Pseudo-Modelldifferenzen und die Phasenabhängigkeit von Metamodellen

Udo Kelter
Fachbereich Elektrotechnik und Informatik
Universität Siegen
kelter@informatik.uni-siegen.de

Abstract: Beim Vergleichen von Dokumenten werden manchmal Unterschiede angezeigt, die man als inhaltlich belanglos ansieht; solche Differenzen werden als Pseudodifferenzen bezeichnet. Wir betrachten dieses Phänomen für den speziellen Fall des Vergleichs von Modellen, deren Struktur durch ein Metamodell definiert wird, wie z.B. in der UML. Einen großen Teil der Pseudodifferenzen kann man darauf zurückführen, daß Metamodelle selbst abhängig von Entwicklungsphasen auf der Metaebene sind. Die Pseudodifferenzen entstehen hier, weil "spätphasige" Metamodelle benutzt werden. Weitere Typen von Pseudodifferenzen entstehen infolge von Editierkommandos bzw. elementaren Änderungen in abstrakten Syntaxgraphen, die nur durch mehrere zusammenhängende Änderungen auf der nächsttieferen Ebene realisiert werden können, ferner infolge suboptimaler Differenzen.

1 Einleitung

Beim Vergleichen von Dokumenten werden manchmal Unterschiede angezeigt, die man als inhaltlich belanglos ansieht; solche Differenzen werden als **Pseudodifferenzen** bezeichnet. Ein sehr einfaches Beispiel sind Leerzeichen am Ende von Textzeilen in einem Textdokument: man sieht sie nicht, egal wieviele vorhanden sind. Ein etwas komplexeres Beispiel mit Texten besteht darin, die Zahl der Leerzeichen zwischen zwei Worten belanglos zu finden, oder allgemeiner jeden beliebig geformten Leerraum zwischen zwei Worten als gleichwertig zu betrachten. Ein drittes Beispiel ist eine XML-Datei, in der zwei Realweltobjekte und eine Beziehung zwischen diesen Objekten repräsentiert werden, und zwar jedes Objekt durch ein XML-Element und die Beziehung durch zwei gleiche Werte in je einem Attribut in diesen beiden Elementen, z.B einem `ID`- und einem `IDREF`-Attribut. Welcher konkrete Wert in diesen beiden Attributen steht, ist belanglos, die dargestellte Beziehung bleibt die gleiche. Zwei XML-Dateien, die die gleichen Entitäten und Beziehungen repräsentieren, können daher umfangreiche textuelle Differenzen aufweisen.

Pseudodifferenzen müssen von echten Differenzen unterschieden werden, u.a. weil sie in Differenzdarstellungen stören und man dort nur echte Differenzen sehen will; bei Mischungen erzeugen sie unnötige Konflikte.

Die vorstehenden Beispiele legen es nahe, Äquivalenzen zwischen Dokumentzuständen

zu definieren und von einer Pseudodifferenz zu reden, wenn die Zustände äquivalent sind. Nur in einfacheren Fällen kann man Äquivalenzen lokal definieren (z.B. ein Leerzeichen ist äquivalent zu n Leerzeichen). Bei komplizierteren Dokumentstrukturen, z.B. der o.g. Beziehung in einer XML-Datei, liegt es nahe, von der textuellen Darstellung auf einen abstrakten Syntaxbaum überzugehen. Allerdings hilft dies in diesem Beispiel auch nicht weiter, denn die beiden Attributwerte, die die Beziehung darstellen, erscheinen auch im Syntaxbaum. Die offensichtliche Lösung besteht darin, zu einer noch abstrakteren Darstellung des Inhalts der XML-Datei überzugehen, die die Beziehung direkt enthält, hier also zu einem abstrakten Syntaxgraphen mit typisierten Knoten und Kanten.

In diesem Papier betrachten wir das Problem der Pseudodifferenzen speziell für Differenzen zwischen Modellen, deren Struktur wie z.B. in der UML durch ein Metamodell definiert ist. Eine Hauptthese ist, daß viele Pseudodifferenzen dadurch entstehen, daß "spätphasige" Metamodelle benutzt werden. Dies sind Metamodelle, die auf der Meta-Ebene technologiespezifische Anteile enthalten. Diese Formen von Pseudodifferenzen können nur systematisch behandelt werden, wenn man sich die Phasenabhängigkeit von Metamodellen explizit bewußt macht - letzteres geschieht in der Literatur bisher kaum [1].

Ferner werden teilweise die Phasenabhängigkeit und Meta-Ebenen miteinander verwechselt. Daher behandelt zunächst Abschnitt 2 die Phasenabhängigkeit von Metamodellen ausführlich und zeigt, daß Phasenabhängigkeiten unabhängig von linguistischen und ontologischen Metamodellhierarchien sind. Sobald man Metamodelle eindeutig Phasen zuordnet, kann man die meisten Pseudodifferenzen sehr einfach eliminieren, indem man auf "frühphasige" Metamodelle übergeht (s. Abschnitt 3).

Weitere Typen von Pseudodifferenzen entstehen infolge von elementaren Änderungen in abstrakten Syntaxgraphen, z.B. dem Löschen einer Kante, die nur durch mehrere zusammenhängende Änderungen in den technologieabhängigen Modellen realisiert werden können (s. Abschnitt 4). Ein analoger Effekt entsteht eine Größenstufe darüber durch elementare Änderungen in abstrakten Syntaxgraphen, die nicht isoliert durchgeführt werden können, sondern nur als Teil einer inhaltlich sinnvollen Editieroperation.

Es verbleiben einige unschärfer definierte Formen von Pseudodifferenzen, die abhängig von der Methode, wie Differenzen gewonnen werden, auftreten. Hier sind zustandsbasierte und protokollbasierte Verfahren zu unterscheiden, weil sie zunächst eigene Formen von überflüssigen Bestandteilen von Differenzen oder ungünstigen Darstellungen erzeugen können. In Abschnitt 5 zeigen wir, daß das Problem in beiden Fällen im Kern auf ein allgemeineres Optimierungsproblem hinausläuft, nämlich unter mehreren denkbaren Darstellungen der Unterschiede zwischen zwei Modellen eine günstige zu wählen.

[1] In einigen vielzitierten Publikationen über Metamodelle, z.B. [1, 2, 7, 10], wird die Phasenabhängigkeit von Metamodellen nicht diskutiert. Ob die Problematik übersehen oder bewußt ausgeklammert wurde, weil es primär um konzeptuelle Meta-Ebenen geht, sei dahingestellt. In [3], wo in Abschnitt 3.2 'From contemplative to operational models' auch praktische Fragen adressiert werden, werden Konversionen zwischen verschiedenen technologiespezifischen Repräsentationen von Metamodellen einfach als "Projektionen" bezeichnet; wie solche Projektionen arbeiten, bleibt offen. Publikationen zur Differenzberechnung von Modellen, z.B. diverse Beiträge zu den CVSM-Workshops [4, 5], gehen durchweg von vorgegebenen Metamodellen aus, die nicht weiter hinterfragt werden. Besonders gilt dies für "generische" Verfahren, die nur eine bestimmte Repräsentation der Metamodelle voraussetzen, z.B. als Ecore-Laufzeitobjekte.

2 Phasenabhängigkeit von Metamodellen

2.1 Paradigmen für Metamodellhierarchien

Metamodellhierarchien werden vor allem anhand des linguistischen und des ontologischen Paradigmas gebildet. Das linguistische Paradigma liegt der UML-Metamodellhierarchie [12] zugrunde, ebenfalls der wesentlich älteren CDIF-Metamodellhierarchie [6, 8]. Ontologische Metaebenen sind im Kern völlig unabhängig von den linguistischen [1, 2] und können, wenn man von linguistischen Ebenen ausgeht, auf jeder Ebene unabhängig voneinander entstehen. Bei beiden Paradigmen kann man die Entitäten der unteren Ebene als Instanzen von Entitäten der nächsthöheren Ebene auffassen, aber die Bedeutung der "ist-Instanz-von"-Beziehungen ist grundsätzlich anders.

In diesem Abschnitt fassen wir die wesentlichen Charakteristika dieser Hierarchien zusammen; auf dieser Basis können wir im folgenden Abschnitt klären, ob und wie Phasenabhängigkeiten von Metamodellen mit diesen Paradigmen korrelieren.

Merkmale linguistischer semantischer Ebenen. Die Metamodellebenen der UML basieren auf dem linguistischen Paradigma. Dieses ist analog zu den Metasprachebenen der Linguistik (Objektsprache, Metasprache, Meta-Metasprache, ...) definiert:

- Eine Aussage der Metasprache betrifft die Objekt*sprache als ganze* (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 macht ein Metamodell Aussagen bzw. repräsentiert Wissen über alle Modelle des zugehörigen Typs, namentlich wie die Modelle strukturiert und zu interpretieren sind. Metamodelle sind keine vereinfachten oder gekürzten Varianten der von Modellen. Die Gegenstandsbereiche beider Ebenen sind verschieden.

In beiden Fällen enthält die Metaebene 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.

Merkmale ontologischer semantischer Ebenen. Eine ontologische Begriffshierarchie basiert auf einer Grundmenge von Entitäten, z.B. der Menge aller Tiere, und klassifiziert diese Entitäten anhand mehr oder minder abstrakter Begriffe, z.B. Hund - Raubtier - Säugetier - Wirbeltier. Ein abstrakterer Begriff gibt für seine Unterbegriffe und die zugehörigen Entitäten gemeinsame Merkmale und ggf. auch Merkmalsausprägungen vor. Auf Entitäten einer untergeordneten Gruppe treffen daher alle Merkmale und Merkmalsausprägungen aller übergeordneten Gruppen zu[2]. Die Gegenstandsbereiche aller Klassifikationsstufen sind *gleich*; im obigen Beispiel handelt es sich um auf allen Ebenen um

[2] Bei der Modellierung dieser Daten ist neben Typhierarchien oft das Typ-Instanz-Muster sinnvoll verwendbar.

Mengen von Lebewesen, die Teilmengen voneinander sind; wenn man die komplette Begriffshierarchie betrachtet, liefern die Ebenen dieses Baums jeweils andere Zerlegungen der Gesamtmenge aller hier betrachteten Lebewesen.

Wenn nun eine Begriffshierarchie so gestaltet ist, daß alle Wege von der Wurzel zu den Blättern gleich lang sind, kann man *durchgängige Ebenen* bilden und jeder Ebene einen Namen geben. Ein Beispiel sind die biologischen Klassifikationsstufen

Familie - Ordnung - Klasse - Unterstamm - Stamm.

Beispielsweise ist die Gruppe der Hunde eine Familie, und die Gruppe der Wirbeltiere *ist ein* Unterstamm. Man kann also von einer "is-a"-Beziehung zwischen "Wirbeltiere" und Unterstamm reden. Begriffe wie Stamm oder Ordnung klassifizieren keine einzelnen Lebewesen mehr, sondern Begriffe, insofern stehen sie auf einer höheren semantischen Ebene und stellen Meta-Begriffe dar. Für eine gegebene Menge von Entitäten kann es mehrere Klassifikationsmethoden geben, die zu unterschiedlichen Begriffshierarchien mit verschieden vielen Ebenen führen. Ein System von Klassifikationsstufen ist daher sehr eng verbunden mit einer konkreten Begriffshierarchie.

Bei Begriffshierarchien mit stark schwankenden Pfadlängen kann man i.d.R. keine sinnvollen Ebenen bilden und daher auch keine Klassifikationsstufen definieren (außer daß man die Ebenen von der Wurzel aus einfach durchnumeriert). In einem solchen Fall kann man also keine höhere ontologische semantische Ebene mehr bilden (während man in linguistischen Hierarchien prinzipiell immer höhere Ebenen bilden kann).

2.2 Modelle in den Entwicklungsphasen

Modelle treten in verschiedenen Entwicklungsphasen bzw. Verfeinerungsstufen eines Systems auf. Wenn Modell M2 die Weiterentwicklung von M1 ist, kann man häufig M2 als Ergebnis einer Transformation *trf* von M1 verstehen, zu der die weiteren Modellteile *Addendum* hinzugefügt wurden, also in einer mathematischen Schreibweise:

$$M2 = trf(M1) \cup Addendum$$

Ausgangspunkt sollten Modelle sein, die frei von technologiespezifischen Details sind (*platform independent model, PIM* [11]). Diese können in einem oder mehreren Schritten in technologiespezifische Modelle (*platform specific model; PSM*) und letztlich in Quellcode bzw. andere laufzeitrelevante Dokumente transformiert werden. Die Details dieser Transformationsketten hängen vom Typ der Modelle ab. Praxisrelevant sind solche Transformationen bei Datenmodellen, Zustandsmodellen und Ablaufstrukturmodellen.

Datenmodelle. Technologiefreie Modelle der Nutzdaten einer Applikation sind z.B. ER-Diagramme oder Analyse-Klassendiagramme, die wir i.f. als Analysedatenmodelle bezeichnen. Ein Analysedatenmodell wird in einem oder mehreren Schritten weiterentwickelt zu Datentypdefinitionen in einer konkreten Programmiersprache oder zu einem Schema für eine Datenbank oder für XML-Dateien (s. Bild 1), die für eine transiente bzw. persistente Darstellung der Nutzdaten benötigt werden. Die Typdefinitionen in Programmen bzw. Schemata in Datenverwaltungssystemen sind die präzisesten und detailliertesten Ent-

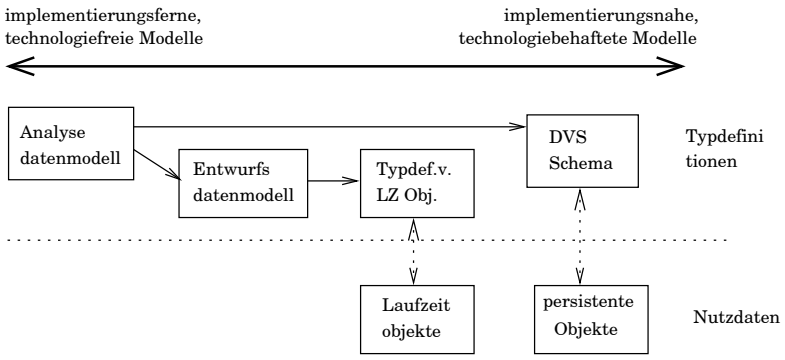

Abbildung 1: Modelle und Typdefinitionen eines Systems in verschiedenen Phasen

wicklungsdokumente; sie können nach einer Übersetzung bzw. Installation von einem Laufzeitsystem, das transiente oder persistente Instanzen verwalten kann, instantiiert werden.

Während die Schemata für die persistente Datenhaltung i.d.R. in einem einzigen Schritt aus den Analysedatenmodellen abgeleitet werden, sind für die transiente Seite mehrere Schritte üblich. Ein typisches Zwischenprodukt ist ein Entwurfsdatenmodell, das gegenüber dem Analysedatenmodell zusätzliche Modellelemente enthält, z.B. Containerklassen, und in dem die Navigationsrichtungen der Beziehungstypen festgelegt sind. Die Festlegung von Navigationsrichtungen ist ein Beispiel für eine technologiespezifische Entwurfsaufgabe bei transienten Daten, die für persistente Daten nicht existiert und die zu unterschiedlichen Transformationsketten führt.

Zustands- und Ablaufmodelle. Technologiefreie Zustandsmodelle von Systemen sind Zustandsübergangsdiagramme, Petri-Netze, *state machines* der UML und viele weitere Varianten. Technologiefreie Ablaufmodelle sind z.B. Aktivitätsdiagramme der UML. Endprodukte von Transformationsketten sind hier Teile des Quellcodes, die das Verhalten des Systems mitbestimmen, oder entsprechende Ressourcen (z.B. Zustandsübergangstabellen), die interpretiert werden. Die Transformationsketten müssen an die jeweilige Architektur des Zielsystems und die dort verwendeten Technologien angepaßt werden. Im Prinzip ergibt sich die gleiche Struktur wie in Bild 1), also mehrere von den technologiefreien Modellen ausgehende Transformationsketten.

“Modelle von Modellen”. Im Zusammenhang mit Pseudodifferenzen stellt sich die Frage, ob bei den Verfeinerungsstufen von Modellen auch eine Metamodellhierarchie vorliegt und ob und wie sie mit linguistischen bzw. ontologischen Hierarchien zusammenhängt.

Üblicherweise definiert man ein **Modell** eines (komplizierten, teuren, noch nicht vorhandenen, ...) Systems S als ein einfacheres System M, das interessierende Merkmale von S wiedergibt. Gemäß dieser Definition ist

– ein Analysedatenmodell ein Modell eines Entwurfsdatenmodells,
– ein Entwurfsdatenmodell ein Modell der resultierenden Typdefinitionen in einer Programmiersprache,
– der Quellcode ein Modell des bei der Übersetzung entstehenden Maschinen- oder Bytecodes bzw. des letztlich entstehenden lauffähigen Systems.

Wenn wir unter einem Metamodell generell das "Modell eines Modells" verstehen würden, dann folgte aus den vorstehenden Aussagen:

– ein Analysedatenmodell ist ein Modell eines Modells der Typdefinitionen, somit also ein *Metamodell* der Typdefinitionen.
– Wenn wir das Entwurfsdatenmodell in unserem Entwicklungsprozeß weglassen würden, wäre das Analysedatenmodell nur noch ein *Modell* der Typdefinitionen.
– Wenn wir den Entwurfsvorgang in zwei Schritte aufteilen, wird unser Entwurfsdatenmodell zu einem *Meta-Metamodell* der Typdefinitionen.

Die vorstehenden Beispiele zeigen, daß man *schrittweise Verfeinerungen bzw. Entwicklungsstufen eines Systems nicht sinnvoll als Metamodell-Ebenen auffassen kann*. Die Zahl der Entwicklungsstufen bzw. Transformationsschritte hängt nur vom individuellen Entwicklungsprozeß ab und ist insofern völlig willkürlich. Die Zahl der Entwicklungsstufen ist nicht an den Modellen selber erkennbar. Ferner ist die Zahl der Entwicklungsstufen für unterschiedliche Zieldokumente, z.B. Typdefinitionen der transienten bzw. persistenten Darstellungen derselben Daten, unterschiedlich.

Alle Modelle, die schrittweise Entwicklungsstufen eines Systems sind, modellieren das gleiche Endprodukt und haben daher *den gleichen Gegenstandsbereich*! Entwicklungsstufen von Modellen sind daher völlig orthogonal zu linguistischen Metaebenen, denn linguistische Metaebenen haben unterschiedliche Gegenstandsbereiche.

Entwicklungsstufen korrelieren auch nicht mit ontologischen Ebenen. Ausgangspunkt ontologischer Metaebenen ist immer eine Gesamtmenge von zu klassifizierenden Entitäten, die alle konzeptuell auf dem gleichen Niveau nebeneinander stehen. Modelle des gleichen Systems auf verschiedenen Entwicklungsstufen stehen in diesem Sinne nicht unabhängig nebeneinander, sondern überlappen inhaltlich erheblich. Klassifiziert wird allenfalls die Gesamtmenge aller denkbaren Realisierungen des Systems, das durch die gegebenen Analysemodelle spezifiziert ist; der Klassifikationsbaum würde dann aus allen Verfeinerungen bestehen, die auf der jeweils nächsten Entwicklungsstufe denkbar sind. Diese Gesamtmenge interessiert aber gar nicht, ihre Struktur ist nicht Modellierungsgegenstand.

2.3 Phasenabhängigkeit der Metamodelle eines Modelltyps

Aus den vorstehenden Betrachtungen ergibt sich unmittelbar, daß alle Modelltypen, die in Bild 1 auf der Ebene "Typdefinitionen" angeordnet sind, eigene Repräsentationsformen benötigen, also insb. dann, wenn man die Modelle selber maschinell verarbeiten muß, *eigene (linguistische) Metamodelle* benötigen!

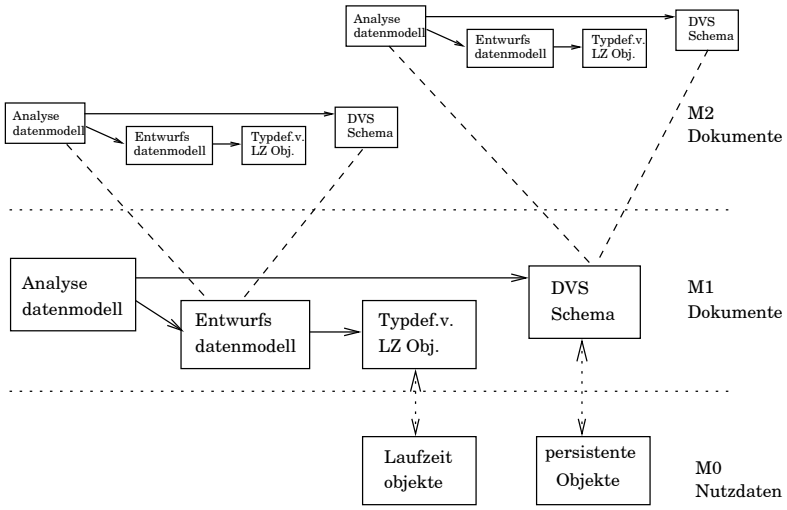

Abbildung 2: Phasenabhängige Metamodelle von Modellen

Wir betrachten als Beispiel eines Modells auf Ebene M1 der OMG-Modellhierarchie ein Entwurfs-Datenmodell, s. Bild 2. Man könnte glauben, mit einem einzigen (M2-) Metamodell für diesen (M1-) Modelltyp auszukommen. Dies trifft aber nicht zu: wir benötigen

1. *abstraktere* Metamodelle (also Analyseklassendiagramme), die man in den frühen Phasen der Entwicklung von Modellierungswerkzeugen einsetzen wird; derartige Metamodelle werden z.B. in der Spezifikation der UML [13] eingesetzt;
2. *implementierungsnahe* Metamodelle für *transiente* Repräsentationen von Modellen als Laufzeitobjekte in einer bestimmten Programmiersprache, z.B. in Modelleditoren;
3. *implementierungsnahe* Metamodelle für die *persistente* Repräsentationen von Modellen in Dateien oder Datenbanken (üblicherweise als Schema bezeichnet), z.B. zum Dokumentaustausch oder zur Archivierung.

Im Prinzip liegen die gleichen Verhältnisse wie in Bild 1 gezeigt vor, nur ist dort der Begriff "Nutzdaten" der Ebene M0 zu ersetzen durch "Repräsentationen von Modellen", also die Nutzdaten von Modelleditoren. Modelle sind, wenn sie z.B. in MDD-Ansätzen maschinell verarbeitet werden, völlig normale Daten, zu deren Modellierung und Implementierung die üblichen Entwicklungsschritte zu durchlaufen sind. Diese Erkenntnis ist eigentlich trivial, wird in der Literatur aber weitgehend übersehen.

Übersehen wird ferner meist auch, daß Daten zusammen mit den Operationen auf den Daten entwickelt werden sollten; Operationen auf Modellen sind vor allem die Editierkommandos von Modelleditoren; auf diese gehen wir anschließend noch näher ein.

3 Pseudodifferenzen infolge implementierungsnaher Metamodelle

In der Einleitung genannt war als ein Beispiel für Pseudodifferenzen ein Paar von XML-Dateien, die die gleichen Entitäten und Beziehungen repräsentieren – die Datei könnte ein Modell repräsentieren – und die trotzdem umfangreiche textuelle Differenzen aufweisen. Der Begriff Pseudodifferenz unterstellt dabei das in Bild 3 gezeigte Szenario: Es liegen zwei technologiespezifische Repräsentationen (spätphasige Modelle) R1 und R2 vor, die das "gleiche" abstrakte Modell repräsentieren und die trotzdem Unterschiede aufweisen. Allgemeiner ist das Szenario auf separat darstellbare, identische Teile von zwei verschiedenen abstrakten Modellen zu beziehen. Eine der beiden Repräsentationen kann durch Anwendung der Transformationsfunktion *trf* entstanden sein, die andere z.B. durch Editiervorgänge, die im Endeffekt nichts verändert haben, z.B. Laden und unverändertes Abspeichern in einem Editor. R1 und R2 müssen keine Versionen voneinander sein, sondern können unabhängig entstanden sein.

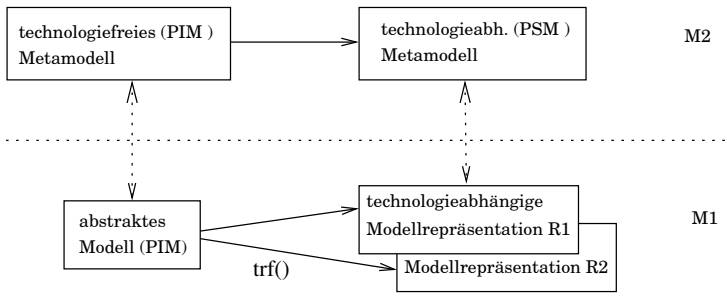

Abbildung 3: Szenario einer Pseudodifferenz

Wir unterstellen, daß die abstrakten Modelle als abstrakte Syntaxgraphen angesehen werden können. Die technologiespezifischen Implementierungen dieser abstrakten Syntaxgraphen enthalten Details, die konzeptuell nicht auftreten und die Pseudodifferenzen verursachen können. Details hierbei hängen stark davon ab, wie (un-) geschickt die Transformationsfunktion *trf* gestaltet ist und wie die jeweiligen Werkzeuge arbeiten. Häufig entstehen Pseudodifferenzen bei der Implementierung folgender konzeptueller Strukturen:

- Beziehungen (Kanten im abstrakten Syntaxgraph): Diese werden in persistenten Repräsentationen durch zwei gleiche Werte bzw. in transienten Repräsentationen oft durch zwei gegenläufige Referenzen repräsentiert. Die Datenwerte sind beliebig austauschbar, die Werte dieser Referenzen sind zufällig. Ein Austausch führt zu zwei lokalen Pseudodifferenzen.
- ungeordnete Kollektionen von Modellelementen, die im abstrakten Syntaxgraph durch ein Wurzelobjekt und von dort ausgehende Kanten eines bestimmten Typs bestimmt werden: In XML-Dateien sind Kollektionen durch die Dokumentreihenfolge geordnet, durch unterschiedliche Anordnungen können hier Pseudodifferenzen entstehen.
- Analog gilt für geordnete Kollektionen, daß die Ordnung in relationalen Tabellen oder anderen Strukturen, die per se ungeordnet sind, explizit implementiert werden muß,

z.B. durch laufende Nummern, und hier bei der Neuvergabe der Nummern Pseudodifferenzen entstehen können.

Vermeidung von Pseudodifferenzen. Pseudodifferenzen sind normalerweise unerwünscht. Es gibt zwei grundlegende Methoden, um sie zu vermeiden:

1. normierte Darstellungen in den implementierungsnahen Modellrepräsentationen,
2. Rekonstruktion der frühphasigen Modelle und Vergleich auf deren Basis.

Die erste Methode ist nur unter speziellen Randbedingungen, auf die hier nicht eingegangen werden soll, realisierbar. U.a. muß jedes abstrakte Modellelement einen universell eindeutigen Identifizierer haben, um eindeutige Darstellungen von Beziehungen zu ermöglichen. Ferner müssen irrelevante und relevante Teile der Modelle unterscheidbar sein, was bei der Gestaltung der spätphasigen Metamodelle beachtet werden muß. Verglichen werden die spätphasigen Modelle, die Vergleichsfunktion wird aber dahingehend modifiziert, Unterschiede in den irrelevanten Teilen auszublenden.

Die zweite Methode kann bei transienten Repräsentationen von Modellen oft durch passende Interfaces oder Adapter zu den ohnehin vorhandenen Modellrepräsentationen realisiert werden, die überflüssige Teile ausblenden. Das Kopieren des Modells kann dann vermieden werden. Die zweite Methode empfiehlt sich besonders für persistente Repräsentationen von Modellen in XML-Dateien. Standardverfahren zum Vergleich von Texten versagen hier weitgehend, wenn spezielle Vergleichsverfahren implementiert werden, müssen die Dateien ohnehin in transiente Darstellungen eingelesen werden und können dabei konvertiert werden.

In vielen Modelleditoren kann man den "Konstruktionsfehler" beobachten, daß einfach die komplette transiente Modellrepräsentation mit Standardverfahren in eine XML- (bzw. XMI-) Darstellung konvertiert wird. Das Resultat enthält dann nicht nur die XML-spezifische Besonderheiten, sondern auch noch Zufälligkeiten aus der transienten Repräsentation und schlimmstenfalls noch Hilfsdaten für interne Editorfunktionen.

4 Nichtatomare Implementierungen konzeptueller Editieroperationen

Die Metamodelle der UML und anderer Modellierungssprachen sind reine Datenmodelle. Implizit unterstellt wird, daß die (M1-) Modelle als abstrakte Syntaxgraphen repräsentiert werden. Die Metamodelle beschreiben nur noch die Typen der Knoten, Kanten, Attribute und weitere Details dieser Graphen.

Modelle modellieren jedoch eigentlich nicht nur Daten, sondern Systeme, die Funktionen anbieten. Als Funktionen implizit vorausgesetzt werden die elementaren Graphoperationen, namentlich das Erzeugen und Löschen von Knoten und Kanten oder das Setzen von Attributen.

Bei den meisten Modelltypen und Typen von Modellelementen der UML ist jede zulässige elementare Operation im abstrakten Syntaxgraphen, sofern sie im Rahmen von OCL-

Constraints überhaupt zulässig ist, ein sinnvoller Editierschritt. Beim Vergleich zweier abstrakter Syntaxgraphen, in denen ein Modellelement vorhanden ist bzw. fehlt, ist der Rückschluß möglich, daß es erzeugt bzw. gelöscht worden ist.

Nichtatomare Implementierungen elementarer Operationen im abstrakten Syntaxgraphen. Der vorgenannte einfache Rückschluß ist bei spätphasigen Modellen leider nicht immer möglich. Ein Beispiel ist das Erzeugen oder Löschen einer Kante im abstrakten Syntaxgraphen: bei vielen Implementierungen der abstrakten Syntaxgraphen führt dies zu zwei lokalen Unterschieden in den spätphasigen Modellen. Je nach Modelltyp und Implementierung der Kanten können sich auch mehr als zwei Unterschiede in den spätphasigen Modellen ergeben. Wenn man nun einen dieser lokalen Unterschiede als echt, also Repräsentanten der konzeptuellen Änderung ansieht, sind alle anderen "unecht". Bei mehreren zusammenhängenden Änderungen im abstrakten Syntaxgraphen können komplizierte Konstellationen "unechter" Änderungen in den spätphasigen Modellen entstehen.

Nichtatomare Operationen im abstrakten Syntaxgraphen. Darüber hinaus sind manche elementaren Operationen im abstrakten Syntaxgraphen für sich alleine nicht zulässig bzw. sinnvoll. Ein Beispiel sind Assoziationen in Klassendiagrammen. Im abstrakten Syntaxgraphen wird eine Assoziation durch einen Knoten für die Assoziation und wenigstens zwei Knoten für die Assoziationsenden und eventuell weitere Knoten repräsentiert; hinzu kommen diverse Kanten. Aus Benutzersicht sind einzelne Änderungen an diesen Knoten und Kanten nicht sinnvoll[3], sinnvolle Editieroperationen sind beispielsweise

- das Anlegen bzw. Löschen einer ganzen Assoziation, wofür ein ganzer Teilbaum im abstrakten Syntaxgraphen angelegt bzw. gelöscht werden muß
- das Ändern der Klasse, die eine Rolle einnimmt.

Die Menge der nichtatomaren Operationen ergibt sich in solchen Fällen *nicht automatisch aus den Datenstrukturen*, sondern muß in einer bewußten Entwurfsentscheidung festgelegt werden. Weitere Beispiele in Zustandsmodellen, bei denen elementare Operationen im abstrakten Syntaxgraphen nicht verwendet werden können, sind in [9] beschrieben.

Wenn man die Wurzel des Teilbaums, der eine Assoziation implementiert, als Repräsentanten der Assoziation ansieht und die Löschung dieses Knotens als Löschung der Assoziation, sind alle anderen dadurch implizierten Löschungen von Knoten und Kanten in diesem Teilbaum "unecht". Ähnliche Fälle treten bei allen konzeptuellen UML-Modellelementen auf, die durch mehrere Knoten und Kanten repräsentiert werden. OCL-Constraints, die ein isoliertes Ändern einzelner Knoten oder Kanten verbieten, verursachen ebenfalls implizierte Änderungen; Beispiele hierfür sind diverse *subsets*-Constraints zwischen Mengen von Kanten.

Man kann den Begriff Pseudodifferenz auf die vorgenannten "unechten" Änderungen ausdehnen. Hauptmerkmal der Situation ist, daß eine atomare Operation auf der Ebene von Benutzerkommandos oder im abstrakten Syntaxgraphen nur durch mehrere lokale Operationen auf der nächsttieferen Ebene (abstrakter Syntaxgraph bzw. spätphasiges Modell)

[3] Ferner führen sie zu inkonsistenten abstrakten Zuständen, die kein gültiges Modell mehr darstellen.

realisiert werden kann und nicht jede Änderung auf der unteren Ebene als Repräsentant einer konzeptuellen Änderungen gewertet werden kann.

5 Suboptimale Differenzen

Über die bisher diskutierten strukturellen Ursachen hinaus können "belanglose" bzw. uninteressante Anteile von Differenzen durch "suboptimale" Differenzen entstehen. Details hängen von der Methode ab, wie Differenzen bestimmt werden und welche "natürliche" Repräsentation von Differenzen sich daraus ergibt:

1. **Zustandsbasierte Verfahren** vergleichen die Zustände der beiden Modelle. Sie bestimmen zunächst "gleiche", also korrespondierende Modellelemente, die den Durchschnitt der Elementmengen darstellen. Im Prinzip liegt hier eine symmetrische Differenz vor, die als Tabelle von Korrespondenzen zwischen Modellelementen und Einfügungen in die gemeinsamen Teile darzustellen ist.
2. **Protokollbasierte Verfahren**: diese zeichnen Editierkommandos in Editoren auf.

Ein Optimierungsproblem bei zustandsbasierten Verfahren entsteht durch komplexe Editierkommandos. Ein Beispiel hierfür in einem Zustandsmodell ist das Verschieben eines Zustands in eine innere Region eines anderen Zustands. Diese Änderung kann auch (umständlich) durch Benutzerkommandos realisiert werden, die den alten Zustand löschen und ihn in der inneren Region neu anlegen. Bei einem Zustandsvergleich findet man zunächst genau diese elementaren Editierschritte. Aus Benutzersicht ist diese Differenzdarstellung aber sehr ungeschickt und mit uninteressanten Details belastet. Mit einer Verschiebung kann man die Änderung wesentlich besser darstellen.

Bei protokollbasierten Verfahren kann z.B. ein Modellelement zuerst erzeugt und später wieder gelöscht worden sein, d.h. auch hier können uninteressante Teile auf treten.

Bei beiden Methoden, Differenzen zu bestimmen, treten jeweils eigene Fälle auf, in denen die Differenz umständlich bzw. mit verzichtbaren Details belastet erscheint, also durch eine bessere Differenz ersetzt werden sollte; Details können hier aus Platzgründen nicht diskutiert werden. Solche suboptimalen Differenzen kann man allenfalls noch am Rande unter dem Begriff Pseudodifferenz subsumieren, sie sind letztlich Sonderfälle eines allgemeineren Optimierungsproblems und sie haben strukturell andere Ursachen als die vorher diskutierten Arten von Pseudodifferenzen.

6 Zusammenfassung

Wenn man Modelle vergleicht, können sie technisch nur in einer technologiespezifischen persistenten oder transienten Repräsentation vorliegen. Hierbei kommt es regelmäßig zu Pseudodifferenzen, also lokalen Unterschieden, die als irrelevant angesehen werden. Hauptziel dieses Papiers ist eine Klassifizierung der Ursachen für diese Unterschiede.

Die wichtigste Ursache für Pseudodifferenzen sind technologiespezifische Anteile in den Repräsentationen der Modelle. Die zugehörigen Metamodelle kann man als technologiebehaftet oder "spätphasig" bezeichnen, weil sie in den späten Entwicklungsphasen von Werkzeugen, die Modelle verarbeiten, benötigt werden. Für jeden M1-Modelltyp existieren daher mehrere technologiefreie bzw. -behaftete Metamodelle. Unterschiedliche M1-Modelltypen haben im Prinzip immer eigene Metamodelle.

Pseudodifferenzen zwischen Modellen stören, interessiert ist man nur am Vergleich des "konzeptuellen" Inhalts der Modelle. Letzteren kann man in den meisten Fällen als abstrakten Syntaxgraphen auffassen. Man muß also von den Unterschieden in den technologiespezifischen Repräsentationen auf konzeptuelle Unterschiede zurückschließen. Wenn die Metamodelle geschickt gewählt sind, findet man eindeutige Repräsentanten der Knoten, Kanten und Attribute des abstrakten Syntaxgraphen; von Änderungen an diesen Repräsentanten kann auf konzeptuelle Änderungen zurückgeschlossen werden.

In manchen Fällen stellen indes elementare Änderungen im abstrakten Syntaxgraphen keine sinnvollen Editieroperationen dar; hier ist eine weitergehende Abstraktion erforderlich, die über die reine Datenmodellierung hinausgeht, nämlich die Definition eines Editierdatentyps, der die zulässigen Operationen auf abstrakten Modellen festschreibt. Das Problem, geeignete Repräsentanten für die Durchführung von Editieroperationen zu finden, stellt sich hier erneut, aber eine Abstraktionsstufe höher.

Literatur

[1] Atkinson, Colin; Kühne, Thomas: Rearchitecting the UML Infrastructure; ACM Transactions on Modeling and Computer Simulation 12:4, p.290-321; 2002

[2] Atkinson, Colin; Kühne, Thomas: Model-Driven Development: A Metamodeling Foundation; IEEE Software 20:5, p.36-41; 2003

[3] Bézivin, Jean: On the Unification Power of Models; Software and Systems Modeling 4:2, p.171-188; 2005

[4] Ebert, Jürgen; Kelter, Udo; Systä, Tarja: Proc. 2008 Intl. Workshop on Comparison and Versioning of Software Models; ACM, ISBN 978-1-60558-045-6; 2008

[5] Ebert, Jürgen; Kelter, Udo; Systä, Tarja: Proc. 2009 ICSE Workshop on Comparison and Versioning of Software Models; IEEE, ISBN 978-1-4244-3714-6; 2009

[6] Flatscher, Rony G.: Metamodeling in EIA/CDIF—Meta-Metamodel and Metamodels; ACM ToMaCS 12:4, p.322-342; 2002

[7] Hesse, Wolfgang: More matters on (meta-)modelling: remarks on Thomas Kühne's "matters"; Journal on Software and Systems Modeling 5:4, p.387-394, December 2006; Springer; 2006

[8] Imber, Mike: The CASE Data Interchange Format (CDIF) standards; p.457-474 in: Software Engineering Environments; Ellis Horwood; 1991 ISBN 0-13-832601-0

[9] Kelter, Udo; Schmidt, Maik: Comparing State Machines; p.1-6 in [4]

[10] Kühne, Thomas: Matters of (Meta-) Modeling; Journal on Software and Systems Modeling 5:4, p.369-385, December 2006; Springer; 2006

[11] MDA Guide Version 1.0.1; OMG, Doc. omg/2003-06-01; 2003

[12] Meta Object Facility (MOF) Core Specification, Version 2.0; OMG, formal/06-01-01; 2006

[13] Unified Modeling Language: Superstructure, Version 2.0; OMG, Doc. formal/05-07-04; 2006