Strukturbezogener Vergleich von Modellversionen mit graphbasierten Optimierungsalgorithmen

Sabrina Uhrig[1] und Bernhard Westfechtel[2]

Abstract: Der Einsatz von modellgetriebener Entwicklung in der industriellen Praxis setzt insbesondere voraus, dass die Versionskontrolle für Modelle adäquat unterstützt wird. In diesem Zusammenhang spielen Algorithmen zum Vergleich von Modellversionen eine zentrale Rolle. Die bisher entwickelten Algorithmen sind Heuristiken, die teilweise eindeutige Objektbezeichner voraussetzen. In diesem Aufsatz beschreiben wir ein von eindeutigen Objektbezeichnern unabhängiges, strukturbezogenes Verfahren zum Vergleich von Modellversionen. Der Vergleich wird auf ein Optimierungsproblem abgebildet, das mit einem graphbasierten Algorithmus gelöst wird. Der Algorithmus ist als Bestandteil eines EMF-basierten Rahmenwerks implementiert, das die Integration und Evaluation alternativer Vergleichsalgorithmen unterstützt.

1 Einleitung und Abgrenzung zu verwandten Arbeiten

Modellgetriebene Softwareentwicklung zielt darauf ab, den Aufwand zur Erstellung komplexer Softwaresysteme mit Hilfe von ausführbaren Modellen zu reduzieren, die auf einer höheren Abstraktionsebene liegen als Programmcode. Die Anwendung modellgetriebener Softwareentwicklung in der industriellen Praxis wird u.a. dadurch gehemmt, dass die Versionskontrolle für Modelle noch nicht ausreichend unterstützt wird. Defizite bestehen insbesondere hinsichtlich der Verfahren zum Vergleich und zum Mischen von Modellversionen [FW07, BE09].

In diesem Aufsatz beschränken wir uns auf den *Vergleich* von Modellversionen. Traditionelle *textbasierte Verfahren*, wie sie seit langer Zeit in Systemen zur Softwarekonfigurationsverwaltung eingesetzt werden [HS77], haben sich als ungeeignet erwiesen, da sie zeilen- und nicht strukturbasiert arbeiten und ein Vergleich auf der Ebene von Textrepräsentationen, die als Speicherformat dienen, keine sinnvollen Resultate auf konzeptioneller Ebene liefert. Dies hat die Entwicklung von *strukturbezogenen Verfahren* motiviert, die jedoch bisher unter folgenden Beschränkungen leiden:

Viele Verfahren setzen *eindeutige Objektbezeichner* voraus, um “gleiche” Objekte miteinander zu identifizieren [MGH05, OWK03, RW98, AP03]. Dies erleichtert den Vergleich erheblich, da nicht mehr nach Korrespondenzen gesucht werden muss. Auf eindeutigen Objektbezeichnern basierende Verfahren haben jedoch eine Reihe gravierender

[1] Angewandte Informatik I, Universität Bayreuth, 95440 Bayreuth, sabrina.uhrig@uni-bayreuth.de
[2] Angewandte Informatik I, Universität Bayreuth, 95440 Bayreuth, bernhard.westfechtel@uni-bayreuth.de

Nachteile. Sie sind nur innerhalb eines Werkzeugs einsetzbar, das diese Bezeichner vergibt. Beim Austausch von Modellen über Werkzeuggrenzen hinweg können die Objektbezeichner jedoch verloren gehen. Weiterhin hängt das Ergebnis des Vergleichs von der Ediergeschichte ab. Schließlich liefert selbst innerhalb eines Werkzeugs ein Vergleich keine sinnvollen Ergebnisse, wenn die Versionen unabhängig voneinander entstanden sind. Einige Verfahren verwenden zwar keine eindeutigen Objektbezeichner, identifizieren jedoch Objekte über vom Modellierer vergebene *Namen*. Namen werden als *Orientierungspunkte* verwendet, die den weiteren Vergleich steuern [XS05, BP08, KWN05]. Werden die Namen geändert, so schlägt die Identifikation fehl.

Alle bekannten Verfahren sind *Heuristiken* und verfolgen dabei unterschiedliche Strategien: Bei SiDiff [KWN05] sowie UMLDiff [XS05] erfolgt das Matching auf der Basis lokaler Entscheidungen mit Hilfe heuristischer Ähnlichkeitsfunktionen. Die wesentlichen Unterschiede bestehen in der Abarbeitungsreihenfolge und im verwendeten Datenmodell. Der in SiDiff verwendete Baum aus Kompositionsbeziehungen erweitert durch Querverweise wird zunächst bottom-up durchlaufen, wobei die paarweisen Ähnlichkeitswerte für die jeweiligen Elemente eines Typs ermittelt werden. Können Elemente (typischerweise über ihre Namensattribute) eindeutig zugeordnet werden, werden die Ähnlichkeitswerte noch nicht zugeordneter Söhne angepasst, woraus sich weitere Zuordnungen ergeben können. UMLDiff hingegen verfolgt einen Top-Down-Ansatz und arbeitet auf aus Quellcode generierten UML-Klassendiagrammen, wodurch mehr Information in die Ähnlichkeitsberechnung einfließen kann. EMFCompare [BP08] ist aufgrund der Top-Down-Durchlaufstrategie dem UMLDiff-Verfahren nach eigener Auskunft am ähnlichsten, hat diesen Ansatz jedoch auf EMF-Modelle verallgemeinert. Einen völlig anderen Ansatz verfolgt Similarity-Flooding [MGMR02]. In diesem Matching-Verfahren auf Graphen mit Kantenmarkierungen werden die Ähnlichkeitswerte, ausgehend von den Ähnlichkeiten der Namen, über eine Fixpunktiteration an die Nachbarelemente propagiert. Die Ähnlichkeitswerte sind auch hier heuristisch, werden aber global berechnet, die korrespondierenden Elemente werden nach Abschluss der Berechnung über verschiedene Metriken ausgewählt. Das Verfahren wurde unseres Wissens nicht auf Klassendiagramme übertragen.

Alle diese Verfahren basieren nicht auf einem mathematisch definierten Optimierungskriterium, so dass sich über die Optimalität der berechneten Lösung keine Aussage treffen lässt. Dies erschwert die Bewertung der Verfahren erheblich, zumal keine allgemein akzeptierten Benchmarks zur Verfügung stehen. In diesem Aufsatz beschreiben wir ein strukturbezogenes Verfahren zum Vergleich von Modellversionen [Uhr08], das sich durch folgende Eigenschaften auszeichnet:

Unabhängigkeit von eindeutigen Objektbezeichnern Es werden keine eindeutigen Objektbezeichner vorausgesetzt. Das Ergebnis des Vergleichs hängt nicht von werkzeugspezifischen Zusatzinformationen, sondern nur von den logischen Inhalten der zu vergleichenden Modelle ab.

Definiertes Optimalitätskriterium Das Verfahren basiert auf einem *Kostenmodell*, das Edieroperationen auf Modellen und daraus gebildeten *Edierskripten* Kosten zuordnet. Gesucht wird ein Edierskript mit minimalen Kosten, das das Ausgangsmodell in das Zielmodell des Vergleichs überführt.

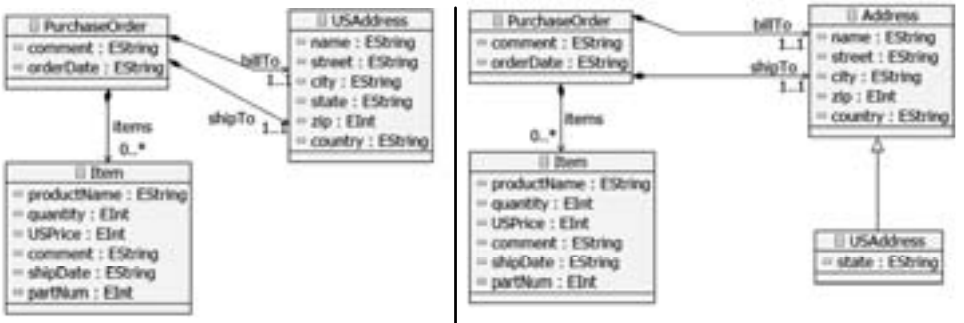

Abb. 1: Beispiel: Vergleich von Klassendiagrammen

Geringe Sensitivität gegenüber Namensänderungen Beim strukturellen Vergleich werden Namen als gewöhnliche Attribute behandelt (nicht als Orientierungspunkte). Bei hinreichend hoher struktureller Übereinstimmung werden auch unterschiedlich benannte Modellelemente miteinander identifiziert.

Verwendung von Optimierungsalgorithmen aus der Graphentheorie Der Vergleich von Modellversionen wird in ein Optimierungsproblem transferiert, das mit bekannten Algorithmen aus der Graphentheorie gelöst werden kann.

Implementierung in einem EMF-basierten Rahmenwerk Das Verfahren ist in einem EMF-basierten Rahmenwerk implementiert, das die Integration alternativer Vergleichsalgorithmen unterstützt und damit das Experimentieren mit und das Evaluieren von Algorithmen zum Vergleich von Modellversionen ermöglicht.

2 Motivation

Wie bereits erwähnt, führt das von uns entwickelte Verfahren einen strukturellen Vergleich durch, durch den Modellelemente auch dann miteinander identifiziert werden können, wenn sie unterschiedliche Namen haben (und zwar ohne Verwendung eindeutiger Objektbezeichner). In diesem Abschnitt betrachten wir nun zwei Beispiele, die den Vorteil dieses Ansatzes verdeutlichen. Abbildung 1 zeigt zwei Ecore-Klassendiagramme zur Modellierung von Kaufaufträgen. In Diagramm *A* werden Liefer- und Rechnungsadresse mit der Klasse USAddress modelliert. In Diagramm *B* wurde die Modellierung von Adressen verallgemeinert. Die Klasse Address enthält fast alle Attribute der alten Klasse USAddress, die nun als Unterklasse von Address definiert wird und das für US-Adressen spezifische Attribut state enthält.

Unser Verfahren berechnet eine minimale Distanz zwischen den beiden Diagrammen. Aufgrund ihrer strukturellen Ähnlichkeiten identifiziert es die Klasse USAddress in Diagramm *A* mit der Klasse Address in Diagramm *B* und ermittelt folgende Differenzen:

1. Umbenennen von USAddress in Address

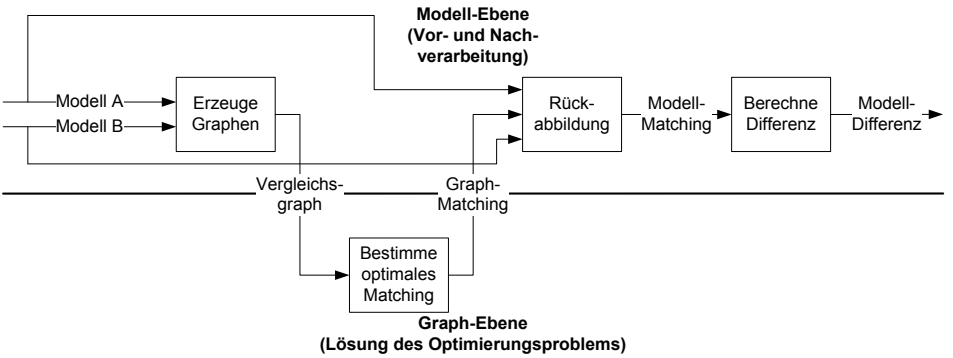

Abb. 2: Graphbasierter Ansatz zum Vergleich von Modellversionen

2. Löschen des Attributs `state`
3. Einfügen der Klasse `USAddress` als Unterklasse von `Address` mit Attribut `state`

Im Gegensatz dazu identifiziert beispielsweise EMF Compare [BP08] Klassen über den Namen und ordnet daher die mit `USAddress` benannten Klassen einander zu. Dies führt zu einer erheblich größeren Modelldifferenz, da fünf Attribute aus der Klasse `USAddress` gelöscht und in die neue Klasse `Address` eingefügt werden müssen.

Um den Effekt von Umbenennungen zu demonstrieren, haben wir in einem weiteren Beispiel ein 8 Klassen umfassendes Diagramm aus dem Englischen ins Französische übersetzt. Unser Verfahren ordnet alle Klassen korrekt einander zu. EMF Compare identifiziert dagegen lediglich drei Klassen, deren Namen zufälligerweise hinreichend ähnlich sind.

3 Ansatz

3.1 Überblick

Abbildung 2 vermittelt einen Überblick über den von uns verfolgten Ansatz zum Vergleich von Modellversionen. Zunächst wird in einem *Vorverarbeitungsschritt* aus den zu vergleichenden Modellen *A* und *B* ein *Vergleichsgraph* konstruiert. Auf den Vergleichsgraphen wird ein *generischer Graphalgorithmus* angewendet, der aus der Graphentheorie bekannt ist [BG61]. Mit Hilfe dieses Algorithmus wird ein *optimales Graph-Matching* berechnet. Die *Rückabbildung* transformiert anschließend das Matching auf der *Graph-Ebene* zurück auf die *Modell-Ebene*. Damit ist bekannt, welche Modellelemente in *A* und *B* miteinander identifiziert werden. Aus den identifizierten Elementen wird schließlich die *Modell-Differenz* berechnet, d.h. es wird ermittelt, welche Elemente aus *A* gelöscht bzw. geändert wurden und welche Elemente in *B* neu eingefügt wurden.

Der generische Graphalgorithmus zur Lösung des Optimierungsproblems hängt nicht von dem *Metamodell* ab, auf dem die zu vergleichenden Modelle basieren. Die auf der Mo-

dellebene angeordneten Schritte des Verfahrens sind dagegen vom Metamodell abhängig. Die dafür benötigten Algorithmen sind für jedes Metamodell spezifisch zu entwickeln.

3.2 Graph-Matching

Es existiert eine Fülle von *graphentheoretischen Algorithmen* für eine große Bandbreite von Problemen. Es stellt sich die Frage, welche Ansätze für den Vergleich von Modellversionen geeignet sind. Der Isomorphiebegriff auf Graphen bzw. die Suche nach einem maximalen gemeinsamen Untergraphen sind für unsere Zwecke zu restriktiv. Für den Vergleich von Modellversionen erweisen sich Ansätze zum *Graph-Matching*[KN05] als besser geeignet. Wir betrachten hier folgende Problemvariante:

Optimierungsproblem 1 (Maximales Graph-Matching mit minimalen Kosten) *Gegeben sei ein bipartiter Graph $G = (V, E)$ mit Knotenmenge $V = V_A \cup V_B$, wobei $V_A \cap V_B = \emptyset$, und Kantenmenge $E \subseteq V_A \times V_B$. Ferner sei $c : E \to \mathbb{R}^+$ eine Kostenfunktion, die jeder Kante nichtnegative reellwertige Kosten zuordnet.*

Gesucht ist ein Matching $M \subseteq E$ mit folgenden Eigenschaften:

1. *Jeder Knoten aus V ist zu höchstens einer Kante aus M inzident.*
2. *M ist maximal mit dieser Eigenschaft, d.h. es gibt kein anderes Matching mit höherer Kardinalität.*
3. *M hat minimale Kosten, d.h. für jedes andere maximale Matching M' gilt $\sum_{e \in M} c(e) \leq \sum_{e \in M'} c(e)$.*

Der Vergleich von Modellen lässt sich auf ein Graph-Matching-Problem abbilden. Dazu konstruiert man für zwei Modelle A und B einen Graphen, dessen Knotenmenge in zwei Partitionen V_A und V_B zerfällt. Zunächst fügt man für die Modellelemente aus A und B entsprechende Knoten in V_A und V_B ein. Für jede mögliche Zuordnung (*Ersetzung*) von Modellelementen fügt man anschließend eine Kante in E ein, deren Kosten sich aus den Operationen errechnet, mit denen man das Quellelement in das Zielelement überführt. *Erzeugungen* von Modellelementen in B werden modelliert, indem man für jedes Modellelement aus B einen ε-Knoten (für ein nicht existierendes Modellelement) in V_A einfügt. Der ε-Knoten wird mit dem Knoten aus V_B durch eine Kante verbunden, der die Kosten für das Erzeugen des Modellelements zugeordnet werden. Schließlich behandelt man das *Löschen* von Modellelementen auf analoge Weise.

3.3 Anwendung auf den Modellvergleich

Ein Algorithmus zum Modellvergleich soll *effizient* sein und ein Ergebnis liefern, das einerseits *optimal* im Sinne von Problem 1 ist und andererseits den Anforderungen des Benutzers entspricht (*Validität*).

Effizienz *Effiziente Algorithmen* zur Lösung des Optimierungsproblems 1 sind aus der Literatur bekannt [Jun08]. Der von uns verwendete Algorithmus hat polynomielle Laufzeit (Abschnitt 3.4).

Optimalität Hinsichtlich der *Optimalität des Ergebnisses* ergibt sich folgendes Problem: Im Optimierungsproblem 1 wird vorausgesetzt, dass sich den Kanten des bipartiten Graphen die Kosten *statisch* zuordnen lassen, d.h. die Kosten werden berechnet, bevor der Algorithmus gestartet wird. Beim Modellvergleich sind jedoch die Kosten insofern *dynamisch*, als sie von der Abbildung des *Kontextes* eines Modellelements abhängen.

Bildet man beispielsweise Klassen aufeinander ab, so hängen die Edierkosten nicht nur von den lokalen Eigenschaften dieser Klassen ab (d.h. von den Attributen und Operationen), sondern auch von der Abbildung der Kontextklassen, die über Assoziationen oder Vererbungsbeziehungen verbunden sind. Diese Beziehungen lassen sich nur dann aufeinander abbilden, wenn ihre Enden miteinander identifiziert werden.

Dieses Problem lässt sich lösen, indem man für das Graph-Matching statt der (unbekannten) *realen Kosten* c_{real} untere Schranken (*Mindestkosten*) c_{lb} verwendet:

$$c_{lb} \leq c_{real} \tag{1}$$

Die Mindestkosten werden bestimmt, indem man zwischen *kontextabhängigen* und *kontextfreien Kosten* unterscheidet und die kontextabhängigen Kosten unter der optimistischen Annahme berechnet, dass der Kontext eines Modellelements "passend" abgebildet wird. Insgesamt führt dies zu folgendem Vorgehen:

1. Das Optimierungsproblem 1 wird für die Mindestkosten gelöst.
2. Für das berechnete Matching werden die realen Kosten und die Mindestkosten berechnet:

$$C^*_{real} = \sum_{e \in M} c_{real}(e) \tag{2}$$

$$C^*_{lb} = \sum_{e \in M} c_{lb}(e) \tag{3}$$

3. Das Optimierungsproblem für die realen Kosten ist gelöst, wenn die realen und die Mindestkosten übereinstimmen:

$$C^*_{real} = C^*_{lb} \tag{4}$$

Gleichung 4 liefert somit ein *hinreichendes Optimalitätskriterium*. Leider ist dieses Kriterium nicht zugleich notwendig: Die optimale Lösung kann höhere Kosten haben als die berechneten Mindestkosten. Falls $C^*_{real} > C^*_{lb}$, ist also nicht bekannt, ob die berechnete Lösung optimal ist. Wir kommen auf dieses Problem in Abschnitt 6 noch einmal zurück.

Validität In unserem Verfahren werden wie in vielen anderen Ansätzen Differenzen mit Hilfe von *Edierskripten* dargestellt. Die Kosten eines Edierskripts werden durch Summation der Kosten der einzelnen Operationen ermittelt. Dabei werden komplexe Operationen (z.B. das Löschen einer Klasse) auf Elementaroperationen (Löschen von Attributen und Methoden) zurückgeführt und — im einfachsten Fall — Elementaroperationen gleich gewichtet (*Einheitskostenmodell*).

Auch bei einer erfolgreichen Lösung des Optimierungsproblems ist nicht sichergestellt, dass das auf diese Weise berechnete Ergebnis die Anforderungen des Benutzers erfüllt (*Validierung*). Um dies zu erreichen, bieten sich folgende Maßnahmen an:

Schwellenwerte für Zuordnungen Ist man an einem Edierskript mit minimalen Kosten interessiert, so ist die Änderung eines Modellelements (z.B. einer Klasse) kostengünstiger, als das Quellelement zu löschen und das Zielelement einzufügen. Sind die Modellelemente jedoch nicht "hinreichend ähnlich", so ist es sinnvoller, eine Zuordnung zu vermeiden, da der Benutzer die Differenz nicht (nur) auf der Ebene von Elementaroperationen bildet. Nicht angemessene Zuordnungen lassen sich z.B. über *Schwellenwerte* für die Ähnlichkeiten ausschließen.

Unterschiedliche Behandlung von Elementaroperationen Es kann sinnvoll sein, von der Gleichgewichtung aller Elementaroperationen) abzuweichen. So spielen etwa *Namensattribute* eine besondere Rolle bei der Zuordnung von Modellelementen. Dies kann beispielsweise berücksichtigt werden, indem man das Ändern von Namensattributen stärker gewichtet. Ferner kann sich eine feingranulare Analyse der Änderungen durch einen Vergleich von Zeichenketten empfehlen, um zu erreichen, dass Modellelemente mit ähnlichen Namen mit größerer Wahrscheinlichkeit identifiziert werden.

3.4 Lösung des Optimierungsproblems

Zur Lösung des Optimierungsproblems 1 reduzieren wir dieses auf die Berechnung eines *maximalen kostenminimalen Flusses* in einem Netzwerkgraphen ([NM02], Kapitel 2). Ein *Netzwerkgraph* ist ein gerichteter Graph $N = (V, E)$, der eine ausgezeichnete Quelle r und eine ausgezeichnete Senke s enthält. Den Kanten $e = (i, j)$ sind als Minimal- und Maximalkapazitäten natürliche Zahlen $\lambda_{ij} \leq \kappa_{ij}$ sowie nichtnegative reellwertige Kosten c_{ij} zugeordnet. Die Flüsse werden mit ϕ_{ij} bezeichnet. Im Folgenden setzen wir $\lambda_{ij} = 0$ voraus (die Minimalkapazität verschwindet bei den von uns konstruierten Netzwerkgraphen). Das Optimierungsproblem lässt sich dann wie folgt formulieren:

Optimierungsproblem 2 (Maximaler kostenminimaler Fluss) *Minimiere für den maximal erreichbaren Fluss f die Kosten*

$$\sum_{(i,j)\in E} c_{ij}\phi_{ij} \tag{5}$$

so, dass der Fluss balanciert ist

$$\sum_{\{j:(i,j)\in E\}} \phi_{ij} - \sum_{\{k:(k,i)\in E\}} \phi_{ki} = \begin{cases} f & \text{if i = r} \\ -f & \text{if i = s} \\ 0 & \forall\, i \in V \backslash \{r,s\} \end{cases} \tag{6}$$

und die Kapazitäten der Flusskanten einhält:

$$0 \leq \phi_{ij} \leq \kappa_{ij},\ (i,j) \in E \tag{7}$$

Es seien n und m die Zahlen der Modellelemente in A und B; o.B.d.A. nehmen wir $n \geq m$ an. Der Netzwerkgraph wird nach folgenden Regeln konstruiert (unter der vereinfachenden Annahme, dass sich alle Modellelemente in A und B aufeinander abbilden lassen):

1. Erzeuge einen Quellknoten r und einen Zielknoten s.
2. Füge für alle Modellelemente aus A und B entsprechende Knoten ein. Die Knotenmengen seien im Folgenden mit V_A und V_B bezeichnet.
3. Füge für jeden Knoten $a \in V_A$ eine Kante (r,a) mit Markierung $(0,1,0)$ ein (Minimalkapazität, Maximalkapazität, Kosten).
4. Füge für jeden Knoten $b \in V_B$ eine Kante (b,s) mit Markierung $(0,1,0)$ ein.
5. Füge für jedes Knotenpaar $(a,b) \in V_A \times V_B$ eine Kante mit Markierung $(0,1,c_{ab})$ ein, die die Ersetzung von a durch b mit Kosten c_{ab} repräsentiert.
6. Füge einen Knoten ε ein und erzeuge Löschkanten $(a,\varepsilon$ mit Markierung $(0,1,c_{a\varepsilon})$ bzw. Erzeugungskanten (ε,b) mit Markierung $(0,1,c_{\varepsilon b})$ für alle $a \in V_A$ und $b \in V_B$. Verbinde den Knoten ε ferner mit dem Zielknoten s mit einer Kante der Markierung $(0,n-m,0)$.

Der Netzwerkgraph ist somit i.W. ein bipartiter Graph, der um einen Quell- und einen Zielknoten ergänzt wird. Ferner enthält er einen Knoten ε, dessen einlaufende Kanten zum Löschen und dessen auslaufende Kanten zum Einfügen von Modellelementen verwendet werden. Ein maximaler Fluss hat die Stärke $|V_A|$. Allen von r ausgehenden Kanten wird ein Fluss der Stärke 1 zugeordnet. Für jedes $a \in V_A$ wird genau eine auslaufende Kante beschickt. Dadurch wird festgelegt, ob a durch ein $b \in V_B$ ersetzt oder gelöscht wird. Die Balancierungsregeln stellen ferner sicher, dass jedes $b \in V_B$ entweder neu erzeugt wird oder ein $a \in V_A$ ersetzt. Als Beispiel zeigt Abbildung 3 den Netzwerkgraphen, der für die Diagramme aus Abbildung 1 konstruiert wird (mit vertauschten Rollen von A und B).

Zur Lösung des Optimierungsproblems haben wir den Algorithmus von *Busacker* und *Gowen* [BG61] implementiert (dargestellt z.B. in [NM02, Jun08]). Die Laufzeitkomplexität liegt in $O(n_V n_E f)$, wobei n_V und n_E für die Zahl der Knoten bzw. Kanten des Netzwerkgraphen sowie f für die Kapazität des berechneten Flusses stehen. Bei Netzwerkgraphen,

die nach obiger Vorschrift konstruiert werden, lässt sich die Komplexität als Funktion von $n_A = |V_A|$ ausdrücken: $|f| = n_A$, $n_V \leq 2n_A + 3$, und $n_E \leq n_A^2 + 4n_A$. Damit ergibt sich insgesamt $O(n_A^4)$ als obere Schranke für die Laufzeitkomplexität.

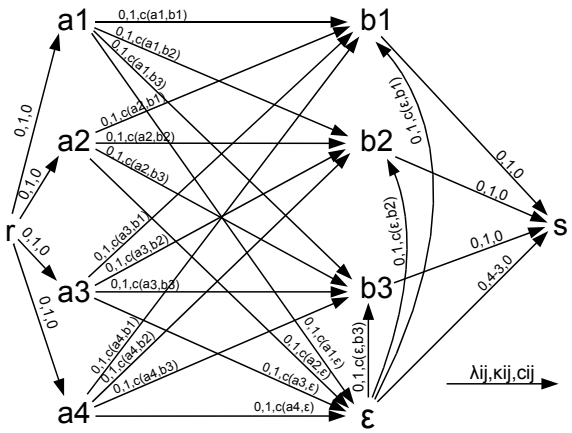

Abb. 3: Netzwerkgraph für den Vergleich der Klassendiagramme aus Abbildung 1

4 Vergleich von Klassendiagrammen

Das beschriebene Verfahren zum Vergleich von Modellversionen wurde für Ecore-Klassendiagramme implementiert. Der Graph eines Klassendiagramms wird dabei als hierarchische Struktur aus Kompositionsbeziehungen mit Querverbindungen betrachtet. Die Attribute, Operationen, Vererbungskanten und Assoziationen einer Klasse bilden jeweils einen Baum, der über die Assoziationen und Vererbungskanten wiederum mit anderen Bäumen verbunden sein kann.

Abbildung 4 zeigt das dabei verwendete Metamodell, eine Teilmenge des Ecore-Metamodells. Um das Problem in seiner Komplexität zu reduzieren, wurde der Typ von Attributen, Rückgabeparametern und Übergabeparametern vereinfacht als String behandelt. Sichtbarkeiten und der Modifizierer *static* sind in diesem Modell nicht erfasst. Ebenso wenig werden derzeit Unterpakete unterstützt. Stattdessen gehen wir davon aus, dass alle Klassen in einem Grundpaket liegen. Im folgenden Abschnitt werden die auf diesem Modell zulässigen Edieroperationen und deren Kosten behandelt.

4.1 2-Ebenen-Ansatz

Um die Kosten für die Zuordnung zweier Klassen zu berechnen, müssen auch die Kosten für die Zuordnungen derer Unterelemente berücksichtigt werden. Dabei wurden folgende Annahmen hinsichtlich der zulässigen Edieroperationen getroffen:

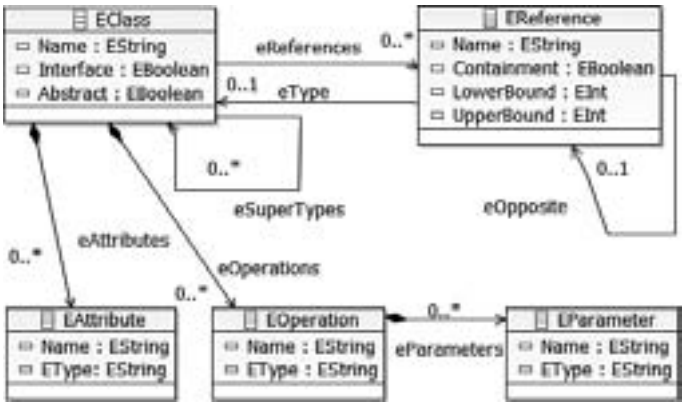

Abb. 4: Implementierter Teil des Ecore-Metamodells

Keine Typänderung Der Typ von Elementen kann nicht geändert werden, d.h. dass ein Attribut nicht in eine Methode umgewandelt werden kann. Eigenschaften jedoch können geändert werden. So kann eine Klasse zum Interface werden, indem das entsprechende *Interface*-Attribut gesetzt wird.

Vollständige Teilbaum-Zuordnung Falls die Klassen *a* und *b* einander zugeordnet werden, werden auch die Unterelemente von Klasse *a* den entsprechenden Unterelementen von Klasse *b* zugeordnet, oder gelöscht. Unterelemente von Klasse *b*, die dabei nicht zugeordnet werden konnten, werden erzeugt.

Mit diesen Annahmen beschränken sich die auf diesem Modell zulässigen Edieroperationen auf die elementaren Operationen Erzeugen und Löschen von Elementen und das Ändern von Eigenschaften. Das Verschieben von Elementen wird nicht als eigenständige Edieroperation betrachtet, sondern wird durch eine Folge von Einfüge- und Löschoperationen erzielt. Auch Assoziationen und Vererbungskanten werden jeweils als gemeinsames Unterelement der beiden beteiligten Klassen betrachtet. Daher kann das Assoziationsende einer Assoziation zwischen den Klassen *a* und *b* nicht von *b* nach *c* verschoben werden. Stattdessen muss die Assoziation zwischen *a* und *b* gelöscht und eine andere Assoziation zwischen *a* und *c* erzeugt werden. Das Gleiche gilt auch für Vererbungskanten.

Jeder zulässigen Edieroperation werden Kosten zugewiesen. Das hier verwendete Kostenmodell enthält nur positive reelle Kosten und ist symmetrisch hinsichtlich inverser Operationen. So verursacht das Löschen eines Elements die gleichen Kosten wie das Einfügen des gleichen Elements. Diese setzen sich aus den Löschkosten des Elements selbst und den Löschkosten aller Unterelemente zusammen. Die Löschkosten eines Elements selbst wiederum sind größer oder gleich der Summe der Kosten, alle Eigenschaften dieses Elements zu ändern. Insgesamt orientieren sich die Kosten an einem *Einheitskostenmodell*, bei dem jede Elementaroperation mit Kosten 1 gewichtet wird. Abweichungen davon ergeben sich nur durch nachträgliche Verfeinerungen, z.B. indem die Edieroperation zum Ändern der Multiplizität einer Assoziation in zwei Operationen zum Ändern der oberen bzw. unteren Schranke aufgeteilt wurde.

Tabelle 1 zeigt einen Ausschnitt aus dem Kostenmodell, anhand dessen nun exemplarisch die Kosten für die Edieroperationen auf Methoden erläutert werden. Falls sich die Namensattribute a und b zweier Methoden unterscheiden, werden die Kosten für die Änderungsoperation mit Hilfe der *längsten gemeinsamen Untersequenz lcs* ([HS77]) berechnet, wobei sich Edierkosten innerhalb des Intervalls $[0,1]$ ergeben (siehe Formel 8).

$$lcs = 1 - \frac{2 * lcs.length}{a.length + b.length} \tag{8}$$

Abweichungen im Rückgabetyp der Methode werden mit Kosten 1 bewertet. Beim Einfügen bzw. Löschen eines Übergabeparameters entstehen Kosten in Höhe von 2 (Addition der Einheitskosten für Name und Typ des Parameters). Wird die komplette Methode gelöscht, entstehen Kosten in Höhe von 2 für das Löschen der Methode selbst und zusätzlich 2 für jeden gelöschten Übergabeparameter dieser Methode.

EOperation	
Änderung des Namensattributs	lcs
Änderung des Rückgabetyps	1
Erzeugen/Löschen einer Methode	2 + Kosten für das Einfügen/Löschen aller Übergabeparameter
EParameter	
Änderung des Namensattributs	lcs
Änderung des Typs	1
Einfügen/Löschen eines Übergabeparameters	2

Tab. 1: Kostenmodell für Methoden

Aus dem 2-Ebenen-Ansatz ergibt sich ein hierarchischer Aufbau von Optimierungsproblemen. Für die Berechnung der Kosten für die Zuordnung zweier Klassen müssen die Kosten für die Zuordnung aller Unterelemente berücksichtigt werden. Dies beinhaltet weitere Zuordnungsprobleme für die Zuordnung der Attribute, Methoden, Assoziationen und Vererbungsbeziehungen, die nach der Übertragung in ein entsprechendes Netzwerk ebenfalls mit dem Busacker-Gowen-Algorithmus gelöst werden können. Erst nachdem die Zuordnungen der Unterelemente berechnet sind, kann auch das Optimierungsproblem auf Klassenebene gelöst werden.

4.2 Kontextabhängige Kosten und deren Abschätzung

Für die Kantenbeschriftungen des Netzwerkgraphen werden statische Kosten benötigt (vgl. Abschnitt 3), d.h. die Kosten der Zuordnung zweier Klassen a und b muss von den Zuordnungen der anderen Klassen entkoppelt sein, da sonst ein kombinatorisches Problem mit exponentiellen Aufwand entstünde. Bei den Attributen und Methoden können die Zuordnungskosten exakt berechnet werden, da es sich in diesen Fällen um lokale Eigenschaften handelt. Für die exakte Berechnung der Kosten für die Zuordnung der Assoziationen

und Vererbungskanten von a und b werden jedoch Informationen über die Kontextklassen benötigt: Da Assoziationen ebenso wie Vererbungskanten in diesem Modell nicht verschoben werden können, dürfen zwei Assoziationen von a und b im Fall der Abbildung von a auf b nur dann identifiziert werden, wenn auch deren Assoziationsenden c und d in der berechneten Lösung identifiziert werden.

Daher arbeiten wir mit abgeschätzten Kosten für die Zuordnung von Assoziationen und Vererbungsbeziehungen, die eine untere Schranke für die tatsächlichen Kosten darstellen. D.h. wir treffen die optimistische Annahme, dass in unserem obigen Beispiel die Klassen c und d einander zugeordnet werden, und berücksichtigen nur eventuelle Kosten für die Namensänderung der Assoziation oder Änderungen der Multiplizitäten. Dieses relaxierte Problem mit den Kostenabschätzungen wird nun mit dem Busacker-Gowen-Algorithmus gelöst. Sobald die Zuordnung der Klassen feststeht, können die vorher abgeschätzten Kosten für die Zuordnung der Assoziationen und Vererbungskanten exakt bestimmt werden. Falls die Klassen c und d aus unserem Beispiel doch nicht miteinander identifiziert werden, liegen die realen Kosten über den abgeschätzten Kosten.

4.3 Schwellenwerte

Wie in Abschnitt 3 bereits erwähnt wurde, kann es sinnvoll sein, auf die Zuordnung der Elemente in Form von Schwellenwerten Einfluss zu nehmen. An dieser Stelle werden nun die beiden Möglichkeiten der technischen Realisierung solcher Schwellenwerte behandelt:

1. Soll die Zuordnung der Klassen a und b auf jeden Fall verhindert werden, kann einfach die Verbindungskante zwischen a und b aus dem Netzwerkgraphen (siehe Abbildung 3) entfernt werden.

2. Falls die Zuordnung der beiden Klassen lediglich erschwert werden soll, ist es möglich, die Kosten c_{ab} der Verbindungskante zwischen a und b zu erhöhen. Wenn die Kosten der Kante erhöht werden, bis $c_{ab} > c_{a\varepsilon} + c_{\varepsilon b}$ gilt, wird die Kante ab sicher nicht in der Zuordnung enthalten sein, da es günstiger wäre, die Klasse a zu löschen und Klasse b zu erzeugen, als die Klassen aufeinander abzubilden.

In einer Implementierung des Verfahrens wurden Schwellenwerte wie in der (allgemeineren) zweiten Variante realisiert. Dabei werden die Kanten der Netzwerkgraphen zunächst mit den Kosten bewertet, die sich aus dem Kostenmodell ergeben. Im Anschluss daran erfolgt eine Iteration über alle Verbindungskanten des bipartiten Graphen auf Klassenebene, die gemäß einem vorgegebenen Kriterium gegebenenfalls die Kosten der Kanten erhöht. Das modifizierte Optimierungsproblem wird ebenfalls mit dem Busacker-Gowen-Verfahren gelöst, und für die modifizierten Kosten c'_{real} gilt:

$$c'_{real} \geq c_{real} \tag{9}$$

5 Rahmenwerk

In Abschnitt 3 haben wir einen allgemeinen graphbasierten Ansatz zum Vergleich von Modellversionen beschrieben, den wir in Abschnitt 4 auf Ecore-Klassendiagramme angewendet haben. Das dort beschriebene Matching-Verfahren wurde als Eclipse-Plugin implementiert und bildet zusammen mit weiteren Plugins des Eclipse Modeling Frameworks ([SBPM09]) ein Rahmenwerk, das die Integration alternativer Vergleichsalgorithmen unterstützt und damit das Experimentieren mit und das Evaluieren von Algorithmen zum Vergleich von Ecore-Klassendiagrammen ermöglicht (Abbildung 5). Um Ecore-Klassendiagramme zu erzeugen, können verschiedene bereits existierende auch graphische Editoren verwendet werden, wie z.B. von Ecore Tools[3]. Über die Benutzerschnittstelle unseres Plugins können die zu vergleichenden Diagramme und das zu verwendende Verfahren ausgewählt werden. Das Verfahren berechnet die Korrespondenzen, d.h. die aufeinander abgebildeten Elemente. Diese werden dann in ein EMF Match Model überführt, das aus einem Baum von Paaren identifizierter Elemente und deren identifizierten Unterelementen sowie aus einer Liste derjenigen Elemente besteht, die nicht zugeordnet werden konnten. Das EMF Match Model dient zusammen mit den Diagrammen als Eingabe für den Differenzberechnungsalgorithmus von EMF Compare, der aus den berechneten Korrespondenzen die Unterschiede ableitet und diese in einem EMF Diff Model speichert. Schließlich können die berechneten Differenzen im EMF Compare UI Editor in der Baumsicht visualisiert werden.

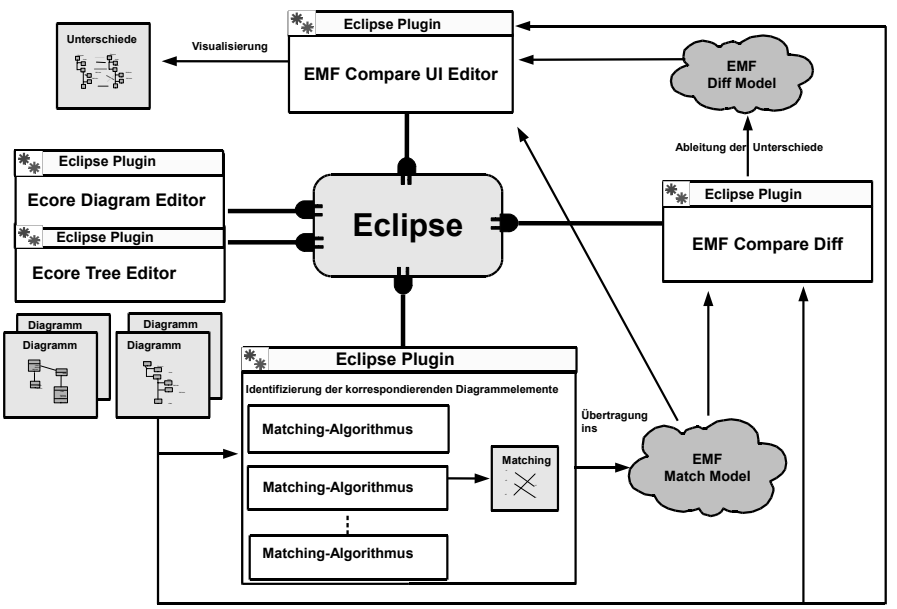

Abb. 5: Rahmenwerk für den Modellvergleich

[3] http://www.eclipse.org/modeling/emft/?project=ecoretools

6 Evaluierung

Die Vorteile des strukturbasierten Vergleichs von Klassendiagrammen, der sich nicht auf eindeutige Objektbezeichner oder die Identifikation von Namen stützt, wurde bereits in Abschnitt 2 erläutert. In diesem Abschnitt werden wir die Fragestellungen untersuchen, wie gut die untere Schranke ist, wie oft die optimale Lösung gefunden wird und in welcher Höhe die Kosten im Durchschnitt von der optimalen Lösung abweichen. Bei den Tests wurde die Version des Algorithmus eingesetzt, bei der die Zuordnung der korrespondierenden Elemente ausschließlich über das Kostenmodell gesteuert wird. Der Einsatz von Schwellenwerten kann aus Sicht des Benutzers zu nachvollziehbareren Lösungen führen, stellt jedoch einen nachträglichen Eingriff ins Kostenmodell dar.

Sind die geschätzten Kosten mit den realen Kosten identisch, so wurde mit dem Verfahren eine optimale Lösung gefunden. Auch in Fällen, in denen diese untere Schranke nicht erreicht wird, kann eine optimale Lösung gefunden worden sein, da die Erreichbarkeit der unteren Schranke nicht garantiert wird (vgl. Abschnitt 3.3). Um in diesen Fällen überprüfen zu können, ob die gefundene Lösung optimal ist, wurde eine Brute-Force-Lösung implementiert, welche die optimalen Kosten mit Aufwand $O(n!)$ ermittelt.

Die für die Evaluierung verwendeten Testmengen *A*, *B* und *C* enthalten jeweils verschiedene manuell erstellte Versionen eines Klassendiagramms. Daraus ergibt sich, dass sich die Diagramme innerhalb einer Testmenge ähnlich sind und sich die Diagramme aus verschieden Testmengen sehr stark unterscheiden. Die Testmenge *AB* ist die Vereinigung der Testmengen *A* und *B*. Die Größe der Klassendiagramme in diesem Test wurde durch die langen Laufzeiten des Brute-Force-Algorithmus limitiert. Die in Testmenge *A* enthaltenen Klassendiagramme sind die kleinsten mit bis zu 4 Klassen, Testmenge *B* enthält Klassendiagramme mit bis zu 8 Klassen, die Diagramme der Testmenge *C* bis zu 11 Klassen.

Die Ergebnisse der Tests sind in Tabelle 6 zu sehen. Die ersten beiden Spalten enthalten die verwendete Testmenge und die Anzahl der paarweise durchgeführten Vergleiche. In der 3. Spalte ist angegeben, in wieviel Prozent der Testfälle die Kosten der unteren Schranke erreicht wurden. In diesen Fällen konnte der Algorithmus eine optimale Lösung finden und verifizieren. Der Prozentsatz der tatsächlich gefundenen optimalen Lösungen steht in der 4. Spalte der Tabelle. Im Fall der Testgruppe B zeigt sich, dass obwohl die hinreichende Optimalitätsbedingung nur in 5,71% der Fälle erfüllt ist, die optimale Lösung dennoch in 96,19% der Fälle erreicht wird. Die 5. Spalte enthält die durchschnittliche Abweichung der Kosten von den Kosten der optimalen Lösung. Dabei entsprechen 100% den Kosten der optimalen Lösung. Hierbei werden in allen Testmengen gute Ergebnisse erzielt, auch in der Testmenge *AB*, in der sehr unterschiedliche Diagramme verglichen werden, liegen die durchschnittlichen Kosten nur bei 1,10 Prozentpunkten über den Kosten der optimalen Lösung. In den letzten beiden Spalten der Tabelle stehen die Laufzeiten, die auf einem Arbeitsplatzrechner (2,53 GHz, 4GB RAM) benötigt werden. In der Testmenge *AB* werden 1035 Tests in 1249 ms ausgeführt, so dass sich eine durchschnittliche Dauer von 1,12 ms pro Test ergeben. Aufgrund des polynomialen Aufwands des Verfahrens benötigt der Vergleich zweier Klassendiagramme mit 65 Klassen bzw. 80 Klassen mit 96 Änderungen im Mittel von 10 Tests immer noch lediglich 2255 ms. Im Gegensatz dazu benötigt die Brute-

Force-Lösung auf dem gleichen Rechner für zwei Klassendiagramme zu je 11 Klassen über 6 Stunden.

Die Ergebnisse dieser Evaluierung lassen sich folgendermaßen zusammenfassen: Die untere Schranke ist zwar ein relativ schlechtes Kriterium für Optimalität, dennoch findet der Algorithmus in sehr vielen Fällen eine optimale Lösung und insgesamt ist die durchschnittliche Abweichung von den optimalen Kosten gering. Darüber hinaus zeigen die Laufzeiten, dass dieses Verfahren auch in der Praxis gut einsetzbar ist. Im interaktiven Einsatz fällt die Laufzeit des Algorithmus im Vergleich zu anderen Aktionen, wie dem Aufbau des EMF Compare Editors, der die Unterschiede visualisiert, kaum ins Gewicht.

Menge	#	EC = RC	RC = BF	RC/BF	Gesamtzeit[ms]	∅[ms]
A	465	36,77%	96,77%	100,45%	191	0,41
B	105	5,71%	96,19%	100,30%	401	3,82
AB	1035	20,44%	70,58%	101,10%	1249	1,12
C	55	3,64%	60,00%	103,18%	361	6,56

Abb. 6: Ergebnisse

7 Zusammenfassung

In diesem Aufsatz wurde ein Verfahren zum Modellvergleich vorgestellt, das die Berechnung der korrespondierenden Modellelemente in ein vereinfachtes Optimierungsproblem überführt, das mit Hilfe eines graphentheoretischen Algorithmus mit polynomiellem Aufwand gelöst werden kann. Es ist möglich, die berechnete Zuordnung über die Ausgestaltung des Kostenmodells und den Einsatz von Schwellenwerten zu beeinflussen. Das Verfahren wurde bereits erfolgreich auf den Vergleich von Ecore-Klassendiagrammen übertragen und in ein EMF-basiertes Rahmenwerk zum Modellvergleich eingebettet. Die Ergebnisse der Evaluierung sind ermutigend sowohl in Bezug auf die Optimalität der berechneten Lösung als auch in Bezug auf die Laufzeiten. Im Rahmen zukünftiger Arbeiten ist es geplant, diesen generischen Ansatz auf weitere Diagrammarten anzuwenden.

Literaturverzeichnis

[AP03] Marcus Alanen und Ivan Porres. Difference and Union of Models. In *Proc. UML 2003*, LNCS 2863, Seiten 2–17, San Francisco, CA, 2003. Springer-Verlag.

[BE09] Lars Bendix und Pär Emanuelsson. Requirements for Practical Model Merge - An Industrial Perspective. In *Proc. (MODELS 2009)*, LNCS 5795, Seiten 167–180, Denver, CO, 2009. Springer-Verlag.

[BG61] R.G. Busacker und P.J. Gowen. A Procedure for Determining a Family of Minimum Cost Flow Networks. Bericht 15, John Hopkins University, Operations Research Office, Baltimore, MD, 1961.

[BP08] Cédric Brun und Alfonso Pierantonio. Model Differences in the Eclipse Modelling Framework. *UPGRADE*, IX(2):29–34, April 2008.

[FW07] Sabrina Förtsch und Bernhard Westfechtel. Differencing and Merging of Software Diagrams - State of the Art and Challenges. In *Proc. ICSOFT 2007*, Seiten 90–99, Barcelona, Spain, Juli 2007.

[HS77] J.W. Hunt und T.G. Szymanski. A Fast Algorithm for Computing Longest Common Subsequences. *Communications of the ACM*, 20(5):350–353, Mai 1977.

[Jun08] Dieter Jungnickel. *Graphs, Networks and Algorithms*. Springer-Verlag, Berlin, 2008.

[KN05] Sven Oliver Krumke und Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Leitfäden der Informatik. Teubner Verlag, Wiesbaden, 2005.

[KWN05] Udo Kelter, Jürgen Wehren und Jörg Niere. A Generic Difference Algorithm for UML Models. In *Software Engineering 2005*, LNI 64, Seiten 105–116, Essen, 2005.

[MGH05] Akhil Mehra, John C. Grundy und John G. Hosking. A generic approach to supporting diagram differencing and merging for collaborative design. In *Proc. ASE 2005*, Seiten 204–213, Long Beach, CA, 2005.

[MGMR02] S. Melnik, H. Garcia-Molina und E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and ist Application to Schema Matching. In *Proc. 18th International Conference on Data Engineering*, Seiten 117–128, San Jose, CA, 2002.

[NM02] Klaus Neumann und Martin Morlock. *Operations Research*. Carl Hanser Verlag, 2002.

[OWK03] Dirk Ohst, Michael Welle und Udo Kelter. Differences between versions of UML diagrams. In *Proc. ESEC/FSE-11*, Seiten 227–236, Helsinki, 2003.

[RW98] J. Rho und C. Wu. An Efficient Version Model of Software Diagrams. In *Proc. Asia Pacific Software Engineering Conference*, Seiten 236–243, Taiwan, 1998.

[SBPM09] Dave Steinberg, Frank Budinsky, Marcelo Paternostro und Ed Merks. *EMF Eclipse Modeling Framework*. Addison-Wesley, Upper Saddle River, NJ, 2009.

[Uhr08] Sabrina Uhrig. Matching Class Diagrams: With Estimated Costs Towards the Exact Solution? In *Proc. CVSM 2008*, Seiten 7–12, Leipzig, 2008.

[XS05] Zhenchang Xing und Eleni Stroulia. UMLDiff: an algorithm for object-oriented design differencing. In *Proc. ASE 2005*, Seiten 54–65, Long Beach, CA, 2005.