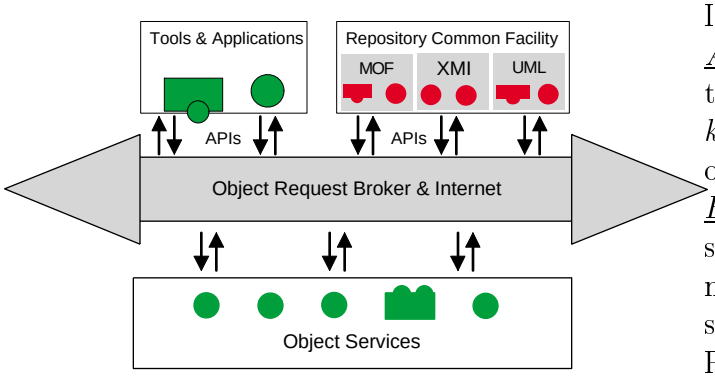

Abbildung 1: Die Object Management Architecture

In Abbildung 1 ist die *Object Management Architecture* (Abk. OMA) dargestellt. Zentrales Element ist der *Object Request Broker*, die Vermittlungsinstanz der objektorientierten Middleware *Common Request Broker Architecture* (Abk. CORBA). Dieser programmiersprachen- und plattformunabhängige Infobus verbindet die bereitgestellten Basisdienste (*Object Services*), wie Persistenz- und Transaktionsmanagement, mit den verschiedenen Applikationen. Alle durch die OMG verabschiedeten Industriestandards fügen sich in diese Architektur ein (siehe [Sol97]).

Darüber hinaus wird eine Zusammenfassung weiterer, auf der OMA aufsetzender Standards, als *Repository Common Facility* bezeichnet. Der in der Praxis wohl bekannteste dieser Standards ist die schon erwähnte UML. Sie definiert, auf Basis eines gemeinsamen — ebenfalls in UML modellierten — Metamodells die verschiedenen UML-Diagrammsprachen (wie Klassendiagramm, Aktivitätsdiagramm, Sequenzdiagramm, Use-Cases,...) (vgl. hierzu: [OHI+99, p. 1-3]).

Im wesentlichen stellt das Metamodell, als Teil der abstrakten Semantikbescheibung zur UML, die Beziehungen zwischen den Modellelementen (wiederum in UML) dar. Mithin sind die erlaubten Konstrukte der Modellierungssprache in der Modellierungssprache selbst beschrieben. Insgesamt betrachtet ist die Semantikfestlegung, bestehend aus Metamodell und well-formedness rules, die in einer formalen Sprache der *Object Constraint Language* (Abk. OCL) beschrieben werden, nur für Werkzeughersteller und Methodiker interessant. Während Erstere diese abstrakte Syntaxbeschreibung quasi als Pflichtenheft übernehmen, ist sie für Methodenentwickler Ausgangspunkt weiterer Überlegungen.

3 Metamodellierung

Sicherlich stellt sich dem aufmerksamen Leser die Frage: „wenn die UML in UML modelliert wird, um die Korrektheit und Widerspruchsfreiheit zu garantieren, wodurch wird die Korrektheit für das Metamodell, welches wiederum ein UML-Modell ist, sichergestellt?“.

Die Antwort ist so komplex wie einfach gleichermaßen: Durch ein weiteres (UML-)Modell. Dieses „Meta-Metamodell“ genannte, und als Standard *Meta Object Facility* (Abk. MOF) normierte (siehe [CII+99]), Sammlung von Klassendiagrammen modelliert die Syntax der Metamodelle.
Das konkrete Modell wird also durch ein Modell modelliert welches wieder durch ein Modell modelliert ist....

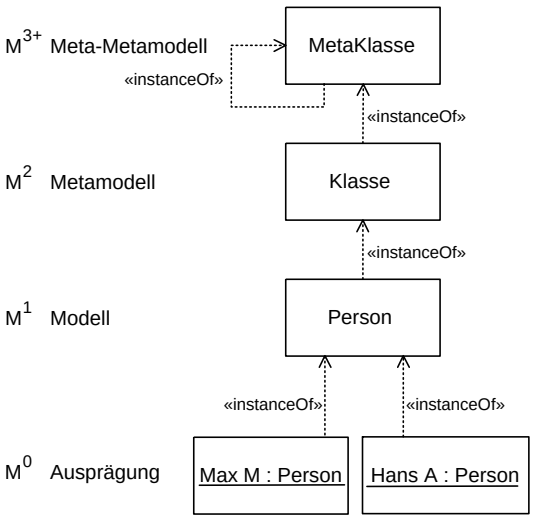

Abbildung 2: OMG Metamodell-Architektur

Offensichtlich läßt sich Metaebenenbildung mühelos fortsetzen, ohne zwingend sinnvoll zu terminieren. Jedoch ergeben sich, unter weiterer Anwendung der *Instance-of*-Beziehung als Metaisierungsprinzip, ab der Meta-Metaebene keine sinnvollen Aussagen über die modellierten Modelle, mithin kein Informationsgewinnung durch die Metaisierung, mehr. Aus den, im Diagramm der Abbildung 2 dargestellten, Beziehungen zwischen den Modellierungsebenen ergeben sich folgende Aussagen:

- M^0 — M^1: `Max M` ist eine `Person`
- M^1 — M^2: `Person` ist eine `Klasse`
- M^2 — M^3: `Klasse` ist eine `Metaklasse`
- M^3 — M^4: `Metaklasse` ist eine `Meta-Metaklasse`
- ...

Oder allgemeiner: M^n — M^{n+1}: $[\text{Meta}]_{n-2}$-Klasse ist eine $[\text{Meta}]_{n-1}$-Klasse, für $n \geq 2$.

Intuitiv wird klar, das sich für den Anwender der Modellierungssprache einerseits — jedoch auch für den Methodiker andererseits spätestens ab der Meta-Metamodell-Ebene keine sinnvoll interpretierbare Information mehr ergibt. Auch erscheint der Informationsgewinn weiterer Metaebenen (M^{4+}) fraglich.

Jedoch ist ein willkürlicher Hierarchieabbruch vor der Hintergrund einer gewünschten formalen Validierung der Metamodell-Hierarchie, analog der formalen algorithmischen Syntaxprüfung eines konkreten Modelles (hier: UML-Modells), nicht gerechtfertigt.

Die in Abbildung 3 dargestellte zirkuläre Abhängigkeit des Meta-Metamodells von sich selbst (M3+) ist die pragmatische Lösung für dieses Problem. Die Meta Object Facility geht den Weg, Ebene 3 als selbstbeschreibendes Modell aufzufassen. Wie durch die Abhängigkeitsbeziehung angedeutet, ist MOF eine Ausprägung von MOF.

Daher kann das MOF-Meta-Metamodell durch sich selbst auf Korrektheit geprüft werden. Deshalb spricht die MOF-Spezifikation [CII+99] an dieser Stelle auch von einem Prozeß der an die Befreiung Baron Münchhausens aus dem Sumpf (er zieht sich im Märchen am eigenen Schopfe heraus) erinnert.

Jeder der bisher dargestellten Modellebenen kommt eine dedizierte Stellung innerhalb der Modellierungsarchitektur zu: Mit Hilfe des Meta-Metamodells lassen sich beliebige Metamodelle konkreter Modellierungssprachen definieren; das gemeinsame Meta-Metamodell läßt sie zu objektorientierten Modellierungssprachen (das sind im vorliegenden Beispiel: Sprachen die objektorientierte Konzepte verwenden) werden. Diese Konstruktion der Metaebenen ermöglicht es, verschiedene objektorientierte Beschreibungssprachen ineinander zu überführen. Ein klarer Vorteil der hierarchischen Anordnung der einzelnen Modellebenen liegt in der schrittweisen Verfeinerung der semantischen Konzepte durch wiederholte Anwendung desselben Metaisierungsprinzips. [2] Hierdurch wird die Überfrachtung der konkreten Meta-Metaebenen (M2) mit unnötiger, sich teilweise überschneidender Semantikdefinition (wie „Was ist eine Klasse?“) vermieden. Der Bildungsprozeß des Meta-Metamodells *faktorisiert* somit, ähnlich dem in von Wirfs-Brock vorgeschlagenen bottom-up-Ansatz zur Konstruktion (abstrakter) Elternklassen ([WWW90]), gemeinsame Elemente der M2-Ebene in die gemeinsame Meta-Metaebene heraus.

Für die Anwendung des Transferstandards XMI bedeutet die Vier-Schichten Metamodell-Architektur einen zusätzlichen Freiheitsgrad dergestalt, daß die MOF als Basis weiterer Metamodelle genutzt werden kann. Metamodellstandards zur Workflow- und Data-Warehouse Modellierung befinden sich derzeit innerhalb der OMG in Entwicklung ([OMG98]).

XMI transportiert beliebige MOF-Instanzen, d.h. Ausprägungen eines Meta-Metamodells, was eine deutliche Erweiterung des Einsatzspektrums gegenüber einem ausschließlich auf die UML zugeschnittenen textuellen Format bedeutet. Als Resultat ist XMI in der Lage, sowohl Metamodelle, gleichbedeutend mit der Syntaxdefinition einer Modellierungssprache, als auch deren Ausprägungen, die in den Modellierungssprachen erstellten Modelle, zu übertragen.
Insbesondere können zukünftige Sprachen — infolge der Abbildung ihrer Metamodelle in MOF — ebenso problemlos als XMI-Datenstrom übertragen werden, ohne das Transferformat abändern oder erweitern zu müssen.

4 Beispielanwendung

Um die Betrachtungen auf den in der Praxis am häufigsten auftretenden Fall zu beschränken, wird im folgenden ausschließlich der Transfer von UML-Modellen via XMI betrachtet.

Selbstverständlich ist es möglich in XMI, Modelle aller angebotenen Diagrammsprachen der UML, zu codieren, jedoch soll das einfache Klassendiagramm der Abbildung 3 als durchgängiges Beispiel dienen.

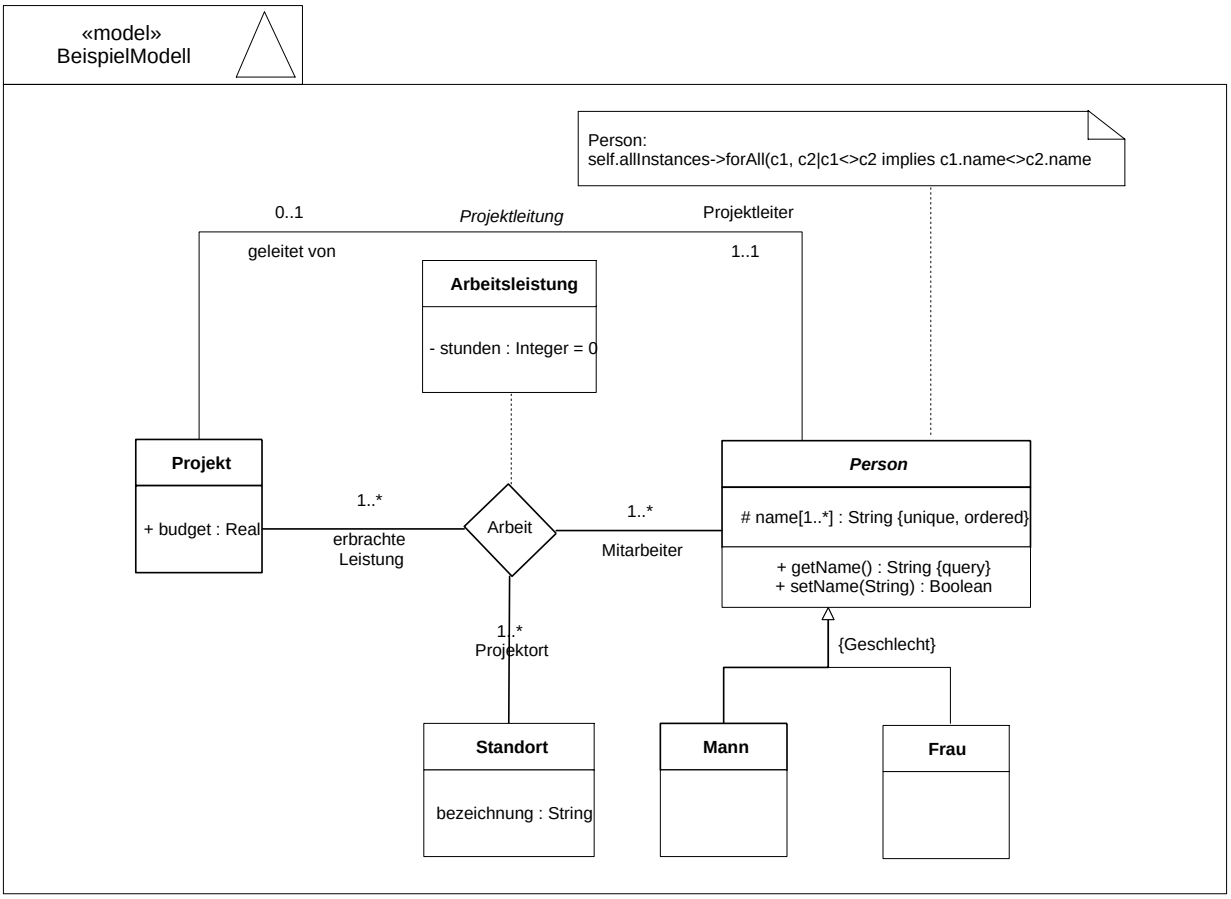

Abbildung 3: Beispielklassenmodell

Das Modell der Graphik 3 bildet einen kleinen Teil des Problembereichs *Projektorganisation* ab.
Eine (als abstrakte Klasse realisierte) `Person` kann an mehreren `Projekten` als `Mitarbeiter` teilnehmen. Darüberhinaus kann sie (die Person) optional ein `Projekt` leiten, d.h. sie übernimmt die Rolle `Projektleiter`. Jedes `Projekt` hat mindestens einen `Mitarbeiter` und genau einen `Projektleiter`. Desweiteren können `Projekte` an verschiedenen `Standorten`

von verschiedenen `Mitarbeiten` simultan bearbeitet werden. Der ternäre-Beziehungstyp `Arbeit` stellt diesen Sachverhalt dar. Dieser, durch den Beziehungstyp entstehenden, eindeutigen Kombination ist die `Arbeitsleistung`, gemessen in `Stunden`, zugeordnet.
Die `Person` verfügt zudem über das mengenwertige Attribut `name`, welches die einzelnen Namen (erster Vorname,..., Nachname, akad. Grade, etc.) aufnimmt.
Der Sichtbarkeitstyp dieses Attributs ist *protected*, d.h. es ist ausschließlich in der definierenden Klasse selbst, sowie in den direkten Kindklassen, sichtbar (das `#`-Symbol deutet dies graphisch an).
Der Eigenschaftswert `ordered` ist in der UML vordefiniert (siehe [OHI+99, p. 2-22]), zur Spezifikation einer geordneten Menge. Im konkreten Beispiel legt er die beschriebene Reihenfolge der Namen einer Person fest. Ferner verfügt das Attribut `name` über einen Eigenschaftswert `unique`, der nicht Bestandteil der offiziellen UML-Norm ist. Hierbei handelt es sich um eine benutzerdefinierte Erweiterung der Modellierungssprache (die Darstellung der *UML Extension Mechanisms* würde den Rahmen dieses Beitrags sprengen, deshalb sei auf die einschlägige Literatur — wie [OHI+99, p. 2-65ff] — verwiesen). Dieser Erweiterung der Ursprache ist eine mit OCL formulierte Integritätsbedingung (im Beispielmodell der Abbildung 3 als Anmerkung (in UML-Terminologie: *Note*) zugeordnet. Die formale Zusicherung spezifiziert die Eindeutigkeit des Attributs `name` über alle Ausprägungen der Klasse `Person`. Für die folgenden Betrachtungen der Transfermöglichkeiten soll lediglich das eventuelle Vorhandensein nicht standardisierter aber metamodellkonformer Modellelemente (Erweiterungen der UML) festgehalten werden.
Als Subklassen der `Person` sind, differenziert (im UML-Sprachgebrauch: *diskriminiert*) nach `Geschlecht`, `Mann` und `Frau` definiert. Sie können über die `Person` hinaus weiter spezialisierendes Verhalten oder zusätzliche Eigenschaften (in der Graphik nicht dargestellt), haben.

5 Gewinnung der XMI-Darstellung

Die Übersetzung beliebiger UML-Diagramme (im Beispiel ein Klassendiagramm) vollzieht sich in zwei Schritten, die mit den beschriebenen Metaebenen korrespondieren.
Zunächst wird das, in der Terminologie der umseitig dargestellten Abbildung 2 auf der M1-Schicht angesiedelte, Modell als Instanz des zugehörigen Metamodells ausgedrückt. Anschließend wird dieses Metamodell, angereichert um die gewonnenen konkreten Objekte, gemäß der zugehörigen DTD in den XMI-Stream übersetzt.
Die XMI-Norm definiert für alle MOF-basierten Metamodelle allgemein, und das der UML im besonderen, eine XML-*Document Type Definition* (Abk. DTD). Sie liefert die Grammatik der Transfersprache. Ferner enthält die Norm Generierungsrichtlinien, welche abstrakt die Transformation beliebiger Metamodelle in XMI-DTDs beschreiben. Die normierte UML-DTD stellt bereits eine Anwendung dieser Prinzipien dar.

Ausgehend von Darstellung des Modells als Metamodell-Instanz kann der XMI Datenstrom ohne hinzufügen weiterer Information erzeugt werden. Dies ist in der Praxis die notwendige Voraussetzung der maschinellen Generierung der XMI Darstellung auf Basis der vorhandenen Modelle sowie der zugehörigen Metamodelle.

Das Diagramm der Abbildung 4 greift auf das Metamodell der aktuellen UML-Sprachfestlegung (siehe [OHI+99, pp. 6-5ff]) zurück. Es enthält jedoch nur die am Beispiel beteiligten Metaklassen. Andere Details (ungenutzte und die Übersichtlichkeit beeinträchtigende) sind ausgeblendet.

Einzelheiten des Beispiels finden sich als Objekte vom Typ der zugeordneten Metaklasse wieder. Die zugehörigen (Meta-)Eigenschaften des Modellelements werden als Attributwerte des Metaobjekts dargestellt. Das gesamte Metamodell ist ausschließlich in Klassendiagrammen modelliert, welche die Semantik der einzelnen Sprachkonstrukte festlegen.

Wie erwähnt, entsteht unter Einbezug der Meta-Ebenen eine unauflösbare meta-zirkuläre Beziehung innerhalb der Modellhierarchie. Zwar ist jedes UML-Modell durch Abgleich mit dem UML-Metamodell, unter Einbezug der Mapping-Semantikdefinition, auf syntaktische Korrektheit prüfbar (gleiches gilt analog für das UML-Metamodell selbst, relativ zur Meta Object Facility), jedoch verbleibt das semantische Verhältnis der Modellebenen weiter ungeklärt. De facto wird es durch die syntaktische Korrektheit kaschiert.
Letztlich stellt sich die Frage, ob eine wie auch immer geartete Formalisierung überhaupt in der Lage ist — oder wäre — die semantische Korrektheit zu prüfen.

Trotz der diskutierten Problematik, stellt die Vier-Ebenen Metamodell-Architektur einen hinreichend mächtigen und intuitiv verständlichen Mechanismus dar, der bei beherrschbarer Komplexität die Einordnung beliebiger Sprachen — unter einem gemeinsamen Meta-Metamodell — erlaubt.
Die somit geschaffene Basis erweist sich im folgenden als tragfähiges und gleichzeitig fle-

xibles Fundament zur Generierung der Transferdarstellung. Der Ableitungsprozess soll im weiteren sukzessive, beginnend mit den Klassen, ausschnittsweise veranschaulicht werden.

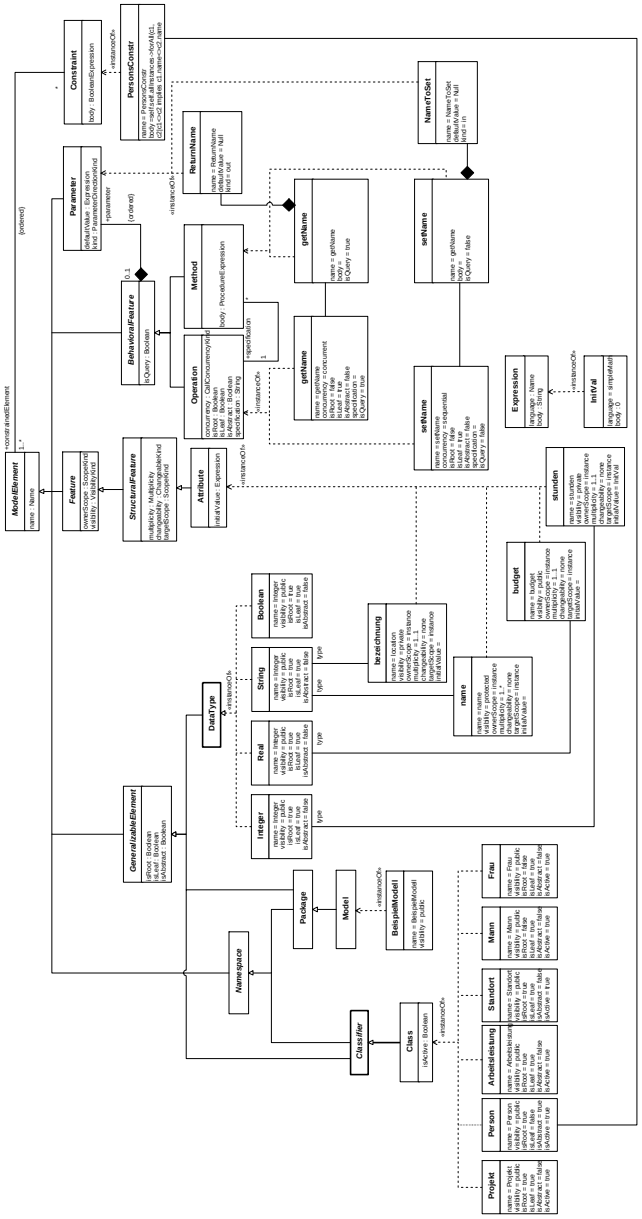

Abbildung 4: Darstellung des Beispiels (Abbildung 3) als Metamodell-Instanz

Die im Metamodell aufgebaute Hierarchie dient in erster Linie der Zusammenfassung und Gruppierung von Sprachkonstrukten mit gleicher Semantik. So sind (wie im Ausschnitt der Abbildung 5 zu dargestellt) neben den Klassen auch Datentypen generalisierbar; sie werden beide von der gemeinsamen Metaklasse `GeneralizableElement` abgeleitet.

Bei näherer Betrachtung der Objekte des Klassendiagramms der Abbildung 4 (die vollständige Vererbungshierarchie für `Person` ist in Abbildung 6 dargestellt) fällt der erste Effekt der Metaisierung ins Auge: der höhere Uniformitätsgrad der dargestellten Information. Dieser Aspekt wird durch die faktische Reduktion des Sprachumfangs erreicht. Zwar stünden rein formal auf den Metaebenen ($M2^{+}$) dieselben Sprachkonzepte wie in der definierten Modellierungssprache zur Verfügung, jedoch ist die Sprachsemantik (das Metamodell) durch vergleichsweise wenige Primitive — im wesentlichen Klasse, Attribut, Assoziation, Abhängigkeitsbeziehungen, Vererbung, Multiziplitäten, Paketierung (vgl. [OHI+99, p. xiv, 2-6f, 2-13ff, 2-67, 2-75f, 2-83f, 2-104, 2-118, 2-123f, 2-130f, 2-160, 2-172, 2-179f, 182]) — ausdrückbar.

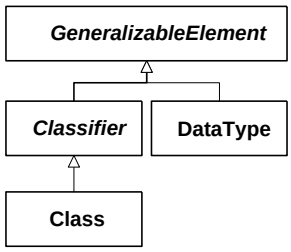

Abbildung 5: Metamodellausschnitt

Als Resultat erscheinen die modellierten Sachverhalte deutlich transparenter.

Als markantes Beispiel zeigen sich die Benennungen der einzelnen Elemente. Während die UML den Namen des Modellelements wie Klasse, Attribut oder des Modells selbst unterschiedlich darstellt (siehe Abbildung 3) — als fett gedruckte Zeichenkette im Kopf des Klassensymbols, als Namen zwischen Multiziplitätsangabe und Typ im Attributbereich oder als Beschriftung des stilisierten Hängeordners für das Modell — wird dieser Sachverhalt auf Metaebene durch das immergleiche Attribut `name` verkörpert. Ebenso werden im graphischen Modell symbolhaft formulierte Angaben, wie die Sichtbarkeit, durch textuelle Attribute der Metaobjekte abgebildet.

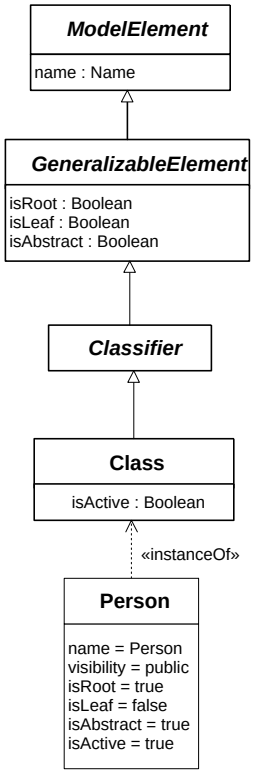

Abbildung 6: Vererbungshierarchie (vgl. [OHI+99, fig. 2-8, p. 2-17])

Die in Graphik 7 herausgegriffene Abbildung der Attribute `name`, `budget` und `stunden` des Modells auf das Metamodells zeigt die beschriebene Überführung vorher symbolisch dargestellter Information. Die unterschiedliche Sichtbarkeit (im UML-Jargon: *visibility*) ist als Wert eines Metaattributs expliziert, somit ist die Symbolik („#“ in Abbildung 3) durch die Klartextzuordnung quasi „entschlüsselt“. Wenngleich trotzdem Besonderheiten der Modellierungssprache erhalten bleiben. Hier sei auf die Inhaltscodierung der `multiplicity` verwiesen, die vom Grundsatz der Klartextexplizierung symbolhafter Information abrückt, und die im Modell dargestellte Angabe (`1..*` bzw. `1..1`) unverändert übernimmt, ohne sie in Unter- und Obergrenze zu teilen, sowie den Stern in *unlimited* zu überführen.

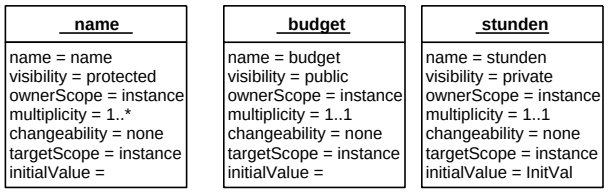

Abbildung 7: Attributdarstellung im Metamodell

Ebenfalls zur Kategorie explizierter Information zählt der `ownerScope`, welcher festlegt, ob es sich um ein Klassenattribut (graphisch durch Unterstreichung abgebildet — im Beispiel nicht benutzt) handelt, oder ob jede konkrete Ausprägung der Klasse (*instance*) einen eigenen Speicherplatz zur Wertaufnahme zur Verfügung stellt (siehe [OHI+99, p. 2-34]).

Desweiteren überführt die Abbildung in das Metamodell vorher implizit ausgedrückte Gegebenheiten in eine offensichtlichere Variante. Gut sichtbar wird dies am Attribut `isActive` der Metaklasse `Class`. Hier wird spezifikationsgemäß der Vorgabewert gesetzt.
Als Resultat ist es nicht notwendig über die XMI-Darstellung hinausgehende Information zur Rekonstruktion des Ursprungsmodells heranzuziehen. Insbesondere ist kann eine zusätzliche Referenz auf die verwendete Modellierungssprache (bzw. deren Version) entfallen, da alle notwendigen Informationen im XMI-Stream abgelegt wird.

Metaattribute wie `isRoot` lassen sich einer weiteren Kategorie zuordnen. Sie stellen aus dem Kontext des Modellelements ableitbare Information dar. Im Falle von `isRoot`, ob es sich um eine Klasse ohne Elternklassen (also das Wurzelelement einer Vererbungshierarchie handelt).

Jenseits der auf das Beispielmodell zurückführbaren Komponenten fallen zusätzliche Metaobjekte ins Auge: die „gewöhnlichen“ Datentypen objektorientierter Programmiersprachen (`Integer`, `Real`, `Boolean`, `String`) und `InitVal`. Die explizite Definition der Datentypen von der im Modell der Abbildung 3 enthaltenen Attribute verwundet zunächst, definiert doch das UML-Metamodell bereits diese Typen und wendet sie zur Definition der Sprachsemantik weidlich an (siehe beispielsweise die Definition der Multiplicity [OHI+99, p. 2-75]). Jedoch schlägt sich diese Festlegung nicht auf den definierten Sprachschatz nieder.
Begründet wird dies üblicherweise mit der weitestgehenden Vermeidung einschränkender Definitionen an die entstehende Modellierungssprache, die den Vorgriff auf spätere programmiersprachliche Umsetzungen, bürgen.
Eine Assoziation setzt jeweils die Typdefinition mit dem darauf basierenden Attribut in Beziehung (vgl. Abbildung 8).

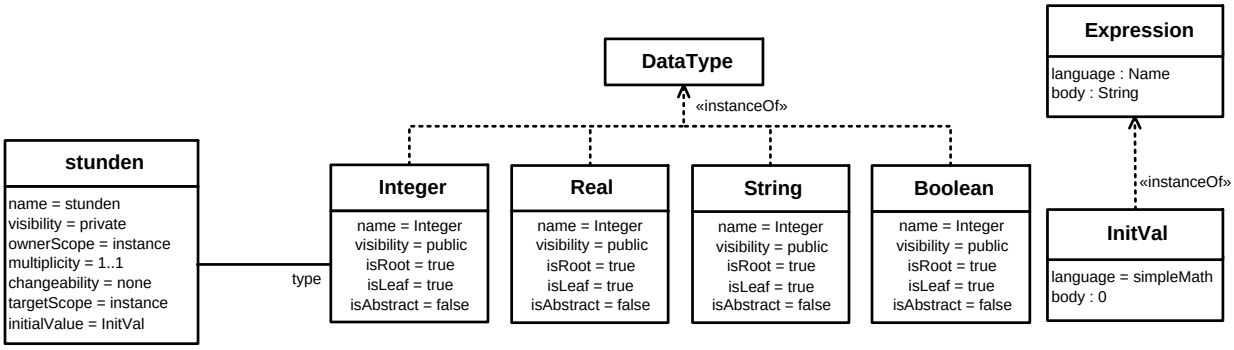

Abbildung 8: Darstellung der Basistypen und Vorgabewerte im Metamodell aus Bild 4

Das Objekt `InitVal` verkörpert den Vorgabewert des Attributs `stunden`. Hinausgehend über die im UML-Klassendiagramme dargestellten Einzelheiten findet im Metamodell auch die Sprache, die zur Darstellung der Wertinitialisierung verwandt wird, Eingang. Der Inhalt des Metaattributs `initialValue` referenziert ein anderes Metaobjekt (im Beispiel `InitVal`) das die Wertfestlegung näher beschreibt.
Ausgehend von der Darstellung des UML-Modells als Metamodellinstanzen ist die Überführung in XMI denkbar einfach. Entlang der baumartigen Hierarchie direkt umschließender Modellelemente (d.h. Beispielmodell enthält u. a. die Klasse `Person`, welche das Attribute `name` vollständig umschließt) wird die im Metamodell dargestellte Information ausgegeben. Diese Betrachtungsweise wird auch gut durch die graphische Darstellung der Klassenstrukturen, wie in Abbildung 3, gestützt.

Die Angabe des vollständigen XMI-Streams würden den Rahmen des vorliegenden Beitrages sprengen, deshalb sind nachfolgend nur Ausschnitte als Beispiel angegeben. Der ungekürzte Beispielcode ist über die Homepage des Autors verfügbar
(`http://www.krumbach.de/home/jeckle/index1.htm`)

Beginnend mit dem Modell werden die Attribute des Metamodells abgearbeitet. Die XMI-Norm definiert Generierungsrichtlinien (*generation principles*), welche die Gewinnung der XMI-Repräsentation aus dem Metamodell festlegen. Durch diese Formalisierung wird ein Determinismus eingeführt, der sichergestellt, daß neue Document Type Definitionen zukünftiger Metamodelle schnell (da maschinengestützt) und eindeutig (es liegen immer dieselben Prinzipien zugrunde) generiert werden können.

XMI-Streams sind generell zweistufig aufgebaut, zunächst sind in einem Kopfelement alle administrativen und verwaltungsrelevanten Daten untergebracht. An diesen schließt sich der eigentliche Nutzdateninhalt, der sog. *content*, an.

```
<?xml version = "1.0"?>
<!DOCTYPE XMI SYSTEM "uml.dtd">
<XMI xmi.version="1.0">
<XMI.header>
<XMI.metamodel xmi.name="uml" xmi.version="1.3"/>
</XMI.header>
```

Das Codefragment zeigt den Beginn des XMI-Streams. Im Vorspann wird zunächst die Identifikation als XML-Dokument vorgenommen. Anschließend wird die zur Validierung notwendige UML-DTD referenziert.
Nachfolgend beginnt der, durch den Tag XMI.header eingeleitete Nutzdatenbereich. Zunächst wird das für den Transfer maßgebliche Metamodell spezifiziert. Dies stellt eine Einschränkung gegenüber der im vorhergehenden getroffenen Aussage zur Metamodellunabhängigkeit dar. Der für den Transfer relevante Anteil des Metamodells ist aus dem XMI-Stream konstruierbar, jedoch nicht validierbar, da keine Fehler im Metamodell (d.h. die Gültigkeit des Metamodells hinsichtlich seines Metamodells) entdeckt werden können.

Jeder Tag des entstandenen Dokuments wird zunächst durch den Namen des Metamodell-Packages eingeleitet, in dem die gerade bearbeitete Metaklasse definiert ist, im Beispiel `Foundation.Core`. Daran schließt sich der eindeutige Name der Metaklasse und des Metaattributs an. Die einzelnen Komponenten des Tag-Names werden durch Punkte getrennt. Für den Metamodellkundigen entstehen aufgrund dieser durchgängigen Namenskonvention sprechende Namen. Als Beispiel sei der letzte Tag des folgenden Codefragments herausgegriffen: Von rechts nach links betrachtet sagt sie, ohne Konsultation des zugrundeliegenden Metamodells aus, daß ein (Meta-)Attribut `isAbstract` einer Klasse `GeneralizableElement` existiert, und diese Klasse im Package `Core` des Modells `Foundation` definiert ist (die umgekehrte Lesrichtung resultiert aus der beliebigen Schachtelungstiefe der Packages). Mit diesem Determinismus kann der für das codierte Modell relevante Bereich des zugrundeliegenden Metamodells einfach und sicher ermittelt werden. Durch den Einschluß kompletter Tags entsteht auch im XMI-Dokument eine hierarchische Struktur. Der aus der diskutierten Baumstruktur des Modells herrührende Aufbau bleibt somit auch

im XMI-Strom gewahrt.

```
<XMI.content>
<Model_Management.Model xmi.id="ModelBeispielModel">
<Foundation.Core.ModelElement.name>
BeispielModel
</Foundation.Core.ModelElement.name>
<Foundation.Core.ModelElement.visibility xmi.value="public"/>
<Foundation.Core.GeneralizableElement.isRoot xmi.value="true"/>
<Foundation.Core.GeneralizableElement.isLeaf xmi.value="true"/>
<Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
```

Schrittweise werden im Nutzdatenbereich die verschiedenen enthaltenen Elemente des Modells abgearbeitet. Zunächst, wie dargestellt, das Modell selbst. Deutlich sichtbar ist die Überführung der Metamodelldarstellung aus Abbildung 4 in die XMI-Elemente, insbesonders der Erhalt der Vererbungshierarchie (`GeneralizableElement`).

```
<Foundation.Core.Class xmi.id="ClassPerson">
<Foundation.Core.ModelElement.name>
Person
</Foundation.Core.ModelElement.name>
<Foundation.Core.ModelElement.visibility xmi.value="public"/>
<Foundation.Core.GeneralizableElement.isRoot xmi.value="true"/>
<Foundation.Core.GeneralizableElement.isLeaf xmi.value="false"/>
<Foundation.Core.GeneralizableElement.isAbstract xmi.value="true"/>
<Foundation.Core.Class.isActive xmi.value="true"/>
```

Zur Illustration der Codierung einer Klasse sei die `Person` als Beispiel herausgegriffen. Jedes Metamodellelement wird mit einer eindeutigen Identifikation (`xmi.id`), durch einen besonderen XML-Datentyp, versehen. Damit werden grundlegende Validierungsschritte, hier die Eindeutigkeit von Identifiern und Korrektheit von darauf aufbauenden Referenzen, auf die generische Ebene des XML-Parsers verlagert.
Auffallend an diesem Beispiel ist die Ähnlichkeit in XMI-Darstellung zwischen dem zuvor dargestellten `Model` und der `Class`, sie rührt, wie in Abbildung 6 bzw. 5 aufgezeigt, von der Vererbungshierarchie — d.h. der gemeinsamen Elternklasse der beiden Metaklassen her.

Als letztes Beispiel dieses kurzen Überblicks sei die bereits im Abschnitt über die Metamodell-Darstellung des konkreten Modells diskutierte Explizierung der Datentypen dargestellt:

```
<Foundation.Core.DataType xmi.id="DataTypeInteger">
<Foundation.Core.ModelElement.name>
Integer
</Foundation.Core.ModelElement.name>
<Foundation.Core.ModelElement.visibility xmi.value="public"/>
<Foundation.Core.GeneralizableElement.isRoot xmi.value="true"/>
<Foundation.Core.GeneralizableElement.isLeaf xmi.value="true"/>
<Foundation.Core.GeneralizableElement.isAbstract xmi.value="false"/>
</Foundation.Core.DataType>
```

Alle im Modell verwandten Datentypen, sei es zur Definition von Attributen, Operationsparametern oder deren Rückgabewerten, müssen über diesen Mechanismus explizit definiert werden.

6 Mehr als „nur“ Modellaustausch

Zusammenfassend kann XMI als zukünftiger Standard zum objektorientierten Modellaustausch (zumindest im Umfeld der OMG Standards wie UML) angesehen werden. XMI ist nicht der erste Versuch der Etablierung eines sprach- und methodenneutralen Transferformates (das *Case Tool Data Interchange Format* (Abk. CDIF) und die Bemühungen im Rahmen der ISO-STEP-EXPRESS-Bewegung seien hier nur exemplarisch erwähnt), jedoch deuten die breite Unterstützung der Bemühungen im Vorfeld, und erste Prototypen namhafter Hersteller, eine rasche Umsetzung an.
Durch die OMG-Schirmherrschaft über den Standard ist eine werkzeugunabhängige Ausrichtung der Bestrebungen — und damit einhergend, Gleichbehandlung der verschiedenen konkurrierenden Marktteilnehmer — sichergestellt.
Abschließend seien weitere Einsatzmöglichkeiten des XMI-Formates im objektorientierten Systementwicklungsprozeß kurz umrissen:
Unter anderem werkzeugunabhängige...

- Codegenerierung aus OO-Modellen
- Modellvalidierung
- Metrikenberechnung
- Langzeitspeicherung
- Versionsverwaltung
- Systemunabhängiges Export-/Importformat für Datawarehouse-Anwendungen
- ...

7 Kritische Bewertung des Ansatzes *XMI*

Trotz des beachtlichen, Momentums das die XML-basierten Sprachen und Technologien mittlerweile vorantreibt, sollen einige kritische Randbemerkungen zur vorgestellten Sprache XMI nicht unterbleiben.
Wie am aufgezeigten Beispiel ablesbar nimmt der XMI-Stream schnell, selbst bei relativ „trivialen“ Modellen, eine „beachtliche“ Größe an. Diese Aspekt, als hemmender Faktor, tritt (vermeintlich) lediglich bei Mensch-Maschine-Interaktion mit XMI als Schnittstelle in den Vordergrund. Das Größenverhalten der erzeugten Modellinformation ist auf die Struktur von XML-Dateien zurückzuführen, die insbesondere bei grober Disproportionalität zwischen Meta- und Nutzdaten auftritt. Hierbei übersteigt die Größe der öffnenden und schließenden Tags deutlich die der eingeschlossenen Information

(z.B.: `<Foundation.Core.StructuralFeature.multiplicity>`
`1`
`</Foundation.Core.StructuralFeature.multiplicity>`
Verhältnis Metadaten zu Nutzdaten: 93 zu 1(!))
Gleichzeitig offenbart sich eine weitere Schwäche der XML-Sprachen im allgemeinen, und XMIs im besonderen, die fehlende Übersichtlichkeit und Erfaßbarkeit (engl. Human readability). Es wird versucht beiden Punkten mit einer Argumentationslinie zu begegnen, welche den Inter-Tool-Austausch gegenüber der Interaktion mit dem Tool betont. Technisch soll hierdurch das Einsatzgebiet von XMI auf Bereiche der reinen Maschine-Maschine-Kommunikation verengt werden — von der Intention der UML eine „menschen les- und verstehbare“ gleichwertige textuelle Modellierungssprache zur Seite zu stellen ist damit — zunächst — Abstand genommen.
Die Idee einer solchen textuellen Sprache als Ergänzung zu den bestehenden graphischen UML-Diagrammsprachen wurde in jüngster Zeit jedoch wieder durch die OMG aufgegriffen, und in einem entsprechenden *Request for Proposals* (siehe [OMG99a]) zur Einreichung diesbezüglicher Normungsvorschläge aufgefordert.

8 Existierende Implementierungen

Zwar ist es im augenblicklichen Entwicklungsstand (noch) schwer konkrete Realisierungen herauszugreifen, jedoch soll ein kurzer Überblick des XMI-Einsatzes in der Industrie gegeben werden.
Nahezu alle namhaften CASE-Toolhersteller haben bereits erste Studien oder Prototypen vorgelegt. Hierbei scheint sich die (technisch mit weniger Aufwand behaftete) Export-Schnittstelle schon in der jeweils nächsten Werkzeugversion als Standard abzuzeichnen. Problematischer einzuschätzen ist hingegen die Import-Funktionalität, da sie auf ein entsprechend gestaltetes Repository des Werkzeuges zurückgreift (d.h. greifen sollte), und an diesem unter Umständen größere Änderungen notwendig sind. Aus dieser Sicht heraus er-

geben sich zunächst Wettbewerbsvorteile für diejenigen Hersteller, deren Werkzeuge bereits ein UML-verträgliches Metamodell besitzen.
Bereits im Herbst letzten Jahres haben die XMI-Partner ein prototypisches Entwicklungsszenario, unter Einbezug der entsprechenden Tools, präsentiert, das die Integration verschiedener Systeme (darunter Entwicklungsumgebung, Modellierungstool, Datenbankmanagementsystem) zeigt.

Für konkrete Produktinformation sei auf die Internetauftritte der jeweiligen Hersteller verwiesen.

Notes

[1] In bezug auf die UML sei auf die laufenden Aktivitäten hingewiesen, sie in absehbarer Zeit (voraussichtlich im Laufe des kommenden Jahres) unverändert in den Rang eines offiziellen ISO-Standards zu erheben.

[2] Der Korrektheit halber sei angemerkt das es sich bei der Kopplung der einzelnen Modellebenen innerhalb der OMG-Metamodell-Architektur derzeit (d.h. UML v1.3) „nur“ um *lose Metamodellierung* handelt, in der nicht wie bei der *strikten* Variante, jedes Element einer Modellebene n direkte Ausprägung eines der Modellebene $n+1$ zugeordneten Modellelements sein muß. Für UML Version 2.0 ist die Definition eines strikten Metamodell-Mappings angestrebt ([OMG99b]).

Literatur

[Boo94] Grady Booch. *Object Oriented Design with Applications.* Addison-Wesley, 2nd edition, 1994.

[CII+99] Cooperative Research Centre for Distributed System, International Business Machines Corporation, International Computers Limited, Objectivity, Inc., Oracle Corporation, System Software Associates, and Unisys Corporation. *Meta Object Facility.* Object Management Group, Framingham, July 1999. (joint revised submission), Document: ad/99-07-03.

[HJK+99] Peter Hruschka, Nicolai M. Josuttis, Hartmut Kocher, Hartmut Krasemann, and Markus Reinhold. *Erfolgreich mit Objektorientierung – Vorgehensmodelle und Managementpraktiken für die objektorientierte Softwareentwicklung.* Oldenbourg, München, 1999.

[Jac94] Ivar Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, Reading, 1994.

[OHI+99] Object Management Group, Hewlett-Packard Company, International Business Machines Corporation, ICON Computing, i-Logix, IntelliCorp, Electronic Data Services Corporation, Microsoft Corporation, ObjecTime Limited, Oracle

Corporation, Platinum Technology, Inc., Ptech Inc., Rational Software Corporation, Reich Technologies, Softeam, Sterling Software, Taskon A/S, and Unisys Corporation. *OMG Unified Modeling Language Specification Version 1.3*. Object Management Group, Framingham, June 1999. Document: ad/99-06-06.

[OMG98] Common Warehouse Metadata Interchange – Request For Proposal. Object Management Group, Framingham, September 1998. Document: ad/98-09-02.

[OMG99a] A Human-Usable Textual Notation for the UML Profile for EDOC – Request For Proposal. Object Management Group, Framingham, March 1999. Document: ad/99-03-12.

[OMG99b] UML 2.0 Request For Information. Object Management Group, Framingham, August 1999. document: ab/99-08-08.

[RBP+91] James Rumbaugh, Michael R. Blaha, William Premerlani, Frederick Eddy, and William Lorenson. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, 1991.

[Sol97] Richard M. Soley. *Object Management Architecture Guide*. John Wiley & Sons, New York, 3rd edition, 1997. Document: ab/97-05-05.

[Wor98] World Wide Web Consortium. Extensible Markup Language (XML) 1.0. World Wide Web Consortium, February 1998. REC-xml-199808210, URL: (as of February, 10th 1999) http://www.w3c.org/xml.

[WWW90] Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. *Designing Object-Oriented Software*. Prentice-Hall, Englewood Cliffs, 1990.