Korrespondenzberechnung auf Klassendiagrammen

Von der Universität Bayreuth
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

von

Sabrina Uhrig, geb. Förtsch

aus Nürnberg

1. Gutachter: Prof. Dr. Bernhard Westfechtel
2. Gutachter: Prof. Dr. Jürgen Ebert

Tag der Einreichung: 26.05.2011
Tag des Kolloquiums: 28.07.2011

Zusammenfassung

Für die modellgetriebene Softwareentwicklung werden Werkzeuge benötigt, die das Arbeiten mit Modellen unterstützen. Ein besondere Stellung nehmen Vergleichswerkzeuge ein, die dem Entwickler nicht nur veranschaulichen, was sich zwischen zwei Versionen eines Modells geändert hat, sondern auch Differenzen liefern, auf deren Grundlage zwei Versionen miteinander zu einem Dokument verschmolzen werden können. Vergleichsverfahren arbeiten üblicherweise in zwei Schritten. Zunächst werden die korrespondierenden Modellelemente bestimmt, bevor dann im zweiten Schritt die Unterschiede auf Basis der Korrespondenzen ermittelt werden. Die vorliegende Arbeit beschäftigt sich mit dem ersten, schwierigeren Schritt, der Korrespondenzberechnung, mit einem Schwerpunkt auf Klassendiagrammen, die das Kernstück der objektorientierten Modellierung darstellen. In diesem Rahmen wurden zwei Korrespondenzberechnungsverfahren für den Vergleich von Ecore-Klassendiagrammen entwickelt, die strukturbasiert arbeiten und keine eindeutigen Objektbezeichner verwenden.

Während die bisher bekannten Vergleichsverfahren für Klassendiagramme, die keine eindeutigen Objektbezeichner verwenden, die Korrespondenzen über Ähnlichkeitsheuristiken bestimmen, wurde für das erste Verfahren ein neuartiger Ansatz verfolgt. Analog zu der Definition eines Edierabstandes für Bäume oder Graphen wurde ein Edierkostenmodell in Verbindung mit einer Menge zulässiger Änderungsoperationen auf Klassendiagrammen entworfen. Auf diese Weise kann die Berechnung der Korrespondenzen zwischen zwei Klassendiagrammen als Optimierungsproblem dargestellt werden, eine Zuordnung mit minimalen Edierkosten zu bestimmen. Aufgrund der kontextabhängigen Kosten für die Edieroperationen auf Assoziationen und Vererbungskanten würde eine Bestimmung der optimalen Lösung die Bewertung aller $O(n!)$ verschiedenen Kombinationsmöglichkeiten der Klassenpaare erfordern. Daher wird in dem vorgestellten Verfahren stattdessen ein relaxiertes Optimierungsproblem gelöst. Mit Hilfe der Abschätzung der Kosten für Assoziationen und Vererbungsbeziehungen durch eine untere Schranke lässt sich das Optimierungsproblem auf ein Netzwerkflussproblem reduzieren, das mit Hilfe eines Verfahrens aus der Graphentheorie mit polynomiellem Aufwand gelöst werden kann. In einem Teil der Fälle lässt sich über ein leicht überprüfbares hinreichendes Optimalitätskriterium nachweisen, dass die so berechnete Lösung für das relaxierte Optimierungsproblem auch für das ursprüngliche Optimierungsproblem optimal ist. Entsprechend den Ergebnissen einer durchgeführten Evaluation weichen die berechneten Edierkosten nur wenig von den optimalen Edierkosten ab. Das Verfahren unterscheidet sich von allen heuristischen Verfahren dadurch, dass ein objektives Kriterium, die Edierkosten der berechneten Zuordnung, existiert, mit dem beurteilt werden kann, inwieweit das Verfahren die vorgegebenen Ziele erreicht hat.

Als zweites Verfahren zum Vergleich wurde ein ähnlichkeitsbasiertes, heuristisches

Verfahren entwickelt, das für die möglichen Korrespondenzpaare Ähnlichkeitswerte berechnet und mit Hilfe eines heuristischen Auswahlverfahrens die korrespondierenden Elemente über die Maximierung der Gesamtähnlichkeit bestimmt. Beide Verfahren wurden in einem Plugin für Eclipse implementiert, das zusammen mit Komponenten des Eclipse Modeling Frameworks ein Rahmenwerk zum Modellieren und Vergleichen von Ecore-Klassendiagrammen bildet. Dabei wurde der Vergleich von Klassendiagrammen mit Paketen in zwei Stufen unterteilt. Über einen Vergleich aller Klassenpaare unabhängig von deren Lage in der Pakethierarchie werden zunächst die Korrespondenzen auf Klassenebene gebildet. Darauf aufsetzend werden die korrespondierenden Pakete unter der Vorgabe der Klassenkorrespondenzen bestimmt. Das Rahmenwerk bietet eine Auswahlsicht, in der verschiedene Klassen- und Paketebenenverfahren kombiniert werden können.

Die Vergleichsergebnisse des edierkostenbasierten und des ähnlichkeitsbasierten Verfahrens wurden in einer Evaluation einander gegenübergestellt. Die vorliegende Arbeit erweitert den Stand der Technik in diesem Gebiet somit nicht nur um Korrespondenzberechnungsverfahren, sondern liefert auch Erkenntnisse über die Eignung des Edierabstandes zwischen Modellen als Kriterium für die Bewertung von Modelldifferenzen und darüber, wie sich ein edierkostenbasiertes Verfahren im Vergleich zu einem ähnlichkeitsbasierten Verfahren verhält.

Abstract

In the field of model-driven software development tools are needed which supply the work on models, especially the comparison of models. With such a tool a developer can visualize the changes between two versions of a model and use the differences to merge the two versions. Usually the comparison of models consists of two steps, the identification of the corresponding model elements and the interpretation of the differences based on the computed correspondences. The present study deals with the computation of the correspondences - which is the more difficult step of both - and focuses on class diagrams since class diagrams are the most relevant type of diagram in object-oriented modeling. In this context, two algorithms for the computation of correspondences between Ecore class diagrams have been developed. Both are structure-based and do not make use of unique object identifiers.

While all known differencing algorithms for class diagrams which do not rely on unique object identifiers use heuristic similarity values, the approach of our first algorithm is new. On the analogy of edit distance for trees or graphs, a set of edit costs and feasible edit operations have been defined for class diagrams. This way the computation of corresponding elements is treated as an optimization problem to find correspondences associated with minimal edit costs. As the costs for edit operations on association edges and inheritance edges of classes depend on how the other classes are matched, a computation of the optimal solution requires the evaluation of all $O(n!)$ possibilities to combine the class correspondences. Therefore the presented edit costs-based algorithm solves a relaxation of the original optimization problem instead. An estimation of the costs for association and inheritance edges with a lower bound enables the reduction of the problem on a network flow problem solvable with a graph algorithm in polynomial time. In some cases a sufficient optimality criterion - which can be easily verified - proves that the optimal solution for the relaxed problem holds for the original problem too. According to the results of our evaluation there is a small deviation of the computed edit costs from the optimal edit costs. This algorithm differs from all existing heuristic approaches in that there exists an objective criterion - the edit costs - to check the resulting correspondences for optimality.

A second similarity-based algorithm has been developed, in order to compare the two different approaches. The algorithm computes similarity values for all possible correspondences and finally selects the corresponding elements with the largest total similarity using a heuristic greedy strategy. Both algorithms have been implemented as an Eclipse plug-in, which - together with several plug-ins of the Eclipse Modeling Framework - form a framework for modeling and differencing Ecore class diagrams. The comparison of class diagrams which also contain packages is divided in two parts. First all classes of all packages are compared and the correspondences on the class level are determined.

Based on these correspondences the correspondences on the package level are computed. The framework offers a view to select different matching algorithms for the class and the package level.

The results of the two differencing algorithms have been compared in an evaluation. Thus the contribution of the present study consists not only in two differencing algorithms for class diagrams but also in insights on the suitability of edit distance for the comparison of models and on how an edit cost-based algorithm behaves compared to a similarity-based algorithm.

Danksagung

Mein besonderer Dank gilt meinem Doktorvater, Herrn Prof. Dr. Bernhard Westfechtel. Durch seine sehr gute Betreuung und seine zahlreichen hilfreichen Anregungen hat er wesentlich zum Gelingen dieser Dissertationsschrift beigetragen. Ebenso möchte ich mich bei Herrn Prof. Dr. Jürgen Ebert von der Universität Koblenz-Landau für die Übernahme des Zweitgutachtens bedanken.

Mein Dank gilt weiterhin meinen ehemaligen und aktuellen Kollegen am Lehrstuhl, namentlich Dr. Bernhard Daubner, Dr. Thomas Buchmann, Dr. Alexander Dotor sowie auch Sabine Winetzhammer. Besonders bedanke ich mich bei Dr. Bernhard Daubner, der mich dazu ermutigt hat, als Diplom-Wirtschaftsmathematikerin ein Promotionsvorhaben in der Informatik in Betracht zu ziehen, sowie bei Dr. Alexander Dotor, der durch seine Teilnahme an der Evaluation einen direkten Beitrag zu dieser Arbeit geleistet hat. Auch möchte ich Stephanie Meerkamm, meiner Kollegin am benachbarten Lehrstuhl, für die nicht nur fachlichen Gespräche in den Mittagspausen danken. Vielen Dank auch an die weiteren Lehrstuhlmitglieder Monika Glaser und Bernd Schlesier, die stets für einen reibungslosen Betrieb des Lehrstuhls und ein gutes Arbeitsumfeld sorgen.

Zum Schluss möchte ich mich noch sehr bei meinen Eltern und meinem Ehemann Peter für die moralische Unterstützung bedanken und auch bei Anneliese Uhrig, die mir freundlicherweise das Korrekturlesen dieser Arbeit auf Rechtschreibfehler abgenommen hat.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

In der modellbasierten Softwareentwicklung werden Modelle zur Beschreibung der Softwaresysteme in der Planungsphase vor der eigentlichen Implementierung eingesetzt, um die Qualität und Wartbarkeit der resultierenden Software zu verbessern. Mit zunehmender Komplexität der zu entwickelnden Softwaresysteme erhöht sich der mögliche Nutzen einer Produktivitätssteigerung, der durch eine Automatisierung im Softwareentwicklungsprozess erreicht werden kann. Die modellgetriebene Sofwareentwicklung schließt die Lücke zwischen der abstrakten Problembeschreibung durch Modelle und der Implementierung, indem aus ausführbaren Modellen Quellcode erzeugt werden kann [FR07]. Damit erreicht die Entwicklung von Software einen höheren Abstraktionsgrad, wobei gleichzeitig die Modelle im Softwareentwicklungsprozess eine zentrale Rolle einnehmen: Sie werden von begleitenden Modellen zu Analyse- und Dokumentationszwecken zu „first-class"-Artefakte aufgewertet [FV09]. Die Vorteile der modellgetriebenen Softwareentwicklung sind offensichtlich: Die Codegenerierung aus den Modellen spart nicht nur Zeit, sondern senkt auch die Fehleranzahl im Vergleich zu einer manuellen Programmierung.

Da während des modellgetriebenen Softwareentwicklungsprozesses insbesondere bei verteilter Entwicklung verschiedene Versionen von Modellen entstehen, ist es sinnvoll, die Modelle genauso wie den Quellcode bei herkömmlicher Implementierung unter Versionsverwaltung zu stellen. Eine gute Werkzeugunterstützung für die Versionierung von Modellen, die auch die Differenzberechnung zwischen Modellen und das Verschmelzen zweier Modelle beinhaltet, ist für einen effizienten Softwareentwicklungsprozess unverzichtbar [ELH+05] und eine Voraussetzung dafür, dass sich die modellgetriebene Softwareentwicklung in der Zukunft weiter durchsetzt.

Die Differenzberechnung von Modellen beantwortet nicht nur die Frage, was sich zwischen zwei Modellversionen geändert hat, sondern stellt auch die Basis für das Verschmelzen zweier Versionen dar. Um zwei Modelle, die unabhängig voneinander erstellt wurden, zu einer gemeinsamen Version zusammenzuführen, was als Verschmelzen bezeichnet wird, werden die Unterschiede der beiden Versionen benötigt. Differenzen zwischen Modellen können in Versionsverwaltungssystemen auch dazu verwendet werden, um verschiedene Versionen platzsparend zu speichern, indem eine Basisversion und entsprechende Deltas gespeichert werden [CW98]. Doch die Differenzberechnung ist nicht nur innerhalb von Versionsverwaltungssystemen nützlich, wie die nachfolgenden weiteren Anwendungsgebiete belegen. Die Differenzberechnung stellt ein Analysewerkzeug dar, mit dem die Entwicklung von Modellen im Verlauf einer Historie nachvollzogen werden kann [WHK]. Auch falls Modelle nicht zur Generierung von Quellcode, sondern

nur zur Dokumentation verwendet werden, kann es interessant sein, die durch Reverse Engineering gewonnenen Modelle zu einem späteren Zeitpunkt des Projekts mit der anfänglichen Modellbeschreibung zu vergleichen. In [XS06] werden Modelle aus Quellcode gewonnen, die gezielt mittels Differenzberechnung untersucht werden, um komplexere Umstellungen im Quellcode, sogenannte Refactoringmaßnahmen, zu erkennen. Eine weitere mögliche Anwendung ist der Bereich des Model Transformation Testings [KPP06], in dem die Differenzberechnung dazu verwendet wird, um die exakte Ausführung von Modelltransformationsregeln zu überprüfen, die ein Ausgangsmodell in ein Zielmodell überführen sollen. Ferner wird die Differenzberechnung in [BHE09] auch eingesetzt, um hinreichend ähnliche Software Cases aufzufinden, die wiederverwendet werden können.

1.2 Vergleich von Klassendiagrammen

Die vorliegende Arbeit befasst sich mit dem Vergleich von Klassendiagrammen, da das Klassendiagramm in der Objektorientierung die grundlegende Diagrammart zur Modellierung von Datenaspekten darstellt. Klassendiagramme werden als teilweise ausführbare Modelle bezeichnet, da aus ihnen wie zum Beispiel im Eclipse Modeling Framework (EMF) [SBPM09] Klassen mit Feldern und elementaren Operationen wie Zugriffs- und Änderungsmethoden für Werte und Links generiert werden können. Manche Werkzeuge bieten darüber hinaus die Möglichkeit, auch die Inhalte von Methoden zu generieren, wobei dazu die Klassendiagramme meist um eine weitere Diagrammart zur Modellierung der Methodenrümpfe erweitert werden (z. B. Storydiagramme in Fujaba [Zün01]).

Die Verfahren, die in dieser Arbeit vorgestellt werden, wurden für Klassendiagramme konzipiert und implementiert. Dennoch beziehen sich die Definitionen und Aussagen in dieser Arbeit teilweise auf Modelle, falls sich diese verallgemeinern lassen. Im letzten Kapitel dieser Arbeit wird auch darauf eingegangen, inwieweit sich die beschriebenen Verfahren für Klassendiagramme auf allgemeine Modelle übertragen lassen.

Modell vs. Diagramm Ein Modell bezeichnet in dieser Arbeit eine Struktur von getypten Elementen, die eine Kompositionsstruktur bilden[1]. Ein Element kann Eigenschaften besitzen und durch getypte Beziehungen mit anderen Elementen verknüpft sein. Die zulässigen Typen, deren Eigenschaften und Beziehungen werden meist in einem Metamodell festgelegt. Ein Diagramm hingegen bezeichnet eine Darstellung eines Modells, wobei es zu einem Modell verschiedene Darstellungsformen geben kann.

Da zu einem Modell beliebig viele Diagramme erstellt werden können, die sich zwar in der Darstellung, aber nicht in der Semantik unterscheiden und die Diagramme darüber hinaus oft zusätzlich technische Details speichern, die der Repräsentation dienen, wie z. B. Positionen der Diagrammelemente, werden in dieser Arbeit keine Diagramme, sondern die zugrunde liegenden Modelle verglichen. Die Klassendiagramme müssten in

[1] Die Forderung einer Kompositionsstruktur stellt keine Einschränkung der Allgemeinheit dar, da sich für jedes Modell eine Kompositionsstruktur durch Ergänzung eines Wurzelknotens, der mit jedem Modellelement verbunden ist, einführen lässt.

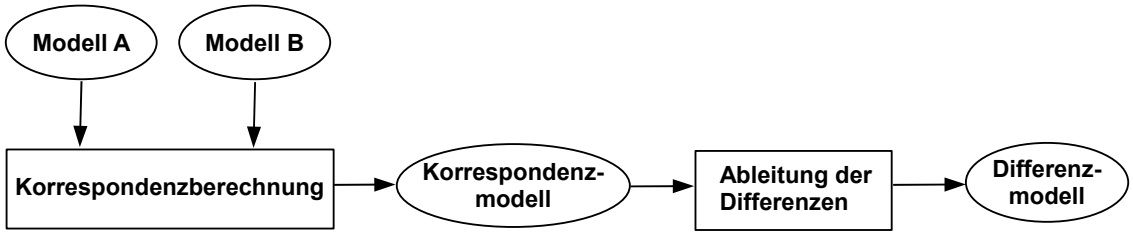

Abbildung 1.1: Die zwei Stufen der Differenzberechnung

dieser Arbeit daher streng genommen als Klassenmodelle bezeichnet werden. Da die Verwendung des Begriffs Klassenmodell jedoch nicht üblich ist und im Kontext der Arbeit klar ist, dass die Modelle und nicht ihre Repräsentationen verglichen werden, wird im Folgenden weiterhin von Klassendiagrammen oder kurz Diagrammen gesprochen.

Einteilung der Vergleichsverfahren in Phasen Vergleichsverfahren, auch als Differenzberechnungsverfahren bezeichnet, für Modelle lassen sich in zwei Phasen unterteilen. In einem ersten Schritt werden die korrespondierenden Modellelemente bestimmt. Dabei wird entschieden, welche Elemente aus den beiden Modellen miteinander identifiziert werden, d. h. als gleich oder hinreichend ähnlich betrachtet werden. Im zweiten Schritt werden anhand der Korrespondenzbeziehungen die Unterschiede in Form von Änderungsoperationen abgeleitet. Diejenigen Elemente, die im anderen Modell kein korrespondierendes Element besitzen, werden als gelöscht oder eingefügt interpretiert. Weiterhin werden alle korrespondierenden Elemente auf Unterschiede in ihren Eigenschaften und Beziehungen überprüft.

In einem modellorientierten Ansatz bietet es sich an, die Ergebnisse beider Stufen jeweils als Modell abzuspeichern. Die Ergebnisse der Korrespondenzberechnung bilden ein Korrespondenzmodell, die ermittelten Unterschiede werden in einem Differenzmodell dargestellt (vgl. Abbildung 1.1). Der Aufwand der ersten Phase kann je nach verwendetem Verfahren sehr unterschiedlich sein. Liegen die Informationen über die korrespondierenden Elemente bereits vor, da sie über eindeutige Objektbezeichner in den Modellen gespeichert sind, oder da die durchgeführten Änderungsoperationen zwischen den beiden Versionen protokolliert wurden, fällt für diese erste Phase kaum Aufwand an. Wenn die Korrespondenzen andernfalls berechnet werden, kann mit beliebigem Aufwand nach den besten Korrespondenzbeziehungen gesucht werden. Diese Arbeit behandelt die erste und schwierigste Phase der Differenzberechnung, die Berechnung der Korrespondenzen. Für das Ableiten der Unterschiede wird in den vorgestellten Verfahren der Differenzalgorithmus von EMF Compare [MSW09a] eingesetzt.

1.2.1 Herausforderung bei der Korrespondenzberechnung zwischen Klassendiagrammen

Die Korrespondenzberechnung auf Klassendiagrammen stellt in Hinblick auf den Aufwand ein schwieriges Problem dar. Klassendiagramme enthalten nicht nur eine Baum-

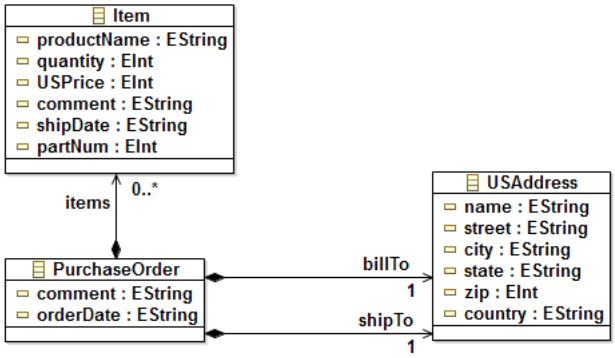

Abbildung 1.2: Klassendiagramm A

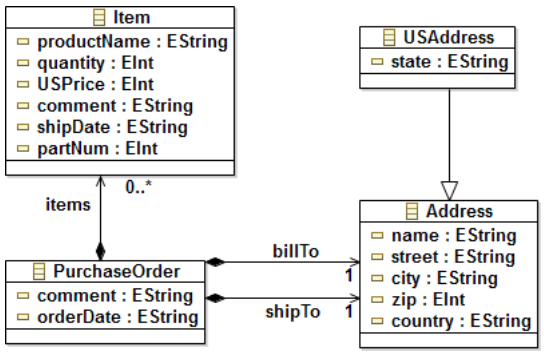

Abbildung 1.3: Klassendiagramm B

struktur, die durch die Kompositionsbeziehungen der Klassen und deren Unterelemente gebildet wird, sondern auch Querverbindungen in Form von Assoziationen und Vererbungskanten, die diesen Baum zu einem allgemeinen Graphen erweitern. Daher sind Vergleichswerkzeuge, die auf flachen Strukturen wie Textdateien arbeiten, oder Baumvergleichsverfahren nicht ausreichend, um Klassendiagramme zu vergleichen. Der exakte Vergleich von Graphen stellt jedoch in der Mathematik ein komplexes Problem dar, für das keine effizienten Lösungsverfahren bekannt sind.

Die Korrespondenzberechnung bildet die Grundlage für das Ableiten der Differenzen. Das folgende Beispiel veranschaulicht, wie sich die berechneten Korrespondenzen auf die anschließende Differenzbildung auswirken. In den Abbildungen 1.2 und 1.3 sind zwei Klassendiagramme zur Modellierung von Kaufaufträgen abgebildet, die auf einem Beispielmodell aus [SBPM09, S. 70] basieren. 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.

Die beiden Klassendiagramme werden mit zwei verschiedenen Vergleichsverfahren verglichen, die sich nur in den Korrespondenzberechnungsverfahren unterscheiden und jeweils den gleichen Algorithmus zum Ableiten der Differenzen verwenden. Das EMF Compare Verfahren [MSW09a], das im Eclipse Modeling Framework integriert ist und

in Abschnitt 3.2.2 genauer beschrieben wird, ordnet die beiden gleichnamigen Klassen `USAddress` und `USAddress` einander zu. Dies führt zur Ableitung folgender Differenzen:

1. Hinzufügen der Klasse `Address` mit den fünf Attributen `name`, `street`, `city`, `zip` und `country`
2. Änderung des Referenzziels der Assoziation `shipTo`
3. Änderung des Referenzziels der Assoziation `billTo`
4. Einfügen einer Vererbungsbeziehung zwischen `USAddress` und `Address`
5. Löschen der fünf Attribute `name`, `street`, `city`, `zip` und `country` in der Klasse `USAddress`

Das eigenentwickelte edierkostenbasierte Verfahren, das in Abschnitt 4.2 dargestellt wird, identifiziert die Klasse `USAddress` aus Diagramm A mit der Klasse `Address` aus Diagramm B und ermittelt folgende Differenzen:

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

Der Vollständigkeit wegen zeigen die Abbildungen 1.4 und 1.5 die beiden resultierenden Differenzberichte visualisiert im EMF Compare UI Editor, der in Abschnitt 5.4.3 genauer behandelt wird. Beim Vergleich der beiden Vergleichsergebnisse fällt auf, dass das edierkostenbasierte Verfahren Korrespondenzen berechnet, die im Vergleich zum Ergebnis des EMF Compare Verfahrens zu der kleineren Differenz gemessen an der Anzahl der Änderungsoperationen führen. Dies ist darauf zurückzuführen, dass das edierkostenbasierte Verfahren eine minimale Distanz zwischen den beiden Diagrammen sucht, wobei die Diagrammelemente in Hinblick auf ihre strukturellen Ähnlichkeiten verglichen werden. Das EMF Compare Verfahren verwendet hingegen eine Ähnlichkeitsheuristik, die sich vergleichsweise stark an den Namen der Elemente orientiert.

Da beide Verfahren den gleichen Algorithmus zum Ableiten der Differenzen verwenden und sich nur in der ersten Phase, der Korrespondenzberechnung, unterscheiden, wird anhand des Beispiels deutlich, wie wichtig eine gute Korrespondenzberechnung ist, um auch gute Differenzen zu erhalten. Daher erscheint es sinnvoll, die relevanten Kriterien für gute Differenzen bereits in der Phase der Korrespondenzberechnung mit einzubeziehen. Die bisherigen Evaluationen von Vergleichsergebnissen (siehe z. B. [MGMR02, KWN05]) basieren lediglich auf der Befragung von Testpersonen, die die Qualität der berechneten Unterschiede beurteilen. Da meines Erachtens noch kein formalisierbares Kriterium für die Güte von Differenzen bekannt ist und dieses daher nicht in die Korrespondenzberechnung einfließen kann, wird in dieser Arbeit das Problemfeld von der anderen Seite aus angegangen. Der Ansatz besteht darin, ein formales Kriterium für die Berechnung von Korrespondenzen über den minimalen Edierabstand einzuführen und zu überprüfen, ob

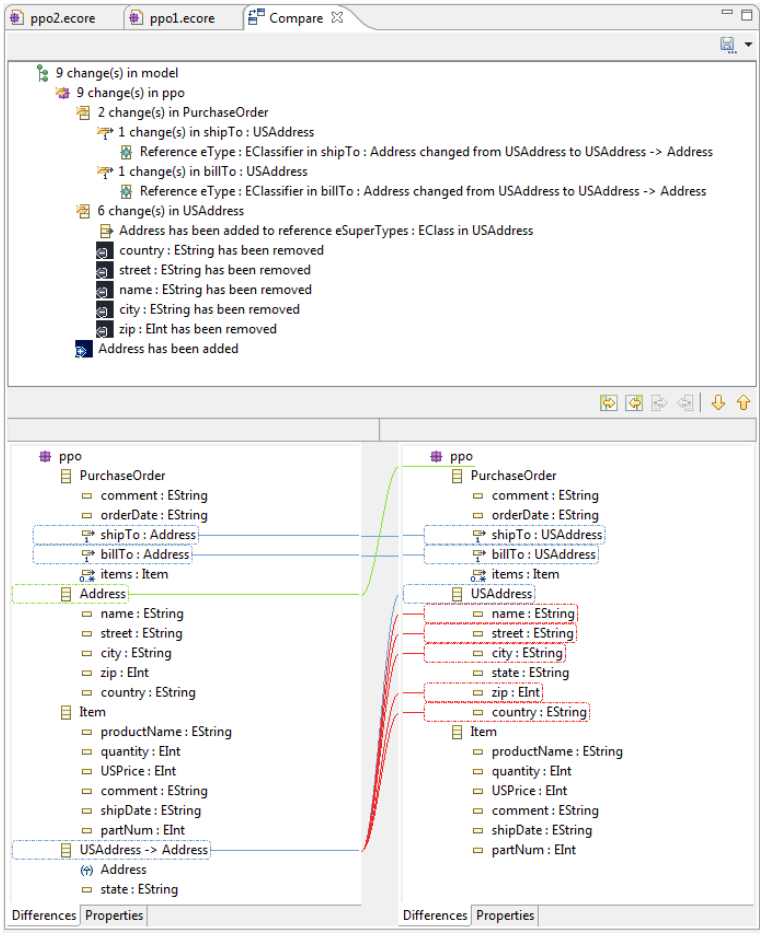

Abbildung 1.4: EMF Compare: Vergleichsergebnis der Klassendiagramme A und B

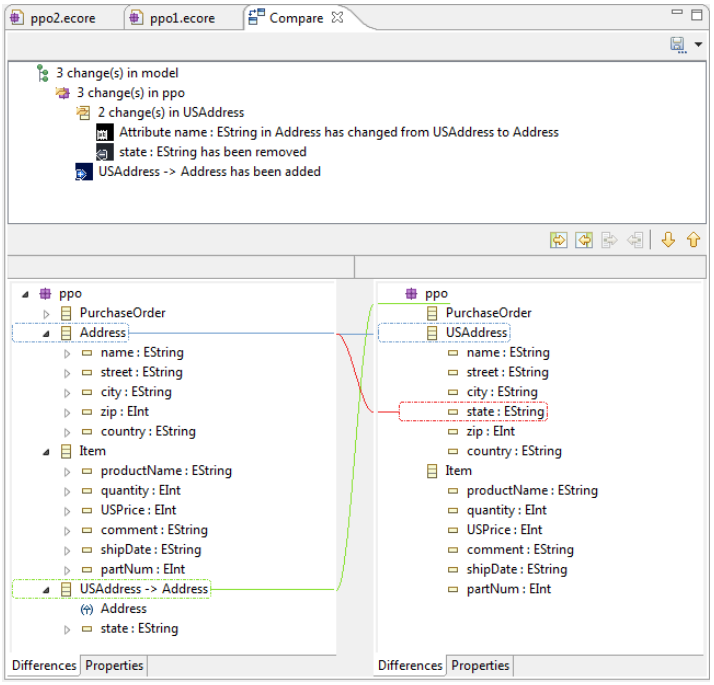

Abbildung 1.5: ECVerfahren: Ergebnis des Vergleichs der Klassendiagramme A und B

dies zu besseren Differenzen führt. Die Korrespondenzberechnung wird auf diese Weise als Optimierungsproblem aufgefasst.

Bevor auf die Anforderungen eingegangen wird, die Vergleichsverfahren idealerweise erfüllen, wird ein kursorischer Überblick über bisherige Vergleichsverfahren für Klassendiagramme gegeben. Eine ausführlichere Darstellung und Bewertung folgt in Kapitel 3 dieser Arbeit.

1.2.2 Kurzüberblick über den Stand der Technik

Die bisher bekannten Vergleichsverfahren für Klassendiagramme lassen sich in zwei Kategorien einteilen. Die Verfahren der ersten Gruppe verzichten auf die eigentliche Korrespondenzberechnung und verwenden stattdessen eindeutige Objektbezeichner zur Identifikation korrespondierender Elemente. Jedem Element, im Folgenden auch als Objekt bezeichnet, wird bei dessen Erzeugung ein unveränderlicher Bezeichner zugeordnet, wobei jeder Bezeichner jeweils nur einmal vergeben und auch nicht wiederverwendet wird, falls das Objekt gelöscht wird. Beim Kopieren eines Objekts wird auch dessen Bezeichner kopiert. Wird jedoch ein Objekt gelöscht und ein gleiches Objekt erneut erstellt, unterscheiden sich die beiden Objekte in ihren Bezeichnern. Bei der Bestimmung der korrespondierenden Elemente werden genau diejenigen Elemente miteinander identifiziert, deren Bezeichner identisch sind. Dies besitzt den Nachteil, dass die Korrespondenzen in Abhängigkeit der Entstehungsgeschichte der zu vergleichenden Modelle gebildet werden. Meines Erachtens sollte ein Korrespondenzberechnungsverfahren jedoch lediglich die aktuellen Zustände der beiden Modelle vergleichen und damit auch Elemente als korrespondierend identifizieren, die semantisch gleich sind, auch wenn sie verschiedene Bezeichner besitzen. Beispiele für Verfahren, die eindeutige Objektbezeichner verwenden, sind [AP03, OWK03, MGH05].

Verfahren, die nicht auf eindeutigen Objektbezeichnern basieren, ermitteln die Korrespondenzen stattdessen über Ähnlichkeitsheuristiken. Beispiele für solche Verfahren sind [KWN05, XS05, LGJ07, BP08, RV08]. Diese Verfahren unterscheiden sich im Wesentlichen in den verwendeten Ähnlichkeitsheuristiken und den Strategien, wie die Modelle auf der Suche nach korrespondierenden Elementen traversiert werden. Heuristische Verfahren für den Vergleich von Klassendiagrammen scheinen gut akzeptiert zu sein. Allerdings verlassen sich diese Verfahren, soweit sie im Rahmen dieser Arbeit getestet wurden, zu sehr auf die Identifikation von korrespondierenden Elementen über Namen und weisen daher Schwächen auf, falls Diagramme mit vielen Umbenennungen verglichen werden.

Es existieren somit bereits Vergleichsverfahren für Klassendiagramme, die Verfahren mit eindeutigen Objektbezeichnern liefern jedoch meines Erachtens keine befriedigenden Lösungen. Auch die getesteten heuristischen Verfahren weisen Defizite auf, da sie die Namen von Elementen zu stark gewichten oder nicht alle möglichen Klassenkorrespondenzen berücksichtigen. Ein weiterer Kritikpunkt der heuristischen Verfahren besteht darin, dass die Gesamtlösung über eine Folge einzelner guter Korrespondenzen gebildet wird, wobei bei diesen lokalen Einzelentscheidungen nicht die globale Situation betrachtet wird und auch bereits getroffene Entscheidungen nicht mehr revidiert werden. Darüber hinaus bin ich der Ansicht, dass vor der Anwendung von heuristischen Verfah-

Tabelle 1.1: Anforderungen an Differenzberechnungsverfahren für Modelle

(A1)	Genauigkeit
(A1a)	Korrektheit
(A1b)	Güte der Ergebnisse in Hinblick auf ein vorgegebenes Kriterium
(A2)	Hoher konzeptioneller Level
(A3)	Domänenunabhängigkeit
(A4)	Werkzeugunabhängigkeit
(A5)	Unabhängigkeit von der Ediergeschichte
(A6)	Effizienz
(A7)	Benutzerfreundliche Darstellung der Unterschiede
(A8)	Leichtgewichtiger Ansatz

ren zunächst untersucht werden sollte, ob das Problem nicht auch mit einem exakten oder approximativen Verfahren gut lösbar ist und vielleicht die besseren Differenzen berechnet.

1.2.3 Anforderungen

Differenzberechnungsverfahren auf Modellen sollten allgemein acht grundlegende Anforderungen erfüllen, die in Tabelle 1.1 aufgelistet sind. Im Vergleich zu der ursprünglichen Beschreibung der Anforderungen in [FW07] ist die erste Anforderung in dieser Darstellungsform in zwei Teilanforderungen untergliedert.

Ein Differenzberechnungsverfahren soll die Unterschiede zwischen zwei Modellversionen so genau wie möglich berechnen (A1). Dies beinhaltet zum einen, dass ein vollständiger Differenzbericht erstellt wird, der in Hinblick auf die betrachteten Eigenschaften der Modelle keine Unterschiede unberücksichtigt lässt sowie keine widersprüchlichen Unterschiede oder nicht vorhandene Unterschiede meldet (A1a). Damit ist die Grundvoraussetzung gegeben, dass die berechnete Differenz für weitere Verwendungszwecke, wie das Verschmelzen zweier Versionen, eingesetzt werden kann. Neben der Korrektheit ist es wünschenswert, dass die berechnete Differenz in Hinblick auf vorgegebene Qualitätskriterien möglichst gut ist (A1b). Was als solches Kriterium herangezogen werden kann, das sich formalisieren und auch messen lässt, darüber existiert noch Unklarheit. Als ein Aspekt dieser Arbeit wird untersucht, ob der Edierabstand zwischen zwei Klassendiagrammen als Kriterium geeignet ist.

Das Verfahren soll außerdem auf einem hinreichend hohen konzeptionellen Level arbeiten (A2), der für den Vergleich von Modellen angemessen ist. Wie in Kapitel 2 erläutert wird, sind Vergleichsverfahren, die auf Text- oder Baumebene angesiedelt sind, für den Vergleich von Modellen nicht zufriedenstellend, da sie die Korrektheit des Ergebnisses im Sinne der Anforderung (A1a) nicht gewährleisten können. Da eine Projektion der Modelle auf eine niedrigere konzeptionelle Ebene zur Differenzberechnung und eine Rückführung der Ergebnisse auf Modellebene die in Kapitel 2 beschriebenen Nachtei-

le mit sich bringt, werden die Korrespondenzen idealerweise bereits auf Modellebene berechnet, da die daraus resultierenden Unterschiede auch auf Modellebene dargestellt werden müssen.

Domainenunabhängige Differenzberechnungsverfahren (A3) sind flexibler einsetzbar, da sie nicht für eine spezielle Diagrammart konzipiert wurden. Dieser Vorteil steht allerdings dem Nachteil gegenüber, dass bei diesen unabhängigen Verfahren modellspezifische Besonderheiten nicht ausgenutzt werden können, was unter Umständen zu schlechteren Laufzeiten oder schlechteren Ergebnissen führen kann. Außerdem sollte das Differenzberechnungsverfahren unabhängig von den verwendeten Werkzeugen einsetzbar sein (A4). Damit entfällt die Möglichkeit, Änderungsoperationen im Werkzeug zu protokollieren und für die Differenzberechnung zu nutzen. In enger Verbindung mit der Werkzeugunabhängigkeit steht auch die nächste Anforderung (A5), die eine Unabhängigkeit von der Ediergeschichte fordert. Die Nachteile von Verfahren, die auf eindeutigen Objektbezeichnern basieren, werden in Abschnitt 3.1.2 behandelt.

Eine klassische Anforderung an Verfahren stellt die Effizienz (A7) dar. An dieser Stelle ist offensichtlich, dass die genannten Anforderungen teilweise miteinander in Konflikt stehen und wohl nicht gleichzeitig erreicht werden. So stehen die bereits genannten Anforderungen (A1) Genauigkeit und (A7) Effizienz mit der letzten Anforderung (A8), dem leichtgewichtigen Ansatz, in einem klassischen Spannungsdreieck zueinander.

1.3 Zielsetzung und Beitrag der Arbeit

1.3.1 Begriffliche Grundlagen

Im Folgenden werden die in dieser Arbeit wesentlichen Begriffe der Themenbereiche Korrespondenz und Differenz festgelegt.

Korrespondenz Fasst man Modelle als Mengen von Modellelementen auf, kann der Begriff der Zuordnung wie folgt eingeführt werden.

Definition 1 (Zuordnung)**.** Seien $A = \{a_1, \dots a_n\}$ und $B = \{b_1, \dots b_m\}$ Modelle. Eine Zuordnung M ist eine 2-stellige Relation $(a_i, b_j) \in A \times B$. Die Elemente $a_i \in A$ und $b_j \in B$ korrespondieren, falls $(a_i, b_j) \in M$.

Zwischen korrespondierenden Elementen (sogenannten Korrespondenzelementen) besteht eine Korrespondenz(beziehung). Gilt $(a_i, b_j) \in M$, wird a_i auch als Korrespondenzpartner von b_j bezüglich M bezeichnet und umgekehrt. Diejenigen Modellelemente, die mit keinem Element korrespondieren, werden als nichtkorrespondierend oder Einzelelemente bezeichnet. Die nichtkorrespondierenden Elemente werden somit indirekt über die Auswahl der korrespondierenden Elemente festgelegt. Beim Ableiten einer Differenz werden die Einzelelemente aus A als gelöschte Elemente, die Einzelelemente aus B als eingefügte Elemente interpretiert.

Beispiel 1. Die Abbildung 1.6 zeigt die Mengen der Modellelemente der Modelle A und B. Für eine Zuordnung $M = \{(a_1, b_1), (a_3, b_5), (a_4, b_2)\}$ ergeben sich die Mengen der nichtkorrespondierenden Elemente $U_A = \{a_2\}$ und $U_B = \{b_3, b_4\}$.

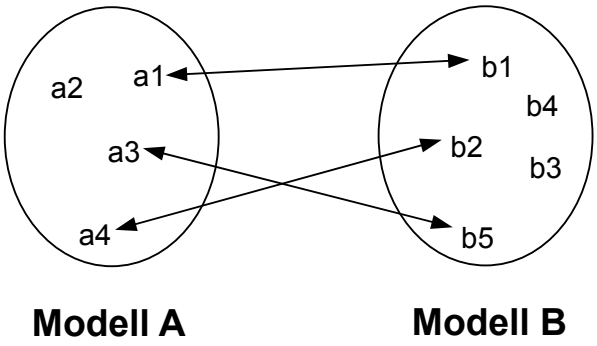

Abbildung 1.6: Korrespondenzbeziehungen zwischen Modellelementen

Über den Begriff der Zuordnung werden noch keine Kriterien vorgegeben, wie die korrespondierenden Elementpaare bestimmt werden. Der Vorgang der Bestimmung der korrespondierenden Elementpaare wird als Korrespondenzberechnung bezeichnet. Die Auswahl der korrespondierenden Elemente erfolgt in einem Korrespondenzberechnungsverfahren, dessen Aufgabe es ist, nach vorgegebenen Kriterien eine möglichst gute Zuordnung zu ermitteln. Dabei ist die Grundentscheidung zu treffen, ob ausschließlich identische oder auch ähnliche Elemente korrespondieren können. Im letzten Fall werden zugeordnete nicht identische Elemente beim Ableiten der Differenz als geänderte Elemente interpretiert.

Je nach Anwendungsfall kann es sinnvoll sein, die Menge der zulässigen Zuordnungen durch weitere Anforderungen zu beschränken. Definition 1 erlaubt es, dass Elemente mehreren Korrespondenzpartnern zugeordnet werden. So ist $M_2 = \{(a_1, b_1), (a_1, b_2), (a_2, b_4), (a_3, b_5)\}$ eine alternative Zuordnung für Beispiel 1. Die Mehrfachzuordnung von a_1 kann gewünscht sein und beim Ableiten der Differenzen als Kopieren des Elements interpretiert werden. Für die eigenentwickelten Korrespondenzberechnungsverfahren in Kapitel 4 wurde hingegen die Entscheidung getroffen, die Menge der zulässigen Zuordnungen dahingehend zu beschränken, dass jedes Element höchstens einen Korrespondenzpartner besitzen darf. Eine Zuordnung, bei der jedes Element höchstens zu einem anderen Element korrespondiert, wird als eindeutige Zuordnung bezeichnet. Die Festlegung der zulässigen Zuordnungen bildet die Basis für die Ableitung der Differenzen in der zweiten Phase der Differenzberechnung.

Abgrenzung von Zuordnung und Matching aus der Graphentheorie Da das später vorgestellte edierkostenbasierte Verfahren das Optimierungsproblem zur Bestimmung eines minimalen Edierabstands mit Hilfe eines Verfahrens aus der Graphentheorie löst, werden grundlegende mathematische Definitionen eingeführt. Insbesondere wird dadurch auch die in Definition 1 angegebene Zuordnung von dem Begriff Matching aus der Graphentheorie abgegrenzt.

Definition 2 (Graph)**.** Ein (ungerichteter) Graph $G = (V, E)$ besteht aus zwei endlichen Mengen V und E. Die Knotenmenge V darf nicht leer sein. Ein Element $a \in V$ wird als Knoten bezeichnet. Die Kantenmenge E enthält zweielementige Mengen von Elementen aus V. Ein Element $e = \{a, b\} \in E$ wird als Kante bezeichnet. Eine Kante $\{a, b\}$ ist inzident zu Knoten a und Knoten b . (nach [Jun08, S. 2])

Ein Matching beschreibt eine Auswahl an Kanten aus der Kantenmenge eines Graphen, so dass jeder Knoten höchstens zu einer ausgewählten Kante inzident ist.

Definition 3 (Matching)**.** Sei $G = (V, E)$ ein Graph. Ein Matching T ist eine Teilmenge von E, so dass jeder Knoten aus V zu höchstens einer Kante aus T inzident ist. (nach [Jun08, S. 129])

Der Zusammenhang zur Berechnung einer Zuordnung nach Definition 1 erschließt sich auf einem bipartiten Graphen. Ein bipartiter Graph ist ein Graph, dessen Knotenmenge sich in zwei disjunkte Teilmengen aufteilen lässt, so dass Kanten nur zwischen Knoten der beiden verschiedenen Mengen, nicht aber zwischen Knoten der gleichen Menge existieren.

Definition 4 (Bipartiter Graph)**.** Ein Graph $G = (V, E)$ heißt bipartit, falls es eine Partition $V = A \cup B$, $A \cap B = \emptyset$ der Knotenmenge V gibt, so dass jede Kante $e \in E$ mit einem Knoten in A und einem Knoten in B inzident ist. (nach [KN05, S. 42])

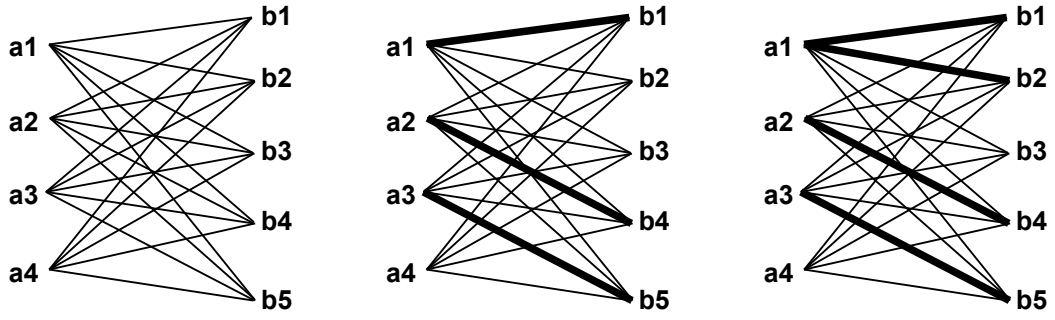

Abbildung 1.7: Links: Bipartiter Graph G_1 zu Beispiel 1, Mitte: Matching auf G_1, rechts: kein zulässiges Matching

Die Abbildung 1.7 (links) zeigt zu Beispiel 1 einen bipartiten Graphen $G_1 = (V, E)$ mit $V = A \cup B$, $A = \{a_1, a_2, a_3, a_4\}$, $B = \{b_1, b_2, b_3, b_4, b_5\}$, der für jede mögliche Korrespondenzbeziehung zwischen den Modellelementen aus A und B, repräsentiert durch die Knoten, eine Kante enthält. Die Menge der ausgewählten Korrespondenzen $M = \{(a_1, b_1), (a_3, b_5), (a_2, b_4)\}$ entspricht in diesem Fall einem Matching T auf G_1 (vgl. Abbildung 1.7 (Mitte)). Die alternative, nicht eindeutige Zuordnung $M_2 = \{(a_1, b_1), (a_1, b_2), (a_2, b_4), (a_3, b_5)\}$ verletzt jedoch die Bedingung eines Matchings, dass jeder Knoten nur zu einer ausgewählten Kante inzident sein darf (vgl. Abbildung 1.7 (rechts)). Zu einer eindeutigen Zuordnung kann ein bipartiter Graph konstruiert werden,

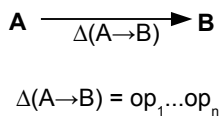

Abbildung 1.8: Gerichtetes Delta aus [CW98]

so dass die eindeutige Zuordnung ein Matching auf diesem Graphen darstellt. Die Begriffe Matching und Zuordnung sind jedoch nicht äquivalent, da dies einerseits nicht für eine nicht eindeutige Zuordnung gilt, und andererseits ein Matching auch auf einem allgemeinen Graphen existieren kann. Dennoch werden die Korrespondenzberechnungsverfahren umgangssprachlich gerne als Matchverfahren bezeichnet.

Differenz In dieser Arbeit bezeichnet eine Differenz zwischen zwei Modellen A und B eine Folge zulässiger Änderungsoperationen, die angewandt auf das Modell A dieses in das Modell B überführen. Die Differenz hängt somit von der Menge der zulässigen Änderungsoperationen ab und ist gerichtet. In [CW98] wird diese Differenz auch als gerichtetes Delta bezeichnet (vgl. Abbildung 1.8). Da die Reihenfolge der Änderungsoperationen relevant sein kann, wird das Delta als Folge und nicht als Menge von Änderungsoperationen dargestellt. Als Änderungsoperationen können nur die Basisoperationen Löschen bzw. Einfügen eines Elements und Ändern einer Elementeigenschaft zugelassen werden oder auch komplexere Änderungen, z. B. Verschiebungen oder Teilbaumlöschung, berücksichtigt werden. Komplexere Änderungsoperationen sind zwar nicht zwingend nötig, da sie sich auf eine Folge von Basisoperationen zurückführen lassen, reduzieren jedoch den Umfang der dargestellten Differenz. Bei der Auswahl der zulässigen Änderungsoperationen ist es sinnvoll, auch zu berücksichtigen, welche Änderungsoperationen im verwendeten Modelleditor möglich sind.

1.3.2 Verfahren zur Korrespondenzberechnung

Da alle in der Literatur bekannten Verfahren zum Vergleich von Klassendiagrammen, die keine eindeutigen Objektbezeichner zum Identifizieren korrespondierender Elemente verwenden, einen ähnlichkeitsbasierten Ansatz verfolgen, startete die vorliegende Arbeit mit der Ausgangsfragestellung, ein edierkostenbasiertes Korrespondenzberechnungsverfahren für Klassendiagramme zu entwickeln, das die Bestimmung einer Zuordnung als Optimierungsproblem betrachtet und für dieses Optimierungsproblem mit akzeptablem Aufwand eine optimale oder annähernd optimale Lösung berechnet. Um zu untersuchen, ob die aufwändiger ermittelten Differenzen des edierkostenbasierten Verfahrens Vorteile gegenüber den Differenzen eines schnelleren ähnlichkeitsbasierten Verfahrens besitzen, wurde ein weiteres ähnlichkeitsbasiertes Korrespondenzberechnungsverfahren konzipiert.

Es wurden somit zwei verschiedene Korrespondenzberechnungsverfahren in verschiedenen Varianten für Klassendiagramme entwickelt und als Eclipse-Plugin für Ecore-Klassendiagramme implementiert. Beide Verfahren arbeiten unabhängig von eindeutigen Objektbezeichnern und strukturbasiert. Bei dem ersten Verfahren handelt es sich

um ein edierkostenbasiertes Verfahren, das das Identifizieren korrespondierender Elemente als Optimierungsproblem auffasst. Gesucht wird eine Zuordnung mit minimalen Edierkosten, die benötigt werden, um das eine in das andere Modell zu überführen. Das Kostenmodell des edierkostenbasierten Verfahrens kann angepasst werden, wobei sichergestellt wird, dass die geänderten Kostenbewertungen den Kriterien einer Metrik entsprechen. Um den Aufwand des Verfahrens polynomiell zu begrenzen, werden die Kosten der kontextabhängigen Eigenschaften durch eine untere Schranke abgeschätzt. Nach Berechnung der Zuordnung lässt sich überprüfen, ob ein hinreichendes, aber nicht notwendiges Optimalitätskriterium erfüllt ist. In diesem Fall wurde die minimale Zuordnung nachweislich gefunden. Andernfalls kann, aber muss die Lösung nicht optimal sein. Neben dem edierkostenbasierten Verfahren mit den abgeschätzten Kosten (EC-Verfahren) wurde außerdem eine Brute-Force-Variante des Verfahrens (BFVerfahren) implementiert. Das BFVerfahren lässt sich aufgrund des Aufwands zwar nur für kleine Klassendiagramme einsetzen, es bestimmt jedoch in jedem Fall eine exakte Lösung und liefert damit Vergleichswerte für die Evaluation des edierkostenbasierten Verfahrens mit abgeschätzten Kosten. Die Ergebnisse der Evaluation zeigen, dass das Verfahren mit abgeschätzten Kosten in vielen Fällen die kostenoptimale Lösung bestimmt und dass die Kosten von den Kosten der optimalen Lösung nur wenig abweichen.

Das ähnlichkeitsbasierte Verfahren bestimmt hingegen eine Zuordnung unter Maximierung der Ähnlichkeiten, wobei die Gewichtung der Ähnlichkeitswerte vor Ausführung des Vergleichs angepasst werden kann. Das Verfahren verfolgt einen Greedy-Ansatz, der nicht zur optimalen Lösung führen muss, aber eine deutlich geringere Komplexität aufweist. Im Evaluationsteil werden die Ergebnisse des ähnlichkeitsbasierten mit denen des edierkostenbasierten Verfahrens verglichen.

Die Verfahren erkennen nativ keine Verschiebungen von Unterelementen der Klassen, es wurde jedoch ein optionaler Nachbearbeitungsschritt entwickelt, der in beschränktem Umfang verschobene Attribute oder Methoden identifizieren kann. Die bisher beschriebenen Verfahren zur Korrespondenzberechnung auf Klassendiagrammen arbeiten zunächst auf flachen Klassendiagrammen, deren Klassen alle in einem gemeinsamen Grundpaket liegen. Um auch Klassen in verschiedenen Paketen zu berücksichtigen, wurde das Verfahren auf Klassenebene zu einem mehrstufigen Ansatz erweitert. Durch einen modularen Ansatz sind Klassen- und Paketebenenverfahren unabhängig voneinander wählbar. In dem anschließenden ähnlichkeitsbasierten Paketebenenverfahren werden die Korrespondenzen der Pakete unter Berücksichtigung der gefundenen Klassenzuordnung gebildet und die jeweilige Zuordnung der Pakete und Klassen in ein gemeinsames Korrespondenzmodell überführt.

Für das anschließende Ableiten des Differenzenmodells aus den berechneten Korrespondenzen und die Visualisierung der Differenzen in einer Baumsicht werden bereits existierende Komponenten genutzt. Das eigenentwickelte Plugin mit den Korrespondenzberechnungsverfahren und Sichten zur Konfiguration und Auswahl der Verfahren bildet mit einem beliebigen Ecore-Editor und dem Differenzberechnungsverfahren von EMF Compare sowie dem EMF Compare UI Editor ein Eclipse-basiertes Rahmenwerk zum Vergleich von Korrespondenzberechnungsverfahren (siehe Abbildung 1.9).

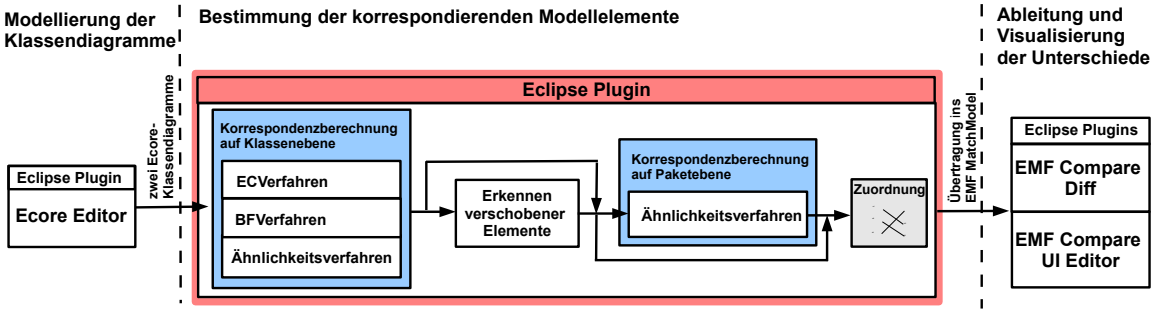

Abbildung 1.9: Vergleich von Klassendiagrammen

1.4 Aufbau dieser Arbeit

Neben der folgenden textuellen Beschreibung des Aufbaus der vorliegenden Arbeit gibt die Abbildung 1.10 einen graphischen Überblick über die Struktur der Kapitel. Abschnitte, die die wesentlichen Beiträge dieser Arbeit enthalten (**E**igener **B**eitrag) oder die sich auf verwandte Arbeiten beziehen (**R**elated **W**ork), sind in dieser Abbildung besonders gekennzeichnet.

Nach dem einleitenden Kapitel, das bereits die Anforderungen an Differenzberechnungsverfahren sowie grundlegende Begriffe eingeführt hat, folgen zwei Kapitel, die im weiteren Sinne verwandte Arbeiten behandeln. Kapitel 2 gibt einen kurzen, nicht vollständigen Überblick über Vergleichsverfahren aus verwandten Bereichen, dem Vergleich von Textdokumenten, Baumstrukturen, Graphen sowie andere Diagrammarten. Dabei wird jeweils kurz darauf eingegangen, warum eine direkte Übertragung dieser Verfahren auf den Bereich der Klassendiagramme nicht sinnvoll ist. In Kapitel 3 folgt eine Darstellung der bisherigen Ansätze für den Vergleich von Klassendiagrammen, die sich im Wesentlichen in Verfahren mit eindeutigen Objektbezeichnern und heuristische Verfahren gliedern lassen. Dabei wird auch darauf eingegangen, weshalb die bisherigen Korrespondenzberechnungsverfahren für Klassendiagramme nicht ausreichen.

Im Hauptkapitel 4 dieser Arbeit wird in Abschnitt 4.1 zunächst das zugrunde liegende Datenmodell für Klassendiagramme eingeführt, das beide eigenentwickelten Verfahren verwenden. Danach folgt in Abschnitt 4.2 eine ausführliche Beschreibung des edierkostenbasierten Verfahrens. Wie die Berechnung einer Zuordnung zwischen Klassen zweier Modelle über die Einführung einer Menge an zulässigen Änderungsoperationen und zugehörigen Kosten als Optimierungsproblem aufgefasst werden kann, wird in Abschnitt 4.2.1 hergeleitet. Der Abschnitt 4.2.2 enthält für dieses Optimierungsproblem ein nichtpolynomielles exaktes Lösungsverfahren, das im Folgenden als Brute-Force-Variante bezeichnet wird und in der Praxis aufgrund des Aufwands jedoch nicht einsetzbar ist. Daher wird in Abschnitt 4.2.3 eine Verfahrensvariante mit polynomiellen Aufwand, das ECVerfahren, vorgestellt, das mit durch eine untere Schranke abgeschätzten kontextabhängigen Kosten arbeitet und so eine optimale Lösung für ein relaxiertes Problem bestimmt. Dazu wird das Optimierungsproblem auf ein Netzwerkflussproblem übertragen (Abschnitt 4.2.4), das schließlich mit einem Verfahren aus der Graphentheorie, dem Algorithmus von Busacker und Gowen gelöst wird (Abschnitt 4.2.5). Der Zusammenhang zwischen

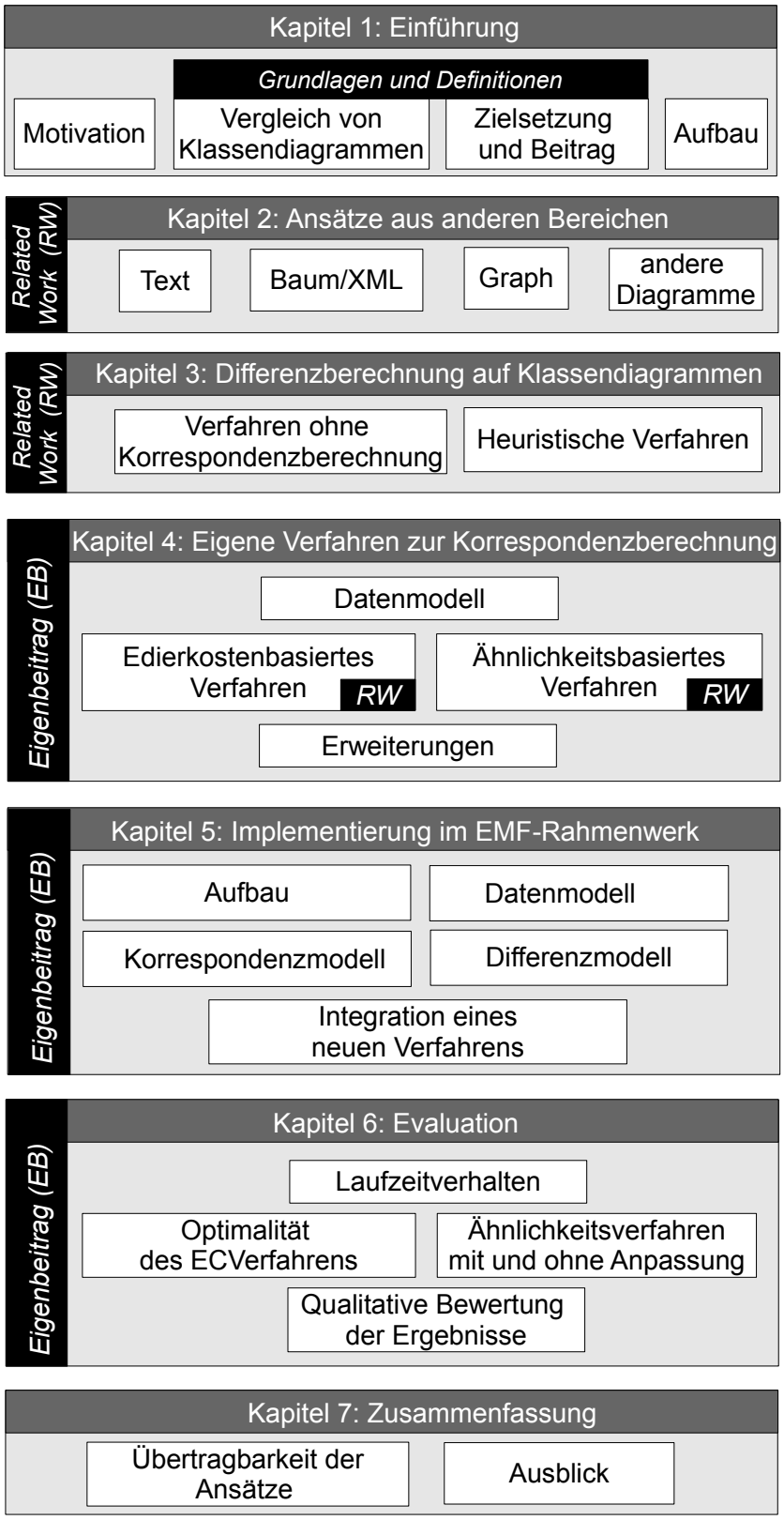

Abbildung 1.10: Graphische Übersicht des Aufbaus der Arbeit mit Kennzeichnung der Abschnitte, die Related Work bzw. eigene Beiträge enthalten.

dem relaxierten und dem ursprünglichen Optimierungsproblem wird in Abschnitt 4.2.6 behandelt. Über ein hinreichendes Optimalitätskriterium kann in manchen Fällen nachgewiesen werden, dass die Lösung des relaxierten Problems auch eine optimale Lösung des ursprünglichen Problems ist. In Abschnitt 4.2.7 wird letztlich beschrieben, wie durch die Definition einer Schwellenwertregel in das Kostenmodell eingegriffen werden kann, um bestimmte Zuordnungen zu verhindern. Eine abschließende Zusammenfassung in Abschnitt 4.2.9 grenzt das Verfahren auch von verwandten Arbeiten ab, die bereits in Kapitel 3 vorgestellt wurden.
Das zweite Verfahren, das ähnlichkeitsbasierte Verfahren, wird in Abschnitt 4.3 behandelt. Wie die Ähnlichkeitswerte zweier Klassen berechnet werden, wird in Abschnitt 4.3.1 angegeben. Aus den Ähnlichkeitswerten wird über ein Auswahlverfahren, das in Abschnitt 4.3.2 beschrieben wird, die Menge der korrespondierenden Klassen bestimmt. Neben der einfachen Variante wird auch ein Auswahlverfahren mit Anpassung der Ähnlichkeiten vorgestellt. Nachdem zwei Klassen als korrespondierend identifiziert wurden, werden die Ähnlichkeitswerte angrenzender Klassen angepasst, die mit den korrespondierenden Klassen über Vererbungsbeziehungen oder Assoziationen verbunden sind. Auch hier folgt abschließend eine Zusammenfassung des Verfahrens mit einer Abgrenzung zu verwandten Arbeiten. In den letzten beiden Abschnitten des Kapitels 4 wird auf Erweiterungen eingegangen. Der Abschnitt 4.4 behandelt die Möglichkeit, in einer Nachbearbeitungsphase verschobene Klassenunterelemente in eingeschränktem Umfang zu erkennen. In Abschnitt 4.5 wird das ähnlichkeitsbasierte Verfahren auf die Paketebene angepasst.

Das Kapitel 5 befasst sich mit der Integration dieser Verfahren in das Eclipse-basierte Rahmenwerk zum Vergleich von Korrespondenzberechnungsverfahren, wobei dargestellt wird, wie die eigenentwickelten Komponenten mit Plugins von EMF Compare zusammenspielen. Nach einem Überblick über das gesamte Rahmenwerk und das eigenentwickelte Plugin in Abschnitt 5.1, werden in den nachfolgenden Abschnitten die verwendeten Datenmodelle beschrieben. Dies beinhaltet das interne Datenmodell in Abschnitt 5.2, sowie das Korrespondenzmodell und das Differenzenmodell von EMF Compare in den Abschnitten 5.3 und 5.4. In Abschnitt 5.5 wird beschrieben, wie weitere Verfahren in dieses Rahmenwerk eingebunden werden können.

Das Kapitel 6 enthält eine Evaluation der Verfahren in Hinblick auf die Laufzeiten und die Ergebnisse. Dabei wurde das edierkostenbasierte Verfahren auf die Optimalität der Ergebnisse untersucht. Außerdem wurden jeweils die verschiedenen Varianten, das edierkostenbasierte Verfahren mit und ohne Schwellenwert sowie das ähnlichkeitsbasierte Verfahren mit und ohne Anpassung der Ähnlichkeiten im 1-Kontext der Klassen, einander gegenübergestellt. Die Evaluation beinhaltet weiterhin eine qualitative Beurteilung der Ergebnisse durch den Anwender.

Das letzte Kapitel 7 schließt diese Arbeit mit einer Zusammenfassung ab, die auch einen Ausblick auf die Übertragbarkeit der beiden Verfahren auf andere Diagrammarten bzw. Instanzen beliebiger Ecore-Modelle sowie auf zukünftige Arbeiten enthält.

2 Ansätze aus anderen Bereichen

In diesem Kapitel werden Vergleichsansätze aus verwandten Bereichen vorgestellt, die aber nicht direkt auf Klassendiagramme übertragbar sind. Da Werkzeuge zur Modellierung von Klassendiagrammen oft einen Serialisierungsmechanismus der Klassendiagramme ins XML-Format anbieten, wird untersucht, ob der einfache Textvergleich (Abschnitt 2.1) oder ein strukturierter Vergleich (Abschnitt 2.2) der XML-Dokumente für die Differenzberechnung zwischen Klassendiagrammen sinnvoll ist. Aufgrund des hierarchischen Aufbaus eines XML-Dokuments basieren die strukturbezogenen Vergleichsverfahren für XML-Dokumente auf Baumalgorithmen. Außerdem werden allgemeinere Vergleichsverfahren für Graphen (Abschnitt 2.3) sowie ausgewählte Verfahren für den Vergleich von anderen Diagrammarten vorgestellt. Die Aufzählung der Verfahren in diesem Kapitel erhebt keinen Anspruch auf Vollständigkeit, soll jedoch einen Überblick vermitteln und vor allem auch aufzeigen, warum spezifische Verfahren für den Vergleich von Modellen, insbesondere Klassendiagrammen, benötigt werden.

2.1 Einfache Textdateien

In diesem Abschnitt werden Vergleichsverfahren für einfache Textdokumente behandelt, wobei mit deren kleinsten Bausteinen, den Zeichenketten, begonnen wird.

2.1.1 Zeichenketten

Der Vergleich von Zeichenketten ist nicht ausreichend, um ganze Klassendiagramme miteinander zu vergleichen. Die Zeichenkettenvergleichsverfahren können jedoch für den Vergleich von Bezeichnern, wie z. B. Klassennamen, eingesetzt werden. So bilden die beiden im Folgenden vorgestellten Ansätze *Longest Common Subsequence (LCS)* sowie die *Levenshtein-Distanz* die Basis für den Zeichenkettenvergleich der eigenentwickelten Verfahren, die in Kapitel 4 beschrieben werden.

Zeichenkettenvergleich mit der Longest Common Subsequence

In [HS77] wird ein Verfahren zur Berechnung der längsten gemeinsamen Teilfolge von Zeichen zweier Zeichenketten angegeben. Das Verfahren bezieht sich zwar auf Textzeilen eines Dokuments, ist aber auch auf Zeichen einer Zeichenkette übertragbar. Seien allgemein $A = a_1, \ldots, a_n$ und $B = b_1, \ldots, b_m$ endliche Folgen der Länge n bzw. m von Zeichen eines Alphabets. So ist $U = u_1, \ldots, u_k$ eine gemeinsame Teilfolge von A und B, falls zwei monoton steigende Folgen von ganzen Zahlen $r_1, \ldots, r_k$ und $s_1, \ldots, s_k$ existieren, so dass

$a_{r_1}, \ldots, a_{r_k} \subseteq a_1, \ldots, a_n$ und $b_{s_1}, \ldots, b_{s_k} \subseteq b_1, \ldots, b_m$ existieren, so dass $u_j = a_{r_j} = b_{s_j}$ für alle $1 \leq j \leq k$. Die längste gemeinsame Teilfolge von A und B ist eine gemeinsame Teilfolge mit größtmöglicher Länge. Die Teilfolge muss dabei in den zu vergleichenden Zeichenketten nicht zusammenhängend enthalten sein. Die Länge der längsten gemeinsamen Teilfolge wird durch die Anzahl der Zeichen der kürzeren Zeichenkette begrenzt.

Listing 2.1 zeigt eine einfache Implementierung eines LCS-Verfahrens in Java, das die längste gemeinsame Teilsequenz mit Aufwand $O(nm)$ in Abhängigkeit der Längen n und m der Zeichenketten bestimmt (vgl. auch [CLRS07, S. 351]). Für den Vergleich zweier Zeichenketten $A = a_1, \ldots, a_n$ und $B = b_1, \ldots, b_m$ wird zunächst eine $(n+1) \times (m+1)$-Matrix aufgebaut, die zeilenweise wie folgt gefüllt wird: Die Einträge der 0. Zeile sowie der 0. Spalte sind 0. Beginnend in der 1. Zeile werden nun die Einträge berechnet. Falls die Zeichen der Positionen i in A und j in B übereinstimmen, wird an der Stelle $[i]\,[j]$ der Matrix die bis zur Position $[i-1]\,[j-1]$ gefundene Länge der Teilfolge um 1 erhöht. Ansonsten wird das bisher gefundene Maximum aus $[i-1]\,[j]$ und $[i]\,[j-1]$ übernommen. Nach Konstruktion der Matrix kann die längste gemeinsame Teilsequenz beginnend bei Position $[n]\,[m]$ ausgelesen werden. Dabei kommt ein rekursives Verfahren zum Einsatz: Falls die Zeichen an den entsprechenden Positionen $[n-1]\,[m-1]$ übereinstimmen, wird das aktuelle Zeichen hinten an die LCS angehängt, die ab Position $[n-1]\,[m-1]$ ausgelesen werden kann. Ansonsten ist das aktuelle Zeichen nicht Teil der LCS und es wird entweder an Stelle $[i]\,[j-1]$ oder $[i-1]\,[j]$ der Matrix weiter gesucht, in Abhängigkeit davon, welcher Eintrag den größten Wert besitzt. Der rekursive Abstieg endet, sobald einer der beiden Indizes den Wert 0 erreicht.

Listing 2.1: LCS-Verfahren in Java

```
double[][] LCS_Matrix(String a, String b){
  int la = a.length();
  int lb = b.length();
  double[][] mx = new double[la + 1][lb + 1];
  for (int i = 1; i < la + 1; i++)
   for(int j = 1; j < lb + 1; j++)
    if(a.charAt(i-1) == b.charAt(j-1))
     mx[i][j] = mx[i-1][j-1] + 1;
    else mx[i][j] = Math.max(mx[i][j-1], mx[i-1][j]);
  return mx;
 }

String auslesen(double[][] mx,String a,String b,int i,int j){
  if(i == 0 || j == 0)
   return "";
  if(a.charAt(i-1) == b.charAt(j-1))
   return LCSauslesen(mx, a, b, i-1, j-1) + a.charAt(i-1);
  if(mx[i][j-1] > mx[i-1][j])
   return LCSauslesen(mx, a, b, i, j-1);
  return LCSauslesen(mx, a, b, i-1, j);
```

}

Beispiel 2 (Längste gemeinsame Teilfolge von `Stunde` und `Student`)**.** Die längste gemeinsame Teilfolge von `Stunde` und `Student` ist `Stude`. Die obere Matrix in Abbildung 2.1 stellt die von Methode `LCS_Matrix` aus Listing 2.1 berechnete LCS-Matrix zu den beiden Zeichenketten dar. Die Pfeile in der unteren Matrix der Abbildung zeigen die Reihenfolge, in der die längste gemeinsame Teilfolge aus der Matrix ausgelesen wird.

		S	t	u	d	e	n	t
	0	0	0	0	0	0	0	0
S	0	1	1	1	1	1	1	1
t	0	1	2	2	2	2	2	2
u	0	1	2	3	3	3	3	3
n	0	1	2	3	3	3	4	4
d	0	1	2	3	4	4	4	4
e	0	1	2	3	4	5	5	5

		S	**t**	**u**	**d**	**e**	n	t
	0	0	0	0	0	0	0	0
S	0	↖ **1**	1	1	1	1	1	1
t	0	1	↖ **2**	2	2	2	2	2
u	0	1	2	↖ **3**	3	3	3	3
n	0	1	2	↑ **3**	3	3	4	4
d	0	1	2	3	↖ **4**	4	4	4
e	0	1	2	3	4	↖ **5**	← **5**	← **5**

Abbildung 2.1: LCS-Matrix des Vergleichs `Stunde`, `Student`

Zeichenkettenvergleich mit der Levenshtein-Distanz

Einen anderen Ansatz zum Vergleich von Zeichenketten verfolgt die Levenshtein-Distanz. Eine Beschreibung der von Vladimir Igor Lewenstein 1965 zunächst in russischer Sprache veröffentlichten Edierdistanz findet sich zum Beispiel in [SK83, S. 18-29]. Bei der Levenshtein-Distanz zweier Zeichenketten a und b wird eine Folge von Edieroperationen bestimmt, die die Zeichenkette a in b überführt. Dabei sind nur elementare Operationen auf einzelnen Zeichen zugelassen. Ein Zeichen kann eingefügt werden, gelöscht werden oder geändert werden. Jeder dieser drei möglichen Edieroperationen werden nichtnegative Kosten zugeordnet. Die Kosten einer Folge von Edieroperationen ergibt sich aus der Summe der Kosten der in der Folge enthaltenen Edieroperationen. Die Levenshtein-Distanz zweier Zeichenketten a und b entspricht den minimalen Kosten aller Folgen von Edieroperationen, die a in b überführen.

Listing 2.2 zeigt eine Implementierung in Java zur Bestimmung der Levenshtein-Distanz zweier Zeichenketten, bei der die Edieroperationen Einfügen, Löschen und Ändern eines Zeichens alle gleichgewichtet mit 1 bewertet werden. Ebenso wie bei der

Berechnung der längsten gemeinsamen Teilfolge entsteht auch hier durch die Konstruktion einer $(n+1) \times (m+1)$-Matrix ein Aufwand von $O(nm)$. Die Einträge der Matrix werden im Einzelnen wie folgt berechnet: Die Einträge der 0. Spalte entsprechen den Zeilenindizes. Ebenso wird die Zeile 0 mit den Spaltenzahlen belegt. Die inneren Werte der Matrix werden wieder zeilenweise bestimmt. Für den Eintrag an der Stelle $[i, j]$ wird der Eintrag an der Stelle $[i-1, j-1]$ übernommen, falls die Zeichen an den Stellen i in a und j in b übereinstimmen. Andernfalls wird der Eintrag $[i, j]$ auf den niedrigsten Vorgängereintrag $[i-1, j-1]$,$[i-1, j]$ oder $[i, j-1]$ gesetzt und um 1 erhöht, da, falls die Zeichen an der aktuellen Position nicht korrespondieren, eine Änderungsoperation nötig ist. Nachdem die Einträge für alle Zeilen berechnet sind, befindet sich der gesuchte Edierabstand im letzten Eintrag der Matrix an Position $[n, m]$.

Für den Vergleich von Zeichenketten der eigenentwickelten Verfahren in Kapitel 4 ist die Berechnung des Edierabstands ausreichend. Aus der erstellten Matrix kann bei Bedarf auch eine Folge der Edieroperationen abgelesen werden. Dazu werden beginnend bei dem letzten Eintrag $mx\,[n, m]$ der Matrix die Entscheidungen des Verfahrens in einem rekursiven Verfahren nachvollzogen, bis der Eintrag $[0, 0]$ erreicht wird. Falls $mx\,[i-1, j] + 1 = mx\,[i, j]$ gilt, entspricht dies dem Einfügen des j-ten Zeichens in b. Falls $mx\,[i, j-1] + 1 = mx\,[i, j]$ gilt, entspricht dies dem Löschen des j-ten Zeichens in b. Ansonsten wird der diagonale Pfad $mx\,[i-1, j-1]$ weiterverfolgt. Falls dabei $mx\,[i-1, j-1]+1 = mx\,[i, j]$ gilt, entspricht dies einer Änderungsoperation, die das i-te Zeichen aus a in das j-te Zeichen von b transformiert. Der Einsatz von Zeigern bei der Berechnung der Matrix im ersten Schritt zur Markierung der getroffenen Entscheidungen, erleichtert das Nachvollziehen der Pfade (vgl. [SK83]). In einem Backtracking-Verfahren können bei Bedarf so alle möglichen Lösungen bestimmt werden.

Listing 2.2: Levenshtein-Abstands-Verfahren in Java

```
double levenshtein(String name,String name2) {
  int n = name.length();
  int m = name2.length();
  int[][] mx = new int[n+1][m+1];

    for (int i = 0; i <= n; i++)
      mx[i][0] = i;
    for (int j = 0; j <= m; j++)
      mx[0][j] = j;

    for (int j = 1; j <= m; j++){
    for (int i = 1; i <= n; i++){
      if( name.charAt(i-1) == name2.charAt(j-1))
        mx[i][j] = mx[i-1][j-1];
      else{
          // Löschen oder Einfügen
        mx[i][j] = Math.min(mx[i-1][j]+1, mx[i][j-1]+1);
          // Ersetzen
```

```
            mx[i][j] = Math.min(mx[i][j], mx[i-1][j-1]+1);
            }
          }
        }
        return mx[n][m];
    }
```

Beispiel 3 (Levenshtein-Distanz zwischen `Stunde` und `Student`)**.** Abbildung 2.2 enthält die Levenshtein-Matrix, die für den Vergleich der Zeichenketten `Stunde` und `Student` konstruiert wird. Der letzte Eintrag enthält die Edierdistanz in Höhe von 3. Die drei benötigten Edieroperationen für die Transformation (`Stunde`→`Student`) sind das Löschen von `n` und das Einfügen von `n` und `t`.

		S	t	u	d	e	n	t
	0	1	2	3	4	5	6	7
S	1	0	1	2	3	4	5	6
t	2	1	0	1	2	3	4	5
u	3	2	1	0	1	2	3	4
n	4	3	2	1	1	2	2	3
d	5	4	3	2	1	2	3	3
e	6	5	4	3	2	1	2	[3]

		S	t	u	d	e	n	t
	↖ 0	1	2	3	4	5	6	7
S	1	↖ 0	1	2	3	4	5	6
t	2	1	↖ 0	1	2	3	4	5
u	3	2	1	↖ 0	1	2	3	4
n	4	3	2	↑ 1	1	2	2	3
d	5	4	3	2	↖ 1	2	3	3
e	6	5	4	3	2	↖ 1	← 2	← [3]

Abbildung 2.2: Levenshtein-Matrix des Vergleichs `Stunde`, `Student`

2.1.2 Textdateien

Auf den Vergleich von Zeichenketten lassen sich Verfahren zum Vergleich einfacher Textdokumente aufbauen, indem diese als eine Folge von Zeilen betrachtet werden. Das Verfahren von Hunt [HS77] basiert auf der längsten gemeinsamen Teilfolge, während das Verfahren von Tichy [Tic84] zusätzlich Blockverschiebungen berücksichtigt. Neben zeilenbasierten Ansätzen existieren auch feingranularere Verfahren, die Zeichen als kleinste, atomare Einheiten eines Textdokuments betrachten. Die Vergleichsverfahren für Textdokumente sind bereits lange im Einsatz, vielseitig einsetzbar und bewährt und berechnen

schnell einen exakten Edierabstand zwischen zwei Dokumenten, ohne auf eindeutige Objektbezeichner oder protokollierte Änderungsoperationen zurückzugreifen [FW07].

Da sich Klassendiagramme als XML-Serialisierung in einer Textrepräsentation darstellen lassen, stellt sich die Frage, ob der Vergleich der serialisierten Klassendiagramme mit zeichenbasierten Textvergleichsverfahren sinnvoll ist. Für die serialisierten Klassendiagramme aus den Abbildungen 1.2 und 1.3 wurde ein Vergleich mit dem Textvergleichsverfahren aus Eclipse durchgeführt. Die Abbildung 2.3 zeigt das Ergebnis des Vergleichs, wobei korrespondierende, nichtidentische Textblöcke zwischen identischen Textblöcken farblich hinterlegt und durch Linien verbunden sind. Eine Korrespondenz unterschiedlicher Textblöcke kann je nach Wahl der zulässigen Edieroperationen als Differenz interpretiert werden, bei der der eine Block durch den anderen Block ersetzt oder der eine Block gelöscht und der andere Block eingefügt wird. Das Vergleichsergebnis der beiden serialisierten Klassendiagramme in Abbildung 2.3 zeigt entscheidende Nachteile des Textvergleichs auf. Obwohl die Reihenfolge der Elemente innerhalb von Klassendiagrammen keine Bedeutung hat, führt die Änderung in der Reihenfolge der Klassen, die Klasse `Item` befindet sich im linken Dokument am Ende des Dokuments und im rechten Dokument in der Mitte, zu Differenzen. Auch Leerzeilen ohne semantische Bedeutung für das Klassendiagramm verursachen Differenzen. Da das textbasierte Vergleichsverfahren zeichenbasiert bzw. zeilenbasiert arbeitet, identifiziert es korrespondierende Blöcke von Zeilen, ohne auf die Zugehörigkeit der Zeilen zu Klassen Rücksicht zu nehmen. Im angegebenen Beispiel wird die Zeile 16 auf der linken Seite mit dem Block von Zeile 16 bis 29 als korrespondierend identifiziert, wobei in der linken Auswahl nur die Kopfzeile der Klasse `USAddress` enthalten ist, während sich die rechte Auswahl aus der Klasse `Item` und Teilen der Klasse `USAddress` zusammensetzt. Auch wenn der textbasierte Vergleich von serialisierten Klassendiagrammen bei einer geringen Zahl an Änderungen in Einzelfällen ausreichend sein kann, sind die ermittelten Differenzen im Allgemeinen jedoch nicht zum Verschmelzen geeignet. Das Verschmelzen zweier Klassendiagramme auf Basis der auf diese Weise berechneten Unterschiede kann zu syntaktisch nicht korrekten Klassendiagrammen führen (vgl. auch [ASW09]).

2.2 Hierarchische Strukturen (Bäume und XML-Dateien)

Im Gegensatz zu einfachen Textdokumenten liegt den XML-Dokumenten eine hierarchische Struktur von Elementen verschiedener Typen zugrunde, die durch Attribute genauer charakterisiert werden können. In der serialisierten Form von Klassendiagrammen werden die Kompositionsbeziehungen von Klassen und deren Unterelementen als Baumstruktur abgebildet. Querverweise auf Modellebene, wie zum Beispiel Vererbungskanten bei Klassendiagrammen, werden in dieser Darstellungsform als Referenzen in Attributform gespeichert. Vor den speziellen Verfahren für XML-Dokumente wird zunächst auf allgemeinere Baumvergleichsverfahren eingegangen. In beiden Fällen werden nur Verfahren betrachtet, die ohne eindeutige Objektbezeichner arbeiten.

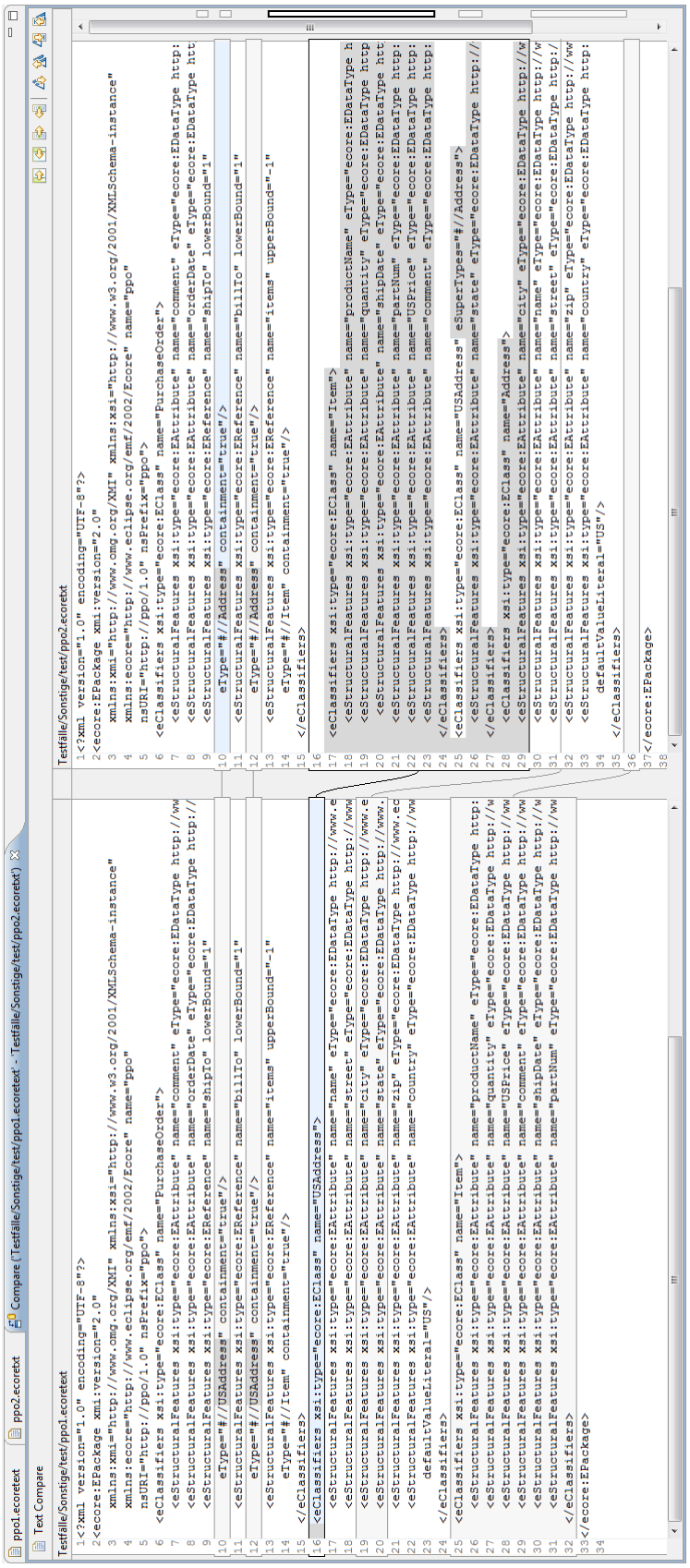

Abbildung 2.3: Ergebnis des Textvergleichs zweier Klassendiagramme in XML-Darstellung. Korrespondierende, nicht identische Textblöcke sind farblich hervorgehoben und durch eine Linie verbunden.

2.2.1 Baumvergleichsverfahren

Ein Baum ist ein zusammenhängender Graph, der keine Zyklen enthält und bei einer Anzahl von n Knoten genau $n-1$ Kanten besitzt (vgl. [Jun08, S. 7]). Wir betrachten im Folgenden Wurzelbäume, die einen ausgezeichneten Knoten, die Wurzel enthalten, der bei einer hierarchischen Anordnung auf Ebene $i = 0$ angeordnet wird. Ein Knoten c der Ebene $i > 0$ besitzt lediglich eine Kante zu einem Knoten der Ebene $i - 1$. Dieser Knoten wird als Vaterknoten von c bezeichnet. Alle Knoten der Ebene i, die mit dem Knoten v aus der Ebene $i - 1$ über eine Kante verbunden sind, werden als Kindknoten des Knotens v bezeichnet. Von einem geordneten Baum spricht man, wenn die Reihenfolge der Kindknoten jedes Knotens relevant ist. Knoten ohne Kindknoten werden als Blattknoten und alle Knoten, die weder Wurzelknoten noch Blattknoten sind, als innere Knoten bezeichnet. Falls die Knoten jeweils einen Namen (engl. label) besitzen, handelt es sich um einen attributierten Baum (engl. labeled tree).

Die Vergleichsverfahren für solche attributierte, ungetypte Bäume lassen sich in zwei Gruppen unterteilen, von denen die Verfahren der einen Gruppe lediglich geordnete Bäume vergleichen, während die anderen allgemeineren Verfahren auf diese Einschränkung verzichten. Für beide Gruppen sind sowohl Ansätze, die eine Zuordnung mit minimalen Kosten im Sinne des Edierabstands berechnen, sowie Alignment-Verfahren bekannt.

Edierdistanz auf Bäumen

Das Tree-Editing Problem zur Berechnung eines kostenminimalen Editskripts zur Transformation eines Baums T_1 in einen Baum T_2 wird in der Literatur auch als Tree-To-Tree-Correction Problem bezeichnet. Die zulässigen Edieroperationen umfassen in den im Folgenden beschriebenen Ansätzen von Zhang und Shasha das Löschen und Einfügen von Knoten und das Ändern von Knotennamen. Wie in Abbildung 2.4 zu sehen ist, weisen die Einfüge- und Löschoperationen innerer Knoten dabei eine Besonderheit auf: Beim Einfügen eines Knotens a als Kindknoten des Knotens b kann beispielsweise auch eine Teilmenge der bisherigen Kindknoten von b als neue Kindknoten von a nach unten verschoben werden. Analog werden die Kindknoten eines gelöschten Knotens als Kindknoten dessen ehemaligen Vaterknotens nach oben verschoben. Bei geordneten Bäumen muss dabei jeweils die Reihenfolge eingehalten werden; die Teilmenge der Kindknoten, die verschoben wird, besteht jeweils aus direkt aufeinanderfolgenden Knoten und deren Reihenfolge wird auch durch die Verschiebung nicht geändert.

In [ZS89] beschreiben Zhang und Shasha einen Abstandsbegriff auf *geordneten* Bäumen. Die zulässigen Edieroperationen werden dabei mit Kosten bewertet, die die Bedingungen einer Abstandsmetrik erfüllen. Zur Berechnung des Abstands zwischen zwei Bäumen wird eine eindeutige Zuordnung mit minimalen Kosten gesucht, die sowohl die Reihenfolge der Kindknoten von links nach rechts als auch die hierarchische Ordnung der Knoten erhält. Das Verfahren zur Berechnung des minimalen Abstands basiert auf dynamischer Programmierung und benötigt einen Zeitaufwand von $O(|T_1| * |T_2| * min(tiefe(T_1), blattknotenzahl(T_1)) * min(tiefe(T_2), blattknotenzahl(T_2)))$. Mit gleicher

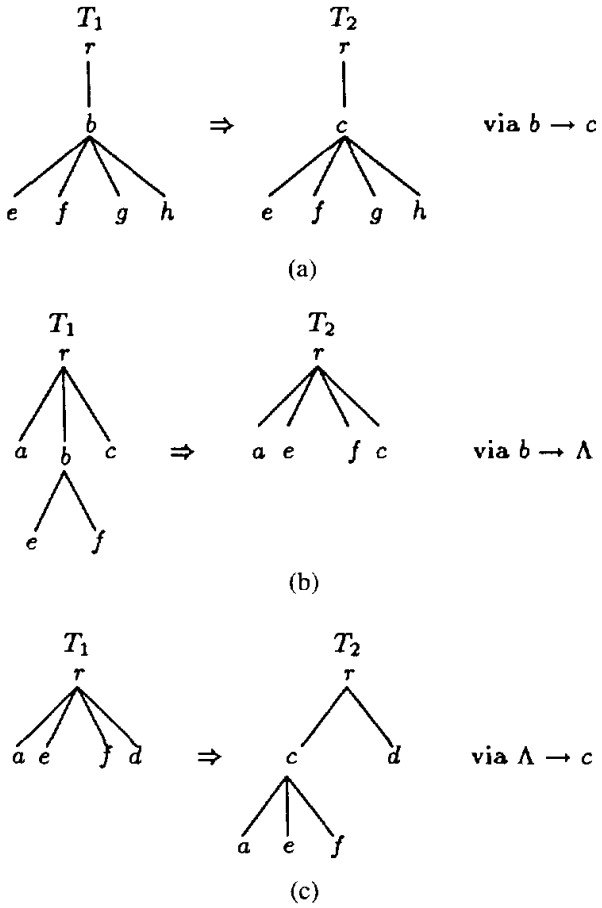

Abbildung 2.4: Darstellung der Edieroperationen auf attributierten Bäumen (entnommen aus [WZJS94]): (a) Umbenennung eines Knotens (b) Löschen eines Knotens (c) Einfügen eines Knotens

Komplexität ist auch die Berechnung der Zuordnung möglich[1], aus der eine Folge von Edieroperationen abgeleitet werden kann.

Der Abstandsbegriff auf *ungeordneten* Bäumen ist analog definiert, wobei im Gegensatz zu geordneten Bäumen nun die Einfüge- und Löschoperationen sowie die gesuchte Zuordnung nicht die Reihenfolge der Kindknoten von links nach rechts beachten müssen. Aufgrund dieser Ausweitung der Kombinationsmöglichkeiten ergibt sich eine deutlich höhere Komplexität der Vergleichsverfahren. Zhang, Statman und Shasha weisen in [ZSS92] nach, dass der Vergleich ungeordneter Bäume NP-vollständig ist. In der Praxis werden daher approximative bzw. heuristische Verfahren mit deutlich besserem Laufzeitverhalten eingesetzt oder bei exakten Verfahren zusätzliche Einschränkungen der Zuordnung gefordert, die die Komplexität des Problems reduzieren. So berechnet das Verfahren, das in [Zha93] beschrieben wird, auf ungeordneten Bäumen eine strukturerhaltende Zuordnung, die getrennte Teilbäume auch auf getrennte Teilbäume abbildet. Der Ansatz basiert auf dynamischer Programmierung, wobei die Berechnung der Edierdistanz - ähnlich wie bei dem eigenentwickelten edierkostenbasierten Verfahren - in ein Netzwerkflussproblem transformiert wird, einen maximalen Fluss mit minimalen Kosten zu bestimmen. Da das in Abbildung 2.5 dargestellte Netzwerk Löschkanten nur für die Elemente der linken Seite des bipartiten Graphen vorsieht, ist es ein geschichtetes Netzwerk, auf dem schnellere Lösungsverfahren eingesetzt werden können. Die Komplexität wird mit $O(m * |f^*| * log_{2+\frac{m}{n}} n)$ bei einem Graph mit n Knoten, m Kanten und einer optimalen Flussstärke von f^* angegeben.

Im Gegensatz zu den vorgestellten Verfahren von Zhang und Shasha werden in den folgenden beiden Verfahren von Chawathe, Garcia-Molina et. al. erweiterte Edieroperationen betrachtet. Chawathe und Garcia-Molina entwickelten mit MH-Diff [CGM97] ein heuristisches Verfahren für ungeordnete Bäume, das ebenfalls auf Edierkosten basiert. Die Menge der zulässigen Edierkosten enthält neben Einfügen, Löschen und Update, wie in den Ansätzen von Shasha und Zhang beschrieben, auch das Verschieben und Kopieren von Teilbäumen, wobei die Kosten der Edieroperationen vom Benutzer vorgegeben werden können. Die gesuchte nichteindeutige Zuordnung wird in diesem Verfahren als kostenminimale Kantenüberdeckung eines bipartiten Graphen berechnet, wobei jeder Knoten zu mindestens einer ausgewählten Kante inzident sein muss, aber auch zu mehreren ausgewählten Kanten inzident sein kann. Um den Aufwand zu reduzieren, werden aus dem aufgestellten bipartiten Knotengraphen Kanten nach heuristischen Regeln entfernt. Damit ergibt sich für eine Knotenzahl von n ein Worst-Case-Aufwand von $O(n^3)$, wobei in [CGM97] angegeben wird, dass das Verfahren in vielen Fällen eher einen quadratischen Zeitaufwand benötigt.

Das Verfahren von Chawathe, Rajaraman, Garcia-Molina und Widom für geordnete Bäume, das in [CRGMW96] beschrieben wird, wurde unter anderem als LaDiff zur Differenzberechnung von Latex-Dokumenten implementiert und verwendet als Modell einen geordneten Baum, dessen Knoten einen Namen und einen Wert besitzen. Neben den Basisoperationen werden auch Verschiebungen von Teilbäumen berücksichtigt. Das

[1] In [ZS89] fehlen die Details für das Verfahren zur Berechnung einer Zuordnung, das laut Zhang und Shasha jedoch implementiert ist.

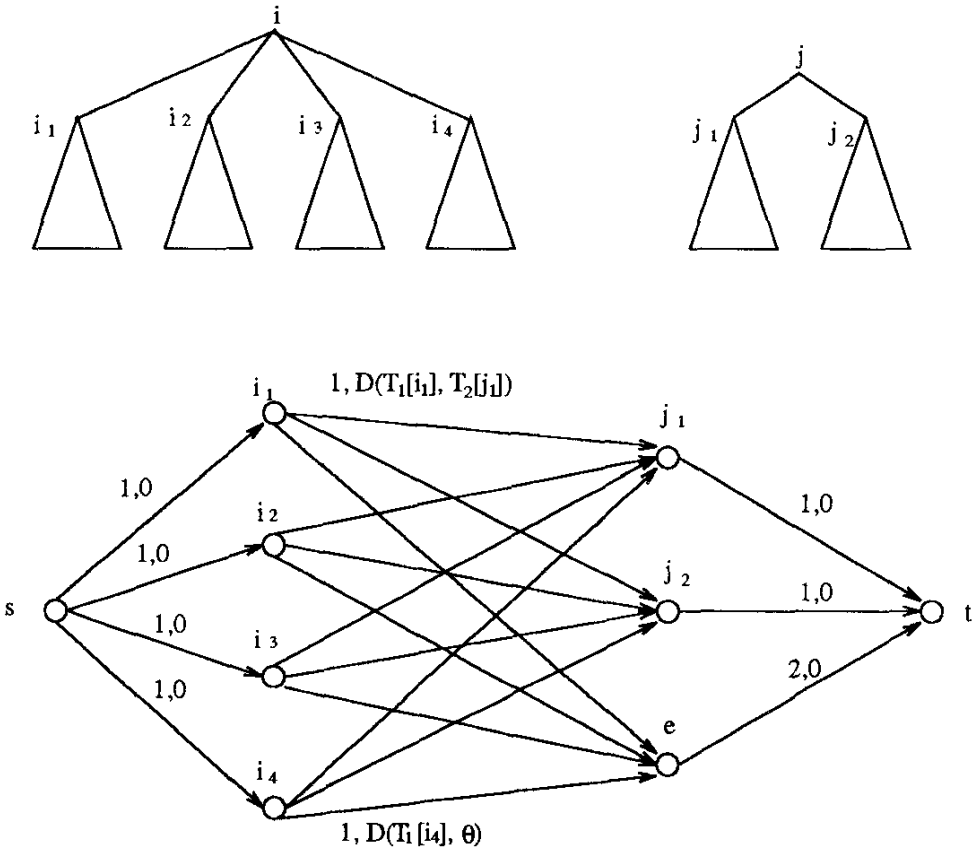

Abbildung 2.5: Übertragung der Teilbaumzuordnung auf ein Netzwerkflussproblem (entnommen aus [Zha93]): Die Knoten $i_1, \ldots, i_4$ repräsentieren die Teilbäume des linken Baums, die Knoten j_1 und j_2 die Teilbäume des rechten Baums. Der Knoten e steht für eine Teilbaumlöschung.

Einfügen und Löschen von Knoten bezieht sich hier, anders als in den Ansätzen von Zhang und Shasha, nur auf Blattknoten. Sollen innere Knoten eingefügt oder gelöscht werden, erfordert dies eine zusätzliche Verschiebung. Änderungsoperationen werden lediglich für den Wert des Knotens angeboten, die Namen, die z. B. Typen repräsentieren können, bleiben unverändert. Es wird eine eindeutige kostenminimale Zuordnung mit folgenden Einschränkungen gesucht:

- Nur Knoten mit gleichem Namen können korrespondieren.
- Es wird eine Höchstgrenze für die Edierkosten zwischen Blattknoten angegeben, bei deren Überschreitung keine Korrespondenz der Knoten mehr möglich ist.
- Innere Knoten können nur dann korrespondieren, wenn ein bestimmter Anteil ihrer Kindknoten korrespondieren.

Weiterhin wird angenommen, dass eine Ordnung der Namen existiert, so dass die verschiedenen Namen nicht in beliebiger Reihenfolge entlang der Hierarchie auftreten können, und dass die gewählte Methode zum Vergleichen der Werte in den Blattknoten diese hinreichend gut in ähnliche und unähnliche Werte unterteilt. Unter diesen Umständen lässt sich eine gute Zuordnung auf Basis von Longest Common Subsequence-Verfahren mit einem Aufwand von $O(ne + e^2)$ bei einer Anzahl von n Blattknoten und einer gewichteten Edierdistanz in Höhe von e berechnen. Dazu werden jeweils Ketten der von links nach rechts angeordneten Elemente mit gleichen Namen gebildet und darauf eine längste gemeinsame Teilfolge berechnet, um erste Zuordnungen zu finden. Die restlichen Korrespondenzen werden mittels linearer Suche gebildet.

Tree-Alignment

Tree-Alignment Verfahren werden als spezielle Form der Berechnung der Edierdistanz auf Bäumen betrachtet, bei der alle Einfügeoperationen den Löschungen vorausgehen [JWZ94]. In [JWZ94] wird jeweils ein Tree-Aligment Verfahren für geordnete und für ungeordnete attributierte Bäume beschrieben. Das Verfahren für zwei geordnete Bäume T_i mit einer Knotenzahl von $|T_i|$ und einem Knotengrad von $deg(T_i)$, $i = 1, 2$, benötigt einen Zeitaufwand von $O(|T_1| * |T_2| * (deg(T_1) + deg(T_2))^2)$. Im Gegensatz zu dem Tree-Edit-Distance Problem auf ungeordneten Bäumen ist das Tree-Aligment Problem auf ungeordneten Bäumen mit polynomiellen Algorithmen exakt lösbar, falls die Zahl der Kindknoten begrenzt ist [JWZ94]. Ohne Einschränkung des Knotengrads ist die Problemstellung NP-schwierig (engl. NP-hard).

Anwendbarkeit der beschriebenen Baumvergleichsverfahren auf Klassendiagramme

Es ist möglich, ein Klassendiagramm als Baum darzustellen, indem die Kompositionsstruktur als Kanten eines spannenden Baums dargestellt werden und Querverbindungen als kontextunabhängige Eigenschaften aufgelöst werden. Abbildung 2.6 zeigt eine solche

mögliche Darstellung des Klassendiagramms aus Abbildung 1.3, wobei das ursprüngliche Klassendiagramm um die Methode `getBillingAmount` erweitert wurde.[2]

Ein Grundpaket dient als Wurzelelement des Baums. Falls alle Klassen und Interfaces wie in Abbildung 2.6 im Grundpaket liegen, bilden diese Klassenelemente die Kindknoten der ersten Ebene. Deren Attribute, Methoden und Assoziationen bilden jeweils die nächste Unterebene. Die Ziele von Assoziationen werden als kontextunabhängige Eigenschaften, z. B. als Zeichenketten behandelt. Vererbungsbeziehungen zwischen Klassen oder Interfaces können entweder als Eigenschaft der Klasse, bzw. des Interfaces oder als kontextunabhängiges Unterelement behandelt werden. Im Fall von Methoden existiert für Übergabeparameter eine weitere Ebene. Während alle bisherigen Unterelementbeziehungen des Baums ungeordnet sind, ist die Reihenfolge der Übergabeparameter von Methoden relevant. Insgesamt ergibt sich auf diese Weise für ein Klassendiagramm, dessen Klassenelemente alle in einem Paket liegen, ein Baum mit dem Grundpaket als Wurzelelement auf Ebene 0 mit maximal 3 Unterebenen. Falls weitere Unterpakete existieren, bilden die Unterpaketbeziehungen die Kompositionsstruktur und die Klassenelemente liegen als Teilbäume wie beschrieben jeweils in ihrem umgebenden Paket. Entsprechend dem Typ besitzt jedes Element, das einen Knoten darstellt, eine Reihe von Eigenschaften. So besitzt beispielsweise ein Attribut die Eigenschaften Name und Typ.

Eine mögliche Menge zulässiger Edieroperationen auf dem Klassendiagrammbaum besteht aus typspezifischen Einfüge- und Löschoperationen sowie Änderungsoperationen. Das Einfügen von Elementen ist für jeden Elementtyp nur an bestimmten Stellen des Baumes erlaubt. So kann beispielsweise ein Attribut nur als Unterelement einer Klasse eingefügt werden. Auch die Löschoperationen müssen auf den Elementtyp angepasst werden. Das Löschen einer Klasse beinhaltet sinnvollerweise das Löschen des kompletten Teilbaums, für den die zu löschende Klasse das Wurzelelement darstellt. Typspezifische Änderungsoperationen erlauben Wertänderungen der Eigenschaften, wie zum Beispiel das Ändern eines Klassennamens. Neben diesen grundlegenden Operationen sind auf Paketebene zusätzlich Verschiebeoperationen sinnvoll, so dass einzelne Klassenelemente oder Unterpakete mit enthaltenen Klassen und Unterpaketen in andere Pakete verschoben werden können.

Die Vergleichsverfahren für geordnete Bäume skalieren zwar wesentlich besser als die Verfahren für ungeordnete Bäume, ein Vergleichsverfahren für geordnete Bäume liefert im Allgemeinen jedoch keine optimale Lösung für ungeordnete Bäume, auch wenn für diese eine künstliche Ordnung erzeugt wird. Im Gegensatz zu allgemeinen ungeordneten Bäumen, deren Knoten lediglich Namen besitzen, enthält die Kompositionsstruktur von Klassendiagrammen verschiedene Knotentypen und ist teilweise ungeordnet und teilweise geordnet. Die für allgemeine Bäume definierten Edieroperationen sind daher nicht direkt auf die Baumstruktur von Klassendiagrammen übertragbar, da abhängig vom Knotentyp nur bestimmte Edieroperationen zulässig sind. So kann ein Übergabeparameter beispielsweise nicht als direktes Unterelement einer Klasse eingefügt werden.

[2] Dabei ist zu beachten, dass der in Abbildung 2.6 gezeigte Baumeditor das Klassendiagramm zwar als Baum visualisiert, das Klassendiagramm aber nicht als solchen behandelt. Die im Editor angebotenen Edieroperationen operieren auf einem Modell mit Querverbindungen.

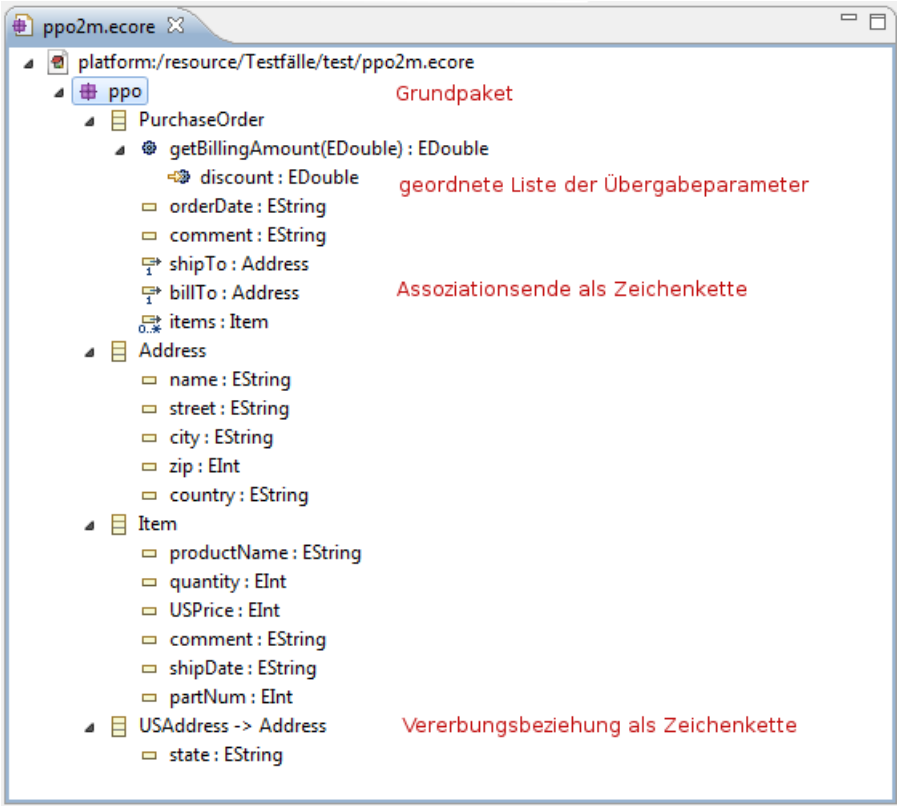

Abbildung 2.6: Das Klassendiagramm aus Abbildung 1.3 erweitert um die Methode `getBillingAmount` im Ecore-Baumeditor der Entwicklungsumgebung Eclipse

Auch ist die Auffassung von Einfüge- und Löschoperationen innerer Knoten, wie in [ZS89, Zha93, CGM97], bei denen Kindknoten in der Hierarchie nach unten bzw. oben mitverschoben werden, auf Ebene der Klassen und ihrer Unterelemente nicht sinnvoll. Die vorgestellten Ansätze sind daher nicht auf Klassendiagramme anwendbar. Weiterhin kann die Projektion eines Klassendiagramms auf eine Baumstruktur die Querverbindungen nicht ausreichend berücksichtigen, wie im folgenden Abschnitt über XML-Vergleichsverfahren deutlich wird.

2.2.2 XML-Vergleichsverfahren

Differenzberechnungsverfahren für XML-Dokumente basieren nach [RPB09] üblicherweise entweder auf Tree-To-Tree Correction oder Sequence-Alignment Verfahren. Damit stellen XML-Vergleichsverfahren spezielle Baumalgorithmen dar, die die spezifische XML-Syntax berücksichtigen. So werden Elementknoten, Attribute und Textknoten unterschiedlich behandelt. Der Großteil dieser Verfahren arbeitet auf einem geordneten Baummodell, da nach der Document Object Model (DOM) Spezifikation die Reihenfolge der Elemente innerhalb von XML-Dokumenten von Bedeutung sein kann [WDC03] und, wie in Abschnitt 2.2.1 erwähnt, die Verfahren für geordnete Bäume deutlich bessere Laufzeiten aufweisen[3].

Beispiele für XML-Vergleichsverfahren, die auf Tree-To-Tree Correction basieren, sind DocTreeDiff [RPB09] und XyDiff [CAM02] für geordnete XML-Dokumente und X-Diff [WDC03] für ungeordnete XML-Dokumente. Das Verfahren DocTreeDiff [RPB09] verwendet als Menge zulässiger Edieroperationen neben Einfüge- und Löschoperationen von Teilbäumen und Änderungsoperationen auch Teilbaumverschiebungen. Das Verfahren wurde für die XML-Darstellung von Office-Dokumenten entworfen und geht davon aus, dass die Bäume sehr viele identische innere Knoten enthalten, die die Struktur des Dokuments beschreiben, und der eigentliche Inhalt in den Blattknoten abgelegt ist. Daher wird zunächst eine größte gemeinsame Teilfolge auf einer reihenfolgetreuen Hashdarstellung der Blattknoten berechnet und darüber die ersten Blattknoten als korrespondierend identifiziert. Ausgehend von diesen Korrespondenzen werden die Baumstrukturen beider Dokumente bis zur Wurzel parallel in einem dynamischen Programmieransatz durchlaufen. Das Verfahren benötigt einen Aufwand von $O(leaves(T) * D + n)$, wobei D die Anzahl der durchgeführten Editoperationen auf Blattebene und n die Anzahl innerer Knoten darstellt.

Das Verfahren XyDiff berücksichtigt Löschoperationen, Einfügeoperationen und Verschiebungen von Teilbäumen sowie Änderungen der Werte von Textknoten und Attributen. Zur Bestimmung korrespondierender Knoten verfolgt das Verfahren eine heuristische Greedy-Strategie: Über einen Signaturvergleich werden möglichst große identische Teilbäume identifiziert. Ausgehend von den korrespondierenden Teilbäumen werden die Korrespondenzen auf Nachfolgerknoten und Vorgängerknoten erweitert. Da nicht alle möglichen Paare von Teilbäumen miteinander verglichen werden, sondern die Größe der

[3] Diese Aussage basiert auf dem Vergleich der exakten Lösungsverfahren des Tree-To-Tree Correction Problems nach Zhang und Shasha für geordnete und ungeordnete Bäume bei gleichen zulässigen Edieroperationen ohne Verschiebungen.

Teilbäume berücksichtigt wird, ergibt sich für das XyDiff-Verfahren ein Aufwand von $O(n * log(n))$.
Der Tree-to-Tree Correction Algorithmus X-Diff für XML-Dokumente, der in [WDC03] beschrieben ist, basiert auf dem vorgestellten Verfahren [Zha93] von Zhang für ungeordnete Bäume. Es wird vorausgesetzt, dass nur Elemente des gleichen Typs korrespondieren können. Neben Einfüge- und Löschoperationen für Blattknoten, werden auch Teilbaumeinfügungen und -löschungen, aber keine Verschiebungen betrachtet. Wie in [Zha93] basiert der Ansatz auf dynamischer Programmierung. Der Berechnung der Edierdistanz zwischen eventuell korrespondierenden Teilbäumen als Netzwerkflussproblem analog zu [Zha93] wird jedoch eine Hashphase vorangestellt. Über den Vergleich von Hashwerten werden identische Teilbäume in verschiedenen Ebenen der Bäume im Vorfeld identifiziert, um die Komplexität der Berechnung der Zuordnung zu reduzieren.

Das Verfahren Faxma, das in [LKT06] beschrieben wird, berechnet die korrespondierenden Elemente zwischen geordneten XML-Dokumenten mit einem Sequence-Alignment Ansatz, dessen Worst-Case-Aufwand in $O(n^2)$ liegt. Das Verfahren transformiert den XML-Baum in ein Token-Sequence-Modell und wendet darauf einen heuristischen zeichenbasierten Vergleich an: Über Hashvergleiche wird in einem Greedy-Verfahren die längste gemeinsame Teilfolge bestimmt. Als Operationen auf den Sequenzen sind Einfüge-, Lösch- und Verschiebeoperationen zugelassen; Update-Operationen sind nicht enthalten. Der Nachteil, den Baum zunächst in eine Zeichenfolge zu übertragen und darauf die Differenzberechnung durchzuführen, liegt darin, dass nicht ohne Weiteres sichergestellt werden kann, dass das Ergebnis der Differenzberechnung nach der Rückabbildung auf die Baumebene syntaktisch korrekt ist. In [RPB09] wird kritisch angemerkt:

> „Due to the linearization of the XML documents in sequences, faxma neglects the XMl hierarchy [...]“

Bei Tests mit dem Verfahren Faxma zeigt sich, dass die Differenzberechnung oder der Differenzbericht nicht feingranular genug ist. Eine Änderung des Zieltyps einer Referenz führt zum Einfügen der gesamten Zeile, in der die Referenz enthalten ist. Da die Zuordnung der Unterelemente unabhängig von den Elementen stattfindet, können bei mehreren gleichen Attributen verschiedener Klassen vertauschte Zuordnungen auftreten.

Anwendung der XML-Vergleichsverfahren auf serialisierte Klassendiagramme

Im Folgenden wird betrachtet, inwieweit die Anwendung von XML-Vergleichsverfahren auf die XML-Dokumente serialisierter Klassendiagramme sinnvoll ist. Dabei wurden die Ergebnisse dreier XML-Vergleichsverfahren, die kommerziellen Softwareprodukte von Oxygen[4] und Altova[5] sowie die frei verfügbare Implementierung des Faxma Verfahrens [LKT06] untersucht.

Diejenigen Verfahren, denen Baumvergleichsverfahren für geordnete Bäume zugrunde liegen, wie das getestete Verfahren von Oxygen sowie das Verfahren Faxma, sind für

[4] http://www.oxygenxml.com/diff
[5] http://www.altova.com/diffdog/diff-merge-tool.html

den Vergleich serialisierter Darstellungen von Klassendiagrammen weniger geeignet, da die Reihenfolge der Elemente in der Regel nicht relevant ist und lediglich bei den Übergabeparametern von Methoden eine Berücksichtigung der Reihenfolge sinnvoll ist. So können zwei XML-Dokumente mit einer abweichenden Reihenfolge der Klassen dennoch zwei semantisch gleiche Klassendiagramme repräsentieren (vgl. Abbildung 2.7). Die angezeigte Differenz würde in diesem Fall jedoch nicht mit einem Unterschied im Modell übereinstimmen. Während die Software von Oxygen lediglich die Möglichkeit bietet, die Reihenfolge der Attribute von Elementen unberücksichtigt zu lassen, kann das Verfahren von Altova so konfiguriert werden, dass Reihenfolgeänderungen der Elemente nicht angezeigt werden. Ob das Verfahren in diesem Fall einen Algorithmus für ungeordnete Bäume verwendet oder aber die angezeigten Differenzen lediglich filtert, ist nicht bekannt.

Auch wenn unklar bleibt, welche Algorithmen in den kommerziellen Werkzeugen verwendet werden, zeigen alle drei Vergleichsergebnisse gemeinsame Fallstricke, deren Ursache auf die Projektion der Modelle auf eine Baumstruktur zurückzuführen ist. Als Beispiel werden hier zwei Klassendiagramme verglichen, die sich nur im Namen einer Klasse unterscheiden, die eine Oberklasse zweier anderer Klassen ist. Die Abbildung 2.8 zeigt das Ergebnis des Modellvergleichs, das nur einen Unterschied, die Umbenennung, meldet. Die Abbildungen 2.9 und 2.10 zeigen die Vergleichsergebnisse der Verfahren von Altova und Oxygen, die beide jeweils drei Unterschiede berichten[6]. Neben der Umbenennung der Klasse werden auch die Änderungen der Referenzen als Unterschiede erkannt, da die Textdarstellungen der referenzierten Objekte verglichen werden.

Die Repräsentation der Unterschiede auf der Ebene der XML-Serialisierung ist von sich aus bereits sehr unübersichtlich. Werden zudem neben den Unterschieden auf Modellebene auch viele nicht relevante Unterschiede angezeigt, die auf Reihenfolgeänderungen basieren oder darauf zurückzuführen sind, dass die Referenzen nicht inhaltlich geprüft wurden, ist das Vergleichsergebnis für den Betrachter nicht hilfreich. Auch für das Verschmelzen von Klassendiagrammen ist der Differenzbericht aus dem XML-Vergleich der serialisierten Darstellung nicht ohne weitere Verarbeitung einsetzbar. In beiden Fällen müssen die berechneten Unterschiede zunächst auf die Modellebene zurückgeführt werden, bevor sie weiterverwendet bzw. visualisiert werden.

2.3 Graphen

Da Vergleichsverfahren für Baumstrukturen für den Vergleich von Klassendiagrammen nicht ausreichen, da diese auch Querverbindungen besitzen, werden nun allgemeine Graphen betrachtet. Der Begriff Graph Matching wird in [CFSV04] wie folgt definiert:

> „Graph matching is the process of finding a correspondence between the nodes and the edges of two graphs that satisfies some (more or less stringent) constraints ensuring that similar substructures in one graph are mapped to similar substructures in the other.“

[6] Das Faxma-Verfahren liefert als Ergebnis ein erweitertes XML-Dokument zurück, das neu eingefügte Knoten sowie Querverweise auf Knoten der verglichenen Dokumente enthält. Auf die Abbildung des Vergleichsergebnisses wird an dieser Stelle aus Gründen der Unübersichtlichkeit verzichtet.

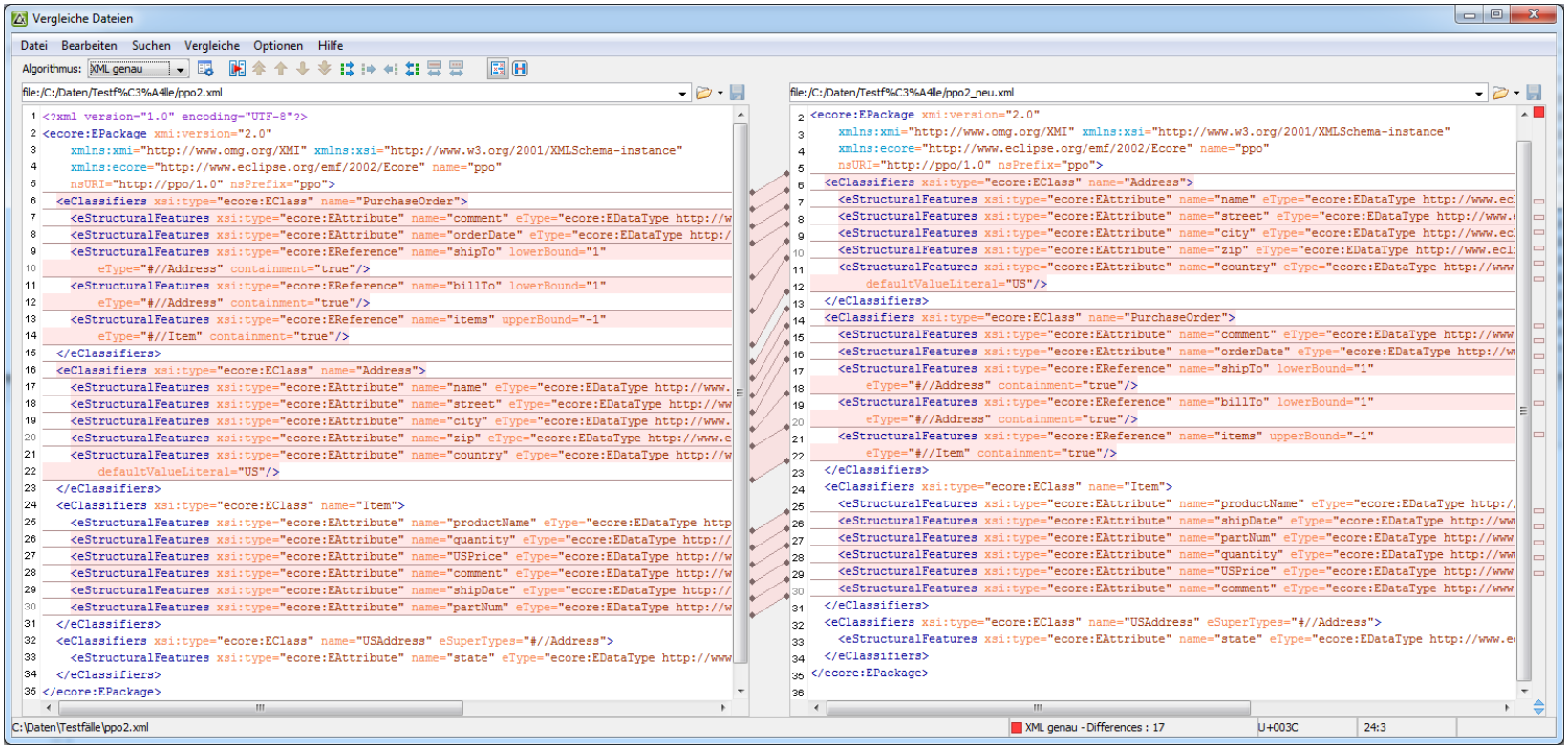

Abbildung 2.7: Ergebnis des XML-Vergleichs mit Oxygen Diff Files 3.0 zweier Klassendiagramme, die semantisch identisch sind, sich jedoch in der serialisierten Darstellung unterscheiden: Die Reihenfolge der Klassen `Address` und `PurchaseOrder` und die Reihenfolge der Attribute innerhalb der Klasse `Item` wurden vertauscht

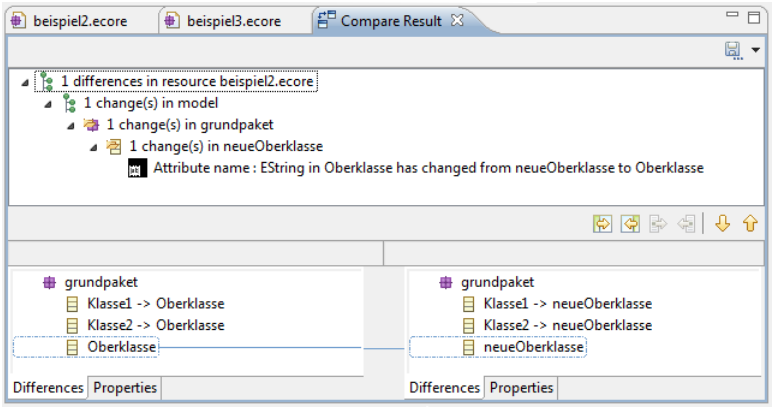

Abbildung 2.8: Modellvergleich der Klassendiagramme A und B

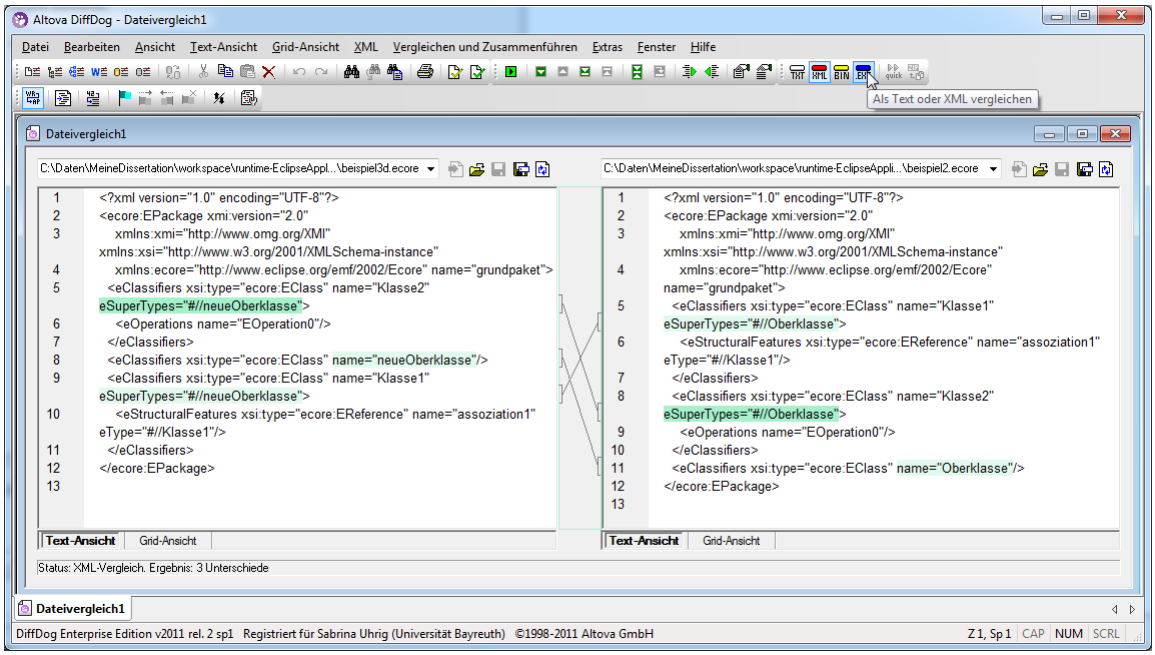

Abbildung 2.9: Ergebnis des XML-Vergleichs mit Altova

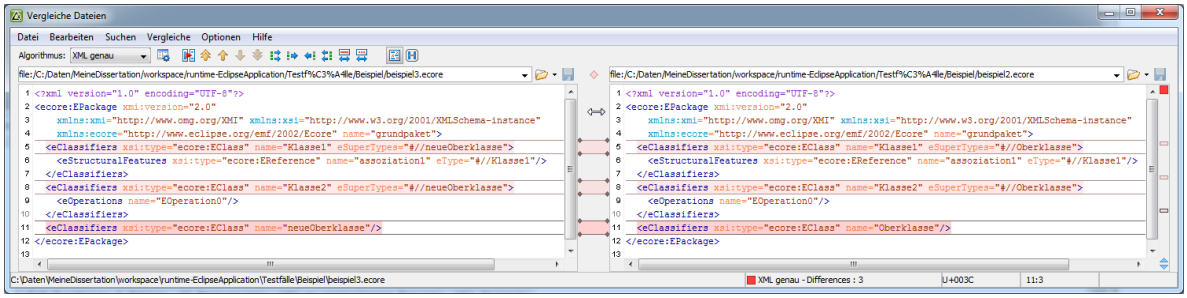

Abbildung 2.10: Ergebnis des XML-Vergleichs mit Oxygen Diff Files 3.0

In [CFSV04] wird ein Überblick über bekannte Graph Matching Verfahren aus Sicht der Mustererkennung gegeben, wobei zwischen *exact graph matching* und *inexact graph matching* unterschieden wird. An dieser Stelle beschränken sich die Ausführungen auf einige wenige ausgewählte Verfahren des nichtexakten Graph Matchings, die neueren Datums als die in [CFSV04] beschriebenen Verfahren sind [NRB06, RNB07], bzw. im Bereich des Modellvergleichs häufig zitiert werden [MGMR02].

2.3.1 Exaktes Graph Matching

Zu den Fragestellungen des exakten Graph Matchings zählen der Test auf Graphisomorphismus, Teilgraphisomorphismus und Homomorphismus zwischen zwei Graphen sowie die Suche nach dem größten gemeinsamen Teilgraphen zweier Graphen [CFSV04]. Der Test auf Graphisomorphismus überprüft, ob zwei gegebene Graphen gemäß Definition 5 zueinander isomorph sind.

Definition 5 (Isomorphe Graphen)**.** Zwei Graphen G=(V,E) und G'=(V',E') sind isomorph wenn eine bijektive Abbildung α: V $\rightarrow$ V' existiert, so dass gilt {a,b} $\in$ E $\Leftrightarrow$ {α(a),α(b)} $\in$ E' $\forall$ a,b $\in$ V. (nach [Jun08, S. 21])

Ein Teilgraphisomorphismus liegt dann vor, wenn ein Graph zu einem knoteninduzierten Teilgraphen eines anderen Graphen isomorph ist. Ein Homomorphismus stellt eine Abschwächung des Isomorphismus dar, bei dem die in Definition 5 genannte Abbildung nicht bijektiv sein muss. Als größter gemeinsamer Teilgraph zweier Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ werden zwei Teilgraphen $G'_1 = (V'_1 \subseteq V_1, E'_1 \subseteq E_1)$ und $G'_2() = (V'_2 \subseteq V_2, E'_2 \subseteq E_2)$ gesucht, die zueinander isomorph sind $G'_1 \cong G'_2$, wobei entweder die Zahl der Knoten oder die Zahl der Kanten des gemeinsamen Teilgraphen maximiert wird.

Die Verfahren des exakten Graph Matchings bestimmen somit kantenerhaltende Zuordnungen. Gilt $G_1 \cong G_2$ und $(x, y) \in E_1$, so muss auch $(\alpha(x), \alpha(y)) \in E_2$ gelten. Falls die Graphen G_1 und G_2 nicht isomorph, sondern nur homomorph sind, gilt die abgeschwächte Bedingung $(\alpha(x), \alpha(y)) \in E_2$ oder $\alpha(x) = \alpha(y)$. Laut [CFSV04] wurde für alle diese Probleme, mit Ausnahme des Graphisomorphismus, der Beweis geführt, dass sie NP-vollständig sind. Für das Isomorphismus-Problem sind polynomielle Verfahren nur für spezielle Graphen, wie z. B. Bäume oder planare Graphen, bekannt. Als typische Lösungsansätze für exakte Graph Matching Probleme werden in [CFSV04] Tiefensucheverfahren bzw. Branch-and-Bound Verfahren genannt.

Eine Ausweitung der Graphisomorphismus-Definition auf Klassendiagramme, die aus Knoten und Kanten unterschiedlicher Typen mit verschiedenen Eigenschaften bestehen, ist meines Erachtens nicht sinnvoll. Ein Test auf Isomorphie zweier Klassendiagramme ist, da meist unterschiedliche Klassendiagramme verglichen werden, wenig aussagekräftig. Auch der Test auf Teilgraphisomorphismus ist, wie im Folgenden gezeigt wird, nur bedingt hilfreich, da die knoteninduzierten Teilgraphen aufgrund der Anforderung der Kantenerhaltung für den Vergleich von Klassendiagrammen ungeeignet sind. Die Graphen A und B in Abbildung 2.11 repräsentieren jeweils ein Klassendiagramm. Diese Klassendiagramme unterscheiden sich darin, dass zwischen der Klasse `Klasse1`

und der Klasse `Klasse2` eine Assoziation entfernt und zwischen den Klassen `Klasse2` und `Klasse3` eine Assoziation eingefügt wurde. Da ein knoteninduzierter Teilgraph als größter gemeinsamer Teilgraph bestimmt wird und zwei Knoten nur dann im gemeinsamen Teilgraphen liegen, falls auch alle Kanten zwischen diesen beiden Knoten übereinstimmen, enthält der größte gemeinsame Teilgraph für das Beispiel nur drei Knoten. Die in der Abbildung dargestellten Graphen C_1 und C_2 stellen die beiden möglichen Lösungen für den größten gemeinsamen Teilgraphen der beiden Graphen A und B dar. Hilfreicher wäre vielleicht eine Menge disjunkter möglichst großer gemeinsamer Teilgraphen im Sinne eines unzusammenhängenden größten gemeinsamen Teilgraphen. Für das Beispiel enthielte diese Menge die Graphen C_1 und C_2. Die Berechnung ist jedoch zu komplex und vermutlich für Klassendiagramme auch nicht besonders gut geeignet, da die in Klassendiagrammen üblichen Änderungen an Assoziationen und Vererbungskanten die gemeinsamen Teilgraphen unter Umständen auf sehr kleine Fragmente reduzieren. Je kleiner die gemeinsamen Teilgraphen werden, desto mehr Informationen über den Kontext der Elemente geht dabei verloren.

2.3.2 Nichtexaktes Graph Matching

Die zweite Gruppe der nichtexakten Graph Matching Verfahren besteht aus Verfahren, die auch nichtidentische Teilgraphen zuordnen können. Statt die Zuordnung zweier Knoten zu verhindern, deren Kanten nicht vollständig aufeinander abgebildet werden können, wird eine solche Zuordnung stattdessen zum Beispiel mit Strafkosten belegt. Optimales nichtexaktes Graph Matching basiert auf einer Minimierung der Edierkosten und benötigt unter Umständen nicht weniger Rechenzeit als exaktes Graph Matching. Ein Edierkostenproblem zwischen zwei Graphen ist, wie in [BK00] gezeigt wurde, unter bestimmten Bedingungen mit der Suche nach dem größten gemeinsamen Teilgraphen identisch.[7] In [CFSV04] werden die Verfahren des nichtexakten Graph Matchings sogar als gewöhnlich deutlich aufwändiger als vergleichbare exakte Graph Matching Verfahren eingeschätzt. Daher werden vor allem suboptimale[8] Verfahren für nichtexaktes Graph Matching eingesetzt.

Im Folgenden werden zwei suboptimale Verfahren vorgestellt, die auf einer Minimierung der Edierkosten basieren. Das Verfahren Similarity-Flooding, das im Anschluss daran behandelt wird, verfolgt stattdessen die heuristische Strategie, Ähnlichkeitswerte von Knoten innerhalb des Graphen an benachbarte Knoten zu propagieren. Alle diese vorgestellten Verfahren basieren auf einem Graph-Modell mit gerichteten Kanten und Kanten- und Knotenbeschriftungen.

[7] In [BK00] werden Korrespondenzen nur zwischen identischen Knoten bzw. Kanten erlaubt, indem die Kosten für Änderungen an Knoten- und Kantenbezeichnungen als ∞ festgelegt werden.

[8] Der Begriff suboptimal wird hier als Oberbegriff für approximative und heuristische Verfahren verwendet.

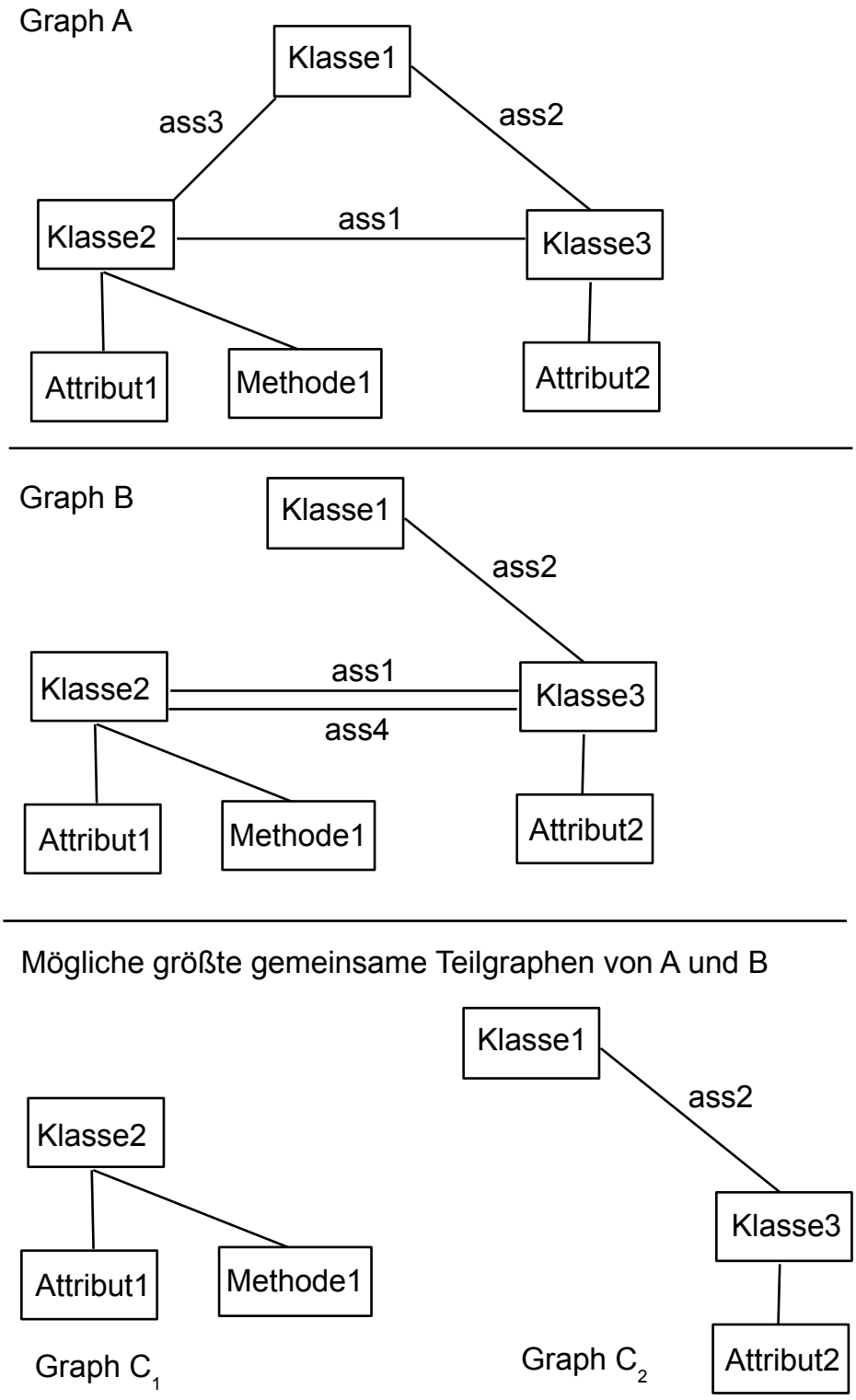

Abbildung 2.11: Mögliche größte gemeinsame Teilgraphen zweier Graphen, die Klassendiagramme repräsentieren.

Edierkostenbasiertes Graph Matching

Im Gegensatz zu den Edierdistanzberechnungen auf Bäumen, bei denen sich die Änderungsoperationen nur auf Knoten beziehen, können die Änderungsoperationen auf Graphen sowohl Knoten als auch Kanten betreffen. Eine gängige Festlegung von Edieroperationen auf Graphen findet sich zum Beispiel in [KN05, S. 343]. Dort werden vier elementare Edieroperationen festgelegt: Das Löschen einer Kante, das Einfügen einer Kante, das Einfügen eines Knotens sowie das Löschen eines Knotens. Hierbei ist allerdings zu beachten, dass die Operationen Löschen eines Knotens und Einfügen eines Knotens nicht zueinander invers sind, da mit dem Löschen eines Knotens gleichzeitig alle inzidenten Kanten ebenfalls gelöscht werden.[9] Auch hier wird eine Folge D von Edieroperationen op_i gesucht, die einen Graphen in einen anderen überführt $D(G) = op_n(...(op_2(op_1(G)))...)$. Jede Operation wird mit Kosten $c(op_i)$ bewertet, so dass sich für die Transformationsfolge Gesamtkosten in Höhe von $c(D)$ ergeben. Ein Paar aus einer Transformationsfolge und einem Graphisomorphismus wird in [KN05] als fehlerkorrigierender Graphisomorphismus bezeichnet und die Kosten der Transformationsfolge als Graph-Edit-Distanz, sofern sie minimal ist, wie der folgenden Definition 6 zu entnehmen ist.

Definition 6. (Fehlerkorrigierender Graph-Isomorphismus und Graph-Edit-Distanz) Ein Tupel (D, α) heißt *fehlerkorrigierender Graphisomorphismus (fkgI)*, falls D eine Folge von Edit-Operationen auf G_1 mit Kosten $c(D)$ und α ein Graphisomorphismus von $D(G_1)$ nach G_2 ist. Die *Graph-Edit-Distanz* $d(G_1, G_2)$ ist dann definiert durch

$$d(G_1, G_2) := min_{(D,\alpha)}\{c(D)|(D, \alpha) \text{ ist fgkI von } G_1 \text{ nach } G_2\}$$

(nach [KN05, S. 345])

In [NRB06] werden von Neuhaus, Riesen und Bunke zwei Varianten des A^*-Algorithmus [HNR68] zur Berechnung der Edierdistanz zwischen zwei Graphen angegeben. Der A^*-Algorithmus stellt eine Familie von Suchverfahren dar, die einen kostenminimalen Pfad in einem Graphen bestimmen. Zur Lösung des Zuordnungsproblems wird in [NRB06] ein Suchbaum im Lösungsraum aufgespannt, dessen Kanten Teilentscheidungen, d. h. die Korrespondenz von Elementen, repräsentieren. Diesen Kanten sind Kosten zugeordnet, die den Kosten der dazu nötigen Edieroperationen entsprechen. So können zu jedem Knoten v im Suchbaum die bisher entstandenen Kosten ermittelt werden, die als $g(v)$ bezeichnet werden. Das Ergänzen weiterer Knoten als Kindknoten eines anderen Knotens in dem Suchbaum wird im Folgenden als Weiterentwickeln bezeichnet und steht für die Festlegung weitere Korrespondenzen in der bisherigen Teillösung. Ein Blattknoten, der nicht mehr weiterentwickelt werden kann, repräsentiert eine Gesamtlösung. In dem Suchbaum wird jeweils der vielversprechendste Pfad, d. h. mit geringsten aktuellen Kosten, weiterverfolgt und weiterentwickelt. Für innere Knoten werden die Kosten

[9] In [KN05] werden diese vier Edieroperationen als elementare Operationen bezeichnet, obwohl das Löschen eines Knotens auch dessen inzidenten Kanten betrifft. Meines Erachtens sollten die elementaren Edieroperationen jeweils nur einen Knoten oder eine Kante des Graphen manipulieren, da ohne diese Einschränkung die Symmetrie der berechneten Edierdistanz im Allgemeinen nicht gewährleistet ist.

abgeschätzt und setzen sich aus bereits bekannten Kosten g durch die getroffenen Teilentscheidungen sowie den abgeschätzten Kosten h für noch notwendige Edieroperationen zusammen. Die abgeschätzten Kosten bilden hierbei eine untere Schranke der tatsächlichen Kosten. Wird ein Blattknoten erreicht und bleiben die Kosten im Vergleich zu den anderen Prognosen minimal, endet das Verfahren mit dieser Lösung. Bei der vollständigen Ausführung des Verfahrens kann es sein, dass alle Knoten bis in die Blattebene entwickelt werden müssen, so dass im ungünstigsten Fall kein Aufwand gegenüber einer Brute-Force-Methode eingespart werden kann. In [NRB06] werden daher zwei verschiedene Strategien verfolgt, um mit geringerem Aufwand eine möglichst gute suboptimale Lösung zu erhalten. In der Variante *A*-Beamsearch* wird nur eine bestimmte Anzahl an Baumknoten mit sehr geringen aktuellen Kosten in der Menge der potentiell weiterzuverfolgenden Knoten behalten. Dadurch werden voraussichtlich weniger erfolgreiche Äste des Suchbaums im Vorfeld ausgedünnt. In der zweiten Variante, *A*-Pathlength*, werden die aktuellen Kosten eines Baumknotens ins Verhältnis zur Länge des Edierpfades gesetzt, um gezielt längere und trotzdem kostengünstige Pfade weiterzuverfolgen.

In [RNB07] wird von Riesen, Neuhaus und Bunke ein weiteres nichtexaktes Graph Matching Verfahren beschrieben, das das Problem einer maximalen Zuordnung der Knoten zweier Graphen unter minimalen Kosten ähnlich wie das eigenentwickelte edierkostenbasierte Verfahren auf einem bipartiten Graphen löst. Zur Bestimmung der Zuordnung wird auf einer Matrix, die den bipartiten Graphen repräsentiert, der Algorithmus von Munkres [Mun57], eine Variante der Ungarischen Methode, mit einem Zeitaufwand von $O(n^3)$ eingesetzt, wobei n der Zahl der Knoten des größeren Graphen entspricht. In [RNB07] werden zwei verschiedene Varianten angegeben: In der Version des *PLAIN-MUNKRES* werden ausschließlich die Kosten für die Edieroperationen der Knoten berücksichtigt. Als *ADJACENCY-MUNKRES* wird die andere Version bezeichnet. Diese berücksichtigt die Kosten für Kantenoperationen bei der Bestimmung eines optimalen Knoten-Matchings insoweit, dass zur Berechnung der Edierkosten zwischen zwei Knoten jeweils ein analoges Matching-Problem für die Zuordnung der Kanten gelöst werden muss. Das Verfahren liefert nach Aussage der Autoren suboptimale Lösungen, die im Bereich der Mustererkennung brauchbare Ergebnisse darstellen.

In beiden vorgestellten Ansätzen der Autoren Riesen, Neuhaus und Bunke wird nicht darauf eingegangen, ob die Kosten der Kantenzuordnungen in den Modellen von der Zuordnung der anderen Knoten abhängig sind und wie dies gegebenenfalls gelöst wird. In [RNB07] wird jedoch von nur lokaler Berücksichtigung der Kantenoperationen gesprochen.

Graph Matching über die Propagierung von Ähnlichkeitswerten

Similarity Flooding, dargestellt in [MGMR02], bezeichnet sich als flexibler Graph Matching Algorithmus, der basierend auf einer Fixpunktberechnung eine Abbildung zwischen den korrespondierenden Elementen zweier Modelle ermittelt. Modelle werden dabei als beliebige Datenstrukturen verstanden, die als gerichtete attributierte Graphen aufgefasst werden. Das Verfahren geht davon aus, dass zwei Elemente ähnlich sind, falls die Elemente in ihrer Nachbarschaft ähnlich sind. Daher werden die Ähnlichkeiten von Ele-

menten in deren Nachbarschaft propagiert, woraus sich auch der Name des Verfahrens ableitet.

Die Vorgehensweise des Verfahrens wird in Abbildung 2.12 dargestellt: Aus den beiden Graphen der zu vergleichenden Modelle wird zunächst ein *pairwise connectivity graph* (PCG) erstellt, wobei jeder Knoten aus dem PCG einem 2-Tupel aus den Knoten der Modelle entspricht. Im PCG der Modelle A und B sind die Knoten (x, y) und (x'y') mit einer Kante von (x, y) zu (x', y') mit der Beschriftung p genau dann enthalten, falls im Modell A eine p-Kante von x nach x' und in Modell B eine Kante p von y zu y' existiert (vgl. Formel 2.1). Somit werden meines Erachtens in den PCG nur Knotenpaare mit ausgehenden oder eingehenden Kanten des gleichen Typs eingefügt und andere Paare, die isolierte Knoten in diesem Graphen darstellen würden, nicht berücksichtigt. Da die Ähnlichkeitswerte isolierter Knoten während der nachfolgenden Fixpunktiteration konstant bleiben, ist die Übertragung dieser Paare in den PCG nicht notwendig. Es ist davon auszugehen, dass diese Ähnlichkeitswerte beim Ablesen der Zuordnung am Ende dennoch berücksichtigt werden, auch wenn dies aus der Veröffentlichung in [MGMR02] nicht explizit hervorgeht.

$$((x, y), p, (x', y')) \in PCG(A; B) \iff (x, p, x') \in A \wedge (y, p, y') \in B \tag{2.1}$$

Aus dem PCG wird schließlich der *propagation graph* abgeleitet, in dem zu den gerichteten Kanten des PCGs auch entgegengerichtete Kanten eingeführt werden und alle Kanten Gewichte erhalten. Das Gewicht einer Kante liegt im Intervall $[0, 1]$ und entspricht dem Faktor, mit dem die Ähnlichkeit entlang der Kante weitergereicht wird. Diese Gewichte können wie im Beispiel dadurch bestimmt werden, dass der Faktor 1 für jeden Knoten zu gleichen Teilen auf alle ausgehenden Kanten des gleichen Typs aufgeteilt wird. Außerdem wird für jeden Knoten ein Ausgangswert für die Ähnlichkeit bestimmt, zum Beispiel durch eine Stringvergleichsmethode. Die Ähnlichkeitswerte der Knoten werden anschließend im Rahmen einer Fixpunktiteration aktualisiert. Da nicht sichergestellt ist, dass die Fixpunktberechnung terminiert, wird diese beendet, falls die Anpassung einen bestimmten Wert unterschreitet oder eine bestimmte Zahl an Iterationen überschritten ist. Aus den berechneten Ähnlichkeitswerten der verschiedenen Knoten des *propagation graphs* werden nun mit Hilfe von Filtern die korrespondierenden Elemente abgelesen. Je nach Wahl des Filters kann die ermittelte Zuordnung eindeutig sein oder aber auch Elemente enthalten, die zu mehreren Elementen korrespondieren. Das Verfahren bietet somit eine Vielzahl an Konfigurationsmöglichkeiten: die Berechnung der initialen Ähnlichkeitswerte, die Bestimmung der Gewichte im *propagation graph*, die Auswahl der Formel für die Fixpunktberechnung sowie die Wahl des Filters.

Die Ermittlung der Korrespondenzen verläuft nicht vollautomatisch, sondern erfordert es, dass das Ergebnis manuell überprüft und gegebenenfalls korrigiert wird. Im Rahmen einer Evaluation des Verfahrens wurde die Anzahl der nötigen Korrekturen als Maß für die Genauigkeit des Verfahrens herangezogen. In [MGMR02] wird außerdem angegeben, dass die Voraussetzung, dass die Graphen gerichtet und attributiert sind, wichtig ist, da sonst keine guten Ergebnisse erzielt werden.

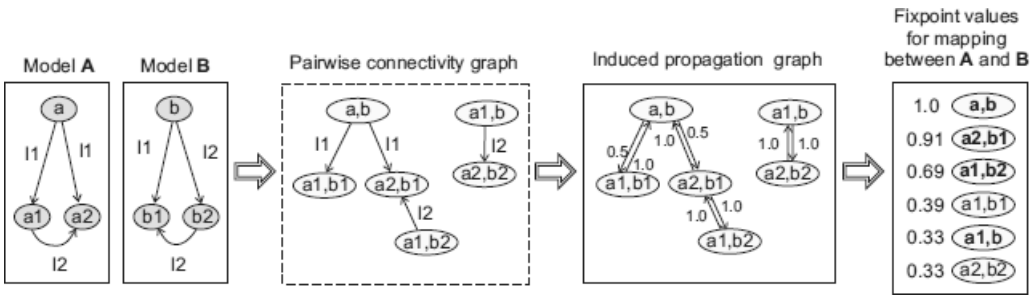

Abbildung 2.12: Ablauf des Similarity Flooding Algorithmus (entnommen aus [MGMR02])

Bewertung der nichtexakten Graph Matching-Verfahren Prinzipiell sind Graphen zur Abbildung von Klassendiagrammen wesentlich besser geeignet als Bäume, jedoch lassen sich bestehende Vergleichsverfahren für Graphen meines Erachtens nur zum Teil und auch nicht ohne Anpassung auf Klassendiagramme übertragen. Teilweise entspricht die verwendete Menge an Edieroperationen nicht den Anforderungen, wie sie für den Vergleich von Klassendiagrammen oder anderen Software-Modellen benötigt würde, so enthalten diese z. B. Spalten und Verschmelzen von Knoten. Je nach Modellierung eines Klassendiagramms als Graph ist weiter zu beachten, dass mögliche Mehrfachkanten, falls man Assoziationen durch Kanten darstellt, nicht in jedem Verfahren berücksichtigt werden können. Alternativ könnte man Mehrfachkanten umgehen, indem Assoziationen als Kante-Knoten-Kante-Konstruktion eingeführt werden. Es ist auf jeden Fall nötig, die Menge der Edieroperationen auf die verschiedenen Typen der Klassendiagrammelemente anzupassen und spezifische Eigenschaften auszunutzen, wie z. B. die Tatsache, dass Attribute und Operationen nur als Kindknoten von Klassenelementen existieren dürfen.

2.4 Andere Diagrammarten mit graphartigen Strukturen

Im Bereich der Softwarearchitektur werden Architekturmodelle bzw. Sichten auf Architekturmodelle verglichen. Dies stellt eine besondere Herausforderung dar, da sich diese Modelle stark in den Namen der Modellelemente unterscheiden können und die Modellelemente nicht immer getypt sein müssen. Nach [MKY06] werden in dem Bereich in der Praxis auch informale Diagramme verwendet, die lediglich aus Kästen und Linien bestehen. Auch wenn eine direkte Übertragung dieser Verfahren auf Klassendiagramme nicht möglich ist, da die Klassendiagramme in der Syntax stärker festgelegt sind, liefern diese beiden Verfahren interessante Ideen.

In [AAAN+06] wird ein Ansatz zum Vergleich von hierarchisch aufgebauten Architektursichten, Component&Connector-Views, beschrieben, der einen polynomiellen Tree-to-Tree-Correction Algorithmus [THRP05] für ungeordnete attributierte Bäume[10] nutzt, um die korrespondierenden Elemente zu identifizieren. Die Edieroperationen umfassen

[10]Durch die Verwendung einer abweichenden Metrik, die zusätzliche Annahmen trifft, wird die polynomielle Laufzeit erreicht, während das allgemeine Edierproblem in [ZSS92] NP-vollständig ist.

Umbenennungen, das Einfügen von Knoten und Löschen von Knoten, wobei bei den Lösch- und Einfügeoperationen die Kindknoten betroffener Knoten ebenso wie in [ZS89] verschoben werden. Für den Anwendungsbereich der Architektursichten sind diese hierarchischen Verschiebungen gut geeignet. Darüber hinaus wird das in [THRP05] vorgestellte Verfahren insoweit erweitert, dass nun auch Verschiebungen in einem beschränkten Umfang erkannt werden können. Das Verfahren beinhaltet einen Branch-and-Bound-Backtracking Algorithmus. Dabei können die Typen der Elemente, soweit vorhanden, berücksichtigt und Korrespondenzen nur zwischen Elementen kompatibler Typen zugelassen werden. Der Suchradius für Verschiebungen ist durch eine obere Schranke begrenzt, um die Komplexität zu beschränken. Ebenso wird empfohlen, bei größeren Bäumen die Zahl der rekursiven Aufrufe zu begrenzen. Mit einem Suchradius, der groß genug ist, und einer unbegrenzten Zahl möglicher Aufrufe bestimmt das Verfahren die exakte Lösung. Die Besonderheit dieses Ansatzes besteht in der Umsetzung der Abbildung einer Graphstruktur auf eine Baumstruktur. C&C Sichten sind nicht nur hierarchisch aufgebaut, sondern besitzen auch Querverbindungen, die Zyklen schließen. Die Sichten werden in Bäume transformiert, indem Knoten, zwischen denen nichthierarchische Querverbindungen bestehen, als Kopien mehrfach in den Baum eingefügt werden. Die kopierten Elemente verweisen dabei über Verbindungskanten zurück zu ihrem ersten Auftreten. Die eigentliche Korrespondenzberechnung findet in zwei Schritten statt: In einem ersten Schritt wird eine Zuordnung der streng hierarchischen Strukturen ohne Querverbindungen ermittelt. Dabei können die kopierten Knoten allerdings widersprüchlich zugeordnet werden. Daher wird der zweite Schritt benötigt, der die übrigen Kanten, die nicht Teil der hierarchischen Struktur sind, bearbeitet.

Ein weiteres Verfahren zum Vergleich von Modellen aus dem Bereich der Architekturmodelle ist der in [MKY06] vorgestellte Ansatz, der nicht mit einem Edierabstand, sondern mit Ähnlichkeitswahrscheinlichkeiten rechnet. Als Elemente eines Diagramms werden neben attributierten Knoten und optional attributierten Kanten, die zwei oder mehr Knoten miteinander verbinden, auch Container und Gruppen verarbeitet. Als Container wird ein Knoten bezeichnet, der wiederum weitere Knoten enthalten kann. Auch Gruppen fassen Knoten zusammen, jedoch kann ein Knoten mehreren Gruppen angehören, aber nur in einem Container liegen. Gruppen können ebenso wie Knoten über Beziehungen verbunden sein. Es werden verschiedene Eigenschaften (engl. features) festgelegt, die für die Korrespondenzbildung berücksichtigt werden sollen. Dies kann zum Beispiel die Beschriftung eines Knotens sein. Es wird eine Ähnlichkeitsfunktion festgelegt, die die Ähnlichkeit zweier Merkmale im Intervall $[0, 1]$ bewertet. Damit eine höhere Aussagekraft der Ähnlichkeitswerte erreicht wird, werden diese unter Vorgabe einer Wahrscheinlichkeitsverteilung mit Hilfe von Bayesscher Statistik (engl. Bayesian inference) in Wahrscheinlichkeiten umgewandelt. Ein Tripel aus Eigenschaft, Ähnlichkeitsmetrik und Wahrscheinlichkeitsverteilung bildet einen Evidencer. Die Aussagen verschiedener Evidencer werden zu einer Aussage kombiniert. Daraus lässt sich dann mit einem Greedy-Verfahren die Zuordnung der Knoten ableiten, die indirekt auch die Zuordnung der Kanten bestimmt. Im Gegensatz zu [AAAN$^+$06] werden hier mehrfache Korrespondenzen zugelassen, die dem Spalten oder Verschmelzen von Knoten entsprechen. Für gute Ergebnisse werden geeignete Ähnlichkeitsfunktionen und Wahrscheinlichkeitsver-

teilungen, die trainiert werden müssen, benötigt.

2.5 Zusammenfassung und Schlussfolgerungen

Es wurde anhand eines Beispiels gezeigt, dass die Anwendung von Textvergleichsverfahren auf serialisierte Klassendiagramme nicht sinnvoll ist. Einerseits werden Unterschiede berichtet, die zwar auf Textebene auftreten, aber für die Semantik von Klassendiagrammen nicht relevant sind, wie beispielsweise Leerzeilen oder Reihenfolgeänderungen ungeordneter Elemente. Andererseits sind die Informationen über korrespondierende Textblöcke für den Vergleich von Klassendiagrammen zu wenig feingranular, da sie insbesondere die klassendiagrammspezifische Syntax unberücksichtigt lassen.

Die vorgestellten edierkostenbasierten Baumvergleichsverfahren sind auf Klassendiagramme ebenfalls schlecht übertragbar, da ihre Edieroperationen ungeeignet sind. Für Klassendiagramme werden stattdessen syntaxgesteuerte Edieroperationen auf getypten Bäumen benötigt, die teilweise geordnet und teilweise ungeordnet sind. Ein grundsätzlicher Nachteil aller vorgestellten Baumvergleichsverfahren besteht darin, dass durch die Projektion der Klassendiagramme auf eine Baumstruktur die Querverbindungen nicht ausreichend berücksichtigt werden können. Dieses Problem zeigt sich auch beim Einsatz von XML-Vergleichsverfahren für die serialisierten Darstellungen von Klassendiagrammen, bei denen die Ziele von Assoziationen lediglich als Text verglichen werden.

Klassendiagramme lassen sich gut als getypte attributierte Graphen darstellen, ohne dass, wie bei der Übertragung auf Bäume, Informationen verloren gehen. Verfahren des exakten Graph Matchings identifizieren jedoch im besten Fall eine Menge identischer Teilgraphen. Weichen die zu vergleichenden Graphen von Klassendiagrammen an vielen Stellen voneinander ab, ist zu erwarten, dass das Ergebnis nicht mehr aussagekräftig ist, um die Unterschiede zwischen den beiden Klassendiagrammen abzuleiten. Verfahren des nichtexakten Graph Matchings können über einen fehlerkorrigierenden Graphisomorphismus auch abweichende Elemente als korrespondierend identifizieren. Dafür liegt der Aufwand für die Berechnung der optimalen Lösungen des nichtexakten Graph Matchings oft sogar über dem Aufwand eines vergleichbaren Verfahrens des exakten Graph Matchings [CFSV04]. Weiterhin erscheinen mir die gewählten elementaren Edieroperationen in der Hinsicht als ungeeignet, dass das Löschen eines Knotens auch das Löschen der Kanten beinhaltet. Wenn dem Löschen eines Knotens ein konstanter Kostenwert zugeordnet wird, unabhängig davon, wieviele Kanten ebenfalls gelöscht werden, widerspricht dies einer symmetrischen Edierdistanz, da der zugehörigen inversen Operation, die dem Einfügen des Knotens und dem Einfügen der Kanten entspricht, unter Umständen abweichende Kosten zugeordnet sein können.

Folglich werden für den Vergleich von Klassendiagrammen eigene, strukturbasierte Verfahren benötigt. Die Schwierigkeit bei der Konstruktion der Verfahren liegt darin, die kontextabhängigen Eigenschaften der Klassendiagramme zwar einerseits zu entkoppeln, um den Aufwand der Verfahren zu begrenzen, die Querverbindungen aber dennoch ausreichend zu berücksichtigen. Dies stellt im Grunde einen Balanceakt zwischen der Behandlung der Klassendiagramme als Baum und als Graph dar. Diese Hybridbehand-

lung einer graphartigen Diagrammstruktur wurde meines Erachtens auch im Verfahren [AAAN+06] im Bereich der Architektursichten umgesetzt, die eigenentwickelten Verfahren verwenden jedoch eine andere Vorgehensweise. Bevor die eigenentwickelten Verfahren in Kapitel 4 vorgestellt werden, wird im nächsten Kapitel zunächst der Stand der Technik bei Vergleichsverfahren von Klassendiagrammen behandelt.

3 Stand der Technik bei der Korrespondenzberechnung auf Klassendiagrammen

Im vorherigen Kapitel wurden Ansätze aus anderen Bereichen behandelt, die nicht für den Vergleich von Klassendiagrammen geeignet sind bzw. noch nicht für Klassendiagramme angepasst wurden. Dieses Kapitel befasst sich mit Verfahren, die zum Vergleich von Klassendiagrammen verwendet werden können, da sie entweder speziell für Klassendiagramme konzipiert wurden oder auf der Meta-Ebene ansetzen und beliebige Instanzen von Modellen vergleichen, die auf einem bestimmten Metamodell basieren.

Abbildung 3.1 zeigt eine Übersicht möglicher Kriterien zur Klassifikation von Differenzberechnungsverfahren auf Klassendiagrammen, dargestellt als Feature-Modell. Die mit einem Sternchen versehenen Optionen stellen Erweiterungspunkte dar, bei denen auf eine Verfeinerung der Kriterien verzichtet wurde, da diese für den betrachteten Bereich der Differenzberechnung von Klassendiagrammen relevant sind. Im Gegensatz zu der üblichen Einsatzweise von Feature-Modellen im Bereich der Konfiguration von Produktlinien sind die aufgelisteten Optionen nicht als binäre Ausprägungen, sondern als Einordnungshilfen bei fließenden Übergängen zu verstehen. Die Kriterien sind in die Bereiche Domaine, Zuordnung, Korrespondenzberechung, Differenzen und Visualisierung unterteilt.

Zum Vergleich von Klassendiagrammen sind, wie bereits erwähnt, sowohl domänenspezifische Verfahren, die für Klassendiagramme entwickelt wurden, als auch generische Verfahren einsetzbar. Die generischen Verfahren lassen sich weiterhin darin unterscheiden, ob sie über eine Konfiguration für spezielle Diagrammarten angepasst werden müssen oder ob sie beliebige Modelle eines Metamodells vergleichen. Ein weiteres Kriterium ist die Art der Zuordnung. Die grundlegende Entscheidung, ob das Verfahren eine eindeutige Zuordnung berechnet, in der jedes Element aus dem Modell A zu höchstens einem Element aus Modell B korrespondiert oder ob auch mehrfache Zuordnungen zugelassen werden, wirkt sich auf die Menge der nativ erkennbaren Edieroperationen aus. Nur im letzten Fall lassen sich Kopier- und Verschmelzungsoperationen bereits bei der Korrespondenzberechnung berücksichtigen. Bislang bekannte Verfahren, die zum Vergleich von Klassendiagrammen eingesetzt werden können, lassen sich in Hinblick auf die Korrespondenzberechnung in von der Ediergeschichte abhängige und zustandsbasierte Verfahren einteilen. Verfahren der ersten Kategorie vermeiden eine aufwändige Bestimmung der korrespondierenden Elemente, indem Unterschiede aus Edierprotokollen abgelesen werden oder die korrespondierenden Elemente über eindeutige Objektbezeichner festgelegt

werden. Die Verfahren der zweiten Kategorie vergleichen die Modelle unabhängig von der Ediergeschichte und führen dabei einen semantischen oder syntaktischen Vergleich durch. Die semantischen Ansätze, wie z. B. [MRR10], berücksichtigen beim Vergleich die durch die Klassendiagramme beschriebene Sprache; im Fall von Klassendiagrammen wird untersucht, welche Objektdiagramme sich aus dem Klassendiagramm ableiten lassen. Im Folgenden werden jedoch nur diejenigen Verfahren betrachtet, die ebenso wie unsere Verfahren einen syntaktischen Vergleich der Modelle vornehmen. Dabei existiert ein fließender Übergang von namensbasierten Verfahren, die die Namen von Objekten anstelle eindeutiger Objektbezeichner verwenden, über signaturbasierte Verfahren, die neben dem Namen weitere Eigenschaften, beispielsweise den Typ, miteinbeziehen, zu den strukturbasierten Verfahren. Die strukturbasierten Ansätze verwenden zwar ebenfalls den Namen von Elementen und deren Eigenschaften zur Bestimmung der korrespondierenden Elemente, basieren jedoch nicht ausschließlich darauf. Die namensbasierten und signaturbasierten Ansätze finden vor allem im Bereich der aspektorientierten Modellierung Anwendung. Die Kernfunktionalitäten und Zusatzfunktionalitäten eines Systems werden in zwei verschiedenen Modellen, einem *aspect model* und einem *core model* modelliert, die schließlich über *model composition* zu einem Modell verbunden werden [RFG+05]. Da diese Ansätze eher im Bereich des Verschmelzens von Modellen als im Bereich der Differenzberechnung einzuordnen sind, werden die namensbasierten und signaturbasierten Ansätze in dieser Arbeit nicht weiter betrachtet. Für den syntaktischen Vergleich von Klassendiagrammen sind exakte Verfahren, approximative Verfahren sowie heuristische Verfahren denkbar, wobei alle aus der Literatur bekannten Ansätze in den Bereich der Heuristiken fallen. Als Kriterium für die Korrespondenzberechnung werden stets Ähnlichkeitswerte berechnet, wohingegen das eigenentwickelte ECVerfahren zeigt, dass auch der Edierabstand zwischen zwei Klassendiagrammen zur Bestimmung der Korrespondenzen herangezogen werden kann. Die bekannten heuristischen Verfahren arbeiten ähnlichkeitsbasiert und unterscheiden sich neben den verwendeten Ähnlichkeitsfunktionen im Wesentlichen durch die verwendete Durchlauftechnik der baumartigen Datenstruktur und darin, ob ein globaler Vergleich aller Elemente durchgeführt wird oder ob nur die Elemente lokal begrenzter Bereiche verglichen werden. In Hinblick auf die dargestellten Differenzen ist es interessant, ob und in welchem Umfang ein Verfahren verschobene Elemente erkennen kann. Dabei kann unterschieden werden, ob mögliche Verschiebungen bereits bei der Korrespondenzberechnung berücksichtigt werden oder ob diese erst in einem Nachbearbeitungsschritt identifiziert werden. Um dem Benutzer die berechneten Differenzen zu veranschaulichen, ist sowohl eine textuelle Beschreibung als auch eine graphische Darstellung möglich. Gerne werden beide Darstellungsformen auch kombiniert. Die Änderungen können in einer Baumsicht graphisch dargestellt werden. Ausführlichere Visualisierungen stellen die beiden verglichenen Diagramme in zwei Fenstern gegenüber oder fassen die Änderungen in einem gemeinsamen Diagramm, dem sogenannten Vereinigungsdokument[1] zusammen.

Im Folgenden werden nun einzelne Verfahren beschrieben, wobei leider nicht zu jedem Verfahren ausreichende Informationen vorliegen, um dieses vollständig in das Klassifika-

[1] Eine Beschreibung eines Vereinigungsdokuments findet sich in [Kel09, S. 12].

tionsschema einzuordnen.

3.1 Von der Ediergeschichte abhängige Verfahren

3.1.1 Protokollbasierte Verfahren

Falls Protokolle der durchgeführten Änderungen vorliegen, ist es möglich, die Unterschiede direkt abzuleiten. Als Beispiel eines Werkzeugs, das die durchgeführten Änderungsoperationen protokolliert, wird hier das Concurrent Object Replication frAmework (CoObRA)[Sch03, SZN04] angeführt, das in Kassel und Siegen entwickelt wurde und in das CASE-Tool Fujaba integriert ist. Es handelt sich dabei um ein allgemeines Verfahren, das für beliebige Objektstrukturen, damit auch für Klassendiagramme, geeignet ist und jede Änderung an einem Objekt in Form von elementaren Änderungsoperationen sofort speichert. Die Listen von Änderungsoperationen werden für die Unterstützung der undo- und redo-Funktionalität und für den Persistenzmechanismus genutzt. Darüber hinaus wäre es denkbar, die Unterschiede zweier aufeinanderfolgenden Versionen aus der Edieroperationsliste abzuleiten, was eine explizite Differenzberechnung überflüssig macht. Diese Funktionalität war in einer früheren Version von CoObRA angedacht, meines Wissens jedoch nicht realisiert.

Anhand von protokollierten Änderungsoperationen können die Unterschiede zwischen zwei Versionen leicht ermittelt werden. Für den Vergleich zweier Modelle, die nicht im gleichen Werkzeug modelliert wurden oder deren Protokolle nicht verfügbar sind, ist ein derartiges Verfahren jedoch nicht einsetzbar. Darüber hinaus werden die Unterschiede immer in Abhängigkeit der Ediergeschichte bestimmt. Zusammenfassend erfüllen derartige Verfahren die Anforderungen (A4) und (A5) der Anforderungsliste aus Abschnitt 1.2.3 nicht.

3.1.2 Verfahren mit eindeutigen Objektbezeichnern

Die im Folgenden beschriebenen Ansätze zur Differenzberechnung auf Modellen setzen voraus, dass die Modellelemente eindeutige, unveränderliche Bezeichner besitzen, über die dann leicht eine Zuordnung korrespondierender Elemente erfolgen kann. Als eindeutige, unveränderliche Objektbezeichner werden eindeutige Kennungen bezeichnet, die einem Modellelement beim Erzeugen zugewiesen und nicht mehr verändert werden. Auch wenn ein Objekt gelöscht wird, bleibt dessen Objektbezeichner eindeutig, da Objektbezeichner gelöschter Elemente nicht wiederverwendet werden. Beim Kopieren von Objekten werden die eindeutigen Bezeichner übernommen. Mit Hilfe dieser Objektbezeichner können korrespondierende Modellelemente einfach identifiziert werden, wenn zwei korrespondierende Elemente genau dann als korrespondierend betrachtet werden, wenn die eindeutigen Objektbezeichner der beiden Modellelemente identisch sind. Neben den drei im Folgenden vorgestellten Verfahren verwenden auch kommerzielle Werkzeuge wie der IBM Rational Software Architect [Let05] oder Visual Paradigm[2] eindeutige

[2]http://www.visual-paradigm.com

Abbildung 3.1: Feature-Modell: Kriterien zur Einordnung von Differenzberechnungsverfahren auf Klassendiagrammen

Objektbezeichner zur Bestimmung der korrespondierenden Elemente.

Ansatz von Alanen und Porres Von Alanen und Porres wird in [AP03] ein Verfahren zum Vergleichen und Verschmelzen für MOF-basierte Modelle beschrieben, das sich auf beliebige Instanzen von MOF-Modellen anwenden lässt und so auch auf Klassendiagramme. Die Modelle werden dabei als Mengen von miteinander verlinkten Elementen aufgefasst. Als Ergebnis des Vergleichsverfahrens wird das gerichtete Delta der Änderungsoperationen bestimmt, wobei die zulässigen Änderungsoperationen die Basisoperationen Erzeugen eines Elements, Löschen eines Elements sowie das Ändern einer Eigenschaft (engl. Feature) sind. Alle Elemente des Modells besitzen einen unveränderlichen Typ. Die Differenzberechnung wird in Alanen und Porres [AP03] als 4-stufiger Prozess beschrieben. Die erste Stufe *map* entspricht der ersten Stufe der in Abschnitt 1.2 vorgestellten zwei Phasen der Korrespondenzberechnung, die über die Zuordnung der eindeutigen Objektbezeichner erfolgt. Die zweite Stufe aus Abschnitt 1.2, das Ableiten der Korrespondenzen, wird in diesem Ansatz [AP03] hingegen noch feiner in die drei Phasen *Create*, *Delete* und *Change* unterteilt. In den jeweiligen Phasen werden entsprechende Änderungsoperationen für das Delta erzeugt. Der Ansatz berücksichtigt dabei sogar geordnete mehrwertige Attributmengen. Darüber hinaus wird auch ein Verschmelzungsverfahren angegeben, das das ermittelte Delta verwendet.

Verfahren von Ohst Im Ansatz von Ohst [OWK03] wie auch im neueren Ansatz in [KWN05](vgl. Abschnitt 3.2.1) werden die zu vergleichenden Klassendiagramme als Graphen aufgefasst, die aus einem spannenden Baum aus Kompositionsbeziehungen bestehen, wobei die enthaltenen Elemente außerdem untereinander durch Beziehungen verknüpft sind. In einem Top-down-Durchlauf werden, beginnend bei der Wurzel, auf jeder Ebene die korrespondierenden Teilbäume über die eindeutigen Objektbezeichner identifiziert. Korrespondierende Elemente werden in Hinblick auf ihre Eigenschaften und Beziehungen verglichen. Die Differenzinformation wird dabei als symmetrische Differenz gespeichert. Eine Verschiebeoperation wird als zusammengesetzte Operation umgesetzt, die ein Objekt löscht und ein anderes Objekt mit dem gleichen Objektbezeichner einfügt. Teilbäume, zu denen auf gleicher Ebene kein Korrespondenzpartner gefunden wird, werden in einer Menge gespeichert, deren Elemente später erneut auf eindeutige Objektbezeichner verglichen werden. Auf diese Weise können auch verschobene Teilbäume erkannt werden. Alle übrig gebliebenen Elemente wurden gelöscht oder erzeugt. Die Differenzen werden in einem Vereinigungsdokument (engl. Unified Document) dargestellt. Dieses Dokument enthält sowohl die korrespondierenden Modellelemente der beiden verglichenen Modelle sowie die Einzelelemente, die je nach Zugehörigkeit in grün oder rot dargestellt werden. Beispiele für Vereinigungsdokumente finden sich in Abschnitt 3.2.1 (z. B. Abbildung 3.2), in dem ein ähnlichkeitsbasierter Differenzberechnungsalgorithmus beschrieben wird, der zur Visualisierung der Unterschiede ebenfalls ein Vereinigungsdokument verwendet.

Pounamu Mit dem Meta-CASE-Tool *Pounamu* [MGH05] lassen sich graphische Entwicklungswerkzeuge spezifizieren und generieren. Dabei unterstützt das Werkzeug auch die Versionierung, die Differenzberechnung und das Verschmelzen von Diagrammen verschiedener Diagrammtypen, die in dem Werkzeug definiert sind und allgemein aus Entitäten und Assoziationen bestehen. Diese Modellelemente werden graphisch als Figuren (engl. shapes) und Konnektoren dargestellt. Die Differenzberechnung wird nicht auf Modellebene, sondern zwischen deren graphischen Repräsentationen, die als Sichten bezeichnet werden, durchgeführt, so dass auch graphische Informationen wie, z. B. die Größe oder Position einer Figur, in die Differenzberechnung einbezogen werden. Der Ansatz zur Differenzberechnung grenzt sich von anderen Ansätzen außerdem dadurch ab, dass nicht die Graphstruktur der Modelle durchlaufen wird, sondern die korrespondierenden Elemente der Diagramme über die eindeutigen Objektbezeichner in zwei Stufen bestimmt werden. Zuerst werden die korrespondierenden Figuren ermittelt. Als Ergebnis entsteht ein gerichtetes Delta, das Änderungsoperationen für unterschiedliche Eigenschaften korrespondierender Figuren sowie Lösch- oder Einfügeoperationen für Einzelelemente der Figuren enthält. Für gelöschte Figuren werden auch Löschoperationen für alle mit der gelöschten Figur verbundenen Konnektoren erzeugt. Analog werden Einfügeoperationen für die Konnektoren eingefügter Figuren erstellt. Im zweiten Schritt werden diejenigen Konnektoren betrachtet, die im ersten Schritt noch nicht behandelt wurden, und entsprechende Operationen in das Delta eingefügt.

Bei diesem Ansatz werden die Konnektoren als Eigenschaft bzw. Bestandteil von Figuren betrachtet. Falls eine der an den Konnektor angrenzenden Figuren gelöscht wird, wird auch der Konnektor gelöscht. Die Edieroperationen sind auf elementare Operationen beschränkt, Verschiebeoperationen werden nicht erkannt.

3.1.3 Bewertung der Verwendung von eindeutigen Objektbezeichnern

Die Bestimmung korrespondierender Elemente mit Hilfe eindeutiger Objektbezeichner besitzt den Vorteil, dass sie wenig aufwändig ist. Darüber hinaus ist das Ergebnis eindeutig und exakt in dem Sinne, dass keine Interpretationsspielräume existieren, welche Zuordnung besser ist. Der schnellen und eindeutigen Identifikation der korrespondierenden Elemente steht jedoch die eingeschränkte Einsatzmöglichkeit der Verfahren gegenüber. Die Verfahren sind dann nicht einsetzbar, falls keine eindeutigen Bezeichner vorhanden sind. So können zwei Modelle, die unabhängig voneinander, gegebenenfalls in zwei verschiedenen Werkzeugen entwickelt wurden, nicht einfach miteinander verglichen werden. Auch wenn der Austausch von eindeutigen Objektbezeichnern über Werkzeuggrenzen hinweg realisiert werden kann, bleibt der Vergleich zweier Modelle, die im gleichen Werkzeug unabhängig voneinander entwickelt wurden, mit derartigen Verfahren so nicht möglich.

Ein weiterer, wesentlicher Nachteil besteht darin, dass Verfahren mit eindeutigen Objektbezeichnern die Differenzen zweier Modelle immer in Abhängigkeit der Entstehungsgeschichte dieser beiden Modelle interpretieren. Erfolgt die Identifikation korrespondie-

render Elemente ausschließlich über die eindeutigen Objektbezeichner, werden auch Unterschiede in inhaltlich identischen Modellen festgestellt, die lediglich durch unterschiedliche Bezeichner verursacht werden. So wird der Fall, dass ein Element verschoben wird, anders behandelt, als wenn es stattdessen gelöscht und an einer anderen Stelle wieder eingefügt wird (vgl. [Ohs04]). Dies ist darauf zurückzuführen, dass verschobene Objekte ihre eindeutigen Bezeichner beibehalten, beim Löschen und erneuten Einfügen eines Objekts mit identischen Eigenschaften sich das neue Element von dem alten Element jedoch im Bezeichner unterscheidet. Eine ähnliche Situation ergibt sich, wenn ein Objekt zunächst gelöscht und dann mit gleichen Eigenschaften an gleicher Position neu erzeugt wird. Verfahren, die Korrespondenzen über eindeutige Objektbezeichner identifizieren, sind im Vergleich zu Verfahren, die im Werkzeug protokollierte Änderungen verwenden, immerhin schon etwas weniger abhängig von der Entstehungsgeschichte, da im ersten Fall nur der aktuelle Zustand der Modelle berücksichtigt wird, während im Edierskript auch zueinander inverse Operationen enthalten sein können.

3.2 Heuristische Verfahren

Verfahren, die sich bei der Differenzberechnung nicht auf eindeutige unveränderliche Bezeichner beziehen, stellen der Differenzberechnung eine Phase voran, in der die Korrespondenzen mittels Heuristiken bestimmt werden.

3.2.1 Strukturbezogene heuristische Verfahren

Unter den strukturbasierten Verfahren werden im Folgenden das SiDiff- und das UMLDiff-Verfahren für den Vergleich von Klassendiagrammen vorgestellt.

SiDiff

Bei dem SiDiff-Verfahren [KWN05] handelt es sich um einen generischen Algorithmus, der sich über eine Konfigurationsdatei für verschiedene Diagrammarten konfigurieren lässt. Der Algorithmus überträgt die in XMI gespeicherten Klassendiagramme in ein proprietäres Datenmodell, das im Wesentlichen einem Baum aus Kompositionsbeziehungen entspricht, in dem die Elemente aber zusätzlich durch Querverweise verbunden sein können. Das Verfahren ist in zwei Phasen unterteilt: Die Matching-Phase beginnt Bottom-up (ähnlich wie in [CRGMW96]) in den Blättern des Baums und vergleicht die Elemente typweise. Dazu wird eine Ähnlichkeitsfunktion verwendet, die die Ähnlichkeit in Abhängigkeit des Elementtyps anhand mehrer Kriterien berechnet und zueinander gewichtet. Im Fall von Klassen werden beispielsweise die Ähnlichkeit des Klassennamens, der Attribute, Methoden, Vererbungsbeziehungen und des Pakets betrachtet. Da die Matching-Phase in den Blättern beginnt, stehen zum Zeitpunkt des Vergleichs von Klassen die Ähnlichkeitswerte der Attribute bereits zur Verfügung. Können Elemente eindeutig zugeordnet werden, werden die Ähnlichkeitswerte noch nicht zugeordneter Söhne auf dieser Basis neu berechnet, wodurch weitere Zuordnungen erfolgen können.

Dies stellt die Top-down-Phase dar, in der die ähnlichsten Elemente zugeordnet werden. Der Ähnlichkeitswert muss dabei jedoch einen Schwellenwert erreichen. Der Name von Elementen spielt eine besondere Rolle im Rahmen der Ähnlichkeitsfunktion. Besitzen zwei Klassen einen identischen Klassennamen, erreicht der Ähnlichkeitswert bereits den geforderten Schwellenwert für die Mindestähnlichkeit eines Korrespondenzpaares (vgl. [KWN05]). Die Berechnung der Zuordnung ist dann beendet, wenn alle Elemente im Bottom-up-Durchlauf der ersten Phase betrachtet und die gefundenen Korrespondenzen Top-down propagiert worden sind. Anschließend kann aus den gefundenen Korrespondenzen eine Differenz abgeleitet werden, wobei auch Verschiebungen erkannt werden. Als Komplexität des Verfahrens wird die Ordnung $O(n^2)$ angegeben, wobei die Größe n der Gesamtzahl der Elemente der zu vergleichenden Modelle entspricht. Da die Datentypen auf der Blattebene von Klassendiagrammen selten eindeutig korrespondieren, wird der Bottom-up-Phase noch eine Hash-Phase vorangestellt, die die Elemente inklusive ihrer Pfadnamen vergleicht. Der Ablauf dieser Hash-Phase ähnelt der des Verfahrens X-Diff [WDC03] für XML Dokumente.

Eine ältere Version des SiDiff-Verfahrens ist als frei verfügbares Plugin[3] für die ältere Fujaba Version 4.3.2 erhältlich, so dass das Verfahren an einigen Beispielen getestet werden konnte. Die in Fujaba modellierten Klassendiagramme werden dabei zunächst nach XMI exportiert. Die beiden XMI-Dateien dienen als Input für das Differenzberechnungsverfahren, das wiederum ein Vereinigungsdokument im XMI-Format erstellt. Dieses Vereinigungsdokument lässt sich mit Hilfe eines Integrator-Plugins in Fujaba visualisieren. Die Abbildung 3.2 zeigt das Ergebnis des Vergleichs der beiden Modelle aus den Abbildungen 1.2 und 1.3, die analog in Fujaba modelliert wurden. Das SiDiff-Verfahren identifiziert hier die beiden gleichnamigen Klassen `USAddress` miteinander.[4] Ein Tippfehler im Bezeichner der Klasse `USAdress` statt `USAddress` führt zu einer anderen Zuordnung der Elemente, wie sie in Abbildung 3.3 zu sehen ist. Die Sensitivität des Verfahrens gegenüber Namensänderungen demonstriert auch das nächste Beispiel. Die Abbildungen 3.4 und 3.5 enthalten zwei Klassendiagramme, die bis auf die Übersetzung der Namen vom Englischen ins Französische identisch sind. Wie die Abbildung 3.6 zeigt, reichen die strukturellen Gemeinsamkeiten dem SiDiff-Verfahren nicht aus, um die umbenannten Klassen zuzuordnen. Die nicht umbenannte Klasse `EDate` wird als einzige Korrespondenz identifiziert. Inzwischen gibt es bereits eine neuere Version des SiDiff-Verfahrens, die jedoch nicht frei verfügbar ist. Durch den Einsatz von mehrdimensionalen Suchbäumen [Tre07] bzw. Finger-Print-Verfahren [Pie08] wurden die Laufzeiten sowie die Ergebnisse verbessert.

[3] http://www.fujaba.de/projects/differences-with-sidiff.html

[4] Warum im Differenzbericht in Abbildung 3.2 die Kompositionsbeziehungen `shipTo` und `billTo` nicht als geändert gekennzeichnet sind, obwohl sich durch die Zuordnung der Klassen `USAddress` für die Assoziationen eine Änderung des Zieltyps ergibt, ist unklar. Laut Beschreibung in [KWN05] müsste der Unterschied als *reference difference* gekennzeichnet werden: „Corresponding elements, whose references are different in the two original documents have a reference difference with references to both targets“.

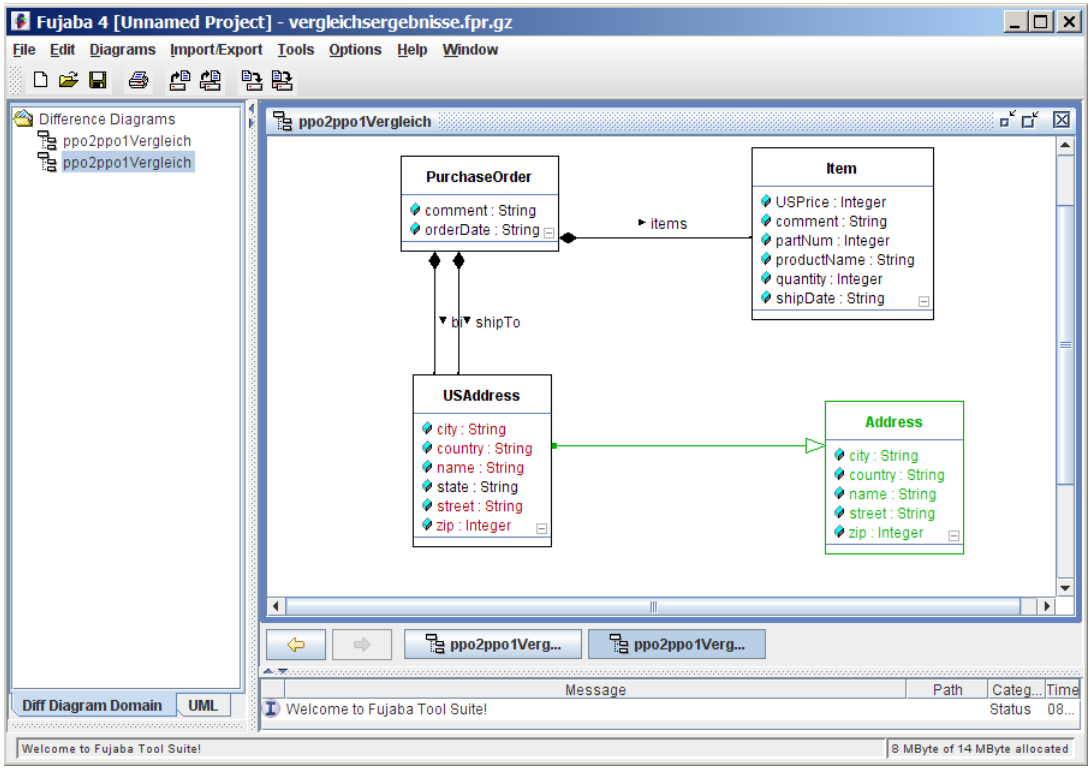

Abbildung 3.2: SiDiff-Plugin für Fujaba: Vergleich der Modelle A und B

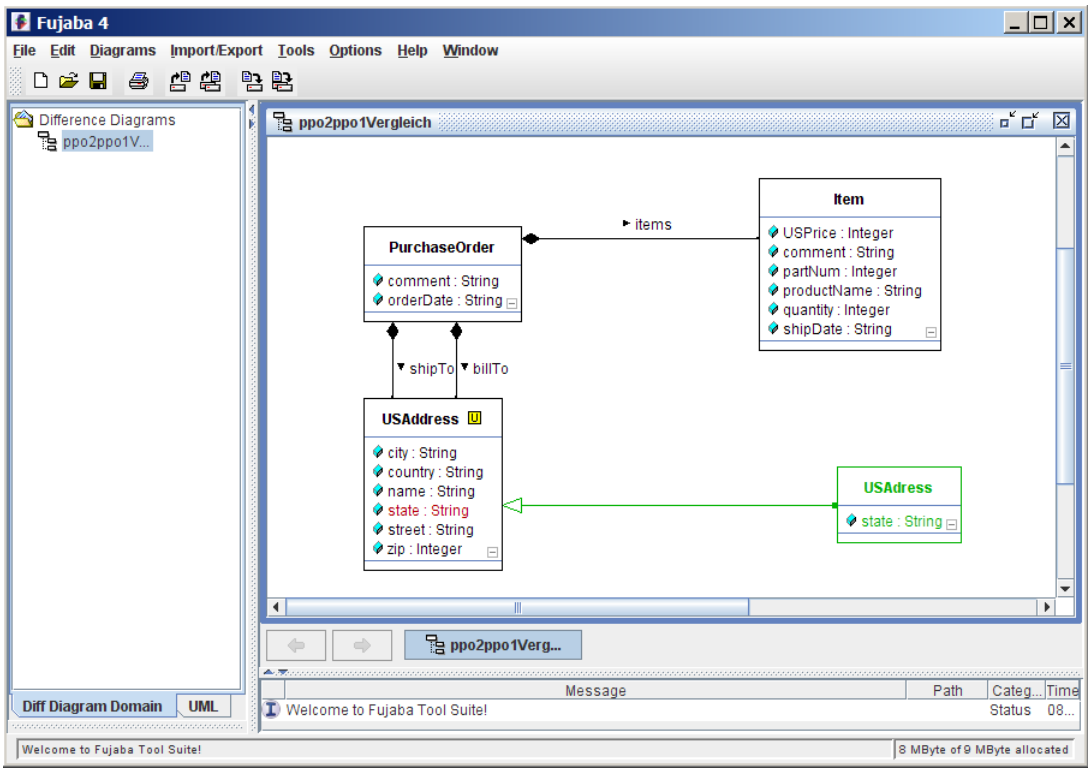

Abbildung 3.3: SiDiff-Plugin für Fujaba: Vergleich der Modelle A und B2 (Tippfehler `USAdress` statt `USAddress`)

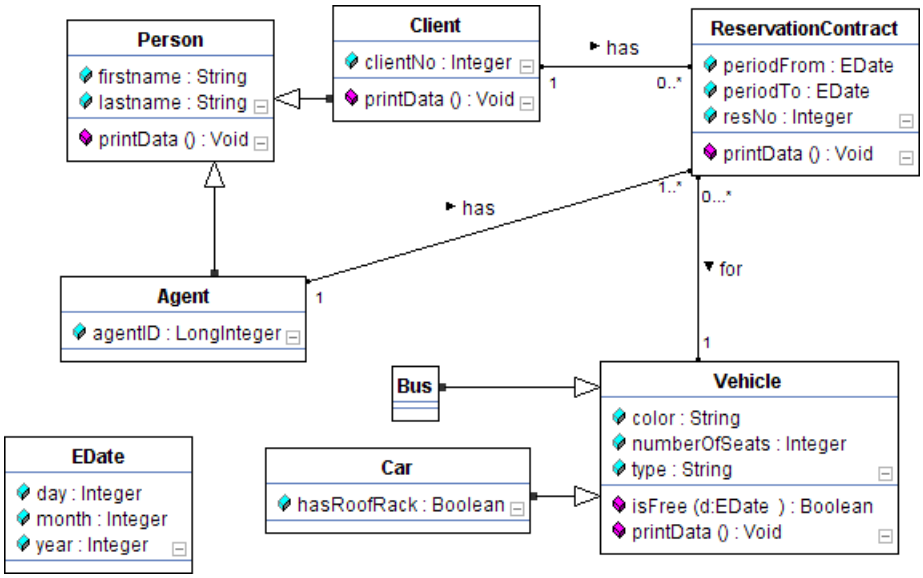

Abbildung 3.4: Modell E (modelliert in Fujaba)

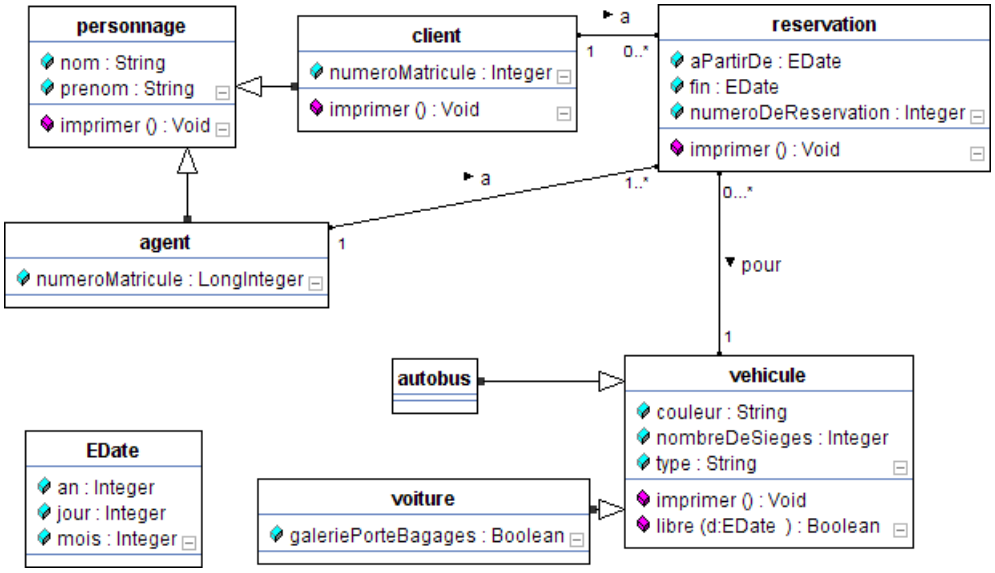

Abbildung 3.5: Modell F (modelliert in Fujaba)

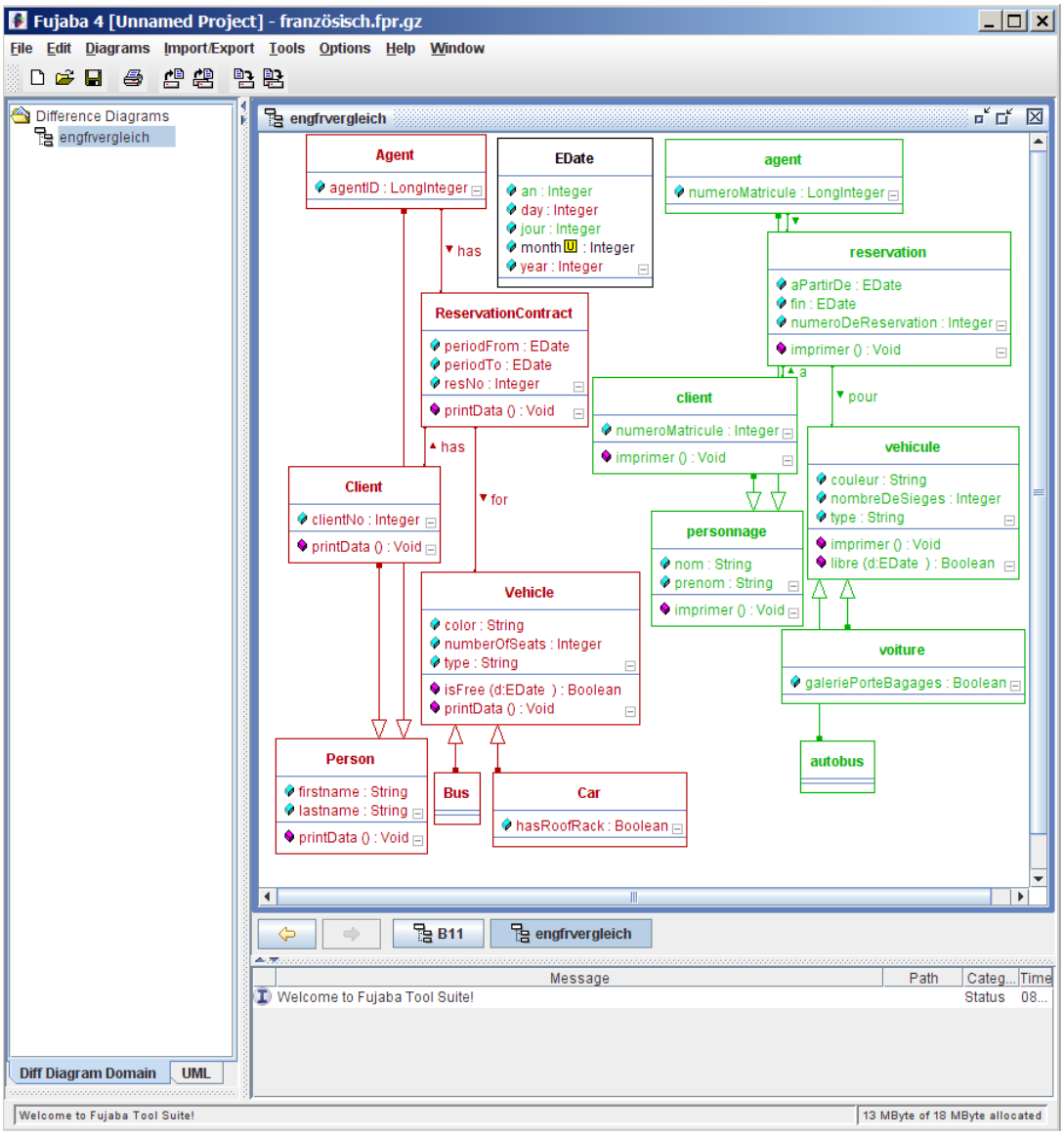

Abbildung 3.6: SiDiff-Plugin für Fujaba: Vergleich der Modelle E und F

UMLDiff

Das UMLDiff-Verfahren [XS05, XS07] wurde speziell für den Vergleich von UML-Klassendiagrammen, die durch Reverse-Engineering aus Java-Quellcode gewonnen werden, konzipiert und ist als Eclipse-Plugin im JDEvAn-Tool (Java Design Evolution and Analysis)[5] implementiert. Da auch der Java-Quellcode der zu vergleichenden Klassendiagramme vorhanden ist, können bei der Bestimmung der Korrespondenzen viel mehr Informationen genutzt werden, als es beim Vergleich modellierter Klassendiagramme möglich ist. So können auch die Implementierungen der Methoden, Zugriffe auf gleiche Variablen usw. berücksichtigt werden. Das Verfahren ist zwar für Java implementiert, lässt sich aber auch auf eine andere objektorientierte Programmiersprache anpassen.

Die Klassendiagramme werden in ein UML 1.0 konformes Metamodell übertragen und als gerichtete Graphen mit einem spannenden Baum aus Kompositionsbeziehungen betrachtet. Die Elemente der Klassendiagramme, die den gleichen Typ besitzen, werden in zwei verschiedenen Stufen des Verfahrens über zwei verschiedene Kriterien als korrespondierend identifiziert:

1. **Namensähnlichkeit:**
   Falls zwei Elemente den gleichen Namen besitzen oder ihre Namen gemäß einer Namensähnlichkeitsheuristik ähnlich sind, gelten diese Elemente als sicher korrespondierend.

2. **Strukturelle Ähnlichkeit:**
   Falls zwei Elemente ähnliche Beziehungen zu bereits korrespondierenden Elementen besitzen, können diese als korrespondierend betrachtet werden.

Nachdem in Stufe 1 die ersten korrespondierenden Elemente anhand des Namens identifiziert sind, werden diese Korrespondenzen als Orientierungspunkte (engl. landmarks) verwendet, um umbenannte und verschobene Elemente zu erkennen. Schwellenwerte für die Ähnlichkeitswerte legen dabei fest, wann zwei Elemente noch als verschoben oder umbenannt betrachtet werden. Neu gefundene Korrespondenzen in Stufe 2 werden ebenfalls als Orientierungspunkte verwendet, mit deren Hilfe gegebenenfalls weitere Korrespondenzen identifiziert werden können.

Die Suche nach korrespondierenden Elementen beginnt bei den Wurzelelementen der beiden Diagramme. Die Durchlauftechnik des UMLDiff-Verfahrens wird in Abbildung 3.7 der Durchlauftechnik des SiDiff-Verfahrens gegenübergestellt. Das UMLDiff-Verfahren vergleicht in einem Top-down-Durchlauf lediglich diejenigen Elemente miteinander, die sich in Teilbäumen befinden, deren Wurzeln bereits als korrespondierend erkannt wurden. Dadurch bleiben weitere Elemente in diesen Teilbäumen unberücksichtigt. Das Verfahren verlässt sich dabei darauf, dass eine ausreichende Menge von Orientierungspunkten existiert, deren Namen nicht verändert wurden. In der Veröffentlichung von UMLDiff in [XS05] wird angegeben:

[5] http://webdocs.cs.ualberta.ca/∼ stroulia/Zhenchang_Xing_Old_Home/jdevan.html

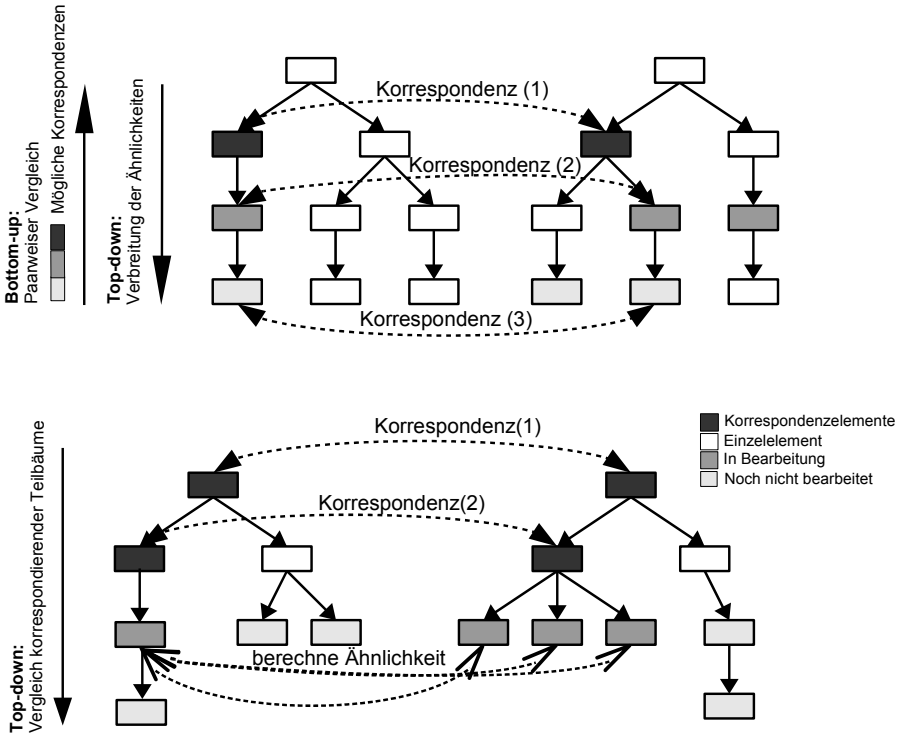

Abbildung 3.7: Verschiedene Durchlauftechniken des SiDiff-Verfahrens (oben) und des UMLDiff-Verfahrens (unten)

> „[...] if a substantial number or a complex set of changes are made to a particular version before it is stored back in the version management system, the accuracy of UMLDiff will most likely suffer[.]“

3.2.2 Metamodellunabhängige Verfahren

Mit dem SiDiff-Verfahren wurde ein Vertreter der generischen Verfahren vorgestellt, die für eine Diagrammart konfigurierbar sind. Bei UMLDiff handelt es sich um einen spezifischen Ansatz zum Vergleich von UML-Klassendiagrammen. Darüber hinaus existieren weitere Ansätze, die nicht auf eine Diagrammart spezialisiert sind, sondern als metamodellunabhängige Verfahren für alle Modelle eines Metamodells geeignet sind, ohne dass eine spezifische Anpassung nötig ist. Zu diesen Ansätzen gehören DSMDiff und EMF Compare, die beide auf Grund der Traversierungsstrategie als eine Art Verallgemeinerung des UMLDiff-Verfahrens einzuordnen sind, sowie das Vergleichsverfahren der Entwicklungsumgebung Maudeling.

DSMDiff

Die Generic Modeling Environment (GME) [LMB+01] erlaubt die Definition domänenspezifischer Sprachen über Metamodelle. Das darin integrierte Differenzberechnungsverfahren DSMDiff [LGJ07] vergleicht domänenspezifische Modelle dieser Umgebung. Die

Knoten werden Top-down, beginnend bei den Wurzelknoten der Modelle, ebenenweise anhand der Kriterien Signaturähnlichkeit und Strukturähnlichkeit verglichen. Bei der Signaturähnlichkeit handelt es sich um eine erweiterte Form der Namensähnlichkeit, bei der zum Beispiel auch der Typ einbezogen wird. Können Knoten nicht aufgrund ihrer Signaturähnlichkeit zugeordnet werden, wird die Strukturähnlichkeit berechnet, die auf den Signaturähnlichkeiten der Verbindungen basiert. Nach der Zuordnung der Knoten ist auch die Zuordnung der Kanten möglich.

EMF Compare

Das Eclipse Modeling Framework stellt auch ein metamodellunabhängiges Verfahren zur Differenzberechnung zur Verfügung [MSW09a, BP08]. Die Bestimmung der korrespondierenden Elemente und das anschließende Ableiten der Differenzen werden von zwei getrennten Modulen, der `MatchEngine` und der `DiffEngine`, durchgeführt. Der Mechanismus zur Ableitung der Differenzen wird als Teil des Rahmenwerks zum Vergleich von Korrespondenzberechnungsverfahren in Kapitel 5 beschrieben. An dieser Stelle soll lediglich das EMF Compare Korrespondenzberechnungsverfahren genauer betrachtet werden.

Um korrespondierende Elemente zu identifizieren, findet ein Baumvergleich statt, bei dem beide zu vergleichende Modelle, beginnend bei den Wurzelelementen, in einer Top-down-Strategie durchlaufen werden. Dabei wird für ein Modellelement aus dem linken Element jeweils das ähnlichste Modellelement aus denjenigen Elementen des rechten Modells bestimmt, die auf gleicher Hierarchieebene liegen. Die Menge der dabei betrachteten Kandidaten wird durch einen Parameter `OPTION_SEARCH_WINDOW` begrenzt, der standardmäßig 100 Elemente in den Vergleich einbezieht.

Um die Ähnlichkeit zweier Elemente zu bewerten, werden bis zu vier Kriterien eingesetzt:

- Ähnlichkeit des Namensattributs
- Ähnlichkeiten der Attribute
- Ähnlichkeiten der Referenzen
- Ähnlichkeit der Elementtypen (bei unterschiedlichen Metamodellen)

Die Ähnlichkeitswerte zweier Elemente in Bezug auf die Attribute und Referenzen werden berechnet, indem je eine Darstellung des Inhalts als Zeichenkette ermittelt wird, die dann über eine Stringvergleichsmethode verglichen werden. Für die Auswertung dieser Kriterien gilt die folgende Regel: Sobald die Namen zweier Elemente identisch sind, gelten diese beiden Elemente als ähnlich. Unterscheiden sich die Namen, gelten die Elemente nur dann als ähnlich, wenn sowohl die Namensähnlichkeit als auch die Ähnlichkeiten der Attribute und Referenzen jeweils einen bestimmten Schwellenwert überschreiten. Werden zwei Elemente als korrespondierend identifiziert, werden die jeweiligen Unterelemente des Paares betrachtet und gegebenenfalls ebenfalls zugeordnet.

Da EMF Compare frei verfügbar ist, wurde das Korrespondenzberechnungsverfahren zum Teil in die Evaluation in Kapitel 6 einbezogen. Das Korrespondenzberechnungsverfahren von EMF Compare berücksichtigt beim Vergleich zwar die Struktur der Modelle, stützt sich dabei jedoch auf die Namen der Unterelemente. Wie das Vergleichsergebnis des Fremdsprachenbeispiels, das auf Seite 69 auch schon mit dem SiDiff-Verfahren verglichen wurde, in Abbildung 3.8 zeigt, werden komplett umbenannte Elemente jedoch schlecht erkannt. Von den 8 Klassen werden durch EMF Compare nur diejenigen 4 Klassen richtig zugeordnet, deren Klassennamen und Namen der Unterelemente trotz Umbenennung hinreichend ähnlich sind (`EDate`, `EDate`), (`Client`, `client`), (`Agent`, `agent`), (`Bus`, `autobus`). Außerdem wird der Teilbaum unter einem nicht zugeordneten Element nicht weiter betrachtet. Die Nachteile, die dadurch entstehen, dass nicht alle Elemente für eine potentielle Korrespondenz berücksichtigt werden, werden in den Evaluationsergebnissen in Kapitel 6 deutlich.

Maudeling

In die Eclipse-basierte Entwicklungsumgebung Maudeling[6] ist ein metamodellunabhängiges Korrespondenzberechnungsverfahren [RV08] integriert, das in Maude[7] implementiert ist. Die Umgebung bietet die Identifikation korrespondierender Elemente wahlweise über eindeutige Objektbezeichner oder über einen strukturellen, ähnlichkeitsbasierten Vergleich an. In den strukturellen, nichtsignaturbasierten Vergleich zweier Elemente werden deren Metaklassen, Namen und strukturelle Eigenschaften miteinbezogen. Für den Ähnlichkeitswert der Metaklassen werden drei Fälle unterschieden: identische Metaklassen, es existiert eine Vererbungsbeziehung zwischen den beiden Metaklassen oder die beiden Metaklassen gelten als verschieden. Für die strukturellen Eigenschaften werden in Abhängigkeit des Datentyps verschiedene Ähnlichkeitsberechnungen verwendet. Handelt es sich bei der strukturellen Eigenschaft um eine Referenz, werden die Ziele der Referenz verglichen, wobei auf den Vergleich weiterer Referenzen der Referenzziele verzichtet wird. Bei nichtidentischen Metaklassen wird in dem Ähnlichkeitswert ein Abzug für strukturelle Eigenschaften eingerechnet, die nur in einer der beiden Metaklassen existieren. Die berechneten einzelnen Ähnlichkeitswerte der Metaklasse, des Namens und der strukturellen Eigenschaften werden schließlich gewichtet und zu einem Ähnlichkeitswert aggregiert. Ein Schwellenwert verhindert, dass Elemente mit verschiedenen Metaklassen als korrespondierend identifiziert werden. Aus den Ähnlichkeitswerten wird schließlich eine eindeutige Zuordnung bestimmt, indem die Summe der Ähnlichkeitswerte maximiert wird. Die genaue Vorgehensweise wird in [RV08] nicht beschrieben:

> „At the end of the process, a sieve is applied to all potential matches in order to pair only those objects that together obtain the biggest rate, taking into account that a specific object can only belong to one match relationship.“

Im Gegensatz zu DSMDiff und EMF Compare werden hier alle Objekte miteinander verglichen. Durch den vollständigen Vergleich steigt zwar der Aufwand, jedoch werden dafür

[6] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/Maudeling
[7] http://maude.cs.uiuc.edu/

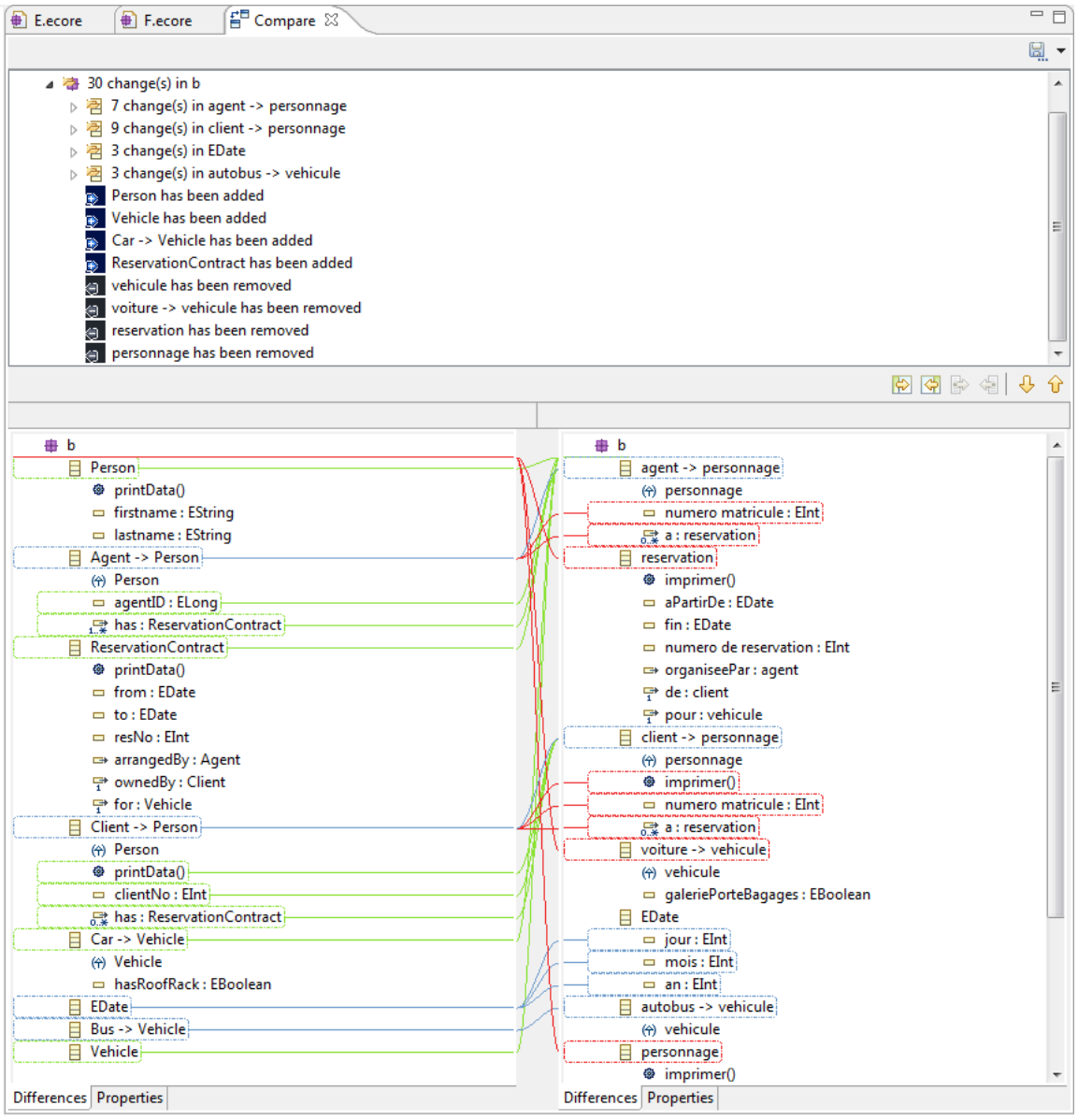

Abbildung 3.8: Differenzbericht des Vergleichs der Modelle A und B aus Beispiel 21 mit EMF Compare

auch Elemente als korrespondierend erkannt, die über mehrere Ebenen der Kompositionshierarchie verschoben wurden. Auch Kindelemente nichtkorrespondierender Elemente werden auf diese Weise berücksichtigt.

3.2.3 Bewertung der heuristischen Verfahren

Während, wie in Kapitel 2 beschrieben, in anderen Bereichen edierkostenbasierte Verfahren existieren, die eine optimale Zuordnung der Elemente unter Vorgabe eines Edierkostenmodells berechnen, stützen sich die Verfahren für den Vergleich von Klassendiagrammen, die nicht auf eindeutigen Objektbezeichnern basieren, auf Ähnlichkeitsheuristiken. Während in den meisten aufgeführten Verfahren nur lokale Ähnlichkeitsentscheidungen getroffen werden, berechnen lediglich die Ansätze [RV08, KWN05] die Ähnlichkeiten auf globaler Ebene. Die heuristischen Verfahren, zu denen Beispiele gezeigt werden konnten, reagieren außerdem sehr empfindlich auf Umbenennungen der Modellelemente.

Da die heuristischen Verfahren nicht auf einem mathematisch definierten Optimierungskriterium basieren, kann über die Optimalität der berechneten Lösung keine Aussage getroffen werden. Dies erschwert die Bewertung der Verfahren erheblich, zumal keine allgemein akzeptierten Benchmarks zur Verfügung stehen. Dem bisherigen Stand der Forschung entsprechend gibt es keine Untersuchungen, ob eine Korrespondenzberechnung auf der Basis von Edierkosten nicht bessere Ergebnisse liefert. In [Kel09, S. 88f.] wird als ein Qualitätskriterium für Differenzen zwar „eine möglichst kleine Zahl von Edieroperationen“ und „möglichst kleine Parameter“ angegeben, „das einzig wirkliche Maß für die Qualität einer Differenz“ sei jedoch, „wie gut sie einem Betrachter die Unterschiede zwischen zwei Dokumenten vermittelt und wie sehr sie damit zusammenhängende Arbeitsschritte erleichtert“.

3.3 Zusammenfassung

Die in diesem Kapitel vorgestellten Verfahren zum Vergleich von Klassendiagrammen basieren entweder auf eindeutigen Objektbezeichnern oder Ähnlichkeitsheuristiken. Die Nachteile dieser Ansätze, die bereits in den Abschnitten 3.1.3 bzw. 3.2.3 beschrieben wurden, werden nun den in Abschnitt 1.2.3 angegebenen Anforderungen für Differenzberechnungsverfahren auf Modellen gegenübergestellt.

Verfahren, die auf eindeutigen Objektbezeichnern basieren, sind zwar sehr effizient (A7), die Umsetzung der Werkzeugunabhängigkeit (A4) ist jedoch schwer umsetzbar und die Unabhängigkeit von der Ediergeschichte (A5) ist keinesfalls realisierbar. Die Verfahren, die Ähnlichkeitsheuristiken verwenden, können die Unabhängigkeit von Werkzeugen (A4) und der Ediergeschichte (A4) hingegen leicht gewährleisten. Kritisch ist für diese Verfahren die Anforderung (A1b). In Hinblick auf die Güte der berechneten Korrespondenzen werden teilweise Abstriche zu Gunsten der Effizienz (A7) gemacht, indem nicht alle möglichen Korrespondenzbildungen berücksichtigt werden [XS05, BP08, LGJ07]. Die getesteten Ansätze weisen darüber hinaus eine zu hohe Sensibilität gegenüber Namensänderungen auf [BP08, KWN05].

Den Klassendiagrammvergleichsverfahren, die auf Ähnlichkeitsheuristiken basieren, fehlt ein vorgegebenes Kriterium für die Güte der berechneten Zuordnung. Es wäre sinnvoll, die Qualität von Zuordnungen zu objektivieren, so dass diese Kriterien bereits bei der Berechnung der Zuordnung berücksichtigt werden können [För07]. Um zu überprüfen, ob die Edierdistanz zwischen zwei Modellen als solches Kriterium herangezogen werden kann, wurde in dieser Arbeit ein edierkostenbasiertes Verfahren entwickelt, das einen Optimierungsalgorithmus aus der Graphentheorie verwendet, und einem ebenfalls entwickelten ähnlichkeitsbasierten Verfahren gegenübergestellt, das auf einem heuristischen Greedy-Ansatz beruht.

Weitere Zielvorgaben, die bei den beiden entwickelten Verfahren umgesetzt wurden, sind, dass sie strukturbasiert und unabhängig von eindeutigen Objektbezeichnern arbeiten, die Korrespondenzen auf globaler Ebene bestimmen und relativ unempfindlich gegenüber Namensänderungen sein sollen. Beiden Verfahren liegt ein gemeinsames Datenmodell zugrunde, das wie in [OWK03, KWN05] einer erweiterten Baumstruktur mit Querverbindungen entspricht. Das Modell wird in Hinblick auf die Differenzen ebenso wie in [AP03] als eine Menge von Elementen betrachtet. Darüber hinaus wird, wie bei den Konnektoren in [MGH05], eine homomorphe Abbildung der Assoziationen unterstellt, so dass Assoziationen nicht verschoben werden können. Die beiden entwickelten Verfahren werden im folgenden Kapitel ausführlich beschrieben.

4 Eigene Verfahren zur Korrespondenzberechnung

In diesem Kapitel werden zwei eigenentwickelte Verfahren zur Berechnung einer Zuordnung sowie Varianten dieser Verfahren beschrieben. Die Verfahren verlassen sich bei der Identifikation der Korrespondenzen weder auf eindeutige Objektbezeichner noch auf die Namensgleichheit der Elemente, sondern berücksichtigen die Struktur der Modelle. Dabei verfolgen die Verfahren unterschiedliche Strategien: Die Bestimmung korrespondierender Elemente erfolgt bei dem in Abschnitt 4.2 dargestellten Verfahren über die Unterschiede der Elemente, die als Edierkosten formalisiert werden. Die Korrespondenzberechnung des Verfahrens in Abschnitt 4.3 basiert im Gegensatz dazu auf den Ähnlichkeiten der Elemente, die über Ähnlichkeitsfunktionen ermittelt werden. Beide Verfahren wurden für den Vergleich von Klassendiagrammen konzipiert, die Ansätze sind jedoch auch auf andere Modelle übertragbar (vgl. Abschnitt 7.1). Die Grundformen dieser Verfahren arbeiten auf Klassenebene und berücksichtigen zunächst keine Verschiebungen von Elementen. In Abschnitt 4.4 wird ein Nachverarbeitungsschritt beschrieben, der es im Nachhinein ermöglicht, auf Klassenebene verschobene Elemente zu erkennen. Um auch Klassendiagramme vergleichen zu können, deren Klassen in verschiedene Pakete gegliedert sind, wird in Abschnitt 4.5 ein zweistufiger Ansatz eingeführt, bei dem die Zuordnung der Pakete unter Berücksichtigung der Klassenzuordnung erfolgt. Dazu wird das ähnlichkeitsbasierte Verfahren auf die Paketebene übertragen.

4.1 Grundlegende Entscheidungen für beide Verfahren: Das Datenmodell

Um die Vergleichbarkeit der Ergebnisse zu gewährleisten, wurde für beide Verfahren das gleiche konzeptionelle Datenmodell für Klassendiagramme zugrunde gelegt, das Abbildung 4.1 zeigt. In Kapitel 5 wird das implementierte Datenmodell genauer behandelt, an dieser Stelle wird lediglich der grundlegende Aufbau eines Klassendiagramms für das Verständnis der im Folgenden beschriebenen Verfahren benötigt. Die Kompositionsbeziehungen der Klassen zu ihren Unterelementen, den Attributen und Methoden bilden zusammen mit der Modellwurzel einen Baum. Dieser Baum wird durch Querverbindungen zwischen Klassen, die gerichtete Assoziationen und Vererbungsbeziehungen darstellen, zu einem Graphen ergänzt (siehe Abbildung 4.2). Im Wesentlichen stellt das konzeptionelle Datenmodell eine vereinfachte Version des Ecore-Metamodells [Ecl09] dar. Die sechs nachfolgenden Einschränkungen wurden während der Konstruktion des

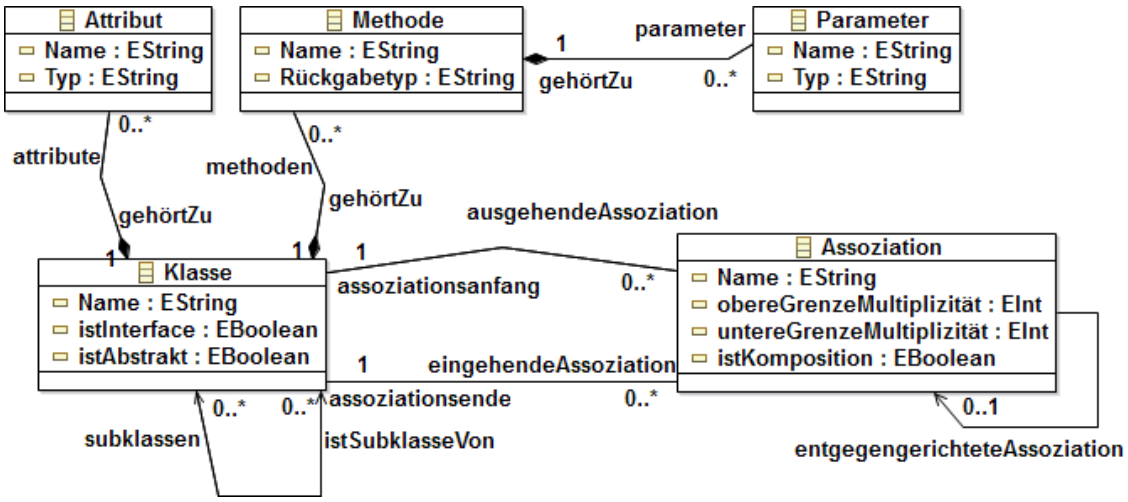

Abbildung 4.1: Das konzeptionelle Datenmodell

kostenbasierten Verfahrens entwickelt, um die Komplexität des Problems auf ein vertretbares Maß zu reduzieren. Um die Vergleichbarkeit zu gewährleisten, wurden diese Einschränkungen auch für das ähnlichkeitsbasierte Verfahren übernommen.

Einschränkungen

1. **Eindeutige Zuordnung**
   Jedes Element korrespondiert zu höchstens einem Element.

2. **Typkonforme Korrespondenzen**
   Elemente können nur dann korrespondieren, falls der Typ übereinstimmt. Die korrespondierenden Elemente dürfen sich jedoch hinsichtlich ihrer Eigenschaften unterscheiden.

3. **Vollständige Teilbaumzuordnung für Klassen**
   Falls die Klassen a und b korrespondieren, können die Unterelemente von a nur mit den Unterelementen von b korrespondieren. Für jeden Typ der Unterelemente wird jeweils eine eigene Zuordnung bestimmt.

4. **Bedingte Zuordnung**
   Assoziationen und Vererbungsbeziehungen können nur dann korrespondieren, falls die Klassen der Assoziationsenden bzw. Vererbungskantenenden korrespondieren. Daraus ergibt sich, dass Korrespondenzen für Assoziationen und Vererbungskanten erst nach Zuordnung der Klassen ermittelt werden können.

5. **Behandlung der Datentypen als Zeichenketten**
   Die Typen von Attributen, Parametern und Rückgabetypen von Methoden werden nicht als Referenz auf einen anderen Datentyp, sondern als Zeichenkette behandelt.

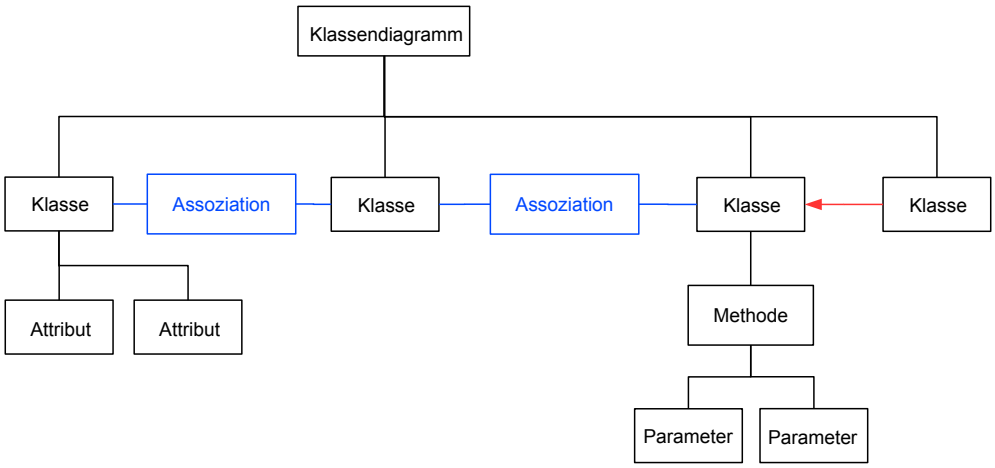

Abbildung 4.2: Die Kompositionsstruktur der Klassen und ihrer Unterelemente bilden zusammen mit der Diagrammwurzel einen Baum, der durch Assoziationen (blau) und Vererbungsbeziehungen (rot) zu einem Graphen erweitert wird.

6. **Zuordnung der Übergabeparameter entsprechend ihrer Reihenfolge**
   Die Parameter von Methoden werden positionsweise zugeordnet.

Mit Ausnahme von Einschränkung 6, die eine Umgehungslösung darstellt, da das verwendete Ecore-Datenmodell bzw. das Korrespondenzmodell von EMF Compare keine Reihenfolgeverschiebung in der Liste der Übergabeparameter erkennt (siehe Abschnitt 4.4), würde das Aufheben einer dieser Einschränkungen wesentliche Änderungen in den Verfahren erzwingen. Insbesondere das edierkostenbasierte Verfahren aus Abschnitt 4.2 nutzt diese Einschränkungen gezielt aus.

Die Einschränkungen 2 bis 5 wurden eingeführt, um den Aufwand des Vergleichs deutlich zu reduzieren. Die Einschränkung 2 bewirkt, dass nur Elemente vom gleichen Typ miteinander verglichen werden müssen. Da Typänderungen für die Elemente eines Klassendiagramms, wie z. B. die Umwandlung einer Methode in eine Klasse, unüblich sind, handelt es sich dabei um eine sinnvolle Einschränkung.
Beim Vergleich zweier potentiell korrespondierender Klassen fließen die Ergebnisse des Vergleichs der Unterelemente je nach Verfahren in Form von Kosten bzw. Ähnlichkeitswerten ein. Statt alle Unterelemente aller Klassen flach miteinander zu vergleichen, werden aufgrund von Einschränkung 3 jeweils nur die Unterelemente der beiden Klassen miteinander verglichen. Daraus ergibt sich ein 2-Ebenen-Ansatz, bei dem die Probleme der Zuordnung der Unterelemente kleinere, einfachere Versionen der Zuordnungsprobleme darstellen. Diese kleineren Probleme werden jeweils mit dem gleichen Verfahren zuerst gelöst, so dass die Ergebnisse des Vergleichs der Unterelemente für die Bewertung der Korrespondenz auf Klassenebene verwendet werden können. Die Einschränkung 3 reduziert zwar den Aufwand des Vergleichs auf ein vertretbares Maß, verhindert aber auch, dass Verschiebungen von Unterelementen von einer zu einer anderen Klasse nativ erkannt werden können. Die Einschränkung 4 ist das Pendant zu Einschränkung 3 und

stellt sicher, dass Assoziationen und Vererbungskanten nicht verschoben werden dürfen. Aufgrund von Einschränkung 5 kann der Vergleich von Attributen und Operationen als kontextfrei aufgefasst werden. Dies stellt einen möglichen Erweiterungspunkt dar, der in die entwickelten Verfahren noch sinnvoll eingearbeitet werden könnte.
Für das Erkennen der Basisoperationen Einfügen, Löschen und Ändern ist eine eindeutige Zuordnung ausreichend. Eine Zuordnung, bei der ein Element zu mehreren Elementen korrespondieren kann, ist erst für das Erkennen von Klassen-übergreifendem Verschieben und Kopieren von Elementen interessant, wie sich in Beispiel 15 aus Abschnitt 4.4 zeigen wird. In Verbindung mit der vollständigen Teilbaumzuordnung (Einschränkung 3) und der bedingten Zuordnung (Einschränkung 4), bei der nur die Unterelemente und Beziehungskanten korrespondierender Klassen verglichen werden, stellt die Forderung nach einer eindeutigen Zuordnung (Einschränkung 1) jedoch keine große zusätzliche Einschränkung mehr dar.

4.2 Korrespondenzberechnungsverfahren basierend auf Edierkosten

In dem Verfahren, das als erste Idee in [Uhr08] veröffentlicht wurde, wird die Korrespondenzberechnung als Optimierungsproblem aufgefasst. Dabei werden alle zulässigen Zuordnungen mit Kosten bewertet und diejenige Zuordnung mit den geringsten Kosten bestimmt.

4.2.1 Kosten der Zuordnung

Um die Kosten einer Zuordnung zu ermitteln, werden alle möglichen Korrespondenzbeziehungen zwischen den Elementen $a \in A$ und $b \in B$ mit Kosten $c(a, b)$ bewertet. Außerdem werden für jedes Element Kosten bestimmt, die berücksichtigt werden, falls das Element ein Einzelelement der Zuordnung darstellt. Diese werden für ein Element $a \in A$ als $c(a, \epsilon)$, für ein Element $b \in B$ als $c(\epsilon, b)$ bezeichnet. Die Gesamtkosten $C(M)$ einer Zuordnung M ergeben sich als Summe der Kosten aller Korrespondenzbeziehungen der Zuordnung sowie den Kosten für die Einzelelemente (siehe Formel 4.1).

$$C(M) = \sum_{(i,j)\in M} c(i,j) + \sum_{\{i\in A|\forall j\in B:(i,j)\notin M\}} c(i,\epsilon) + \sum_{\{j\in B|\forall i\in A:(i,j)\notin M\}} c(\epsilon,j) \tag{4.1}$$

Als Kosten einer Korrespondenzbeziehung (i, j) werden die minimalen Kosten aller Folgen von Änderungsoperationen betrachtet, die das Element i in das Element j überführen (Formel 4.2). Analog werden die Kosten für das Löschen oder Einfügen berechnet, wobei ϵ in der Schreibweise das leere Element bezeichnet.

$$\forall i \in A \cup \{\epsilon\}, j \in B \cup \{\epsilon\} : c(i,j) = min_{\Delta(i\to j)}(C(\Delta(i \to j))) \tag{4.2}$$

Die Kosten einer Folge von Änderungsoperationen werden dabei als Summe der Kosten

der einzelnen Operationen definiert (Formel 4.3).

$$C(\Delta(i \to j)) = \sum_{op_n \in \Delta(i \to j)} c(op_n) \tag{4.3}$$

Die Bildung des Minimums in Formel 4.2 ist nötig, da eine Folge von Änderungsoperationen beispielsweise durch das Einfügen von Operationen, die sich gegenseitig neutralisieren, beliebig verlängert werden kann.

Zulässige Edieroperationen und deren Kosten Um die Kosten zwischen zwei Modellen als Abstand im Sinne einer Metrik verwenden zu können, wird für das in Abschnitt 4.1 vorgestellte Modell nun ein konfigurierbares Kostenmodell definiert, das die folgenden Bedingungen einer Metrik erfüllt:

Definition 7 (Metrischer Raum)**.** Eine Menge X, deren Elemente im Folgenden Punkte genannt werden, heißt ein metrischer Raum, wenn je zwei beliebigen Punkten p und q von X eine reelle Zahl $d(p, q)$ - als Abstand von p und q bezeichnet - zugeordnet ist, so dass gilt

$$d(p, q) \geq 0 \tag{4.4}$$

$$d(p, q) = 0 \Longleftrightarrow p = q \tag{4.5}$$

$$d(p, q) = d(q, p) \tag{4.6}$$

$$d(p, q) \leq d(p, r) + d(r, q) \ \forall \ r \in X \tag{4.7}$$

Eine Funktion, die diese vier Eigenschaften hat, wird Distanzfunktion oder Metrik genannt (nach [Rud05, S. 34]).

Für das Kostenmodell bedeuten die Bedingungen 4.4 und 4.5, dass die Kosten zwischen zwei identischen Modellen 0 sein müssen und jeder Edieroperation, die das Modell verändert, positive Kosten zugeordnet sein müssen. Außerdem muss das Kostenmodell nach Bedingung 4.6 symmetrisch sein, was bedeutet, dass inverse Operationen wie das Löschen und Einfügen von Elementen gleich gewichtet sein müssen. Als Letztes muss das Kostenmodell auch die Dreiecksgleichung aus Bedingung 4.7 erfüllen. Das Löschen eines Elements muss daher mindestens so teuer wie das Ändern aller seiner Eigenschaften sein. Im Folgenden werden nun zunächst die zulässigen Änderungsoperationen und Kosten des Kostenmodells eingeführt, bevor wir am Ende des Abschnitts wieder auf die Frage zurückkommen, ob es sich bei dem beschriebenen Kostenmodell um eine Metrik handelt.

Dazu werden für das Modell elementare Einfüge-, Lösch- und Änderungsoperationen definiert, die in Tabelle 4.1 nach den verschiedenen Elementtypen aufgeschlüsselt aufgelistet sind. Die Operationen sind elementar, da jede Operation jeweils ein leeres Element erzeugt oder löscht oder eine einzelne Elementeigenschaft setzt. So bezieht sich die Operation `löscheLeereKlasse(Klasse)` auf das Löschen einer Klasse, deren Eigenschaften nicht gesetzt sind und die keine Unterelemente (Attribute, Methoden) oder Querverbindungen (Vererbungsbeziehungen oder Assoziationen) besitzt. Soll beispielsweise eine

Klasse mit Attributen und Assoziationen gelöscht werden, müssen zuerst die Klasseneigenschaften zurückgesetzt und die Attribute und Assoziationen gelöscht werden, bevor die Löschoperation auf der Klasse ausgeführt werden kann. Das gleiche Prinzip gilt auch für Methoden. Vor dem Löschen einer Methode müssen der Name der Methode und der Rückgabetyp zurückgesetzt und vorhandene Übergabeparameter gelöscht werden. Bedingung 3 aus Abschnitt 4.1 entsprechend, werden Attribute und Operationen als Unterelemente einer Klasse erzeugt. Da keine Verschiebungen von Attributen oder Methoden unterstützt werden, gibt es keine Edieroperation, die das Vaterelement eines Attributs oder einer Methode manipuliert. Das Gleiche gilt für Parameter von Methoden. Vererbungskanten bzw. Assoziationen werden mit festgelegter Quell- und Zielklasse erzeugt. Gemäß Bedingung 4 aus Abschnitt 4.1 existieren für Assoziationen und Vererbungskanten keine Edieroperationen, die das Quell- oder Zielelement ändern.

Für die Einfügeoperationen und die Änderungsoperationen werden Kosten festgelegt. Die Kosten für die Löschoperationen werden, um die Symmetrie des Kostenmodells zu gewährleisten, nicht festgesetzt, sondern aus den Kosten der entsprechenden inversen Einfügeoperationen abgeleitet. Für Edieroperationen mit abgeleiteten Kosten, die in der zweiten Spalte der Tabelle 4.1 mit einem / gekennzeichnet sind, wird in der Kostenspalte angegeben, wie die Kosten berechnet werden.

Das Erzeugen bzw. das Löschen eines leeren Elements besitzt in der Praxis keine Bedeutung, da alle Eigenschaften des Elements bei dessen Erzeugung mit Standardwerten initialisiert werden. So wird beispielsweise beim Erzeugen einer Klasse im Editor auch der Name der Klasse und der Typ (konkrete Klasse, abstrakte Klasse oder Interface) festgelegt. Beim Einfügen einer Methode wird ferner der Name und der Rückgabetyp festgelegt. Als Einfügeoperationen werden stattdessen also zusammengesetzte Operationen verwendet, die sich in eine Einfügeoperation eines leeren Elements und die einzelnen Initialisierungen aller Eigenschaften dieses Elements zerlegen ließen. Es ist dennoch sinnvoll, zunächst für die elementaren Operationen Kosten festzulegen und die Kosten für zusammengesetzte Operationen aus der Summe der Kosten der darin enthaltenen elementaren Operationen abzuleiten, da so gewährleistet werden kann, dass das Kostenmodell die Dreiecksungleichung 4.7 erfüllt. Tabelle 4.2 zeigt weitere auf dem Modell zugelassene Edieroperationen, die aus den elementaren Operationen zusammengesetzt wurden. Die Bedingungen 4.4 und 4.5 einer Metrik verlangen positive Kosten für alle Edieroperationen, die das Modell verändern, was dadurch erreicht wird, dass für die elementaren Änderungsoperationen einzelner Eigenschaften in Tabelle 4.1 positive Kosten und für die elementaren Einfügeoperationen von leeren Elementen nichtnegative Kosten festgesetzt werden.[1] Dadurch ergeben sich auch für die zusammengesetzten Operationen aus Tabelle 4.2 positive Kosten.

Wahl des Zeichenkettenvergleichsverfahrens Für den Vergleich von Namen können verschiedene Methoden zum Vergleich von Zeichenketten eingesetzt werden, die als Ergebnis Edierkosten innerhalb des Intervalls $[0, 1] \in \mathbb{R}$ liefern. Folgende drei Varianten

[1] Da die Einfüge- und Löschoperationen von leeren Elementen nur in Kombination mit anderen Änderungsoperationen auftreten, ist die Bedingung der Nichtnegativität ausreichend.

Tabelle 4.1: Zulässige elementare Edieroperationen: Die mit Fragezeichen angegebenen Kosten müssen für die Konfiguration des Kostenmodells festgelegt werden, so dass die jeweils vorgegebenen Bedingungen > 0 bzw. ≥ 0 erfüllt werden.

Element		**Edieroperation**	**Kosten**
Klasse	k_1	erzeugeLeereKlasse ()	$? \geq 0$
	$/k_2$	löscheLeereKlasse (Klasse)	k_1
	k_3	setzeKlassenname (Klasse, neuerName)	$? > 0$
	k_4	setzeEigenschaftInterface (Klasse, istInterface)	$? > 0$
	k_5	setzeEigenschaftAbstrakt (Klasse, istAbstrakt)	$? > 0$
Attribut	a_1	erzeugeLeeresAttribut (Klasse)	$? \geq 0$
	$/a_2$	loescheLeeresAttribut (Attribut)	a_1
	a_3	setzeAttributname (Attribut, neuerName)	$? > 0$
	a_4	setzeAttributtyp (Attribut, neuerTyp)	$? > 0$
Methode	m_1	erzeugeLeereMethode (Klasse)	$? \geq 0$
	$/m_2$	löscheLeereMethode (Methode)	m_1
	m_3	setzeMethodenname (Methode, neuerName)	$? > 0$
	m_4	setzeRückgabetyp (Methode, neuerTyp)	$? > 0$
Parameter	p_1	erzeugeLeerenParameter (Methode)	$? \geq 0$
	$/p_2$	löscheLeerenParameter (Parameter)	p_1
	p_3	setzeParametername (Parameter, neuerName)	$? > 0$
	p_4	setzeParametertyp (Parameter, neuerTyp)	$? > 0$
Vererbungskante	v_1	erzeugeVererbungskante (Klasse, Klasse)	$? > 0$
	$/v_2$	lösche Vererbungskante (Vererbungskante)	v_1
Assoziation	r_1	erzeugeLeereAssoziation (Klasse, Klasse)	$? \geq 0$
	$/r_2$	löscheLeereAssoziation (Assoziation)	r_1
	r_3	setzeAssoziationsname (Assoziation, neuerName)	$? > 0$
	r_4	setzeUntereMultiplizität (Assoziation, Untergrenze)	$? > 0$
	r_5	setzeObereMultiplizität (Assoziation, Obergrenze)	$? > 0$
	r_6	setzeEigenschaftOpposite (Assoziation, Assoziation)	$? > 0$
	r_7	setzeEigenschaftKomposition (Assoziation, istKomposition)	$? > 0$

Tabelle 4.2: Kosten der zusammengesetzten Edieroperationen

Element		**Edieroperation**	**Kosten**
Klasse	$/k_6$	erzeugeKlasse (Name, istInterface, istAbstrakt)	$k_1+k_3+k_4+k_5$
	$/k_7$	löscheKlasseOhneUnterelemente (Klasse)	k_6
	$/k_8$	erzeugeKlasseMitUnterelemente (Klasse, n_1 Attribute, n_2 Methoden, n_3 Assoziationen, n_4 Vererbungskanten)	$k_6+n_1a_5+n_2m_7+n_3r_8+n_4v_1$
	$/k_9$	löscheKlasseMitUnterelemente (Klasse)	k_8
Attribut	$/a_5$	erzeugeAttribut (Klasse, Name, Typ)	$a_1+a_3+a_4$
	$/a_6$	loescheAttribut (Attribut)	a_5
Methode	$/m_5$	erzeugeMethode (Klasse, Name, Rückgabetyp)	$m_1+m_3+m_4$
	$/m_6$	löscheMethodeOhneParameter (Methode)	m_5
	$/m_7$	erzeugeMethodeMitParameter (Methode, n_5 Parameter)	$m_5+n_5p_5$
	$/m_8$	löscheMethodeMitParameter (Methode)	m_7
Parameter	$/p_5$	erzeugeParameter (Methode, Name, Typ)	$p_1+p_3+p_4$
	$/p_6$	löscheParameter (Parameter)	$p5$
Assoziation	$/r_8$	erzeugeAssoziation (Klasse, Klasse, Name, Obergrenze, Untergrenze, Assoziation, istKomposition)	$r_1+r_3+r_4+r_5+r_6+r_7$
	$/r_9$	löscheAssoziation (Assoziation)	r_8

stehen bislang zur Auswahl:

1. **Test auf Gleichheit**
   Die einfachste Möglichkeit, zwei Zeichenketten zu vergleichen, ist der Test auf Gleichheit. Falls die beiden Zeichenketten identisch sind, betragen die Edierkosten 0, ansonsten 1.

2. **Längste gemeinsame Teilfolge (LCS)**
   Zwei Zeichenketten, von denen mindestens eine nicht leer ist, werden mit einem LongestCommonSubsequence-Verfahren (z. B. [HS77], vgl. auch Abschnitt 2.1.1) verglichen. Die Anzahl derjenigen Zeichen der beiden Zeichenketten, die nicht Teil der größten gemeinsamen Zeichenfolge sind, wird in Relation zur Gesamtlänge beider Zeichenfolgen gesetzt (siehe Formel 4.8).

   $$lcs = 1 - \frac{2 * lcs.length}{a.length + b.length} \tag{4.8}$$

   Im Fall zweier leerer Zeichenketten werden die Edierkosten auf 0 gesetzt, da die Formel 4.8 für diesen Fall nicht definiert ist. Dass die Ergebnisse der Formel 4.8 im Intervall $[0, 1]$ liegen, wird durch Betrachtung der beiden Extremfälle offensichtlich. Bei zwei disjunkten Zeichenketten besitzt die längste gemeinsame Teilfolge die Länge 0, woraus sich Edierkosten in Höhe von 1 ergeben. Für zwei identische Zeichenketten entspricht die Länge der längsten gemeinsamen Teilfolge genau der Länge der beiden Zeichenketten, so dass die Formel den Wert 0 annimmt. Da die Länge der längsten gemeinsamen Teilfolge maximal die Länge der kürzeren Zeichenkette erreichen kann, liegt das Ergebnis für alle teilweise identischen Zeichenketten zwischen diesen beiden Extremwerten.

3. **Levenshtein-Abstand**
   Bei der Berechnung des Levenshtein-Abstands [SK83, S. 18-29] (vgl. auch Abschnitt 2.1.1) wird eine minimale Folge von Löschoperationen, Einfügeoperationen und Änderungsoperationen auf einzelnen Zeichen ermittelt, die eine Zeichenkette in die andere Zeichenkette transformiert. Um eine Normierung auf das Intervall $[0, 1]$ zu erhalten, wird wie in [YB07] die doppelte Levenshtein-Distanz in Relation zur Summe der beiden Zeichenkettenlängen und der Levenshtein-Distanz gesetzt (siehe Formel 4.9).

   $$lstN = \frac{2 * lst(a, b)}{a.length + b.length + lst(a, b)} \tag{4.9}$$

   Da die Levenshtein-Distanz zwischen zwei identischen Zeichenketten 0 beträgt, nimmt auch Formel 4.9 in diesem Fall den Wert 0 an. Anders als Formel 4.8 erreicht die Formel 4.9 den Extremwert 1 jedoch nur für den Vergleich einer leeren Zeichenkette mit einer anderen beliebigen Zeichenkette. Auch hier gilt die Formel nur, falls mindestens eine Zeichenkette nicht leer ist. Der Vergleich zweier leerer Zeichenketten wird mit 0 bewertet.

Möchte man ein Einheitskostenmodell verwenden, bei dem eine Namensänderung mit Kosten 1 bewertet werden soll, unabhängig davon, wie verschieden die beiden zu vergleichenden Zeichenketten sind, ist die Verwendung der ersten Variante sinnvoll. In den meisten Fällen ist der Test auf Gleichheit jedoch zu wenig differenziert, da somit Tippfehler in Namen zu einem unverhältnismäßig großen Edierabstand führen. In diesen Fällen ist es ratsam, Variante 2 oder 3 zu verwenden. Die Varianten 2 und 3 unterscheiden sich in der Gewichtung der Unterschiede. Wie bereits erwähnt, zeigt sich dieser Unterschied in der Extremwertbetrachtung sehr unterschiedlicher Zeichenketten. Damit die minimalen Kosten der Zuordnung als Abstand verwendet werden können, muss auch der Zeichenkettenvergleich die Kriterien einer Metrik erfüllen. Der Test auf Gleichheit erfüllt die Kriterien der Metrik inklusive Dreiecksungleichung offensichtlich. Für die normierte Levenshtein-Distanz (Formel 4.9) wird in [YB07] der Nachweis geführt, dass die Kriterien einer Metrik erfüllt werden.[2] Die Variante 2 auf Basis der längsten gemeinsamen Teilfolge (Formel 4.8) erfüllt die Dreiecksungleichung jedoch nicht, wie im folgenden Beispiel gezeigt wird.

Beispiel 4 (Gegenbeispiel zur Dreiecksungleichung)**.**
Für die Berechnung der Edierkosten zwischen Zeichenketten mit Hilfe der längsten gemeinsamen Teilsequenz nach Formel 4.8 ist für die Zeichenketten $a =$ "a", $b =$ "ab" und $c =$ "b" die Dreiecksungleichung nicht erfüllt. Mit $c(a,b) = \frac{1}{3}$, $c(b,c) = \frac{1}{3}$ und $c(a,c) = 1$ gilt $c(a,b) + c(b,c) < c(a,c)$.

Die dritte Variante besitzt damit gegenüber der zweiten Variante den Vorteil, dass sie den Kriterien einer Metrik entspricht. Falls die Kostenbewertung des gewählten Zeichenkettenvergleichs die Kriterien der Metrik erfüllt und in dem Kostenmodell aus Tabelle 4.1 die konfigurierbaren Kosten gemäß den angegebenen Bedingungen positiv bzw. nichtnegativ sind, können die Kosten einer Zuordnung als Abstand zwischen den verglichenen Modellen aufgefasst werden. Die Bedingung (4.4) wird erfüllt, indem jeder elementaren Änderungsoperation positive Kosten und den elementaren Einfügeoperationen nichtnegative Kosten zugeordnet werden. $c(i,j) \geq 0 \; \forall \; i \in A \cup \{\epsilon\}$, $j \in B \cup \{\epsilon\}$ und aus Formel 4.1 folgt für $M(A,B)$ damit $C(M) \geq 0$; insbesondere gilt $C(M) = 0 \iff A = B$, da $c(i,j) = 0 \iff i = j$. Damit ist die Bedingung 4.5 erfüllt. Die Symmetrie (4.6) wird durch symmetrische Kostenbewertungen inverser Operationen sichergestellt: Falls $c(i,j) = c(j,i)$ gilt, gilt auch $C(M(A,B)) = C(M(B,A))$. Um die Dreiecksungleichung (4.7) zu erfüllen, muss $c(i,j) \leq c(i,k) + (k,j)$ gelten. Gilt die Dreiecksungleichung für alle Einzeländerungen, gilt sie wegen Formel 4.1 auch für die Kosten der Zuordnung: $C(M(A,C)) \leq C(M(A,B)) + C(M(B,C))$. Die Bedingungen einer Metrik sind damit erfüllt.

Die Konfiguration des Kostenmodells Die Abbildung 4.3 zeigt das Konfigurationsfenster des Vergleichsrahmenwerks mit der Kostenkonfiguration, die für die Korrespondenzberechnung in den Beispielen der Arbeit verwendet wird. Dabei handelt es sich nicht

[2] In [YB07] wird der Beweis für eine verallgemeinerte Levenshtein-Norm geführt. Dieser Beweis ist auch für den Spezialfall gleichgewichteter Kosten der Änderungsoperationen Einfügen/Löschen/Ändern eines Zeichens gültig ($\alpha = 1$).

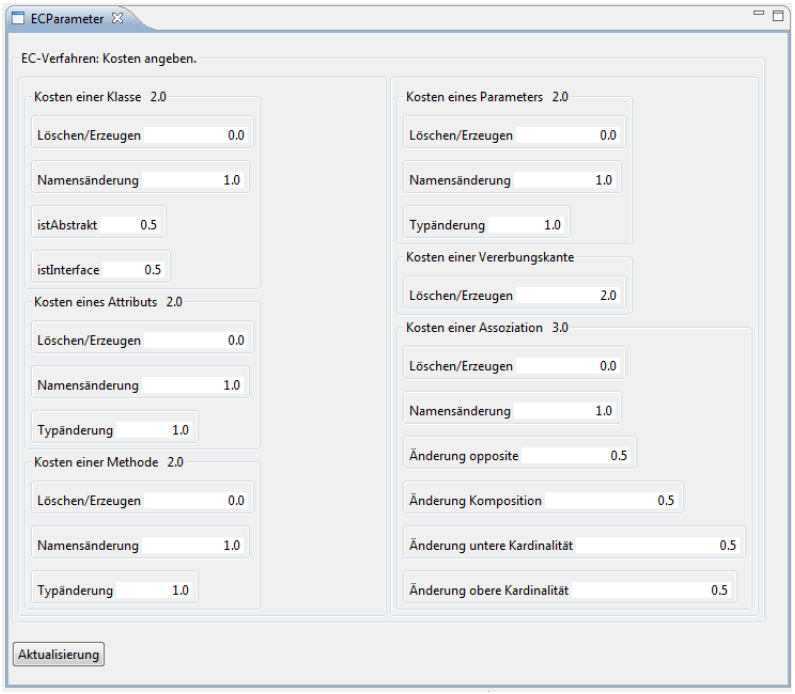

Abbildung 4.3: Konfiguration des Kostenmodells

um ein Einheitskostenmodell, bei dem jeder elementaren Operation die Einheitskosten in Höhe von 1 zugeordnet werden. Die Operationen wurden stattdessen nach vernünftigen Überlegungen gewichtet. So wurden für Änderungen boolescher Eigenschaften die Kosten 0,5 angesetzt, da der Wertebereich 2-elementig ist, während die Kosten für das Ändern von Namen oder Typen die maximalen Kosten 1 betragen können. Für das Löschen bzw. Erzeugen eines leeren Elements wurden dabei keine zusätzlichen Kosten berechnet. Während für alle anderen Operationen konkrete Kosten angegeben werden, entspricht der Wert für die Änderung eines Namenattributs einem Faktor, der mit dem Ergebnis des Zeichenkettenvergleichs aus $[0, 1] \in \mathbb{R}$ multipliziert wird. Bei den angegeben Kosten handelt es sich um einen Vorschlag, der, wie die Ergebnisse der Evaluation in Kapitel 6 zeigen, zu guten Resultaten führt. Das Kostenmodell ist konfigurierbar und kann auch zur Laufzeit des Vergleichsrahmenwerks über das Parameterkonfigurationsfenster geändert werden (vgl. Abschnitt 5). Falls beispielsweise Namensattribute eine besondere Rolle bei der Zuordnung von Modellelementen spielen sollen, kann das Ändern von Namen durch einen höheren Faktor stärker gewichtet werden. Wie bei vielen anderen Vergleichsverfahren mit zahlreichen Konfigurationsmöglichkeiten stellt sich die Frage, wie gut die Parameter eingestellt sind und wie sich eine Änderung der Parameter auswirkt. Die angegebenen Grundeinstellungen wurden mit logischen Überlegungen gefunden, wobei nur begrenzter Aufwand investiert wurde. Mit einer zeitaufwändigen Feinjustierung der Parameter ließen sich die Ergebnisse eventuell noch verbessern. Das folgende Beispiel zeigt, wie mit dieser Kostenkonfiguration die Edierkosten zwischen zwei Methoden berechnet werden.

Beispiel 5 (Kostenberechnung beim Methodenvergleich)**.** Es seien folgende zwei Methoden a_1 und b_1 gegeben, für die die Edierkosten berechnet werden, wobei für den Vergleich von Zeichenketten die zweite Variante, der LCS-Vergleich, verwendet wird:
a_1: `void berechne(int i, String s)`
b_1: `double berechneWert(double s)`
Die Kosten für das Löschen der Methode a_1 betragen:
$c(a_1, \epsilon) = 2$ (Löschen der Methode) + 2*2(Löschen zweier Übergabeparameter) = 6
Die Kosten für das Einfügen der Methode b_1 betragen:
$c(\epsilon, b_1) = 2$ (Löschen der Methode) + 2 (Löschen eines Übergabeparameters) = 4
Für die Korrespondenz der beiden Methoden ergeben sich die Kosten:
$c(a_1, b_1) = 0{,}2$ (Namensänderung) + 1 (Ändern des Rückgabetyps) + 2 (Ändern des Namens und des Typs des 1. Parameters) + 2 (Einfügen eines Parameters) = 5,2
In diesem Fall ist die Korrespondenz (a_1, b_1) mit Kosten in Höhe von 5,2 günstiger als das Löschen von a_1 und das Einfügen von b_1 mit Gesamtkosten in Höhe von 10.

Über ein definiertes Edierkostenmodell können jeder Zuordnung Kosten zugeordnet werden. Die Kosten der Zuordnung können nun als Vergleichskriterium herangezogen werden, um eine möglichst gute Zuordnung im Sinne niedriger Kosten zu bestimmen.

4.2.2 Brute-Force-Verfahren

Die Problemstellung, eine kostenminimale Zuordnung M für die Modelle A und B zu bestimmen, kann wie folgt als Optimierungsproblem formuliert werden:

Optimierungsproblem 1 (Kostenminimale Zuordnung)**.**
Bestimme $M^*(A, B)$ mit $C(M^*(A, B)) \leq C(M(A, B)) \; \forall \; M(A, B)$.

Die Kosten für die Korrespondenz zweier Klassen beinhalten auch die Kosten für die Zuordnung ihrer Unterelemente, insbesondere ihrer Assoziationen und Vererbungskanten (vgl. Abschnitt 4.2.1). In Abschnitt 4.1 wurde die Einschränkung formuliert, dass Assoziationen und Vererbungsbeziehungen nur dann korrespondieren, falls beide beteiligten Klassen entsprechend korrespondieren. Die Kosten einer Korrespondenz zweier Klassen sind in diesem Fall von der Zuordnung anderer Klassen abhängig. Daher kann die minimale Zuordnung nur dadurch in jedem Fall exakt bestimmt werden, dass die Kosten der Zuordnung für alle Kombinationsmöglichkeiten der Klassen der beiden Modelle explizit berechnet werden. Für die m Klassen in Modell A und n Klassen in Modell B werden folgende zwei Fälle unterschieden:

1. **Maximale Zahl an Korrespondenzen** In dieser Variante wird angenommen, dass genau $min(m, n)$ Korrespondenzen und $|m-n|$ Einzelelemente ermittelt werden. Falls $m \geq n$ gilt, existieren $\binom{m}{m-n} * n!$ Kombinationsmöglichkeiten. Diese ergeben sich folgendermaßen: Auf m Positionen sind $m - n$ ϵ-Symbole zu positionieren. Dafür gibt es $\binom{m}{m-n}$ Möglichkeiten. Für jede dieser Möglichkeiten gibt es $n!$ Möglichkeiten, die Elemente aus B auf den restlichen Plätzen zu verteilen.

2. **Berücksichtigung aller möglichen Zuordnungen** Es gibt keine Einschränkung für die Anzahl der Einzelelemente, so dass durch die Zuordnung im maximalen Fall $m + n$ Elemente als Einzelelemente identifiziert werden können. Hier gibt es $\sum_{i=0}^{n} \left(\frac{m!}{(m-i)!} * \binom{n}{i} \right)$ Kombinationsmöglichkeiten, da m Positionen mit $i = 0, \ldots, n$ verschiedenen und $(m - i)$ identischen ϵ-Elementen besetzt werden können.

Der Aufwand ist in beiden Fällen nicht vertretbar, da für das Optimierungsproblem 1 kein polynomielles Lösungsverfahren bekannt ist. Hauptverursacher der Komplexität des Problems sind die kontextabhängigen Kosten. In den nachfolgenden Abschnitten wird beschrieben, wie mit Hilfe abgeschätzter Kosten die Abhängigkeit der Kosten einer Korrespondenz von den anderen Korrespondenzen entkoppelt und der Aufwand damit reduziert wird. Das daraus resultierende Verfahren wird in seinen Einzelschritten beschrieben und schließlich in Abschnitt 4.2.8 zusammengefasst. Die Lösung des Optimierungsproblems 1 mittels vollständiger Enumeration der Kombinationen wurde als Brute-Force-Variante (BFVerfahren) dennoch implementiert, um das Verfahren mit den abgeschätzten Kosten in Hinblick auf die Optimalität der berechneten Ergebnisse zu validieren (vgl. Kapitel 6). Die Standardvariante entspricht der oben beschriebenen Variante 1, die Variante 2 wird in der Implementierung als BFVerfahren mit allen Epsilonkombinationen bezeichnet.

4.2.3 Abschätzung der kontextabhängigen Kosten

Die Kosten einer Zuordnung zweier Klassen lassen sich in kontextabhängige und kontextunabhängige Kosten unterteilen. Die Kosten der Korrespondenzberechnungen für Attribute und Methoden sind kontextunabhängig, die für Assoziationen und Vererbungskanten kontextabhängig.

Die kontextabhängigen Kosten werden entkoppelt, indem die exakten Kosten, die nur im Kontext ermittelt werden können, durch abgeschätzte Kosten ersetzt werden. Die Kosten werden unter der optimistischen Annahme abgeschätzt, dass der Kontext eines Modellelements passend abgebildet wird. Dadurch stellen die geschätzten Kosten c_{lb} eine untere Schranke für die tatsächlichen Kosten c_{real} dar.

$$c_{lb} \leq c_{real} \tag{4.10}$$

Die Kosten für die Zuordnung einer Assoziation oder Vererbungskante werden dabei den beiden angrenzenden Klassen jeweils zur Hälfte angerechnet.

Beispiel 6 (Abgeschätzte Kosten beim Vergleich zweier Klassen)**.** Die Abbildungen 4.4 und 4.5 zeigen zwei Versionen eines Klassendiagramms für die Modellierung eines Bestellsystems, die bereits aus Abschnitt 1.2.1 bekannt sind. Modell B entsteht aus Modell A, indem das Attribut `state` der Klasse `USAddress` in eine neue Unterklasse `USAddress` ausgelagert wird. Die ursprüngliche Klasse wird in `Address` umbenannt. Mit Hilfe der Edierkosten wird nun ermittelt, zu welcher Klasse die ursprüngliche Klasse `USAddress` aus Modell A korrespondiert.

Die kontextunabhängigen Kosten der Zuordnung (`USAddress`, `USAddress`) betragen 5*2 = 10, da in der Klasse des Modells A 5 Attribute mehr vorhanden sind. Die kontextabhängigen Kosten werden wie folgt berechnet: 2*1,5 (Assoziationen) + 1 (Vererbungskante) = 4. Damit ergeben sich Gesamtkosten von c(`USAddress`, `USAddress`) = 14. Analog werden folgende Kosten berechnet:
c(ϵ, `USAddress`) = 2 (Klasse) + 2 (Attribut) + 1 (Vererbungskante) = 5
c(`USAddress`,ϵ) = 2 (Klasse) + 6*2 (Attribute) + 2*1,5 (Assoziationen) = 17.
c(ϵ, `Address`) = 2 (Klasse) + 5*2 (Attribute) + 2*1,5 (Assoziationen) = 15
c(`USAddress`, `Address`) = 0,125 (Namensänderung) + 2 (Attribut) + 1 (Vererbungskante) = 3,125
Für die günstigste Zuordnung M gilt $M = \{$(`Item`, `Item`), (`PurchaseOrder`, `PurchaseOrder`), (`USAddress`, `Address`)$\}$ mit $C(M) = 3,125 + 5 = 8,125$.

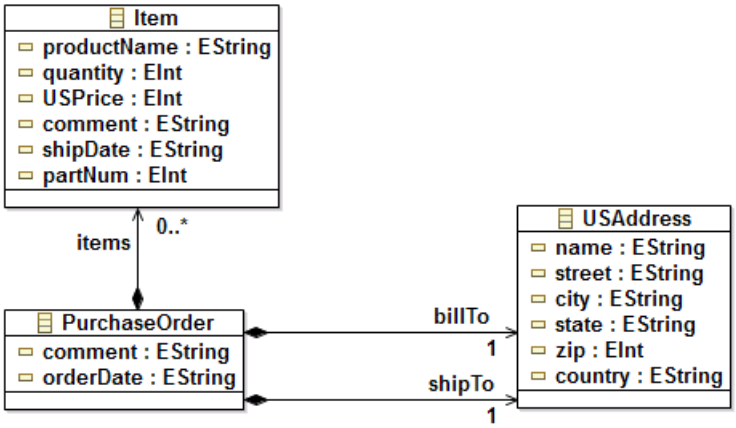

Abbildung 4.4: Modell A für Beispiel 6

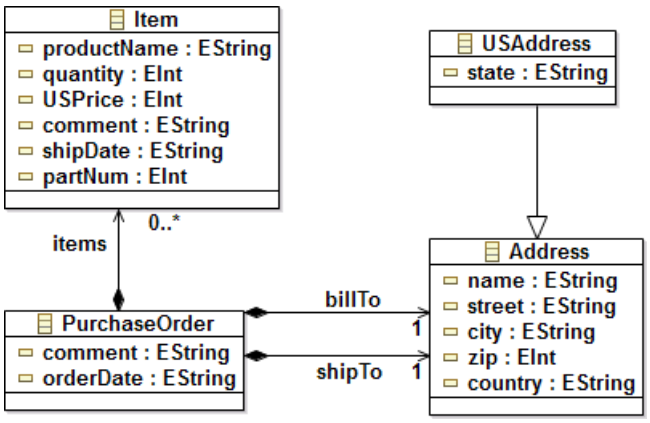

Abbildung 4.5: Modell B für Beispiel 6

Durch die Abschätzung der Kosten entsteht eine Relaxation des ursprünglichen Optimierungsproblems aus Problem 1. Wie die Lösung des relaxierten Problems mit dem

ursprünglichen Problem zusammenhängt, wird in Abschnitt 4.2.6 erläutert. In den folgenden Abschnitten wird das relaxierte Problem als Netzwerkflussproblem formuliert und ein Verfahren angegeben, mit dem dieses Problem gelöst werden kann. Das Verfahren zur Korrespondenzberechnung wurde in der Klasse `ECVerfahren` implementiert (siehe Kapitel 5).

4.2.4 Das Netzwerkflussproblem des relaxierten Problems

Nach Abschätzung der kontextabhängigen Kosten kann das relaxierte Problem als Fragestellung auf einem Netzwerkgraphen formuliert werden. Ein Netzwerkgraph stellt einen Spezialfall eines gerichteten Graphen dar. Ein gerichteter Graph ist ein Graph, dessen Kanten nicht durch eine 2er-Menge sondern durch ein 2er-Tupel von Knoten beschrieben werden. In den weiteren Ausführungen handelt es sich bei allen Kanten um gerichtete Kanten, auch wenn dies nicht explizit angegeben wird. Ein Netzwerkgraph wird wie folgt definiert:

Definition 8 (Netzwerkgraph)**.** Ein Netzwerkgraph ist ein gerichteter Graph $N = (V, E)$, der eine ausgezeichnete Quelle $r \in V$ und eine ausgezeichnete Senke $s \in V$ enthält. Den gerichteten Kanten $e = (i, j)$ sind als Minimal- und Maximalkapazitäten natürliche Zahlen $\lambda_{ij} \leq \kappa_{ij}$ sowie nichtnegative reellwertige Kosten c_{ij} zugeordnet. Die Flussstärke der Kante (i, j) wird mit ϕ_{ij} bezeichnet. Auf den Kanten des Netzwerkgraphen können über die gerichteten Kanten Einheiten von der Quelle zur Senke transportiert werden, wobei die Minimal- und Maximalkapazitäten der Kanten berücksichtigt werden müssen.

$$\lambda_{ij} \leq \phi_{ij} \leq \kappa_{ij},\ (i,j) \in E \tag{4.11}$$

Das Kirchhoffsche Gesetz besagt, dass die Summe der Einheiten, die auf den eingehenden Kanten eines Knotens transportiert werden, mit der Zahl der Einheiten identisch sein soll, die auf den ausgehenden Kanten des Knotens transportiert werden. Dies gilt für alle Knoten bis auf zwei Ausnahmen: die Quelle und die Senke. Das Kirchhoffsche Gesetz lässt sich für den Gesamtfluss in Höhe von n Einheiten wie folgt formal beschreiben:

$$\sum_{(i,j)\in E} \phi_{ij} - \sum_{(k,i)\in E} \phi_{ki} = \begin{cases} n & \text{falls i = r} \\ -n & \text{falls i = s} \\ 0 & \forall i \in V \backslash \{r, s\} \end{cases} \tag{4.12}$$

Ein Fluss auf dem Netzwerk, der die Bedingungen 4.11 und 4.12 erfüllt, wird als zulässiger Fluss bezeichnet (vgl. [NM02]).

Aufbau des Vergleichsgraphen

Für den Vergleich der Modellelemente $a_1,\ldots,a_n$ eines Modells A mit den Elementen $b_1,\ldots,b_m$ eines Modells B mit $n \geq m$ ist ein Netzwerkgraph, im Folgenden auch als Vergleichsgraph bezeichnet, nach folgenden Regeln zu konstruieren, wobei für die Kosten die abgeschätzten Kosten verwendet werden. Falls $n < m$ gilt, sind die Modelle o. B. d. A zu vertauschen. Ein Beispiel eines Netzwerkgraphen ist in Abbildung 4.6 abgebildet.

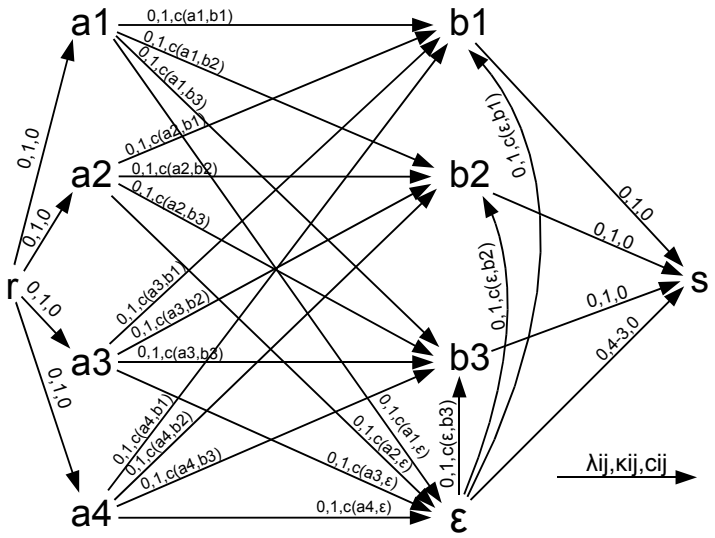

Abbildung 4.6: Netzwerkgraph für den Vergleich von $A = \{a_1, a_2, a_3, a_4\}$ und $B = \{b_1, b_2, b_3\}$

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 $(0, 1, c(a, b))$ ein, die die Ersetzung von a durch b mit Kosten $c(a, b)$ repräsentiert.
6. Füge einen Knoten ϵ ein und erzeuge Löschkanten (a, ϵ) mit Markierung $(0, 1, c(a, \epsilon)$ bzw. Erzeugungskanten (ϵ, b) mit Markierung $(0, 1, c(\epsilon, b))$ für alle $a \in V_A$ und $b \in V_B$. Verbinde den Knoten ϵ auch mit dem Zielknoten s mit einer Kante der Markierung $(0, n - m, 0)$, falls die Kapazität $n - m$ der Kante positiv ist.

Der Netzwerkgraph beinhaltet einen bipartiten Graphen aus den Partitionen der Elemente der beiden Modelle, der um einen Quell- und Zielknoten sowie den Knoten ϵ ergänzt wird. Diejenigen Kanten, die nicht zur Quelle oder Senke inzident sind, werden auch als innere Kanten bezeichnet. Für die Minimalkapazitäten aller Kanten gilt $\lambda_{ij} = 0$. Die eingehenden Kanten des ϵ-Knotens werden zum Löschen und dessen auslaufende Kanten zum Einfügen von Modellelementen benötigt. Die Fragestellung, eine kostenminimale Zuordnung für das relaxierte Problem zu berechnen, entspricht nun der Suche

eines kostenminimalen maximalen Flusses auf dem konstruierten Netzwerkgraphen. Ein maximaler Fluss besitzt die Stärke n.

Da alle von r ausgehenden Kanten eine Maximalkapazität von 1 besitzen, wird bei einem Fluss der Stärke n für jedes $a \in A$ genau eine auslaufende Kante beschickt. Dadurch wird festgelegt, ob a zu einem Element $b \in B$ korrespondiert oder als gelöscht interpretiert wird. Die Balancierungsregeln des Kirchhoffschen Gesetzes garantieren ferner, dass jedes $b \in B$ entweder als erzeugt dargestellt wird oder zu einem $a \in A$ korrespondiert. Durch die Konstruktion des Netzwerkgraphen wird somit sichergestellt, dass es sich bei der Lösung des Netzwerkflussproblems um eine zulässige Lösung des Korrespondenzberechnungsproblems handelt.

Falls das Kostenmodell die Dreiecksungleichung erfüllt, sind die Kosten, ein Element a_i in ein Element b_j zu transformieren, immer günstiger als a_i zu löschen und b_j zu erzeugen. Ein derartiger Pfad ist folglich nur dann im kostenminimalen maximalen Fluss enthalten, falls die Kostenbewertungen die Dreiecksungleichung nicht erfüllen, wie z. B. nach Erhöhung der Kosten einzelner Kanten entsprechend einer Schwellenwertregel, wie sie in Abschnitt 4.2.7 beschrieben wird.

Formal lässt sich das Problem wie folgt beschreiben:

Optimierungsproblem 2 (Netzwerkflussproblem des relaxierten Problems)**.**

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

$$u.d.B. \sum_{(i,j)\in E} \phi_{ij} - \sum_{(k,i)\in E} \phi_{ki} = \begin{cases} n & \text{falls i = r} \\ -n & \text{falls i = s} \\ 0 & \forall i \in V\backslash\{r,s\} \end{cases} \tag{4.13}$$

$$0 \leq \phi_{ij} \leq \kappa_{ij},\ (i,j) \in E \tag{4.14}$$

Da der ϵ-Knoten zu mehr als einer ausgewählten Kante inzident sein darf und der Netzwerkgraph ohne Quelle und Senke aufgrund der Kanten des ϵ-Knotens zu den Knoten der Modellelemente aus B nicht mehr bipartit ist, kann für die Lösung kein Summen-Matching-Verfahren für bipartite Graphen, wie z. B. das Glover-Klingman-Verfahren aus [NM02, S. 299], verwendet werden. Für Ausführungen zu allgemeinen Optimierungsproblemen wird auf [NM02] verwiesen. Im nächsten Abschnitt wird ein Lösungsverfahren für dieses Netzwerkflussproblem dargestellt. Auf die Wahl des Verfahrens wird in Abschnitt 4.2.6 eingegangen.

4.2.5 Lösung des Netzwerkflussproblems mit dem Verfahren von Busacker und Gowen

Die Lösung des Netzwerkflussproblems besteht darin, n Einheiten von der Quelle r auf dem günstigsten Weg zur Senke s zu transportieren. Dazu wurde der Algorithmus von Busacker und Gowen [BG61] implementiert (dargestellt z. B. in [NM02, S. 288] und

[Jun08, S. 299]). Dieses Verfahren löst beliebige Netzwerkflussprobleme, die folgende Voraussetzungen erfüllen:

- nichtnegative Kosten
  $c_{ij} \leq 0 \; \forall \; (i,j) \in E$
- verschwindende Miminalkapazitäten
  $\lambda_{ij} = 0 \; \forall \; (i,j) \in E$
- positive Maximalkapazitäten
  $\kappa_{ij} > 0 \; \forall \; (i,j) \in E$
- antisymmetrisches Netzwerk
  Das Netzwerk enthält kein Paar entgegengesetzt gerichteter Pfeile.

Die aufgezählten Anforderungen sind weniger als Beschränkungen denn als technische Anforderungen zu betrachten, da sich alle Bedingungen durch eine Umformulierung des Problems erreichen lassen. So zum Beispiel die Antisymmetrie: Falls sowohl die Kante (a, b) als auch die Rückkante (b, a) im Netzwerk N enthalten sind, führt dies im Fall, dass auf einer der beiden Kanten ein Fluss fließt, zu parallelen Kanten im Inkrementnetzwerk. Dies lässt sich jedoch gegebenfalls durch Einfügen von Zwischenknoten beheben [NM02]. Bei der Konstruktion unseres Netzwerkgraphen werden alle genannten Anforderungen berücksichtigt, so dass das Verfahren eingesetzt werden kann.

Das Verfahren wird in den nachfolgenden Abschnitten beschrieben. Darüber hinaus zeigt Algorithmus 1 eine Darstellung des Verfahrens in Pseudocode, die in modifizierter Form aus [NM02, S. 288] übernommen wurde. Auf dem Netzwerk, das mit einem 0-Fluss initialisiert ist, wird eine Folge von Flussvergrößerungen durchgeführt, bis die Zielkapazität ω^* des Netzwerkproblems erreicht ist. Dazu wird ein flussvergrößernder kostenminimaler Pfad von r zu s mit Hilfe eines Kürzeste-Wege-Verfahrens bestimmt (Prozedur A). Wird kein derartiger Pfad gefunden, kann die gesuchte Flussstärke auf dem Netzwerk nicht erreicht werden. Ansonsten findet entlang des markierten Pfads eine Flussvergrößerung statt (Prozedur B).

In **Prozedur A** wird ein Kürzeste-Wege-Verfahren auf einem sogenannten Inkrementnetzwerk[3] N' durchgeführt, um nicht nur einen beliebigen, sondern den Pfad mit den geringsten Kosten zu bestimmen, der den Netzwerkfluss erhöht. Das Inkrementnetzwerk $N'(\phi)$ zu einem Netzwerk N ist vom aktuellen Fluss ϕ auf dem Netzwerk abhängig und enthält alle Knoten aus N, sowie diejenigen Kanten aus N, deren Kapazität durch den Fluss ϕ noch nicht ausgelastet ist. Die Maximalkapazitäten der übernommenen Kanten entsprechen den noch freien Kapazitäten der Kanten aus N. Diese Kanten, die in gleicher Richtung sowohl im Inkrementnetzwerk $N'(\phi)$ als auch in N enthalten sind, werden im Folgenden auch als Vorwärtskanten bezeichnet. Weiterhin wird für jede Kante in N, die mindestens eine Einheit transportiert, in das Inkrementnetzwerk eine entgegengerichtete

[3] Die Bezeichnung *Inkrementnetzwerk* wurde aus [NM02] übernommen. In [Jun08] wird es als *auxiliary network* bezeichnet.

Algorithmus 1 Busacker-Gowen-Algorithmus

Gesucht wird ein kostenminimaler Fluss der Stärke ω^* im Netzwerk N.

Verwendete Notationen:
$S(i)$: Menge der Nachfolgerknoten des Knoten i; $M(i)$: Menge der markierbaren Nachfolgerknoten von i; $Q(i)$: Menge neu markierter Knoten (Schlange); d_i: kürzeste bisher gefundene Länge des Weges; p_i: Vorgängerknoten auf dem kürzesten Weg von r zu i

Initialisierung: $\forall\ i = 1, \ldots, n$ setze $M(i) := S(i)$ und $\phi_{ij} := 0\ \forall\ j \in S(i)$

Hauptschritt:

Solange $\omega < \omega^*$
Prozedur A zur Markierung des kostenminimalen flussvergrößernden Pfads
Falls $\epsilon_s = 0$
Ende: Es existiert kein Fluss der Stärke ω' auf N.
Andernfalls *Prozedur B zur Durchführung der Flussvergrößerung*

Prozedur (A): Markierung eines flussvergrößernden kostenminimalen Pfads von der Quelle zur Senke auf dem Hilfsnetzwerk N' (Erweiterter LC-Algorithmus A aus [NM02, S. 208])

Setze $d_r := 0$, $p_r :=$ r, $\epsilon_r := \infty$, $\epsilon_s := 0$,
Schlange $Q := \{r\}$, $d_j := \infty\ \forall\ j = 1 \ldots n$ mit $j \neq$ r
Solange $Q \neq \emptyset$
Entferne i aus Q
$\forall\ j \in M(i)$ mit $d_j > d_i + c_{ij}$
Falls $j \in S(i)$ (Vorwärtsmarkierung)
setze $d_j := d_i + c_{ij}$, $p_j := i$, $\epsilon_j := min(\epsilon_i, \kappa_{ij} - \phi_{ij})$
Andernfalls (Rückwärtsmarkierung)
setze $d_j := d_i$ - c_{ij}, $p_j := -i$, $\epsilon_j := min(\epsilon_i, \phi_{ij})$
Falls $j \notin Q$, füge j in Q ein.
$\epsilon_s = min(\epsilon_s, \omega^* - \omega)$

Prozedur (B): Durchführung der Flussvergrößerung entlang des markierten Pfades um ϵ_s

Setze $i := s$, $\omega := \omega + \epsilon_s$
Wiederhole bis $i = r$
Setze $j := i$, $i := |p_j|$, $M(j) := M(j) \cup \{i\}$
Falls $p_j > 0$ (Flussvergrößerung)
Setze $\phi_{ij} := \phi_{ij} + \epsilon_s$
Falls $\phi_{ij} = \kappa_{ij}$, setze $M(i) := M(i) \setminus \{j\}$
Andernfalls (Rückflussverkleinerung)
Setze $\phi_{ji} := \phi_{ji}$ - ϵ_s
Falls $\phi_{ji} = 0$, setze $M(i) := M(i) \setminus \{j\}$

Kante eingefügt. Diese Kanten werden als Rückwärtskanten bezeichnet. Die Maximalkapazität der Rückwärtskante entspricht der Flussstärke der ursprünglichen Kante in N. Die Kostenbewertungen der Vorwärtskanten werden übernommen, für die Kosten c' der Rückwärtskanten (j, i) gilt $c'_{ji} = -c_{ij}$. Die Minimalkapazitäten aller Kanten bleiben 0. Das Inkrementnetzwerk muss nicht als separater Graph gespeichert werden, sondern kann innerhalb N durch eine weitere Kantenmenge dargestellt werden. So bezeichnet die Menge $M(i)$ in Algorithmus 1 die Menge der ausgehenden Kanten des Knotens i, die im Inkrementnetzwerk enthalten sind.

Das Kürzeste-Wege-Verfahren führt Knotenmarkierungen durch. Die Knotenmarkierung d_j eines Knotens j bezeichnet den Abstand des bislang kürzesten Weges von der Quelle r zu j. Die Länge eines Pfades ergibt sich als Summe der Kosten der verwendeten Kanten. Um einen kürzesten Weg rekonstruieren zu können, wird zu dem Knoten j auch der Vorgängerknoten von j auf dem kürzesten Weg von r zu j in der Variablen p_j gespeichert. Die maximale verfügbare Kapazität auf diesem kürzesten Weg von r zu j wird in ϵ_j gespeichert. Sie beträgt das Minimum der maximalen Kapazitäten der im Pfad enthaltenen Kanten. Die Knotenmarkierung im Kürzeste-Wege-Verfahren beginnt bei der Quelle r mit $d_r := 0$ und $p_r := r$. Für alle Nachfolgerknoten j von r, die im Inkrementnetzwerk enthalten sind, werden nun die Markierungen durchgeführt, falls der Knoten noch nicht markiert war, bzw. aktualisiert, falls der Weg von r über i zu j kürzer als der bislang markierte Weg zu j ist. Die Einträge in d_j, p_j und ϵ_j werden neu berechnet, wobei berücksichtigt werden muss, ob es sich bei der Kante (i, j) um eine Vorwärts- oder eine Rückwärtskante handelt (vgl. Vorwärtsmarkierung und Rückwärtsmarkierung in Algorithmus 1, Prozedur A). Der Knoten j wird, falls sich seine Markierung geändert hat, in die Menge $M(i)$ der neu markierten Knoten aufgenommen, die als Schlange realisiert ist. Aus dieser Menge wird der nächste Knoten i entnommen, für den ebenfalls die Markierungen der Nachfolgerknoten j überprüft und gegebenenfalls aktualisiert werden. Dies wird so lange festgesetzt, bis alle Änderungen berücksichtigt wurden, d. h. die Schlange Q keine weiteren Knoten enthält. Der in Prozedur B bestimmte kürzeste Pfad von r nach s liegt nun implizit in den Markierungen vor. Beginnend bei der Senke $i = s$ kann über p_i der jeweilige Vorgängerknoten bestimmt und so der kürzeste Weg abgelesen werden. Die maximale Kapazität dieses Weges entspricht ϵ_s. Dies legt die Stärke der Flussvergrößerung in Prozedur B fest, wobei beachtet werden muss, dass durch die Flussvergrößerung nicht die vorgegebene Zielflussstärke ω überschritten wird. Bei unserem Optimierungsproblem wird der Fluss in jedem Schritt um genau eine Einheit erhöht, da die Kanten, die von der Quelle r ausgehen, laut Konstruktionsvorschrift jeweils die Maximalkapazität 1 besitzen.

In **Prozedur B** wird eine Flussvergrößerung in Höhe von ϵ_s durchgeführt, falls in Prozedur A ein flussvergrößernder Pfad gefunden wurde. Dazu wird der markierte Pfad beginnend bei der Senke in Richtung Quelle durchlaufen, wobei einerseits die Flüsse auf den Kanten in N angepasst und andererseits die Kanten des Inkrementnetzwerks für den nächsten Schritt aktualisiert werden. Der Iterationsschritt der Prozedur B in Algorithmus 1 bezieht sich auf die markierte Kante (i, j), wobei $j = r$ im ersten Iterationsschritt gilt. Da auf der Kante (i, j) mindestens eine Einheit transportiert wird, kann die entgegengerichtete Kante (j, i) in die Kanten des Inkrementnetzwerks aufgenommen

werden. Für die nachfolgenden Anweisungen muss der Fall unterschieden werden, ob die Kante vorwärts oder rückwärts markiert wurde. Im Falle der Vorwärtsmarkierung wird der Fluss auf der Kante (i, j) um den Betrag ϵ_s erhöht. Falls damit die Maximalkapazität der Kante (i, j) erreicht ist, steht sie für weitere Flussvergrößerungen nicht mehr zur Verfügung und wird daher aus dem Inkrementnetzwerk entfernt. Falls die Kante (i, j) stattdessen rückwärts markiert wurde, wird der Fluss der Kante (j, i) in N um den Betrag ϵ_s reduziert. Falls dabei die Minimalkapazität 0 der Kante (j, i) erreicht wurde, wird diese Kante aus dem Inkrementnetzwerk entfernt.

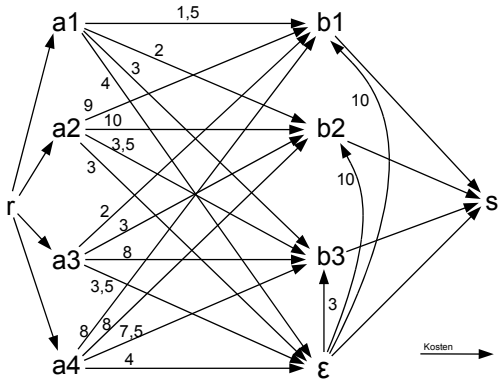

Abbildung 4.7: $N(0)$: Netzwerk mit 0-Fluss $\Leftrightarrow$ $N'(0)$: Inkrementnetzwerk zu $N(0)$

Der Ablauf des Busacker-Gowen-Verfahrens wird im nachfolgenden Beispiel verdeutlicht, wobei auf die Darstellung des Kürzeste-Wege-Verfahrens, das auf jedem Inkrementnetzwerk auszuführen ist, an dieser Stelle verzichtet wird, da dies im anschließenden Beispiel 8 behandelt wird.

Beispiel 7 (Ablauf des Busacker-Gowen-Verfahrens)**.** Zu dem Beispielgraphen in Abbildung 4.7 soll ein kostenminimaler Fluss der Stärke 4 bestimmt werden. Für die im Beispiel abgebildeten Netzwerke wurde dabei folgende Notation verwendet: schwarze, dickgedruckte Pfeile bezeichnen aktive Kanten, auf denen genau eine Einheit transportiert wird. Alle anderen Pfeile entsprechen inaktiven Kanten mit Flussstärke 0. Blaue bzw. rote Pfeile dienen lediglich der Markierung derjenigen Kanten, die auf dem Inkrementnetzwerk für einen kürzesten Weg ausgewählt wurden bzw. Kanten im Netzwerk, die an der Flussvergrößerung beteiligt sind. Die Werte neben den Kanten bezeichnen die Kosten, wobei für Kanten (i, j) ohne Angabe der Kosten $c(i, j) = 0$ gilt.

Schritt 1: $N(0) \rightarrow N(1)$

Der Ausgangsgraph mit Flussstärke 0 wird mit $N(0)$ bezeichnet und ist, da alle Kanten inaktiv sind, identisch mit dem Inkrementnetzwerk $N'(0)$. Auf dem Inkrementnetzwerk $N'(0)$ wird ein kürzester Weg von r nach s bestimmt. Dieser verläuft über die Kanten (r, a_1),(a_1, b_1) und (b_1, s) und besitzt die Länge $0 + 1{,}5 + 0 = 1{,}5$. Da das Minimum der Maximalkapazitäten der verwendeten Kanten 1 beträgt, kann auf diesem Pfad nur eine

Einheit transportiert werden. Da aufgrund der Konstruktion des Graphen die Maximalkapazitäten aller flussvergrößernder Pfade 1 betragen, wird darauf im Weiteren nicht mehr eingegangen. Die Flussvergrößerung in Höhe von 1 wird auf $N(0)$ durchgeführt, indem auf den Kanten (r, a_1),(a_1, b_1) und (b_1, s) der Fluss jeweils von 0 auf 1 gesetzt wird (vgl. Abbildung 4.8). Damit ergeben sich Gesamtkosten von $c = 1,5$ bei einer Flussstärke von $\phi = 1$ auf $N(1)$.

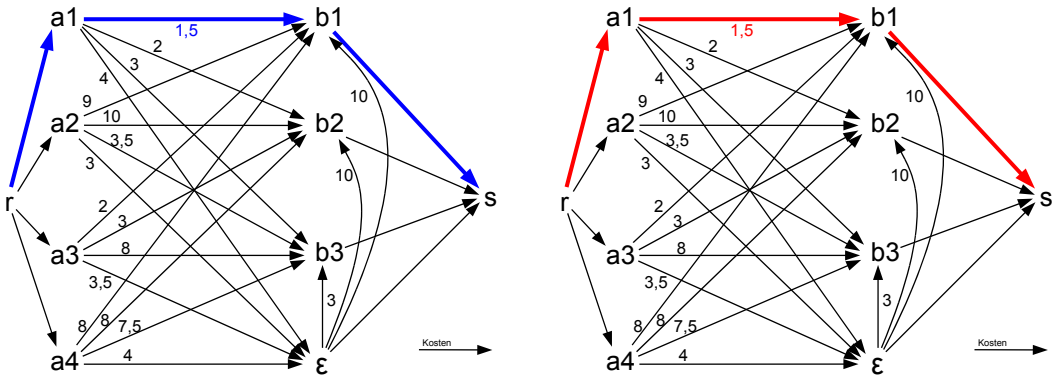

Abbildung 4.8: Links: Kürzester Pfad auf $N'(0)$, Rechts: Flussvergrößerung auf $N(0)$

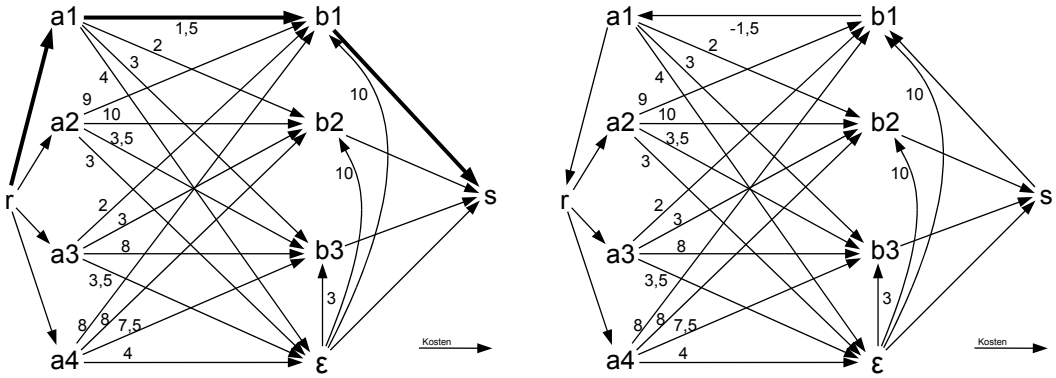

Abbildung 4.9: Netzwerk $N(1)$ (links) und dazugehöriges Inkrementnetzwerk $N'(1)$ (rechts)

Schritt 2: $N(1) \rightarrow N(2)$

Das Inkrementnetzwerk $N'(1)$ wird aus $N(1)$ konstruiert, indem anstatt der aktiven Kanten (r, a_1),(a_1, b_1) und (b_1, s), die ihre Maximalkapazität bereits erreicht haben, jeweils eine entgegengerichtete Kante (a_1, r),(b_1, a_1) und (s, b_1) eingefügt wird. Die Kapazitäten der neuen Kanten werden jeweils auf 1 gesetzt. Die Vorzeichen der Kosten werden gespiegelt, so dass die Kosten für den Transport einer Einheit auf der Kante (b_1, a_1) nun $-1,5$ betragen (vgl. Abbildung 4.9). Die Bestimmung eines kürzesten Pfads auf $N'(1)$ liefert den Weg (r, a_3),(a_3, b_1), (b_1, a_1), (a_1, b_2), (b_2, s) (vgl. Abbildung 4.10, ausführliche Herleitung in Beispiel 8). Auf den ausgewählten Vorwärtskanten wird die Flussvergrößerung durchgeführt, indem die Flusstärke der Kanten jeweils um 1 erhöht wird. Bei der

Kante (b_1, a_1) handelt es sich hingegen um eine Rückwärtskante. Daher wird in $N(1)$ die Flussstärke der Kante (a_1, b_1) um 1 reduziert. Durch die Flussvergrößerung entstehen zusätzliche Kosten in Höhe von 0 + 2 - 1,5 + 2 + 0 = 2,5 und es ergeben sich Gesamtkosten von 4 für den Fluss der Stärke $\phi = 2$ auf $N(2)$. An dieser Stelle zeigt sich, dass bereits getroffene Entscheidungen im Laufe des Verfahrens wieder revidiert werden, um eine global optimale Lösung zu erhalten. Der berechnete Fluss auf $N(1)$ war zwar für die Flussstärke 1 optimal, muss aber kein Teil der Lösung für $\phi = 2$ sein.

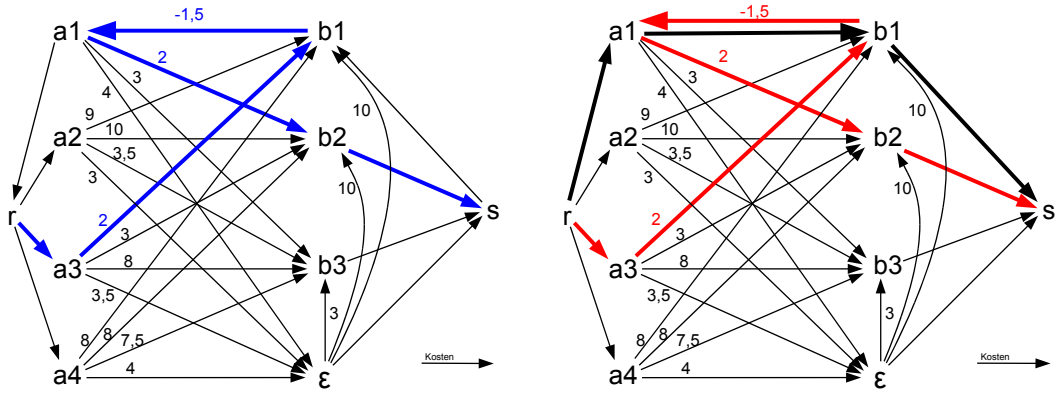

Abbildung 4.10: Links: Kürzester Pfad auf $N'(1)$, Rechts: Flussvergrößerung auf $N(1)$

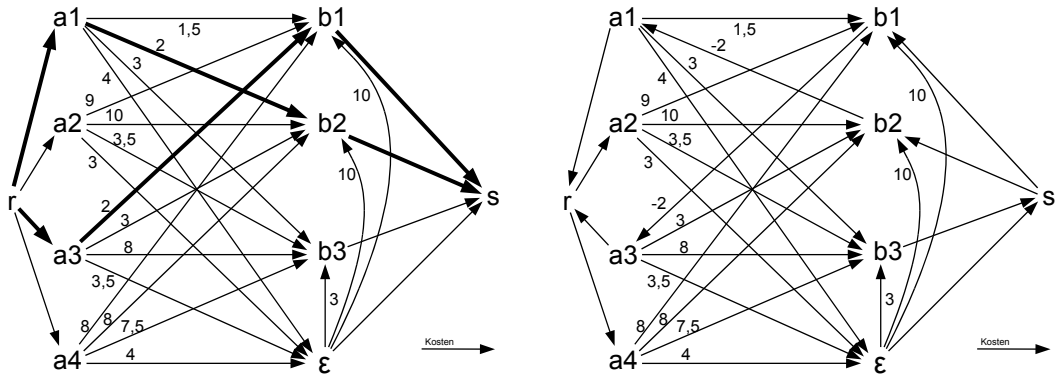

Abbildung 4.11: Links: $N(2)$: Netzwerk N mit 2-Fluss
Rechts: $N'(2)$: Inkrementnetzwerk zu $N(2)$

Schritt 3: $N(2) \rightarrow N(3)$

Abbildung 4.11 zeigt das Netzwerk mit $\phi = 2$ und das dazugehörige Inkrementnetzwerk $N'(2)$. Die Kante (a_1, b_1) ist wieder in $N'(2)$ enthalten, da sie aufgrund der Flussstärke 0 erneut als Vorwärtskante für einen flussvergrößernden kostenminimalen Weg zur Verfügung steht. Als kürzester Weg auf $N'(2)$ wird die Kantenfolge (r, a_2), (a_2, ϵ), (ϵ, s) bestimmt (vgl. Abbildung 4.12). Bei der anschließenden Flussvergrößerung auf $N(2)$ entstehen Kosten in Höhe von 3, womit sich Gesamtkosten in Höhe von 7 für den Fluss der Stärke $\phi = 3$ auf dem Netzwerk $N(3)$ ergeben.

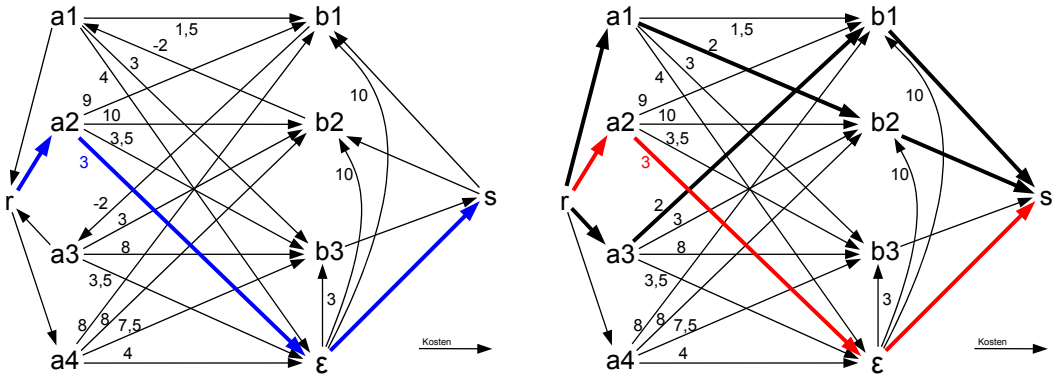

Abbildung 4.12: Links: Kürzester Pfad auf $N'(2)$
Rechts: Flussvergrößerung auf $N(2)$

Schritt 4: $N(3) \rightarrow N(4)$
Das Netzwerk $N(3)$ sowie das zugehörige Inkrementnetzwerk $N'(3)$ sind in Abbildung 4.13 dargestellt. Als kürzester Weg werden die Kanten (r, a_4), (a_4, ϵ), (ϵ, b_3), (b_3, s) ermittelt (vgl. Abbildung 4.14). Bei der Flussvergrößerung entstehen Kosten in Höhe von 7, womit sich die Gesamtkosten für den Fluss der Stärke $\phi = 4$ auf 14 erhöhen. Die Zielflussstärke ist damit erreicht. Das Netzwerk $N(4)$, dargestellt in Abbildung 4.15, enthält einen Fluss mit minimalen Kosten in Höhe von 14.

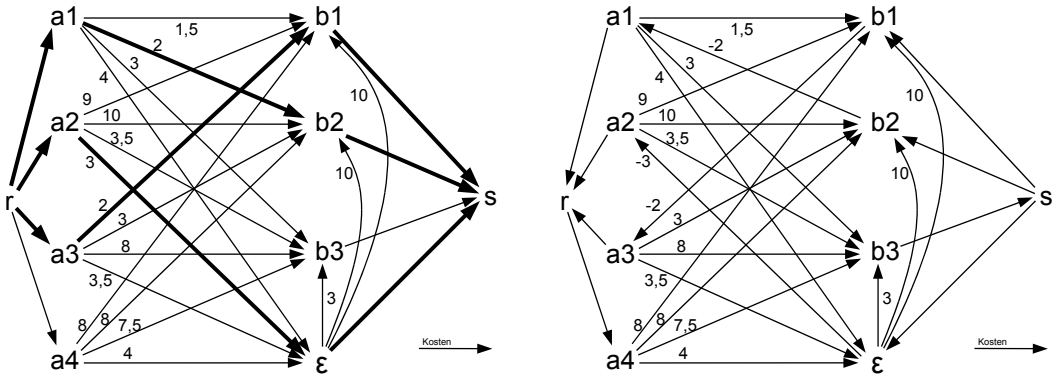

Abbildung 4.13: Links: Netzwerk $N(3)$ mit 3-Fluss
Rechts: Inkrementnetzwerk $N'(3)$

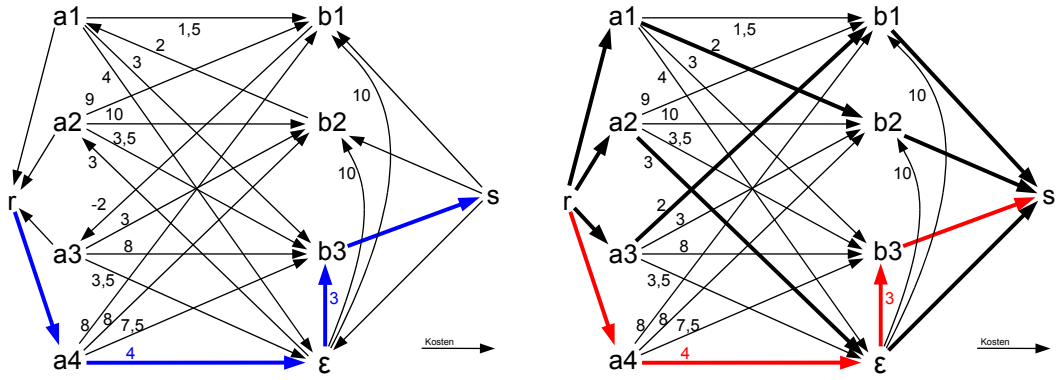

Abbildung 4.14: Links: Kürzester Pfad auf $N'(3)$
Rechts: Flussvergrößerung auf $N(3)$

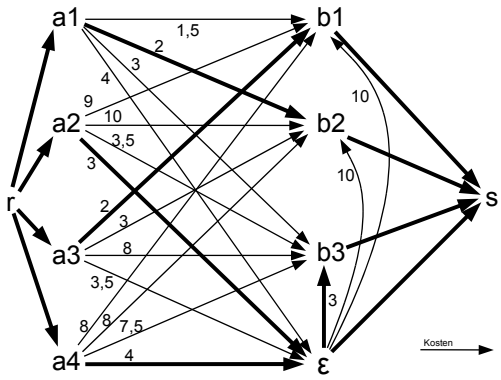

Abbildung 4.15: $N(4)$: Netzwerk mit 4-Fluss

Beispiel 8 (Kürzeste-Wege-Berechnung)**.** Zur Veranschaulichung des Label-Correcting-Verfahrens wird beschrieben, wie der kürzeste Pfad auf dem Inkrementnetzwerk $N'(1)$ aus Abbildung 4.9 berechnet wird. Abbildung 4.17 zeigt die Wertetabellen der im Folgenden beschriebenen Schritte.

Nach der Initialisierung in *Schritt 1*, wird in *Schritt 2* Knoten r bearbeitet. Ausgehend von r sind die Knoten a_2 bis a_4 erreichbar. Da die von r ausgehenden Kanten nicht mit Kosten versehen sind, werden die Abstände $d_{a2}, \ldots, d_{a4}$ auf 0 gesetzt. Als Vorgängerknoten auf dem bislang kürzesten Weg wird bei allen drei Knoten jeweils r eingetragen. Die erreichten Knoten werden am Ende der Schlange Q eingefügt. Im *3.Schritt* wird a_2 aus der Schlange entfernt. Über den Knoten a_2 können die Knoten b_1 bis b_3 sowie ϵ erreicht werden. Als Vorgängerknoten aller vier Knoten wird a_2 eingetragen. Außerdem werden jeweils die Abstände zu r berechnet. Der Wert d für den Knoten b_i ergibt sich aus der Summe $d_{a_2} + c_{a_2 b_i}$. In *Schritt 4* werden die Nachfolger von Knoten a_3 betrachtet. a_3 besitzt 4 Nachfolgerknoten. Da die Wege von r über a_3 nach b_3 oder ϵ nicht kürzer als die bereits gefundenen Wege von r über a_2 nach b_3 und ϵ sind, werden nur die kürzesten Wege nach b_1 und b_2 aktualisiert. Da sowohl b_1 als auch b_2 bereits in der Schlange enthalten sind, werden sie nicht erneut eingefügt. Aus der Bearbeitung der Nachfolger von a_4 in *Schritt 5* ergeben sich keine Verbesserungen der bislang gefundenen Wege. In *Schritt 6* werden die Nachfolger von Knoten b_1 betrachtet. Über die Kante (b_1, a_1) wird nun erstmals Knoten a_1 erreicht. Die Berechnung des Weges von r über b_1 nach a_1 ergibt $2 - 1,5 = 0,5$. Bei der Bearbeitung von b_2 in *Schritt 7* wird schließlich auch ein Weg zur Senke s ermittelt. Bei der anschließenden Bearbeitung von b_3 und ϵ in den *Schritten 8 und 9* ergeben sich keine Verbesserungen. In *Schritt 10* kann der Weg von r nach b_2 über a_1 verkürzt werden. Dadurch wird b_2 erneut in die Schlange eingefügt. In den *Schritten 11 und 12* werden die beiden letzten Elemente s und b_2 abgearbeitet, ohne dass sich die bereits ermittelten Werte verändern.

Der kürzeste Weg von r nach s besitzt die Länge 3 und kann aus der Tabelle des letzten Schritts ausgelesen werden (vgl. Abbildung 4.16). Er ergibt sich aus der Verknüpfung der Vorgängerbeziehungen. $s \leftarrow b_2 \leftarrow a_1 \leftarrow b_1 \leftarrow a_3 \leftarrow r$.

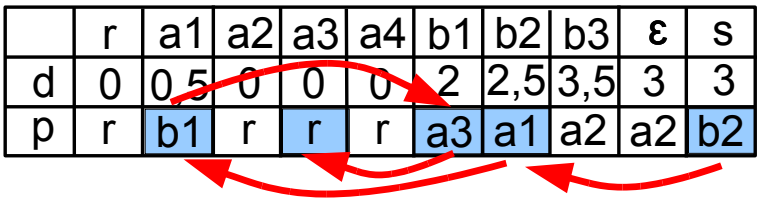

Abbildung 4.16: Ablesen des kürzesten Pfades

Der beschriebene Label-Correcting-Algorithmus berechnet die kürzesten Verbindungen von der Quelle r des Netzwerks zu allen anderen Knoten. Dabei wird ein spannender Baum mit Wurzel r konstruiert, der diejenigen Kanten enthält, welche die kürzesten Wege bilden (vgl. Abbildung 4.18). Die Suche nach dem kürzesten Pfad ist jedoch nicht äquivalent zu dem Aufbau eines minimalspannenden Baums. Ein minimalspannender

1. Initialisierung: Q={r}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0									
p	r									

2. Bearbeitung von r: Q={a2,a3,a4}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0		0	0	0					
p	r		r	r	r					

3. Bearbeitung von a2: Q={a3,a4,b1,b2,b3,ε}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0		0	0	0	9	10	3,5	3	
p	r		r	r	r	a2	a2	a2	a2	

4. Bearbeitung von a3: Q={a4,b1,b2,b3,ε}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0		0	0	0	2	3	3,5	3	
p	r		r	r	r	a3	a3	a2	a2	

5. Bearbeitung von a4: Q={b1,b2,b3,ε}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0		0	0	0	2	3	3,5	3	
p	r		r	r	r	a3	a3	a2	a2	

6. Bearbeitung von b1: Q={b2,b3,ε,a1}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0	0,5	0	0	0	2	3	3,5	3	
p	r	b1	r	r	r	a3	a3	a2	a2	

7. Bearbeitung von b2: Q={b3,ε,a1,s}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0	0,5	0	0	0	2	3	3,5	3	3
p	r	b1	r	r	r	a3	a3	a2	a2	b2

8. Bearbeitung von b3: Q={ε,a1,s}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0	0,5	0	0	0	2	3	3,5	3	3
p	r	b1	r	r	r	a3	a3	a2	a2	b2

9. Bearbeitung von ε: Q={a1,s}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0	0,5	0	0	0	2	3	3,5	3	3
p	r	b1	r	r	r	a3	a3	a2	a2	b2

10. Bearbeitung von a1: Q={s,b2}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0	0,5	0	0	0	2	2,5	3,5	3	3
p	r	b1	r	r	r	a3	a1	a2	a2	b2

11. Bearbeitung von s: Q={b2}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0	0,5	0	0	0	2	2,5	3,5	3	3
p	r	b1	r	r	r	a3	a1	a2	a2	b2

12. Bearbeitung von b2: Q={}

	r	a1	a2	a3	a4	b1	b2	b3	ε	s
d	0	0,5	0	0	0	2	2,5	3,5	3	3
p	r	b1	r	r	r	a3	a1	a2	a2	b2

Abbildung 4.17: Markierungen des Kürzeste-Wege-Verfahrens. Die Änderungen des aktuellen Schritts sind gelb hinterlegt. Die Menge Q bezeichnet den Inhalt der Schlange.

Baum minimiert die Summe der Kosten aller in dem Baum enthaltenen Kanten. Bei der Suche nach dem kürzesten Pfad wird hingegen für jeden Knoten i einzeln der kürzeste Weg von r nach i gesucht.

Aufwand des Busacker-Gowen-Verfahrens Der Aufwand des Busacker-Gowen-Verfahrens lässt sich für einen Netzwerkgraphen mit $|V|$ Knoten und $|E|$ Kanten wie folgt abschätzen: Im Falle ganzzahliger Maximalkapazitäten werden maximal ω^* Iterationsschritte durchgeführt. Jeder Iterationsschritt besteht höchstens aus jeweils einem Aufruf der Prozeduren A und B. Die Prozedur A besitzt die gleiche Laufzeitkomplexität wie das Kürzeste-Wege-Verfahren LC-Algorithmus A aus [NM02], das mit $O(|V||E|)$ angegeben wird. Es fallen maximal $|V| - 1$ Iterationsschritte mit Aufwand $O(|E|)$ an, da in jedem Iterationsschritt jede Kante des Netzwerks höchstens einmal mit konstantem Aufwand bearbeitet wird sowie der längste Weg in einem Netzwerk mit $|V|$ Knoten auf $|V| - 1$ Kanten beschränkt ist. Der Aufwand der Flussvergrößerung in Prozedur B ist von der Zahl der Kanten abhängig, da jede Kante maximal einmal mit konstantem Aufwand bearbeitet wird, und beträgt $O(|E|)$. Insgesamt liegt die Laufzeitkomplexität des Busacker-Gowen-Algorithmus daher für allgemeine Netzwerkprobleme in $O(|V||E|\omega^*)$. Da die Flussstärke ω^* im allgemeinen Fall nicht von der Zahl der Kanten und Knoten im Netzwerk abhängt, handelt es sich bei dem Busacker-Gowen-Verfahren um ein pseudopolynomielles Verfahren [NM02]. Bei Netzwerkgraphen, die nach obiger Vorschrift konstruiert werden, lässt sich die Komplexität jedoch als Funktion von n ausdrücken, wobei n die Anzahl der Knoten auf der linken Seite des bipartiten Graphen bezeichnet: $\omega^* = n$, $|V| \leq 2n + 3$, und $|E| \leq n^2 + 4n$. Damit ergibt sich eine polynomielle Laufzeit mit $O(n^4)$ als obere Schranke.

Ablesen der Korrespondenzen aus dem Vergleichsgraphen Nach Berechnung des maximalen kostenminimalen Netzwerkflusses wird aus der Lösung des Optimierungsproblems eine zulässige Zuordnung M abgelesen. In Hinblick auf die aktiven inneren Kanten werden für jedes Modellelement a_i aus A und b_j aus B die folgenden Fälle unterschieden:

1. Die Kante (a_i, b_j) ist aktiv (siehe Abbildung 4.19)
   Die Elemente a_i und b_j korrespondieren, es gilt $(a_i, b_j) \in M$. Die Gesamtkosten der Zuordnung erhöhen sich um $c(a_i, b_j)$.

2. Die Kante (a_i, ϵ) ist aktiv.
   Das Element a_i aus Modell A ist ein Einzelelement. Die Gesamtkosten erhöhen sich um $c(a_i, \epsilon)$. Das Element a_i wird bei der folgenden Ableitung der Differenzen als gelöschtes Element interpretiert.

3. Die Kante (ϵ, b_j) ist aktiv. Das Element b_j aus Modell B ist ein Einzelelement. Die Gesamtkosten erhöhen sich um $c(\epsilon, b_j)$. Das Element b_j wird bei der folgenden Ableitung der Differenzen als eingefügtes Element interpretiert.

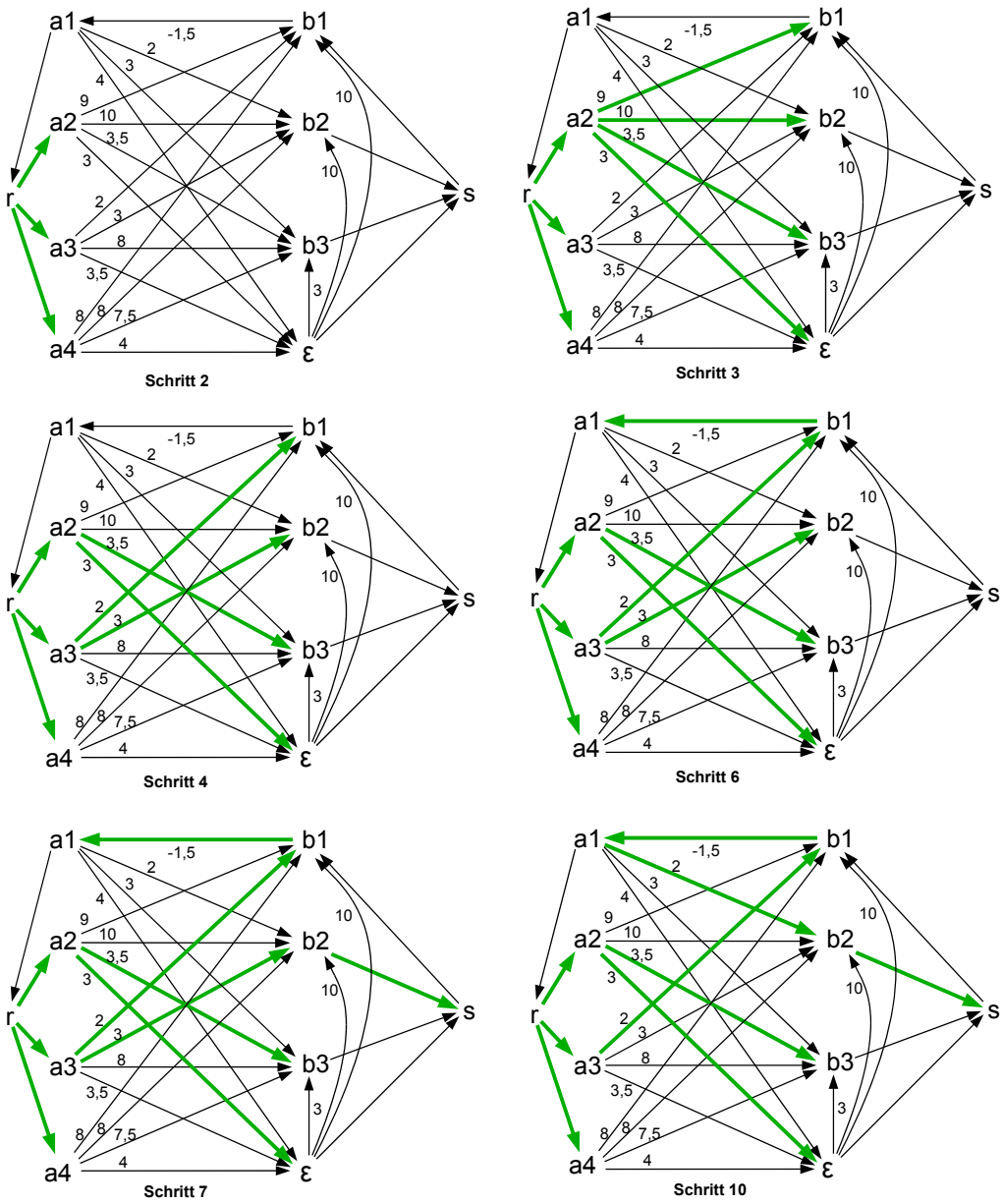

Abbildung 4.18: Zwischenschritte des Kürzeste-Wege-Verfahrens zu Beispiel 8: Die grün markierten Kanten bilden jeweils die bislang gefundenen kürzesten Wege.

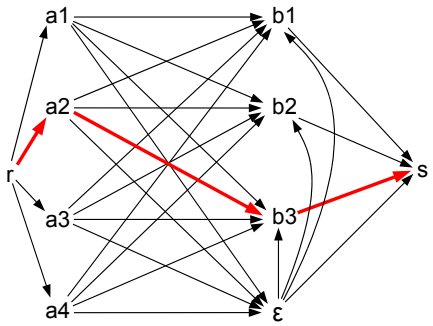

Abbildung 4.19: Auslesen der Korrespondenzen aus dem Netzwerkfluss: Die Modellelemente a_2 und b_3 korrespondieren.

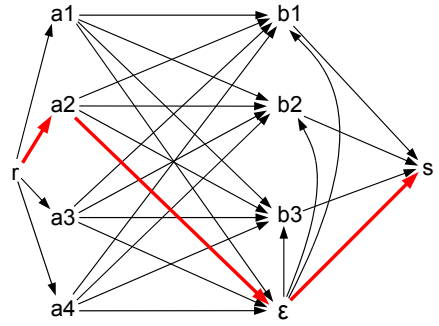

Abbildung 4.20: Auslesen der Korrespondenzen aus dem Netzwerkfluss: a_2 wird gelöscht.

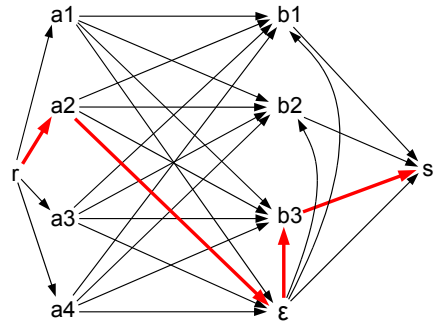

Abbildung 4.21: Auslesen der Korrespondenzen aus dem Netzwerkfluss: a_2 wird gelöscht, b_3 wird erzeugt.

Aufgrund der angegebenen Minimal- und Maximalkapazitäten wird sichergestellt, dass jedes Modellelement genau zu einer inneren aktiven Kante inzident ist. Daher ist die resultierende Zuordnung M eindeutig. Die Flussstärke $\omega^* = n$ stellt sicher, dass für jedes Element entschieden wird, ob es sich um ein Korrespondenzelement oder ein Einzelelement handelt. Dadurch werden alle Kosten der Zuordnung berücksichtigt. Bei der Konstruktion des Netzwerkgraphen werden die Elemente des größeren Modells auf der linken Seite des bipartiten Graphen angeordnet. Die Kapazität der Kante $(\epsilon, s) = n - m$ verursacht, dass $n - m$ Elemente aus Modell A gelöscht werden (siehe Abbildung 4.20). Werden in Modell A mehr als $n - m$ Einzelelemente identifiziert, muss für jedes weitere Element auch ein Element aus B erzeugt werden (siehe Abbildung 4.21), da nach Auslastung der Kante (ϵ, s) jede weitere Einheit über eine Kante (ϵ, b_j) transportiert werden muss. Weitere Details zur technischen Umsetzung folgen in Abschnitt 5.3.2, nachdem das verwendete Korrespondenzmodell eingeführt wurde.

Nachdem der Aufbau eines Vergleichsgraphen, die Lösung des Netzwerkflussproblems und das Ablesen der Zuordnung auf allgemeiner Ebene behandelt wurden, wird im nächsten Abschnitt darauf eingegangen, welche Untergraphen für den Vergleichsgraphen im Fall von Klassendiagrammen erstellt werden.

Die Untergraphen des Vergleichsgraphen Für den Vergleichsgraphen auf Klassenebene müssen die abgeschätzten Kosten für die inneren Kanten ermittelt werden. Die Kosten für das Löschen und Erzeugen von Klassen können direkt abgeschätzt und an die entsprechenden Kanten $a_i \to \epsilon$ bzw. $\epsilon \to b_j$ als Bewertung eingetragen werden. Die Kostenbewertungen der Kanten $a_i \to b_j$ werden aufwändiger ermittelt: Für jede Kante, die einer Zuordnung zweier Klassen entspricht, werden Untergraphen für die Zuordnungen der Attribute, Methoden und Assoziationen erstellt (siehe Abbildung 4.22). Für die Assoziationen werden getrennte Untergraphen für eingehende und ausgehende Assoziationen gebildet, da jeweils nur gleichgerichtete Assoziationen korrespondieren können. Beim Vergleich zweier Assoziationen werden für die abgeschätzten Kosten nur die Eigenschaften der Assoziationen (Name, Multiplizitäten etc.) verglichen, da nicht bekannt ist, auf welche Klassen die Enden der Assoziationen zugeordnet werden. Die Kostenbewertung der Verbingungskante zweier Klassenknoten ergibt sich dann aus der Summe der Kosten der Zuordnungen der Unterelemente der Klassen und den Kosten für die Zuordnung der Klassen selbst. Letztere beinhalten die Kosten für die Änderung von Klasseneigenschaften, wie z. B. den Namen, sowie die abgeschätzten Kosten für die Zuordnung von Vererbungsbeziehungen. Da eine Vererbungskante außer Quelle und Ziel keine weiteren Eigenschaften besitzt, wird lediglich die Anzahl der Vererbungskanten jeder Klasse miteinander verglichen. Daraus ergibt sich die Abschätzung, wieviele Vererbungskanten mindestens gelöscht oder erzeugt werden müssen. Nachdem die Unterprobleme mit dem Busacker-Gowen-Verfahren gelöst sind, stehen die Kosten der Kanten auf Klassenebene fest, so dass dann auch die Zuordnung auf Klassenebene mit dem Busacker-Gowen-Verfahren bestimmt werden kann.

Dass nicht für jedes Paar von Klassen alle Untergraphen erstellt werden müssen, demonstriert das folgende Beispiel.

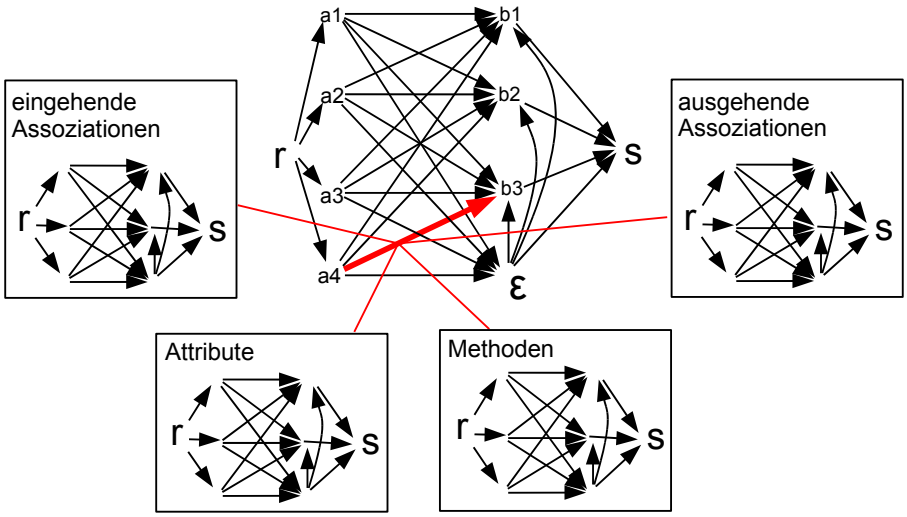

Abbildung 4.22: Unterprobleme einer Korrespondenzkante

Beispiel 9 (Korrespondenzberechnung auf Klassendiagrammen)**.** Für die Modelle A (Abbildung 4.4) und B (Abbildung 4.5) des Bestellsystems aus Beispiel 6 soll nun eine Zuordnung aller Elemente bestimmt werden. Da das Modell A weniger Klassen als das Modell B besitzt, wird der Graph mit vertauschten Rollen von A und B konstruiert. Der Vergleichsgraph auf Klassenebene entspricht dem in Abbildung 4.6, wobei die Modellelemente $a_1, \ldots, a_4$ auf der linken Seite des bipartiten Graphen den Klassen `PurchaseOrder`, `Address`, `Item` und `USAddress` des Modells B entsprechen. Die Modellelemente $b_1, \ldots, b_3$ entsprechen den Klassen `PurchaseOrder`, `USAddress` und `Item` des Modells A. Die durch das Verfahren berechnete Zuordnung der Klassen und ihrer Unterelemente lässt sich aus Listing 4.1 entnehmen, das eine Auflistung aller aktiven Kanten des Vergleichsgraphen und dessen Untergraphen mit zugehörigen Kosten enthält. Aus dem Listing ist auch ablesbar, welche Untergraphen jeweils erstellt wurden. Für den Vergleich der Klassen `PurchaseOrder` und `PurchaseOrder` wurden Untergraphen für ausgehende Assoziationen sowie für Attribute erstellt. Die Untergraphen für eingehende Assoziationen und Methoden werden für den Vergleich dieser beiden Klassen nicht benötigt, da beide Klassen weder eingehende Assoziationen noch Methoden besitzen.

Listing 4.1: Aktive Kanten des Vergleichsgraphen zu Beispiel 9. Im Gegensatz zu Beispiel 9 wurde für den Zeichenkettenvergleich hier nicht das LCS-Verfahren, sondern das Levenshtein-Verfahren angewandt.

```
Gesamtkosten (ECVerfahren (Levenshtein)): 8.222222222222221
Matching:
  PurchaseOrder - PurchaseOrder (0.0)
    Out_Referenzen (0.0)
      Matching:
        shipTo - shipTo (0.0)
        billTo - billTo (0.0)
```

```
      items - items    ( 0.0 )
  Attribut ( 0.0 )
    Matching:
      comment - comment    ( 0.0 )
      orderDate - orderDate    ( 0.0 )
Address - USAddress    ( 3.2222222222222223 )
  In_Referenzen ( 0.0 )
    Matching:
      shipTo - shipTo    ( 0.0 )
      billTo - billTo    ( 0.0 )
  Attribut ( 2.0 )
    Matching:
      name - name    ( 0.0 )
      street - street    ( 0.0 )
      city - city    ( 0.0 )
      e - state    ( 2.0 )
      zip - zip    ( 0.0 )
      country - country    ( 0.0 )
Item - Item    ( 0.0 )
  In_Referenzen ( 0.0 )
    Matching:
      items - items    ( 0.0 )
  Attribut ( 0.0 )
    Matching:
      productName - productName    ( 0.0 )
      quantity - quantity    ( 0.0 )
      USPrice - USPrice    ( 0.0 )
      comment - comment    ( 0.0 )
      shipDate - shipDate    ( 0.0 )
      partNum - partNum    ( 0.0 )
USAddress - e    ( 5.0 )
  Attribut ( 2.0 )
    Matching:
      state - e    ( 2.0 )
```

Aus der Zuordnung lässt sich ablesen, dass die Klasse `USAddress` aus Modell B neu eingefügt wurde. Dabei entstehen geschätzte Kosten in Höhe von 5, von denen 2 auf das Einfügen der Klasse selbst, 1 auf das Einfügen der Vererbungskante und 2 auf das Einfügen des Attributs `state` zurückzuführen sind. Die Klasse `USAddress` wurde in `Address` umbenannt (Kosten in Höhe von 0,2222222222222223) und das Attribut `state` wurde entfernt (Kosten in Höhe von 2). Damit ergeben sich Gesamtkosten in Höhe von 8,2222222222222223 mit einem Anteil an kontextunabhängigen Kosten in Höhe von 6,2222222222222223. Abbildung 4.23 zeigt das Ergebnis des Vergleichs, wie es im EMF Compare UI Editor des Rahmenwerks angezeigt wird. Die verglichenen Modelle werden

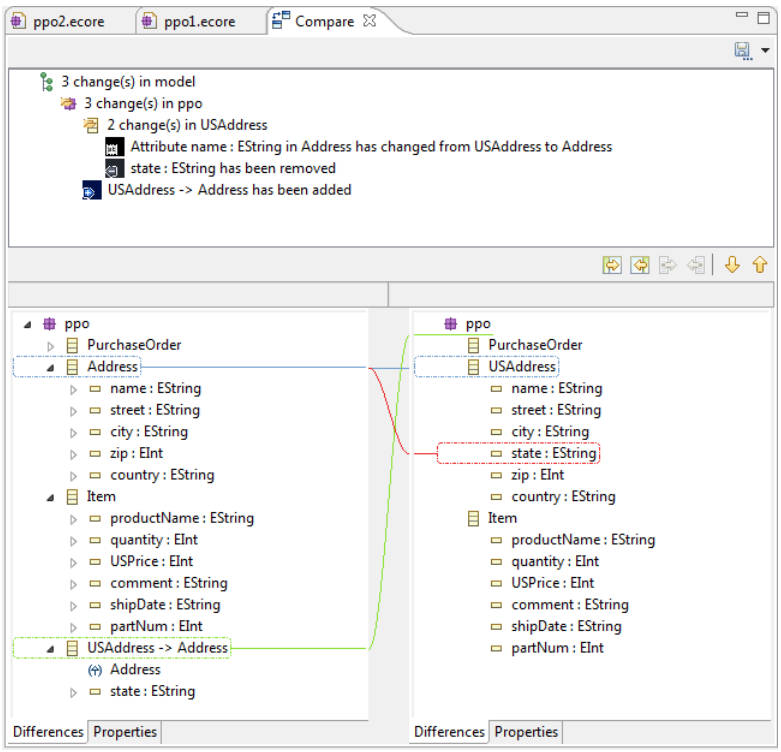

Abbildung 4.23: Ergebnis des ECVerfahrens beim Vergleich der Modelle aus den Abbildungen 4.4 und 4.5

im unteren Teil des Fensters in einer Baumsicht gegenübergestellt, wobei erzeugte Elemente grün, gelöschte Elemente rot und geänderte Elemente blau markiert sind. In der Anzeige von EMF Compare gelten Unterelemente von gelöschten Elementen ebenfalls als gelöscht, werden aber nicht mehr extra mit einer roten Markierung als gelöscht gekennzeichnet. Analog gelten auch Unterelemente von erzeugten Elementen als ebenfalls einfügt, obwohl sie nicht grün markiert werden. Der obere Teil des Fensters enthält eine textuelle Beschreibung der Unterschiede. Für eine genauere Beschreibung dieser Sicht wird auf Abschnitt 5.4.3 verwiesen.

Repräsentierung des Vergleichsgraphen in der Implementierung

Der Vergleichsgraph wird in der Implementierung durch eine Instanz der Klasse `Graph` repräsentiert (siehe Abbildung 4.24). Eine erste Version des Vergleichsgraphen wurde von Diana Rausch im Rahmen ihrer Zulassungsarbeit [Rau08] entwickelt. Das Attribut `typ` beschreibt die Ebene des Graphen, z. B. Klassenebene oder Attributebene. Der boolesche Wert `istGedreht` gibt an, ob der Graph in Bezug auf seinen Vatergraphen gedreht ist. Die Gesamtkosten, die sich nach Lösung des Netzwerkflussproblems ergeben, werden im Attribut `kosten` gespeichert. Ein Graph besitzt eine Liste von Knoten, die über ihre `id` charakterisiert werden. Die `id`s entsprechen den Knotennummern im dargestellten Busacker-Gowen-Verfahren. Positive Knotennummern gewährleisten die Unterscheidbarkeit der Markierungen i und $-i$ in Algorithmus 1, Prozedur A. Mit Ausnahme

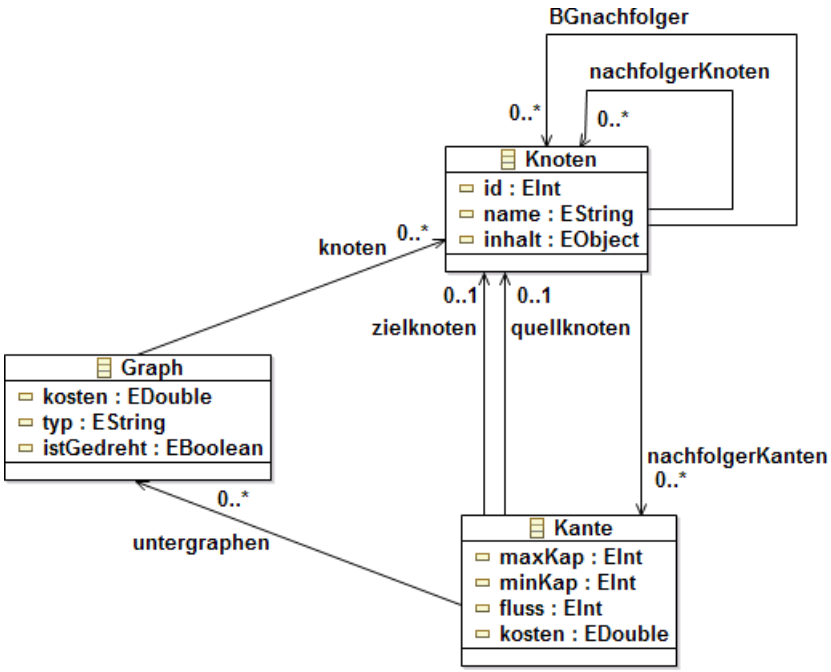

Abbildung 4.24: Modell eines Vergleichsgraphen

der Grundknoten für die Quelle, die Senke sowie den Epsilon-Knoten repräsentieren die Knoten jeweils ein Diagrammelement, auf das über die Referenz `inhalt` verwiesen wird. Um den Graphen in Flussrichtung von der Senke zur Quelle traversieren zu können, besitzt jeder Knoten eine Liste der Kanten, die von dem Knoten ausgehen. Die Kante selbst kennt ihren Quell- und Zielknoten und besitzt eine Reihe von Eigenschaften, die für das Lösen des Netzwerkflussproblems benötigt werden: Die minimale und maximale Kapazität, die Anzahl transportierter Einheiten und die Kosten, die je transportierter Einheit anfallen. Um den hierarchischen Aufbau der Vergleichsgraphen zu modellieren, kann einer Kante Untergraphen angeheftet werden, die wiederum Instanzen der Klasse `Graph` darstellen. Für die Durchführung des Busacker-Gowen-Verfahrens mit dem Label-Correcting-Verfahren zur Berechnung der kürzesten Wege werden für jeden Knoten noch zwei weitere Listen benötigt: Die unveränderliche Menge der Nachfolgerknoten im Netzwerkgraphen ($S(i)$) und die variierende Menge der markierbaren Nachfolgerknoten ($M(i)$) im Inkrementnetzwerk des Busacker-Gowen-Verfahrens.

4.2.6 Zusammenhang des relaxierten und des ursprünglichen Optimierungsproblems

Die Lösung des relaxierten Problems (Optimierungsproblem 2) stellt in jedem Fall eine zulässige Lösung für das Originalproblem (Optimierungsproblem 1) dar. Sei M die Zuordnung, die durch die optimale Lösung des relaxierten Problems mit den abgeschätzten kontextabhängigen Kosten ermittelt wurde. Da nun bekannt ist, welche Klassen korrespondieren, können die Kosten dieser Zuordnung exakt berechnet werden $C(M)^*_{real}$.

Da die Kosten durch eine untere Schranke abgeschätzt wurden, folgt aus Gleichung 4.1 (S. 80) und Ungleichung 4.10 (S. 89) sicher Ungleichung 4.15.

$$C(M)^*_{real} \geq C(M)^*_{lb} \tag{4.15}$$

Falls die tatsächlichen Kosten der Zuordnung mit den abgeschätzten Kosten der Zuordnung übereinstimmen, ist die optimale Lösung des relaxierten Optimierungsproblems auch eine optimale Lösung für das ursprüngliche Problem.

$$C(M)^*_{real} = C(M)^*_{lb} \tag{4.16}$$

Die Gleichung 4.16 liefert ein hinreichendes Optimalitätskriterium. Dieses Kriterium ist jedoch nicht notwendig, da die tatsächlichen Kosten der berechneten optimalen Lösung für das relaxierte Problem höher als die berechneten Mindestkosten sein können. In diesem Fall (Ungleichung 4.17) ist nicht bekannt, ob die ermittelte Lösung für das ursprüngliche Problem optimal ist.

$$C(M)^*_{real} > C(M)^*_{lb} \tag{4.17}$$

Für kleine Beispiele kann die Optimalität der Ergebnisse mit Hilfe der Brute-Force-Implementierung überprüft werden. Auf die Fragen, wie oft mit dem Verfahren die optimalen Kosten erreicht werden und wie weit die Kosten im Fall einer nicht optimalen Lösung von den optimalen Kosten abweichen, wird im Rahmen der Evaluierung in Kapitel 6 eingegangen.

Beispiel 10 (Tatsächliche Kosten des Beispiels 9)**.** Listing 4.2 zeigt die berechneten tatsächlichen Kosten der Zuordnung, die in Beispiel 9 mit dem ECVerfahren bestimmt wurde. Das mehrfache Auftreten der Untergraphen vom Typ `Out_Referenzen` ist darauf zurückzuführen, dass Assoziationen nur dann korrespondieren können, falls beide Assoziationsenden identisch sind. Da die Klasse `PurchaseOrder` ausgehende Assoziationen zu zwei verschiedenen Unterklassen besitzt, werden auch zwei Unterprobleme gelöst. Die tatsächlichen Kosten der gefundenen Lösung sind mit den geschätzten Kosten identisch. Das hinreichende Optimalitätskriterium ist in diesem Fall erfüllt.

Listing 4.2: Aktive Kanten des Vergleichsgraphens zu Beispiel 9

```
Gesamtkosten (Tatsächliche Kosten): 8.222222222222221
Matching:
  PurchaseOrder - PurchaseOrder (0.0)
    Attribut (0.0)
      Matching:
        comment - comment (0.0)
        orderDate - orderDate (0.0)
    Out_Referenzen (0.0)
      Matching:
        items - items (0.0)
    Out_Referenzen (0.0)
```

```
    Matching:
      shipTo - shipTo    ( 0.0 )
      billTo - billTo    ( 0.0 )
Address - USAddress    ( 3.2222222222222223 )
  In_Referenzen ( 0.0 )
    Matching:
      shipTo - shipTo    ( 0.0 )
      billTo - billTo    ( 0.0 )
  Attribut ( 2.0 )
    Matching:
      name - name    ( 0.0 )
      street - street    ( 0.0 )
      city - city    ( 0.0 )
      e - state    ( 2.0 )
      zip - zip    ( 0.0 )
      country - country    ( 0.0 )
Item - Item    ( 0.0 )
  In_Referenzen ( 0.0 )
    Matching:
      items - items    ( 0.0 )
  Attribut ( 0.0 )
    Matching:
      productName - productName    ( 0.0 )
      quantity - quantity    ( 0.0 )
      USPrice - USPrice    ( 0.0 )
      comment - comment    ( 0.0 )
      shipDate - shipDate    ( 0.0 )
      partNum - partNum    ( 0.0 )
USAddress - e    ( 5.0 )
  Attribut ( 2.0 )
    Matching:
      state - e    ( 2.0 )
```

Wahl des Kürzeste-Wege-Verfahrens Das Verfahren von Busacker und Gowen mit dem darin eingebundenen Label-Correcting-Verfahren A wurde von Diana Rausch im Rahmen ihrer Zulassungsarbeit [Rau08] implementiert. Das gewählte Verfahren zur Berechnung der kürzesten Wege kann durch ein anderes geeignetes Kürzeste-Wege-Verfahren, z. B. das Verfahren von Dijkstra [Dij59] ersetzt werden. Dieses Verfahren setzt in seiner ursprünglichen Form voraus, dass alle Kantenbewertungen des Netzwerkes nichtnegativ sind. Dies ist zwar im Inkrementnetzwerk $N(0)$ erfüllt, in den nachfolgenden Inkrementnetzwerken sind jedoch die Rückwärtskanten mit negativen Kosten versehen. In [Gal87] bietet eine abgewandelte Form des Verfahrens eine Lösung für dieses Problem: Das Inkrementnetzwerk wird vor der Kürzeste-Wege-Bestimmung jeweils in

ein Netzwerk mit ausschließlich positiven Kantenbewertungen transformiert. Mit diesem Dijkstra-Verfahren ließe sich bei unserem Netzwerkgraphen der Aufwand für das Kürzeste-Wege-Verfahren im Vergleich zum Label-Correcting-Verfahren A von $O(n^3)$ auf $O(n^2)$ reduzieren. Dies bewirkt eine Reduktion des Gesamtaufwands des Busacker-Gowen-Verfahrens auf $O(n^3)$, was eine signifikante Verbesserung darstellt.

Wahl des Lösungsverfahrens für das Netzwerkflussproblem Das Busacker-Gowen-Verfahren, das hier als Lösungsverfahren für das Optimierungsproblem 2 auf dem von uns konstruierten Netzwerkgraphen verwendet wird, kann auch durch ein anderes Verfahren ersetzt werden. Argumente für den Austausch könnten auf der einen Seite die Aussicht auf andere Ergebnisse sowie auf der anderen Seite die Erwartung kürzerer Laufzeiten sein.

Die Lösung, die vom Busacker-Gowen-Verfahren berechnet wird, ist für das Optimierungsproblem 2 optimal. Ein anderes Verfahren könnte lediglich im Falle mehrerer optimaler Lösungen zu einer anderen minimalen Lösung führen. Da diese optimalen Lösungen zwar die gleichen abgeschätzten Kosten besitzen, sich die realen Kosten der Lösungen jedoch unterscheiden können, kann sich das auf die Qualität des Ergebnisses auswirken. Ausgehend von einer optimalen Lösung des Optimierungsproblems 2 ist es leider nicht möglich, mit vertretbarem Aufwand die anderen optimalen Lösungen zu bestimmen. Dazu müssten nämlich auf dem Netzwerkgraphen mit maximaler Flussstärke alle kostenneutralen Zyklen bestimmt werden.

Inwieweit das Optimierungsproblem mit einem anderen Verfahren in der Praxis schneller gelöst werden kann, wurde noch nicht überprüft, da bei der Arbeit bislang die Qualität der berechneten Korrespondenzen im Vordergrund stand. Bislang ist mir jedoch noch kein Verfahren bekannt, das auf den angegebenen Vergleichsgraphen anwendbar ist und eine geringere Komplexität als $O(n^3)$ besitzt.

4.2.7 Einführung von Schwellenwerten

Die Zuordnung kann nicht nur über die Gestaltung der Kostenfunktion, sondern auch über zusätzliche Schwellenwerte gesteuert werden, die auf dem Netzwerkgraphen mit den abgeschätzten Kosten eingesetzt werden können. Auf diese Weise können weitere Kriterien in die Gewichtung der Kanten einfließen. Ist man zum Beispiel an einem Edierskript mit minimalen Kosten interessiert, so ist die Änderung eines Modellelements kostengünstiger, als das Quellelement zu löschen und das Zielelement einzufügen. Sind die Modellelemente jedoch nicht hinreichend ähnlich, so kann es für den Benutzer intuitiver sein, eine Zuordnung zu vermeiden, da der Benutzer die Differenz nicht (nur) auf der Ebene von Elementaroperationen bildet. Solche aus Sicht des Nutzers nicht angemessenen Zuordnungen lassen sich über eine geeignete Festlegung von Schwellenwerten ausschließen.

Dabei wurden zwei verschiedene Arten von Schwellenwerten für das `ECVerfahren` umgesetzt:

1. **Erhöhung einzelner Kantengewichte** Die Zuordnung der Elemente a und b

kann dadurch erschwert werden, dass das Kantengewicht $c(a, b)$ erhöht wird. Seien $c_{lb}(a, b)$ die abgeschätzten Kosten der Zuordnung (a, b) und $c_+(a, b) > c_{lb}(a, b)$ das erhöhte Kantengewicht. Solange $c_+(a, b) \leq c(a, \epsilon) + c(\epsilon, b)$ gilt, ist es dennoch möglich, dass die Kante von dem Lösungsverfahren als Teil der optimalen Lösung ausgewählt wird. Sobald das Löschen von a und Erzeugen von b günstiger ist als die Transformation von a in b, wird über die Kante (a, b) sicher kein Fluss transportiert werden.

2. **Entfernen einzelner Kanten aus dem Netzwerkgraphen** Soll eine Zuordnung definitiv verhindert werden, was dem Fall $c_+(a, b) > c(a, \epsilon) + c(\epsilon, b)$ entspricht, empfiehlt es sich, die Kante (a, b) einfach aus dem Netzwerkgraphen zu entfernen. Dies reduziert den Aufwand des Lösungsverfahrens für die Suche eines kostenminimalen, maximalen Flusses.

Zu beachten ist, dass sich durch beide Varianten die Gesamtkosten des kostenminimalen, maximalen Netzwerkflusses erhöhen können. Insbesondere bei dem aufwändigen `BFVerfahren` kann es sinnvoll sein, bestimmte Zuordnungen im Vorhinein auszuschließen.

Beispiel 11 (Schwellenwerte)**.** Die Abbildungen 4.25 und 4.26 zeigen zwei Versionen eines Klassendiagramms zur Modellierung einer Autovermietung. Ohne den Einsatz von Schwellenwerten werden die Klassen durch das ECVerfahren (Stringvergleich LCS) wie folgt zugeordnet: $M_1 = \{$(`Agent`, `Agent`), (`Person`, `Person`), (`Client`, `Client`), (`Vehicle`, `Vehicle`), (`Contract`, `ReservationContract`), (`EDate`, `Contract`)$\}$, $U_A = \emptyset$, $U_B = \{$`Car`, `Bus`$\}$. Dabei zeigt sich eine Stärke des Verfahrens im Gegensatz zu anderen Verfahren, die korrespondierende Elemente über Namen identifizieren: Obwohl die Namen der Klassen `Contract` aus Modell A und B identisch sind, werden sie nicht als korrespondierend identifiziert, da die strukturellen Unterschiede zu groß sind. Da das Verfahren jedoch so viele Korrespondenzen wie möglich identifiziert, um den Edierkostenabstand gering zu halten, werden die Klassen `EDate` und `Contract` als korrespondierende Klassen identifiziert, was nicht erwünscht ist. Für die Kosten gilt:
c(`EDate`, ϵ) = 2 + 3*2 (Attribute) = 8
c(ϵ, `Contract`) = 2 + 2 (Attribut) + 3*2 (Methoden) + 1 (Vererbungsbeziehung)= 11
c(`EDate`, `Contract`) = $\frac{11}{13}$ (Namensänderung) + 0,5 (Ändern von `istAbstrakt`) + $\frac{7}{9}$ (Namensänderung `year` $\rightarrow$ `resNo`) + 3*2 (Methoden) + 1 (Vererbungsbeziehung) $\approx$ 9,12
Damit ist die Korrespondenz (`EDate`, `Contract`) günstiger als das Löschen von `EDate` und Erzeugen von `Contract`. Da diese Korrespondenz für den Benutzer nicht intuitiv ist, kann durch eine Schwellenwertregel verhindert werden, dass solch unterschiedliche Klassen korrespondieren. Dies wird z. B. mit folgender Regel erreicht:

> *Falls die Namen der Klassen a und b nicht übereinstimmen und für die kontextunabhängigen Kosten* $c_{fix}(a, b) \geq min(c_{fix}(a, \epsilon), c_{fix}(\epsilon, b))$ *gilt, wird die Kante* (a, b) *aus dem Graphen entfernt.*

Da c_{fix}(EDate, Contract) ≈ 8,12 > min(8,10) wird die Kante (`EDate`, `Contract`) aus dem Netzwerkgraphen entfernt und es ergibt sich die Zuordnung $M_2 = \{$(`Agent`, `Agent`), (`Person`, `Person`), (`Client`, `Client`), (`Vehicle`, `Vehicle`), (`Contract`, `Reservation-Contract`)$\}$, $U_A = \{$`EDate`$\}$, $U_B = \{$`Car`, `Bus`, `Contract`$\}$. Der Differenzbericht des ECVerfahrens sowie die Ergebnisse des EMF Verfahrens für das gleiche Beispiel sind in Anhang A.3.1 abgebildet. Das ECVerfahren erkennt dabei auch die Umwandlung der unidirektionalen Assoziationen `hasArranged/arrangedBy` und `has/ownedBy` aus Modell A in bidirektionale Assoziationen des Modells B.

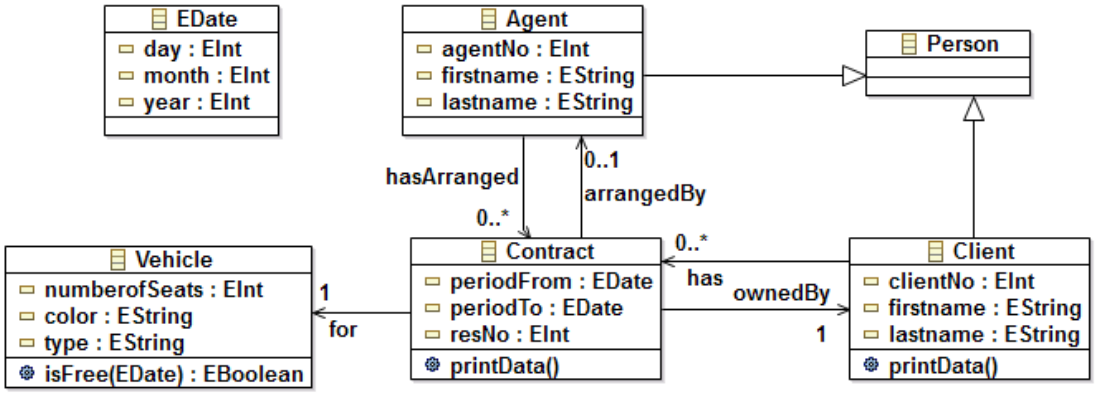

Abbildung 4.25: Modell A zu Beispiel 11

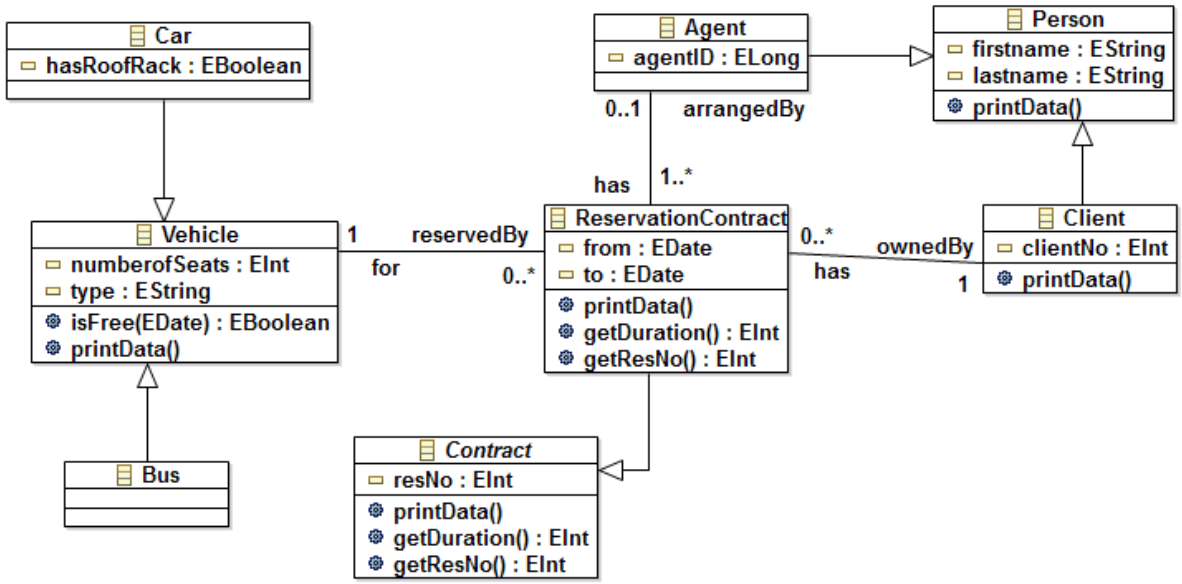

Abbildung 4.26: Modell B zu Beispiel 11

4.2.8 Zusammenfassung des ECVerfahrens

Die Schritte des ECVerfahrens zur Korrespondenzberechnung auf Klassendiagrammen lassen sich wie folgt zusammenfassen:

1. **Konstruktion des Netzwerkgraphen** N
   Für die Zuordnungsmöglichkeiten der Klassen wird ein Netzwerkgraph auf Klassenebene, wie in Abschnitt 4.2.4 beschrieben, zunächst ohne Kostenbewertungen konstruiert. Die Kostenbewertung einer Kante, die einer Zuordnung zweier Klassen entspricht, ergibt sich aus den Kosten für die Zuordnung der Klassen selbst sowie den Kosten der Zuordnung ihrer Vererbungsbeziehungen, Assoziationen, Attribute und Methoden. Dafür werden kleinere Optimierungsprobleme auf Untergraphen der Kante für die Zuordnung der Attribute, Methoden sowie für eingehende und ausgehende Assoziationen gelöst. Für die Bestimmung der Kosten der Kanten für das Einfügen bzw. Löschen von Klassen wird hingegen kein zusätzliches Optimierungsproblem benötigt, da diese direkt abgeschätzt werden können.

2. **Wahlweise Anwendung von Schwellenwerten**
   Bei Bedarf können die Kosten einzelner Kanten durch die Anwendung von Schwellenwertregeln erhöht werden (vgl. Abschnitt 4.2.7).

3. **Bestimmung eines kostenminimalen maximalen Flusses auf** N
   Mit dem Busacker-Gowen-Verfahren, das in Algorithmus 1 beschrieben ist, wird ein kostenminimaler maximaler Fluss mit Kosten C^*_{lb} bestimmt, der eine optimale Lösung des relaxierten Problems (Optimierungsproblem 2) darstellt.

4. **Ablesen der Korrespondenzen**
   Die bestimmte Zuordnung der Klassen lässt sich aus den aktiven Kanten des Netzwerkgraphen auf Klassenebene ablesen (vgl. Abschnitt 4.2.5). Die Zuordnungen der Attribute und Methoden ergeben sich analog aus den aktiven Kanten der Untergraphen der aktiven Kanten auf Klassenebene. Da nun alle Kontextinformationen vorliegen, können die Zuordnungen der Assoziationen und Vererbungsbeziehungskanten ermittelt werden. Für die Assoziationen werden dabei wieder Unterprobleme erzeugt, wobei diesmal nicht nur in eingehende und ausgehende Assoziationen sondern auch in Hinblick auf die unterschiedlichen Assoziationsenden differenziert wird. Dabei werden auch die tatsächlichen Kosten C^*_{real} berechnet.

5. **Überprüfung des hinreichenden Optimalitätskriteriums**
   Falls $C^*_{real} = C^*_{lb}$ gilt, ist die optimale Lösung des relaxierten Optimierungsproblems 2 auch eine optimale Lösung des ursprünglichen Optimierungsproblems 1 (vgl. Abschnitt 4.2.6).

4.2.9 Bewertung und Abgrenzung zu verwandten Arbeiten

In diesem Abschnitt wurde ein Verfahren zum Vergleich von Klassendiagrammen vorgestellt, das unabhängig von eindeutigen Objektbezeichnern oder eindeutigen Namen die

Korrespondenzen auf Basis eines strukturellen Vergleichs bildet. Die Berechnung der korrespondierenden Modellelemente wird dabei als Optimierungsproblem interpretiert, unter Vorgabe der Kostenbewertung der elementaren Edieroperationen eine kostenminimale Zuordnung zu bestimmen. Das Kostenmodell ist im Rahmen der Metrik-Konformität konfigurierbar.

Da die exakte Lösung des Optimierungsproblems aufgrund der Abhängigkeiten der Klassen untereinander nicht mit einem polynomiellen Verfahren berechenbar ist, wurde ein Verfahren mit abgeschätzten Kosten, das ECVerfahren, entwickelt. Dies trennt die Kosten in kontextunabhängige und kontextabhängige Kosten und verwendet für die Abschätzung der kontextabhängigen Kosten, die Zuordnung der Assoziationen und Vererbungskanten, eine untere Schranke. Damit reduziert sich die Komplexität. Das relaxierte Optimierungsproblem wird auf ein Netzwerkflussproblem übertragen, das mit einem Algorithmus aus der Graphentheorie, z. B. dem Verfahren von Busacker und Gowen, gelöst werden kann. Mit den in Abschnitt 4.2.6 vorgeschlagenen Änderungen liegt der Aufwand für die Lösung des Netzwerkflussproblems auf Klassenebene in $O(n^3)$. Für die ausgewählte Zuordnung lassen sich nun, da alle Klassenkorrespondenzen festgelegt sind, die tatsächlichen kontextabhängigen Kosten der Zuordnung ermitteln. Da das Verfahren, um die Komplexität zu reduzieren, eine optimale Lösung für ein relaxiertes Optimierungsproblem mit abgeschätzten kontextabhängigen Kosten berechnet, muss die bestimmte Lösung nicht in allen Fällen für das ursprüngliche Optimierungsproblem optimal sein. Da jedoch mit einer unteren Schranke der kontextabhängigen Kosten gearbeitet wurde, kann ein hinreichendes Optimalitätskriterium abgeleitet werden, das auch ohne großen Aufwand überprüft werden kann. Falls die tatsächlichen Gesamtkosten der Zuordnung die vorher abgeschätzten Gesamtkosten nicht übersteigen, ist garantiert, dass die Lösung des relaxierten Problems auch für das ursprüngliche Problem optimal ist. Die mögliche Definition von Schwellenwertregeln erlaubt ein nachträgliches Eingreifen in das Kostenmodell.

Die Edieroperationen des ECVerfahrens sind anders als bei den in Kapitel 2 beschriebenen Tree-To-Tree-Correction Verfahren auf Klassendiagramme abgestimmt und umfassen sowohl Änderungen an Kanten als auch Knoten. Ähnlich wie in dem in [Zha93] beschriebenen Baumvergleichsverfahren wird das Zuordnungsproblem auf ein Netzwerkflussproblem übertragen, jedoch enthält der eigene Ansatz zusätzliche Kanten zwischen dem ϵ-Knoten und den Knoten der rechten Seite des bipartiten Graphen. Von den Ansätzen von Bunke, Riesen und Neuhaus in [RNB07] und [NRB06] unterscheidet sich der eigene Ansatz nicht nur durch die andere Lösungsstrategie, sondern meines Erachtens von Grund auf durch die Verwendung eines unterschiedlichen Graph-Edit-Modells, das Kantenoperationen anders behandelt. Wie das Verfahren für den Vergleich von Architektursichten aus [AAAN+06], das jedoch auch wieder eine andere Menge an Edieroperationen betrachtet, wird das Klassendiagramm zunächst auf hierarchische Bäume abgebildet. Die Querverbindungen werden im ECVerfahren jedoch über abgeschätzte Kosten berücksichtigt. In beiden Verfahren, dem Verfahren aus [AAAN+06] ebenso wie in dem ECVerfahren, ist die exakte Zuordnung der Querverbindungen bzw. Assoziationen erst nach Zuordnung der Knoten der Baumhierarchie bzw. Klassen möglich.

Das ECVerfahren arbeitet nicht landmarkenbasiert, sondern revidiert auf der Suche

nach einer globalen Lösung gegebenenfalls vorher lokal getroffene Teilentscheidungen wieder, wodurch es sich von den Ansätzen [KWN05, XS05, BP08] abgrenzt. Das Verfahren unterscheidet sich weiterhin von allen in Kapitel 3 vorgestellten heuristischen Verfahren darin, dass ein objektives Qualitätskriterium, die Edierkosten der berechneten Zuordnung, existiert, mit dem beurteilt werden kann, wie gut das Verfahren die vorgegebenen Ziele erreicht. Falls die berechneten kostenoptimalen Lösungen den Erwartungen des Anwenders besser als die ähnlichkeitsbasierten Lösungen entsprechen, könnte der so berechnete Abstand zwischen zwei Klassendiagrammen als Referenzmaßstab für andere heuristische Verfahren herangezogen werden.

4.3 Korrespondenzberechnungsverfahren basierend auf Ähnlichkeiten

Alternativ zu dem edierkostenbasierten Verfahren wurde ein weiteres heuristisch ausgerichtetes Verfahren entwickelt, das die Korrespondenzen auf Basis von Ähnlichkeiten bestimmt und damit eine geringere Komplexität auf Klassenebene erreicht. Bei dem zweiten Verfahren stehen somit nicht die Unterschiede in Form von Edierkosten, sondern die Ähnlichkeiten im Fokus. Ziel des Verfahrens ist es, eine Zuordnung mit einer möglichst großen Gesamtähnlichkeit zu finden, wobei die Ähnlichkeitswerte aller korrespondierenden Elemente summiert werden und Ähnlichkeitswerte von Einzelelementen unberücksichtigt bleiben (siehe Formel 4.18).

$$sim(M) = \sum_{(i,j)\in M} sim(i,j), \quad (i,j) \in A \times B \tag{4.18}$$

Da die Ähnlichkeiten der Klassen durch die Ähnlichkeiten ihrer Unterelemente mitbestimmt werden, werden die Korrespondenzen ebenfalls in einem zweistufigen Ansatz ermittelt. Zur Auswahl der korrespondierenden Elemente wird bei diesem Verfahren kein Optimierungsverfahren, sondern eine heuristische Greedy-Strategie verwendet. Das Verfahren besteht aus zwei Phasen:

1. **Berechnung der Ähnlichkeiten** Für jede mögliche Korrespondenz, d. h. für jedes Paar (i, j) mit $i \in A$, $j \in B$, wird ein Ähnlichkeitswert im Intervall $[0, 1]$ bestimmt. Der Extremwert 1 besagt, dass die beiden Elemente in Hinblick auf alle Kriterien übereinstimmen. Die Ähnlichkeitswerte bilden eine Ähnlichkeitsmatrix, die im Folgenden auch als Vergleichsmatrix bezeichnet wird. In Abschnitt 4.3.1 wird dargestellt, wie die Vergleichswerte ermittelt werden.

2. **Auswahl der Zuordnung** Aus der berechneten Ähnlichkeitsmatrix werden die korrespondierenden Elementpaare ausgelesen. Ein solches Auswahlverfahren wird in Abschnitt 4.3.2 beschrieben.

4.3.1 Berechnung der Ähnlichkeiten

Die Ähnlichkeit zweier Elemente wird durch die Ähnlichkeiten ihrer Eigenschaften sowie die Ähnlichkeiten ihrer Unterelemente bestimmt. Daher wird der Ähnlichkeitswert als gewichtete Summe der Ähnlichkeitswerte der Einzelkriterien berechnet und auf das Intervall $[0, 1]$ normiert. Die Gewichtung der Einzelkriterien kann über Parameter gesteuert werden. Für die jeweiligen Ähnlichkeitswerte der verschiedenen Typen wurden im Einzelnen folgende Berechnungen festgelegt:

Ähnlichkeit eines Klassenpaares/Interfacepaares (a, b)**:** Der Ähnlichkeitswert für Klassen oder Interfaces ergibt sich, wie in Formel 4.19 dargestellt, aus den gewichteten Ähnlichkeitswerten der Klassennamen (s_n), der Klassentypen (s_{kt}), der Attribute (s'_{at}), der Methoden (s'_m), der Assoziationen (s'_a) und Vererbungsbeziehungen (s'_v).

$$sim(a,b) = \frac{(w_1 s_n(a,b) + w_2 s_{kt}(a,b) + w_3 s'_{at}(a,b) + w_4 s'_m(a,b) + w_5 s'_a(a,b) + w_6 s'_v(a,b))}{\sum_{i=1}^{6} w_i} \tag{4.19}$$

Die Ähnlichkeit der Namen kann analog zu den Edierkosten zweier Zeichenketten des ECVerfahrens mit verschiedenen Methoden ermittelt werden (vgl. dazu Formel 4.8 oder 4.9 des ECVerfahrens, S. 85). Die Edierkosten, die im Intervall $[0, 1]$ liegen, werden in einen Ähnlichkeitswert im Intervall $[0, 1]$ umgewandelt, indem sie vom Wert 1 subtrahiert werden.
Als Klassentyp wird zwischen der konkreten Klasse, der abstrakten Klasse und dem Interface unterschieden. Der Wert $s_{kt}(a, b)$ beträgt 1, falls der Klassentyp der beiden Klassen übereinstimmt, ansonsten 0.

$$s_{kt}(a,b) = \begin{cases} 1 & \text{falls a.klassentyp} = \text{b.klassentyp} \\ 0 & \text{sonst} \end{cases} \tag{4.20}$$

Die Formel 4.19 zur Berechnung der Ähnlichkeitswerte eines Paares von Klassen oder Interfaces enthält auch die aggregierten Ähnlichkeitswerte s' für die Attribute, Methoden, Assoziationen und Vererbungsbeziehungen. Um den Gesamtähnlichkeitswert für eine Menge von Unterelementen des gleichen Typs, wie zum Beispiel Attribute, zu berechnen, wird eine weitere Ähnlichkeitsmatrix konstruiert und mit einem Auswahlverfahren eine Zuordnung bestimmt. Die Einzelähnlichkeiten der korrespondierenden Attribute werden summiert und durch die Zahl der Elemente der größeren der beiden Elementmengen geteilt. Auf diese Weise wird eine Normierung der aggregierten Ähnlichkeitswerte auf das Intervall $[0, 1]$ erreicht.

Ähnlichkeit eines Attributpaares (a, b)**:** Die Ähnlichkeit zweier zu vergleichender Attribute ergibt sich aus der Namensähnlichkeit und der Ähnlichkeit des Typs (siehe Formel 4.21). Die Namensähnlichkeit wird mit einem Vergleichsverfahren für Zeichenketten bewertet. Die Ähnlichkeit des Typs wird nach Regel 4.22 bewertet, wobei unterschieden wird, ob der Typ identisch ist, oder zumindest der gleichen Kategorie angehört. Als Kategorien werden hier *Zeichenkette*, *ganzzahlige Zahl* und *Gleitkommazahl* verwendet.

$$s_{at}(a,b) = \frac{w_7 s_n(a,b) + w_8 s_t(a,b)}{w_7 + w_8} \tag{4.21}$$

$$s_t(a,b) = \begin{cases} 1 & \text{falls a.typ = b.typ} \\ 0,5 & \text{falls kat(a.typ) = kat(b.typ)} \\ 0 & \text{sonst} \end{cases} \tag{4.22}$$

Das nachstehende Beispiel veranschaulicht, wie aus den einzelnen Ähnlichkeitswerten s_{at} die Gesamtähnlichkeit s'_{at} berechnet wird.

Beispiel 12 (Berechnung der Attributsähnlichkeit)**.**
Es seien die Klassen a und b mit folgenden Attributen gegeben:
Attribute der Klasse a: `int i, String s, double d`
Attribute der Klasse b: `int i, String s, float f, double d`
$s'_{at}(a,b) = (1+1+1)/4 = 0{,}75$

Ähnlichkeit eines Methodenpaares (a,b)**:** Methoden werden in Hinblick auf den Methodennamen, den Rückgabetyp und die Übergabeparameter verglichen (vgl. Formel 4.23), wobei die Namensähnlichkeit mit einem Vergleichsverfahren für Zeichenketten und die Typähnlichkeit nach Regel 4.22 berechnet werden.

$$s_m(a,b) = \frac{w_9 s_n + w_{10} s_t(a,b) + w_{11} s'_p(a,b)}{w_9 + w_{10} + w_{11}} \tag{4.23}$$

Ähnlichkeit eines Übergabeparameterpaares (a,b)**:** Die Ähnlichkeit von Parametern wird analog zur Ähnlichkeit von Attributen berechnet:

$$s_p(a,b) = \frac{w_{12} s_n(a,b) + w_{13} s_t(a,b)}{w_{12} + w_{13}} \tag{4.24}$$

Ähnlichkeit eines Vererbungsbeziehungspaares (a,b)**:** Zur Berechnung der Ähnlichkeit zweier Klassen wird auch eine Abschätzung der Ähnlichkeit der Vererbungsbeziehungskanten benötigt. Da Vererbungsbeziehungen keine weiteren Eigenschaften besitzen und nur durch die Quell- und Zielklasse charakterisiert werden, wird für die Ähnlichkeit ausgehender Vererbungskanten $s_{v_{aus}}(a,b)$ ein Wert verwendet, der sich aus der Ähnlichkeit der Namen der Zielklassen und der Ähnlichkeit der Klassentypen der Zielklassen zusammensetzt (vgl. Formel 4.25). Analog setzt sich der Wert für eingehende Vererbungskanten $s_{v_{ein}}(a,b)$ aus der Ähnlichkeit der Namen der Quellklassen und der Ähnlichkeit der Klassentypen der Quellklassen zusammen (siehe Formel 4.26).

$$s_{v_{aus}}(a,b) = \frac{w_{14} s_n(a.ziel, b.ziel) + w_{15}(s_{kt}(a.ziel, b.ziel)}{w_{14} + w_{15}} \tag{4.25}$$

$$s_{v_{ein}}(a,b) = \frac{w_{14} s_n(a.quelle, b.quelle) + w_{15}(s_{kt}(a.quelle, b.quelle)}{w_{14} + w_{15}} \tag{4.26}$$

Ähnlichkeit eines Assoziationspaares (a,b)**:** Folgende Vergleichskriterien bilden den Ähnlichkeitswert für gerichtete Assoziationen (siehe Formel 4.27): Vergleich der Assoziationsnamen über ein Stringvergleichsverfahren ($s_n(a,b)$), Vergleich der Multiplizitäten ($s_u(a,b)$), Vergleich der Quellklasse $s_q(a,b)$, Vergleich der Zielklasse $s_z(a,b)$ sowie

der Vergleich der Assoziationseigenschaften `opposite`-Link ($s_o(a,b)$) und Komposition ($s_c(a,b)$). Die Ähnlichkeiten für den Vergleich der Quell- und Zielklasse berechnen sich analog zu Formel 4.26 bzw. 4.25. Für den Ähnlichkeitswert der Multiplizitäten werden die unteren und die oberen Grenzen getrennt voneinander verglichen, wobei eine Übereinstimmung dem Wert 1 und eine Abweichung dem Wert 0 entspricht. Der Gesamtwert setzt sich zu gleichen Teilen aus den beiden Einzelwerten zusammen.

$$s_a(a,b) = \frac{w_{16}s_n(a,b) + w_{17}s_u(a,b) + w_{18}s_q(a,b) + w_{19}s_z(a,b) + w_{20}s_o(a,b) + w_{21}s_c(a,b)}{\sum_{i=16}^{21} w_i} \tag{4.27}$$

Für den Ähnlichkeitswert aller Assoziationen, der für die Ähnlichkeitsbewertung der Klassen in Formel 4.19 verwendet wird, werden die Ähnlichkeiten der eingehenden und ausgehenden Assoziationen jeweils nach der Formel 4.27 berechnet, gewichtet summiert und auf 1 normiert (siehe Formel 4.28).

$$s'_a(a,b) = \frac{w_{22}s'_{a_{ein}}(a,b) + w_{23}s'_{a_{aus}}(a,b)}{w_{22} + w_{23}} \tag{4.28}$$

Auch die Ähnlichkeiten der ein- und ausgehenden Vererbungskanten einer Klasse werden gewichtet summiert und auf 1 normiert (vgl. Formel 4.29).

$$s'_v(a,b) = \frac{w_{24}s'_{v_{ein}}(a,b) + w_{25}s'_{v_{aus}}(a,b)}{w_{24} + w_{25}} \tag{4.29}$$

Die Gewichtung der Ähnlichkeitswerte kann zur Laufzeit des Vergleichsrahmenwerks über eine Sicht eingestellt werden. Die Abbildung 4.27 zeigt die Konfiguration der Gewichte, mit denen in den Testläufen des Kapitels 6 gearbeitet wurde.

4.3.2 Auswahlverfahren

Zur Auswahl der Korrespondenzen aus der Vergleichsmatrix wird ein heuristisches Verfahren verwendet, das die nächste Korrespondenz jeweils über den maximalen Eintrag der Ähnlichkeitsmatrix bestimmt. Daher wird das Verfahren in dieser Arbeit auch als Maxwertverfahren bezeichnet. Es wird zunächst die Grundform des Verfahrens angegeben, wie sie für die Ähnlichkeitsberechnung der Unterelemente von Klassen, z. B. Attribute oder Methoden, verwendet wird. Nach einem kurzen Beispiel wird eine Variante des Verfahrens vorgestellt, die auf Klassenebene wahlweise anstatt des einfachen Auswahlverfahrens eingesetzt werden kann.

Das einfache Auswahlverfahren läuft wie folgt ab: Zunächst wird der Eintrag mit dem höchsten Wert in der Vergleichsmatrix bestimmt. Falls der Eintrag nicht eindeutig ist, wird das erste Vorkommen verwendet (willkürliche Regelung). Der ausgewählte Eintrag in Zeile i und Spalte j entspricht der Korrespondenz zweier Elemente. Da Mehrfachzuordnungen nicht zugelassen werden und jedes Element nur in einer Korrespondenz enthalten sein darf, wird nicht nur der ausgewählte Eintrag sondern alle Einträge der Zeile i und Spalte j aus der Matrix mit den Ähnlichkeitswerten entfernt. Auf die gleiche

Abbildung 4.27: Konfiguration der Gewichte für die Ähnlichkeitswerte

Weise werden auch die weiteren Korrespondenzen bestimmt, bis die maximale Anzahl an Korrespondenzen gefunden wurde.

Beispiel 13 (Maxwertverfahren)**.** Es sei die Vergleichsmatrix aus Abbildung 4.28 gegeben, die die paarweisen Ähnlichkeitswerte der Elemente der Mengen $A = \{a_1, a_2, a_3, a_4\}$ und $B = \{b_1, b_2, b_3\}$ enthält. Der Eintrag in Zeile i und Spalte j entspricht dabei dem Ähnlichkeitswert der Korrespondenz (a_i, b_j). Da die Anzahl der Elemente der kleineren der beiden Mengen 3 beträgt, wird die maximale Zahl an möglichen Korrespondenzen innerhalb von 3 Schritten bestimmt:
1. Schritt: Der Maximalwert beträgt 0,9 und befindet sich in Zeile 2, Spalte 2. Dies entspricht einer Korrespondenz der Elemente a_2 und b_2. Die Einträge der Zeile 2 sowie die Einträge der Spalte 2 werden aus der Vergleichsmatrix entfernt.
2. Schritt: Als nächster Maximalwert wird 0,5 an Position 1,1 bestimmt. Die Elemente a_1 und b_1 werden daher als korrespondierend betrachtet, die Einträge der Zeile 1 sowie der Spalte 1 werden aus der Matrix entfernt.
3. Schritt: Nun ist der Wert 0,45 in Zeile 4, Spalte 3 maximal, was einer Korrespondenz von a_4 und b_3 entspricht.
Damit gilt $M = \{(a_1, b_1), (a_2, b_2), (a_4, b_3)\}$.

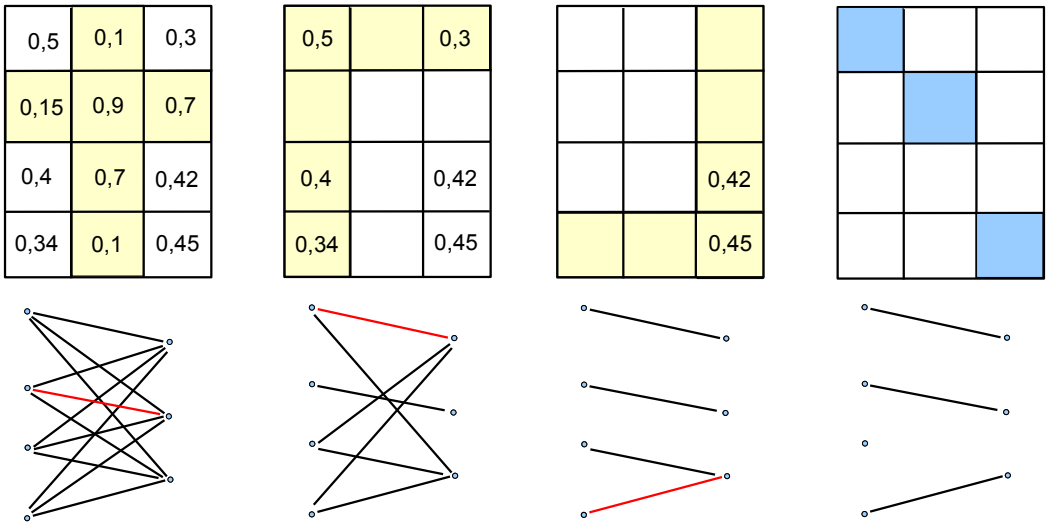

Abbildung 4.28: Vergleichsmatrix zu Beispiel 13 und der Zusammenhang mit der Suche nach einem Matching auf einem bipartiten Graphen

Um zu verhindern, dass sehr verschiedene Elemente einander zugeordnet werden, ist es sinnvoll, eine Grenze für den Ähnlichkeitswert als vorzeitiges Abbruchkriterium vorzugeben. Liegt die Ähnlichkeit des bestimmten Eintrags unterhalb des Grenzwertes, wird das Korrespondenzberechnungsverfahren ohne Festlegung der Korrespondenz abgebrochen, da alle weiteren Einträge den Grenzwert ebenfalls unterschreiten werden. Für die Beispiele in dieser Arbeit wurde ein Grenzwert von 0,5 für die Korrespondenzbildung auf Klassenebene verwendet. Bei Anwendung dieses Schwellenwerts auf Beispiel 13 würden nur die beiden Korrespondenzen (a_1, b_1) und (a_2, b_2) identifiziert werden.

Wie in Abbildung 4.28 (unten) zu sehen ist, ist die hier verwendete Vergleichsmatrix die Repräsentationsform eines bipartiten Graphen. Jeder Eintrag (i, j) der Vergleichsmatrix entspricht im Graphen einer Kante zwischen den Knoten für die Elemente i und j und die Höhe des Eintrags dem Ähnlichkeitswert der beiden Elemente. Beim Löschen von Einträgen aus dieser Vergleichsmatrix fallen auch die entsprechenden Kanten im bipartiten Graphen weg. Um den Aufwand für dieses Verfahren mit dem Aufwand des edierkostenbasierten Verfahrens zu vergleichen, wird auch hier der Aufwand für die Lösung der Korrespondenzberechnung auf Klassenebene in Abhängigkeit von n, der Klassenanzahl des größeren Klassendiagramms, angegeben. Die Konstruktion der Vergleichsmatrix verursacht einen Aufwand, der durch die Größenordnung $O(n^2)$ begrenzt wird. Da bei der derzeitigen Implementierung in jedem Iterationsschritt noch auf alle Einträge der Matrix zugegriffen wird und die Anzahl der Iterationsschritte auf n begrenzt ist, da höchstens n Korrespondenzen gebildet werden können, ergibt sich eine obere Schranke für den Gesamtaufwand von $O(n^3)$.

Das Maxwertverfahrens für die Klassenebene

In der bisher beschriebenen Variante des Verfahrens sind die einmal berechneten Ähnlichkeitswerte in der Vergleichsmatrix konstant. Während diese Grundform für die Bestimmung der Zuordnungen der Unterelemente wie zum Beispiel Attribute und Methoden eingesetzt wird, wird für die Klassenebene eine weitere Variante eingeführt, bei der die Ähnlichkeitswerte in der Vergleichsmatrix aktualisiert werden können.

Nach jeder einzelnen Korrespondenzbildung werden die kontextabhängigen Ähnlichkeitswerte der angrenzenden Klassen angepasst, um bereits identifizierte Korrespondenzen bei den nachfolgenden Entscheidungen zu berücksichtigen. Die Anpassung betrifft beispielsweise die Ähnlichkeitsberechnung von Assoziationen. Sind die Assoziationsenden durch bereits gebildete Korrespondenzen fixiert, kann die Ähnlichkeit der Assoziationen und damit auch die Ähnlichkeit der beteiligten Klassen genauer bestimmt werden. Das Gleiche gilt auch für Vererbungsbeziehungen. Die Aktualisierung der Werte ist nur für die Klassenebene relevant, da dem Datenmodell entsprechend nur für Klassen kontextabhängige Ähnlichkeiten berechnet werden. Um Aufwand zu vermeiden, werden nach der Zuordnung zweier Klassen nicht alle Ähnlichkeitswerte neu berechnet, sondern nur die Ähnlichkeitswerte im betroffenen 1-Kontext aktualisiert. Als 1-Kontext einer Klasse werden dabei diejenigen Klassen bezeichnet, die über eine Kante mit der Klasse verbunden sind. Dies betrifft die Vererbungsbeziehungen zu Ober- und Unterklassen sowie die jeweils anderen Assoziationsenden eingehender und ausgehender Assoziationen. Da nur die Ähnlichkeitswerte von Paaren, deren Klassen noch nicht korrespondieren, für den weiteren Verlauf des Verfahrens relevant sind, wird auf die Anpassung der Werte bereits korrespondierender Paare verzichtet.
Das Maxwertverfahren auf Klassenebene mit Grenzwert und iterativer Anpassung der kontextabhängigen Kosten besteht aus den in Algorithmus 2 angegebenen Schritten. Der Unterschied zwischen dem Auswahlverfahren mit und ohne Anpassung wird im folgenden Beispiel demonstriert.

Algorithmus 2 Maxwertverfahren auf Klassenebene

Berechne die Ähnlichkeitsmatrix $V \in \mathbb{R}^{|A|\times|B|}$
Solange weniger als $min(|A|, |B|)$ Korrespondenzen identifiziert sind
    Bestimme $V[i][j]$ als größten Eintrag der Matrix V
    Falls ($V[i][j] <$ schwelle) Ende
    Die Klassen a_i und b_j korrespondieren
    Bei der Anpassungsvariante: aktualisiere die Ähnlichkeiten im 1-Kontext in V
    Lösche alle Einträge der Zeile i und der Spalte j in V.

Beispiel 14 (Auswahlverfahren mit und ohne Anpassung im Vergleich)**.** Die Modelle der Abbildungen 4.29 und 4.30, die das Gesellschaftsspiel Ludo modellieren, sollen mit dem ähnlichkeitsbasierten Verfahren verglichen werden. Dazu werden zunächst die Ähnlichkeitswerte für die möglichen Klassenpaare in Tabelle 4.3 erstellt, wobei für den

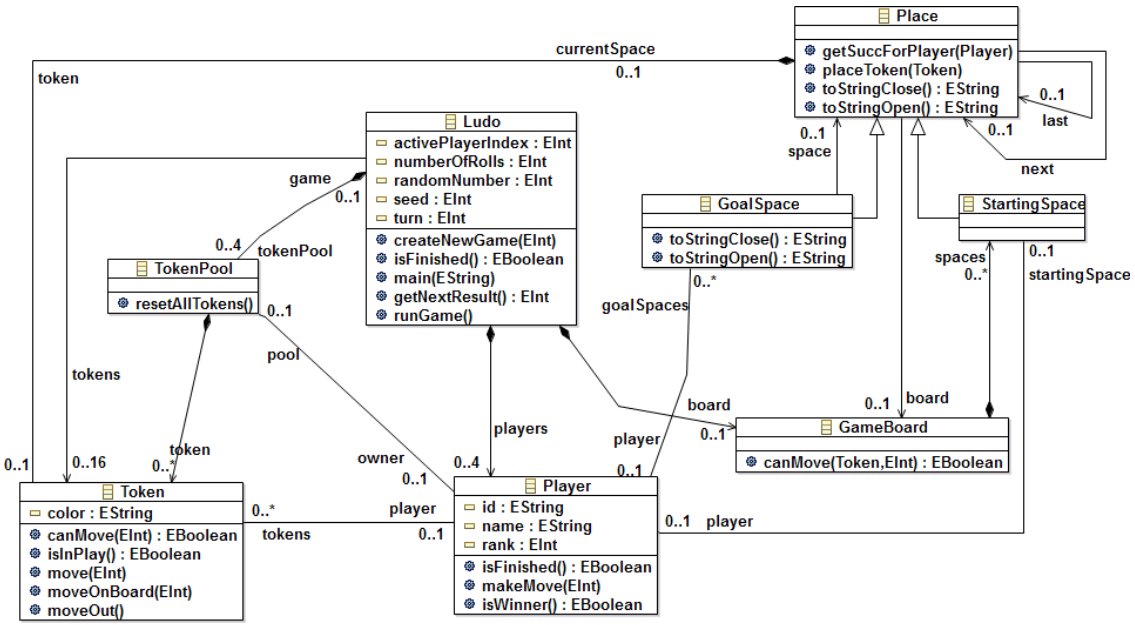

Abbildung 4.29: Modell A für Beispiel 14

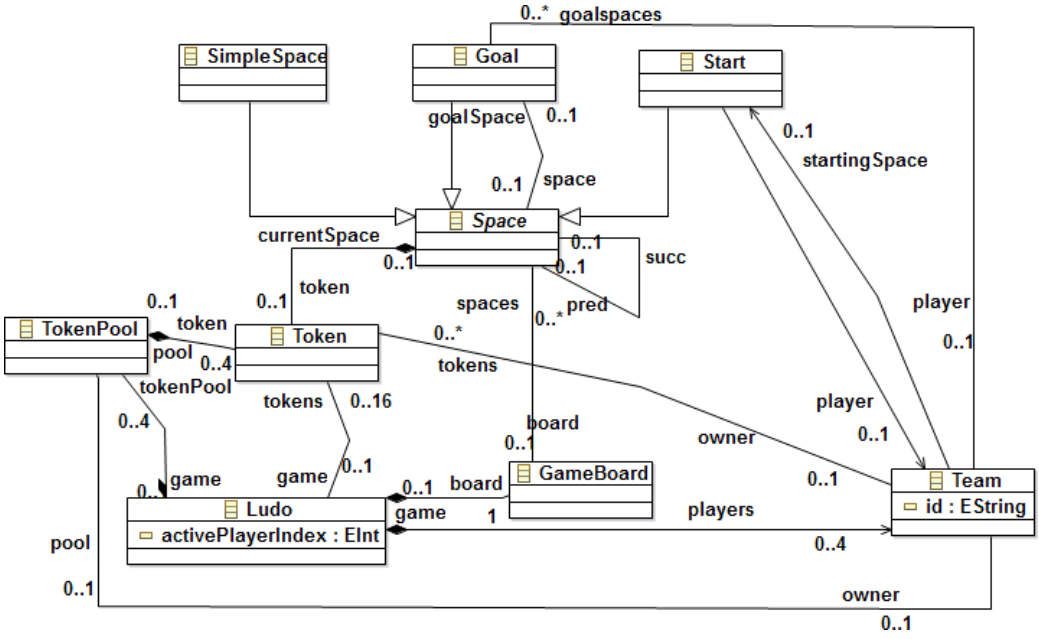

Abbildung 4.30: Modell B für Beispiel 14

Zeichenkettenvergleich die Variante auf Basis der Levenshtein-Metrik verwendet wurde. Tabelle 4.4 zeigt das Ergebnis des einfachen Auswahlverfahrens ohne Anpassung der Ähnlichkeitswerte. Es identifiziert die Korrespondenzen (`TokenPool`, `TokenPool`), (`StartingSpace`, `Start`), (`GameBoard`, `GameBoard`), (`Ludo`, `Ludo`), (`GoalSpace`, `Goal`), (`Token`, `Token`) und (`Player`, `Team`) in dieser Reihenfolge. Die nächste mögliche Korrespondenz `Place`, `Space` wird nicht mehr gebildet, da der Ähnlichkeitswert bereits unter

Tabelle 4.3: V_0: Initiale Ähnlichkeitswerte zu Beispiel 14

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard	0,75	0,4205	0,553	0,4375	0,5054	0,7059	0,6466	0,4468
Ludo	0,4128	0,6048	0,3173	0,481	0,3773	0,4796	0,4395	0,4571
Goal	0,5403	0,3535	0,601	0,3465	0,4424	0,7199	0,5427	0,3925
SimpleSpace	0,488	0,2756	0,5444	0,3012	0,3985	0,7025	0,4722	0,2884
Space	0,3212	0,1795	0,2889	0,2444	0,477	0,4307	0,3345	0,1948
Start	0,5492	0,3493	0,558	0,3313	0,4289	0,7502	0,4883	0,3419
TokenPool	0,6068	0,4235	0,527	0,4402	0,519	0,6775	0,7583	0,4946
Token	0,5776	0,4365	0,4967	0,4799	0,5541	0,6546	0,6586	0,579
Team	0,4172	0,4482	0,3215	0,5551	0,3912	0,4762	0,4375	0,556

dem Grenzwert von 0,5 liegt. Auch die Klasse `SimpleSpace` bleibt ein Einzelelement.

Im Vergleich dazu zeigen die Tabellen 4.5 bis 4.12 die Schritte des ähnlichkeitsbasierten Verfahrens mit Anpassung der Ähnlichkeitswerte im 1-Kontext, die im Folgenden beschrieben werden. Die aktualisierten Werte sind in den Tabellen kursiv gedruckt.

Schritt 1: Das Auswahlverfahren mit Anpassung der Ähnlichkeitswerte wählt ebenfalls (`TokenPool`, `TokenPool`) als erste Korrespondenz aus. Da in Modell A zwischen der Klasse `TokenPool` und den Klassen `Ludo`, `Player` und `Token` und in Modell B zwischen der Klasse `TokenPool` und den Klassen `Ludo`, `Team` und `Token` Assoziationen bestehen, werden die Ähnlichkeitswerte der Paare aus {`Ludo`, `Player`, `Token`} × {`Ludo`, `Team`, `Token`} aktualisiert. Dabei werden die eingehenden und ausgehenden Assoziationen getrennt betrachtet. In der ersten Bewertung der Ähnlichkeiten der Referenzen wurde jeweils eine Ähnlichkeit von 0 für das unbekannte Assoziationsende angenommen. Da nun bekannt ist, dass je nach Orientierung nun nicht nur die Quelle, sondern auch das Ziel der Assoziation identisch ist - oder umgekehrt-, werden die Ähnlichkeitswerte für die Referenzen entsprechend der oben angegebenen Konfiguration um $\frac{1}{6}$ erhöht. Auf der aktualisierten Vergleichsmatrix für Referenzen wird die Gesamtähnlichkeit aller Referenzen neu berechnet und in die Ähnlichkeit des Klassenpaares eingerechnet. Dass sich dadurch nicht immer eine Erhöhung der Ähnlichkeitswerte zweier Klassen ergeben muss, zeigt der Wert an der Stelle `Team`, `Token`. Hier ist die Erhöhung der Ähnlichkeit der einzelnen Referenzen nicht hoch genug, um die Zuordnung der Referenzen zu beeinflussen. Da die Ähnlichkeit aller Referenzen einer Richtung $\frac{1}{12}$ des Ähnlichkeitswertes zweier Klassen bestimmt, kann sich im Idealfall, falls je nur eine Assoziation der Richtung existiert, eine Änderung in Höhe von $\frac{1}{72}$ durch die Propagierung der Ähnlichkeit über eine unidirektionale Assoziation ergeben.

Tabelle 4.4: Ergebnis des einfachen Auswahlverfahrens

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard **(3)**	**0,75**							
Ludo **(4)**		**0,6048**						
Goal **(5)**			**0,601**					
SimpleSpace					0,3985			
Space					0,477			
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token **(6)**								**0,579**
Team **(7)**				**0,5551**				

Tabelle 4.5: V_1: Ähnlichkeitswerte nach Korrespondenzbildung `TokenPool`, `TokenPool` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard	0,75	0,4205	0,553	0,4375	0,5054	0,7059		0,4468
Ludo	0,4128	*0,6145*	0,3173	*0,4899*	0,3773	0,4796		*0,4612*
Goal	0,5403	0,3535	0,601	0,3465	0,4424	0,7199		0,3925
SimpleSpace	0,488	0,2756	0,5444	0,3012	0,3985	0,7025		0,2884
Space	0,3212	0,1795	0,2889	0,2444	0,477	0,4307		0,1948
Start	0,5492	0,3493	0,558	0,3313	0,4289	0,7502		0,3419
TokenPool **(1)**							**0,7583**	
Token	0,5776	*0,4427*	0,4967	*0,4874*	0,5541	0,6546		*0,5831*
Team	0,4172	*0,4475*	0,3215	*0,5626*	0,3912	0,4762		*0,556*

Tabelle 4.6: V_2: Ähnlichkeitswerte nach Korrespondenzbildung `Start`, `StartingSpace` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard	0,75	0,4205	0,553	0,4375	0,5054			0,4468
Ludo	0,4128	0,6145	0,3173	0,4899	0,3773			0,4612
Goal	0,5403	0,3535	0,601	0,3465	0,4424			0,3925
SimpleSpace	0,488	0,2756	0,5444	0,3012	0,3985			0,2884
Space	0,3212	0,1795	0,2889	0,2444	*0,4855*			0,1948
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token	0,5776	0,4427	0,4967	0,4874	0,5541			0,5831
Team	*0,4203*	0,4475	0,3215	*0,5701*	0,3912			0,556

Schritt 2: Der nächste Maximalwert ist 0,7502 und führt zur Korrespondenzbildung von (`StartingSpace`, `Start`). Da die Klassen `StartingSpace` und `Start` jeweils eine Oberklasse, `Place` bzw. `Space`, besitzen, ändert sich in V_2 der Ähnlichkeitswert des Paares `Place`, `Space`. Ihre Unterklassen `StartingSpace`, `Start` sind nun als identisch zu betrachten, weshalb der Ähnlichkeitswert, der vorher bei 0,6923 lag, auf 1 gesetzt wird. Da `Space` drei Unterklassen besitzt und alle Unterklassen ein Zwölftel der Klassenähnlichkeit bestimmen, führt dies mit dem Faktor $\frac{1}{36}$ zu einer Erhöhung des Gesamtwerts um 0,0085. Weitere Erhöhungen der Ähnlichkeitswerte werden durch die eingehenden Assoziationen {`GameBoard`, `Player`} × {`Team`} und die ausgehenden Assoziationen nach `Player` bzw. `Team` verursacht.

Schritt 3: In der neuen Ähnlichkeitsmatrix wird nun der Wert 0,75 an der Stelle `GameBoard`, `GameBoard` als größter Wert bestimmt. Nach der Bildung der Korrespondenz wird im 1-Kontext der Klassen die Ähnlichkeitswerte für alle Paare aus {`Ludo`, `Place`} × {`Ludo`, `Space`} erhöht, da diese Klassen jeweils über eine Assoziation mit `GameBoard` aus Modell A bzw. aus Modell B verbunden sind. Die Klasse `StartingSpace` aus Modell A wäre ebenfalls von den Aktualisierungen betroffen. Da die Klasse bereits zugeordnet wurde und damit die Änderungen keine Auswirkungen mehr auf andere mögliche Zuordnungen haben, wird auf die Durchführung der Anpassung verzichtet.

Schritt 4: Als nächstes Korrespondenzpaar wird `Ludo`, `Ludo` gebildet. Da die Klassen `GameBoard` und `TokenPool` ebenfalls bereits zugeordnet sind, beschränken sich die Aktualisierungen auf die Klassenpaare aus {`Token`, `Player`} × {`Token`, `Team`}, die jeweils über Assoziationen mit `Ludo` aus Modell A bzw. Modell B verbunden sind.

Tabelle 4.7: V_3: Ähnlichkeitswerte nach Korrespondenzbildung `GameBoard`, `GameBoard` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard **(3)**	**0,75**							
Ludo		*0,6187*	0,3173	0,4899	*0,3815*			0,4612
Goal		0,3535	0,601	0,3465	0,4424			0,3925
SimpleSpace		0,2756	0,5444	0,3012	0,3985			0,2884
Space		*0,1829*	0,2889	0,2444	*0,4888*			0,1948
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token		0,4427	0,4967	0,4874	0,5541			0,5831
Team		0,4475	0,3215	0,5701	0,3912			0,556

Tabelle 4.8: V_4: Ähnlichkeitswerte nach Korrespondenzbildung `Ludo`, `Ludo` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard **(3)**	**0,75**							
Ludo **(4)**		**0,6187**						
Goal			0,601	0,3465	0,4424			0,3925
SimpleSpace			0,5444	0,3012	0,3985			0,2884
Space			0,2889	0,2444	0,4888			0,1948
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token			0,4967	*0,4907*	0,5541			*0,5873*
Team			0,3215	*0,5735*	0,3912			*0,5594*

Schritt 5: Ebenso werden nach der Korrespondenzbildung von `GoalSpace`, `Goal` die Ähnlichkeitswerte nur noch für die Klassenpaare {`Player`, `Place`} × {`Team`, `Space`} aktualisiert.

Schritt 6: Als nächstes Korrespondenzpaar werden die Klassen `Token`, `Token` identifiziert. Nach den Anpassungen in {`Player`, `Place`} × {`Team`, `Space`} überschreitet der

Tabelle 4.9: V_5: Ähnlichkeitswerte nach Korrespondenzbildung `Goal`, `GoalSpace` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard **(3)**	**0,75**							
Ludo **(4)**		**0,6187**						
Goal **(5)**			**0,601**					
SimpleSpace				0,3012	0,3985			0,2884
Space				*0,2521*	*0,4977*			0,1948
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token				0,4907	0,5541			0,5873
Team				*0,581*	*0,3943*			0,5594

Wert an der Stelle `Place`, `Space` den Schwellenwert in Höhe von 0,5.

Schritt 7: Die Klassen `Player` und `Team` werden als korrespondierend identifiziert, woraus sich aber keine weiteren Anpassungen ergeben.

Schritt 8: Aufgrund der Anpassung der Ähnlichkeitswerte können nun im 8.Schritt die Klassen `Place` und `Space` zugeordnet werden. Damit wurde die maximale Anzahl an möglichen Korrespondenzen identifiziert und die Klasse `SimpleSpace` bleibt ein Einzelelement. Das Verfahren ist an dieser Stelle beendet. Mit den Aktualisierungen zwischen den Schritten sind in der berechneten Zuordnung die Korrespondenzpaare (`TokenPool`, `TokenPool`), (`StartingSpace`, `Start`), (`GameBoard`, `GameBoard`), (`Ludo`, `Ludo`), (`GoalSpace`, `Goal`), (`Token`, `Token`), (`Player`, `Team`) und (`Place`, `Space`) enthalten.

Diskussion des Verfahrens Die Auswahl der Kanten erfolgt hier anhand eines Greedy-Verfahrens. In [Tur04, S. 40] werden Greedy-Verfahren wie folgt charakterisiert:

> „[Greedy-Verfahren] versuchen eine global optimale Lösung zu finden, indem sie iterativ lokal optimale Lösungen erweitern. Bei der Erweiterung einer partiellen Lösung wird nur die aktuelle Situation betrachtet, d. h. es erfolgt keine Analyse der globalen Situation. Ferner werden einmal getroffene Entscheidungen zur Erweiterung einer partiellen Lösung nicht wieder revidiert [...]“

Dass getroffene Entscheidungen nicht mehr revidiert werden, bedeutet im Fall des Maxwertverfahrens, dass die in den einzelnen Schritten ausgewählten Kanten auch in der

Tabelle 4.10: V_6: Ähnlichkeitswerte nach Korrespondenzbildung `Token`, `Token` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard **(3)**	**0,75**							
Ludo **(4)**		**0,6187**						
Goal **(5)**			**0,601**					
SimpleSpace				0,3012	0,3985			
Space				*0,2607*	*0,5044*			
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token **(6)**								**0,5873**
Team				*0,5885*	*0,4032*			

Tabelle 4.11: V_7: Ähnlichkeitswerte nach Korrespondenzbildung `Player`, `Team` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard **(3)**	**0,75**							
Ludo **(4)**		**0,6187**						
Goal **(5)**			**0,601**					
SimpleSpace					0,3985			
Space					0,5044			
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token **(6)**								**0,5873**
Team **(7)**				**0,5885**				

Tabelle 4.12: V_8: Ähnlichkeitswerte nach Korrespondenzbildung `Place`, `Space` und Anpassung

	GameBoard	Ludo	GoalSpace	Player	Place	StartingSpace	TokenPool	Token
GameBoard **(3)**	**0,75**							
Ludo **(4)**		**0,6187**						
Goal **(5)**			**0,601**					
SimpleSpace								
Space **(8)**					**0,5044**			
Start **(2)**						**0,7502**		
TokenPool **(1)**							**0,7583**	
Token **(6)**								**0,5873**
Team **(7)**				**0,5885**				

endgültigen Zuordnung enthalten sind. Der Vorteil der Greedy-Lösung besteht darin, dass der Aufwand für die Auswahl eines Korrespondenzpaares überschaubar gering ist. Die erreichte Lösung ist in vielen Fällen jedoch nicht optimal. Dies bedeutet, dass eine andere Zuordnung existieren kann, deren Gesamtähnlichkeit höher ist. Statt des beschriebenen Maxwertverfahrens können auch andere Greedy-Auswahlstrategien eingesetzt werden. Alternativ könnte die Suche einer Zuordnung mit maximaler Gesamtähnlichkeit wieder als Optimierungsproblem aufgefasst werden (vgl. Optimierungsproblem 3).

Optimierungsproblem 3 (Ähnlichkeitsmaximale Zuordnung)**.**
Bestimme $M^*(A, B)$ mit $\text{sim}(M^*(A, B)) \geq \text{sim}(M(A, B)) \; \forall \; M(A, B)$

Da es sich in diesem Fall im Gegensatz zu dem Optimierungsproblem 2 des EC-Verfahrens um ein einfaches Summen-Matching-Problem auf einem bipartiten Graphen handelt, kann das Problem z. B. mit dem Glover-Klingman-Verfahren aus [NM02, S. 299] gelöst werden. Dies ist jedoch nur für die nicht iterative Variante sinnvoll, bei der die Zuordnung nicht in Einzelschritten gebildet wird, zwischen denen die Ähnlichkeitswerte angepasst werden. Durch den Austausch des Auswahlverfahrens kann sich die Komplexitätsklasse des Verfahrens erhöhen. Im Fall des Glover-Klingman-Verfahrens mit einer theoretischen Komplexität der Ordnung $O(mn^2)$ für einen bipartiten Graphen mit n Knoten und m Kanten liegt der Aufwand des Verfahrens für die Klassenebene statt in $O(n^3)$ mit dem einfachen Auswahlverfahren wieder in $O(n^4)$.

4.3.3 Bewertung des Verfahrens und Abgrenzung zu verwandten Arbeiten

Das ähnlichkeitsbasierte Verfahren arbeitet wie das andere bereits vorgestellte Verfahren strukturbasiert, ohne eindeutige Objektbezeichner zu verwenden. Es verwendet die gleiche Baumstruktur der Klassendiagramme und unterstellt ebenfalls eine homomorphe Abbildung der Assoziationen. Die Variante mit Anpassung nutzt darüber hinaus die Querverbindungen. Das ähnlichkeitsbasierte Verfahren besitzt im Vergleich zum edierkostenbasierten Verfahren eine geringere Komplexität. Dies wird dadurch erkauft, dass bei dem Greedy-Ansatz lokal getroffene Entscheidungen nicht wieder revidiert werden und über die errechnete Lösung keine Aussage in Bezug auf die Optimalität getroffen werden kann.

Der Ablauf unterscheidet sich von SiDiff [KWN05] und UMLDiff [XS05] und lässt sich nicht in eine klare Top-down- oder Bottom-up-Strategie einteilen. Auf Klassenebene werden die Ähnlichkeitswerte aller möglichen Klassenpaare bestimmt, wobei im Vorfeld potentielle Zuordnungen der Unterelemente berechnet werden. Werden Klassen tatsächlich zugeordnet, werden die vorberechneten Zuordnungen der kontextunabhängigen Unterelemente übernommen. Die Zuordnung der kontextabhängigen Elemente wird erst nach Feststellung aller Korrespondenzen durchgeführt. Im Gegensatz zu den Verfahren mit Top-down-Vergleich ([LGJ07, XS05, BP08]) werden hier alle Klassen miteinander verglichen. Das Verfahren verwendet ebenso wie die anderen in Kapitel 3 vorgestellten Verfahren direkte Ähnlichkeitswerte, ohne diese im Sinne einer Bayesschen Statistik wie in [MKY06] zu bewerten.

In der Variante mit Anpassung der Ähnlichkeiten werden, ähnlich wie bei dem Similarity Flooding Algorithmus [MGMR02], der auf allgemeinen attributierten Graphen arbeitet, Ähnlichkeiten propagiert. Die Propagierung der Ähnlichkeiten findet hier jedoch in einer anderen Phase des Verfahrens statt, da sie jeweils durch die Bildung einer Korrespondenz ausgelöst wird. Als korrespondierend identifizierte Klassen werden als identisch betrachtet, was zur Aktualisierung der kontextabhängigen Ähnlichkeitswerte im 1-Kontext führt. Die Propagierung der Ähnlichkeit ist damit auf den Anteil der kontextabhängigen Ähnlichkeitswerte und auch lokal auf die direkten Nachbarklassen begrenzt. Inwieweit die Variante mit Anpassung der Ähnlichkeitswerte zu einer Verbesserung der Ergebnisse führt, wird in Kapitel 6 untersucht. Der Evaluationsteil befasst sich außerdem mit der Frage, welches der beiden eigenentwickelten Verfahren aus Sicht des Benutzers die besseren Ergebnisse liefert.

4.4 Erkennung von Verschiebungen auf Klassenebene

In den beiden beschriebenen Verfahren werden aufgrund der vollständigen Teilbaumzuordnung, die die Komplexität des Problems reduziert, nativ keine Verschiebungen erkannt. Da Verschiebungen eine kürzere Darstellung der Differenz ermöglichen, wurde eine Methode entwickelt, wie verschobene Elemente in beschränktem Umfang dennoch in der Nachbearbeitung der bestimmten Zuordnung erkannt werden können. Bevor die

Vorgehensweise erläutert wird, wird zunächst unterschieden, welche Arten von Verschiebungen auf unserem Datenmodell zugelassen werden.

Für Klassendiagramme sind drei Arten von Verschiebungen denkbar:

- Verschiebung eines Unterelements in Bezug auf die Kompositionsstruktur von einem Element zu einem anderen Element des gleichen Typs
- Verschiebung von Vererbungskanten oder Assoziationen durch Änderung des Quell- und/oder Zielelements
- Verschiebung von Elementen innerhalb der Reihenfolge einer geordneten Liste

Inwieweit diese in der Nachbearbeitungsphase umgesetzt wurden, wird im Folgenden beschrieben.

Die Kompositionsstruktur besitzt relativ zur Klassenebene zwei Unterebenen: Attribute und Methoden sind direkte Unterelemente von Klassen und die Methoden besitzen ihrerseits Unterelemente, die Übergabeparameter darstellen. Verschiebungen von Attributen oder Methoden von einer Klasse in eine andere Klasse zu erkennen, ist durchaus hilfreich. So ist es im Rahmen von Refactoringmaßnahmen nicht unüblich, dass ein Attribut oder eine Methode einer Klasse in deren Oberklasse verschoben wird. Die Darstellung von Methoden-übergreifenden Verschiebungen von Übergabeparametern würde hingegen meines Erachtens keine relevante Verbesserung des Differenzberichts bewirken. Das Verschieben von Vererbungsbeziehungen erscheint nicht sinnvoll, da eine Vererbungskante außer der Quelle und dem Ziel keine weiteren Eigenschaften besitzt.[4] Assoziationen besitzen hingegen neben der Ziel- und Quellklasse weitere Eigenschaften, wie den Namen und Multiplizitäten etc., so dass das Verschieben einer Assoziation sinnvoll sein kann. Als dritte Variante kommen Verschiebungen von Elementen in Listen, deren Reihenfolge relevant ist, in Frage. In diesem Punkt ist die Realisierung des von EMF Compare zur Verfügung gestellten Datenmodells leider nicht in allen Fällen mit dem konzeptionellen Datenmodell konform (vgl. Abschnitt 5.2). Auf der einen Seite werden technisch-bedingt Verschiebungen erkannt, die auf Modellebene semantisch nicht relevant sind. Dies betrifft z. B. die Reihenfolge von Interfaces, die eine Klasse implementiert. An anderer Stelle werden Verschiebungen nicht unterstützt, wo sie gewünscht wären. Wie in Abbildung 4.31 zu sehen ist, werden Verschiebungen innerhalb der Parameterlisten in der EMF Compare Version 1.0.0 nicht unterstützt. Aus diesem Grund werden die Eingabeparameter zweier zu vergleichender Methoden in beiden beschriebenen Verfahren positionsweise zugeordnet, um eine Reihenfolgeänderung dieser Parameter im Differenzreport visualisieren zu können (vgl. Abbildung 4.32).

Für die Implementierung der Nachverarbeitungsphase wurde von den beschriebenen Arten von Verschiebungen das Erkennen von Klassen-übergreifenden Attributverschiebungen beispielhaft herausgegriffen und wie folgt realisiert: Die Mengen der Einzelobjekte der Klassenzuordnung werden durchlaufen. Falls in beiden Mengen jeweils ein Attribut

[4] Darüber hinaus würden die verwendeten Korrespondenz- und Differenzmodelle von EMF Compare die Anzeige einer derartigen Differenz nicht direkt unterstützen, da wie in der Darstellung des verwendeten Datenmodells in Abschnitt 5.2 zu sehen ist, eine Vererbungsbeziehung lediglich eine Referenz einer Klasse und kein eigenes Element im Ecore-Baumeditor darstellt.

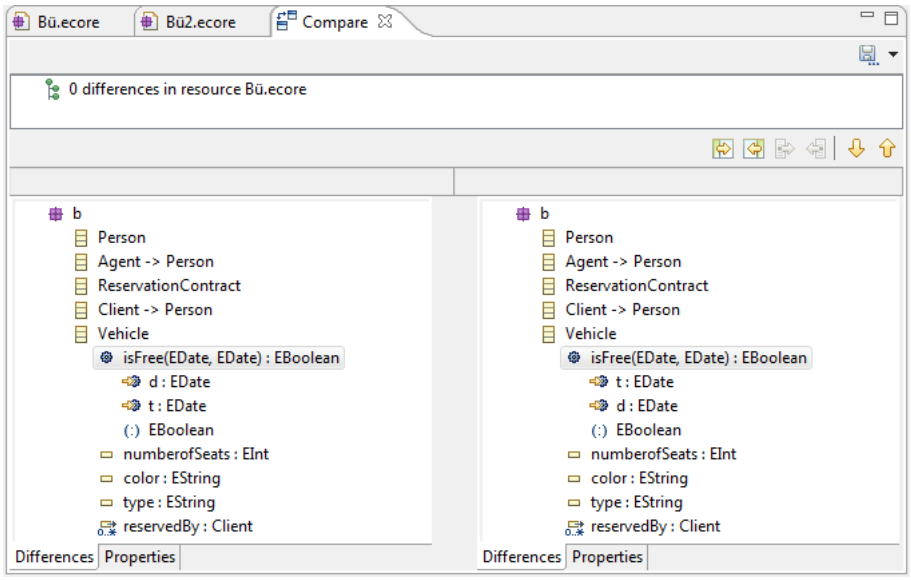

Abbildung 4.31: EMF Compare: die Reihenfolgeänderung in der Liste der Übergabeparameter wird nicht erkannt (EMF Compare Version 1.0.0).

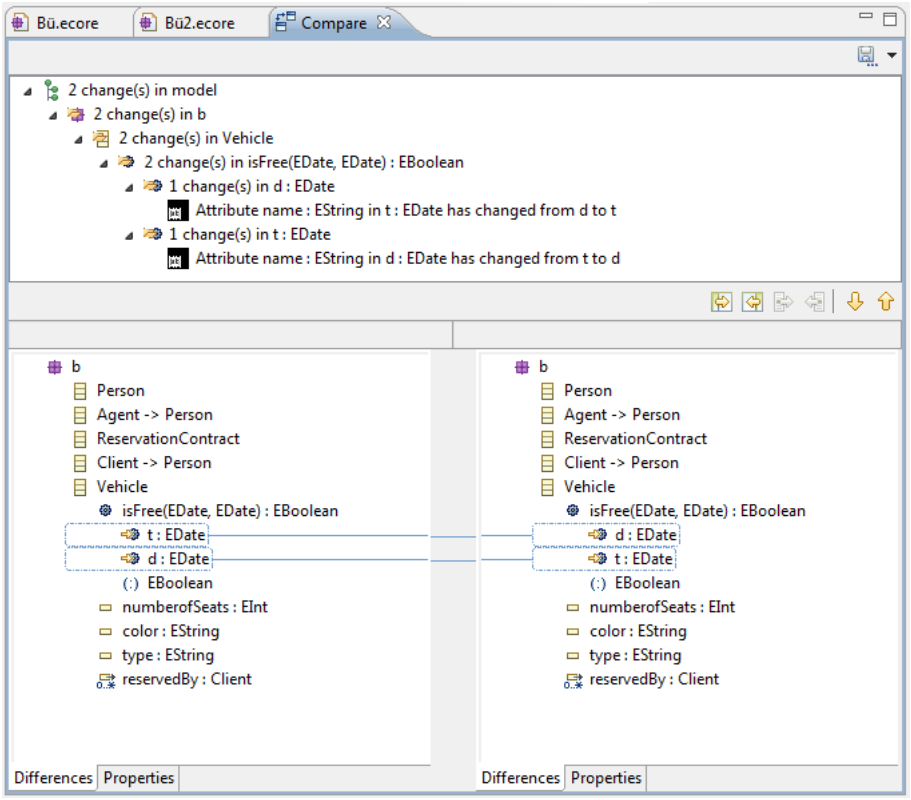

Abbildung 4.32: ECVerfahren: die Reihenfolgeänderung in der Liste der Übergabeparameter wird durch positionsweise Zuordnung erkannt, aber nicht als Verschiebung dargestellt.

mit gleichem Namen vorhanden ist, werden die beiden Elemente aus den Einzelobjektlisten entfernt und bilden eine Korrespondenz. Die Entscheidung, welche korrespondierenden Einzelobjekte zu einem Korrespondenzobjekt umgewandelt werden, wird nach einem Greedy-Prinzip getroffen: Im Fall von mehreren gleichnamigen Einzelelementen eines Modells wird für das erste gefundene Paar ein Korrespondenzobjekt erstellt.

Beispiel 15 (Verschobene Attribute)**.**
Die Abbildungen 4.33 und 4.34 zeigen zwei Differenzberichte des Vergleichs zweier Klassendiagramme. Das Modell auf der rechten Seite ist aus dem Modell der linken Seite entstanden, indem die Attribute `firstname` und `lastname` der Klassen `Agent` und `Client` in die gemeinsame Oberklasse `Person` verschoben wurden. Die Korrespondenzen wurden in beiden Fällen mit dem ECVerfahren berechnet, im ersten Fall ohne und im zweiten Fall mit nachträglichem Erkennen von Verschiebungen. Die Auswahl, welches der gleichnamigen Attribute verschoben und welches gelöscht wurde, erfolgt willkürlich.

Bewertung Analog zum Erkennen von Verschiebungen von Attributen, lassen sich auch verschobene Methoden und Assoziationen erkennen. Mit dieser Option kann der Anwender entscheiden, ob die Darstellung der Differenzen auf Basis der Elementaroperationen Einfügen, Löschen und Ändern angezeigt werden soll oder ob ein Teil der Operationen zu Verschiebungen gruppiert werden soll. Dieses nachträgliche Erkennungsverfahren arbeitet unabhängig von dem vorher eingesetzten Verfahren auf der Liste der Einzelobjekte der Klassenzuordnung. Da die Verschiebeoperationen nicht nativ in das Kostenmodell des edierkostenbasierten Verfahrens aufgenommen werden, entspricht die Kostenbewertung einer Verschiebung immer den Kosten der entsprechenden Einfüge- und Löschoperationen. Auf der anderen Seite gewinnt das Verfahren dadurch auch an Flexibilität. Verschobene Unterelemente einer Klasse können auch dann erkannt werden, falls sich die beiden beteiligten Klassen in verschiedenen Paketen befinden (vgl. mehrstufiges Verfahren in Abschnitt 4.5). Die Identifikation der korrespondierenden Einzelelemente erfolgt bislang über den Namen, das Kriterium kann aber relativ frei gewählt werden. Dabei lassen sich auch zusätzliche Einschränkungen realisieren, wie z. B. Verschiebungen nur für Elemente zuzulassen, die über eine Assoziation oder Vererbungskante miteinander in Verbindung stehen.

Da bislang die Liste der Einzelelemente des Korrespondenzmodells als Menge der Verschiebungskandidaten verwendet wird und die Unterelemente gelöschter oder eingefügter Elemente darin nicht enthalten sein dürfen (vgl. Abschnitt 5.3), werden derzeit Verschiebungen nur zwischen Elementen erkannt, deren Vaterelemente nicht eingefügt oder gelöscht wurden. Für das Beispiel aus Abbildung 4.23 auf Seite 110 kann daher keine Verschiebung des Attributs `state` erkannt werden.

Betrachtet man die nachträglich erkannten Verschiebungen aus Beispiel 15, wird deutlich, dass in diesem Fall das willkürliche Erkennen einer Verschiebung kein zufriedenstellendes Ergebnis ist. Um eine Refactoringmaßnahme zu erkennen, bei der aus zwei Klassen das gleiche Attribut in eine gemeinsame Oberklasse verschoben wird, ist es zielführender, eine Mehrfachverschiebung zu identifizieren, anstatt eine Verschiebung

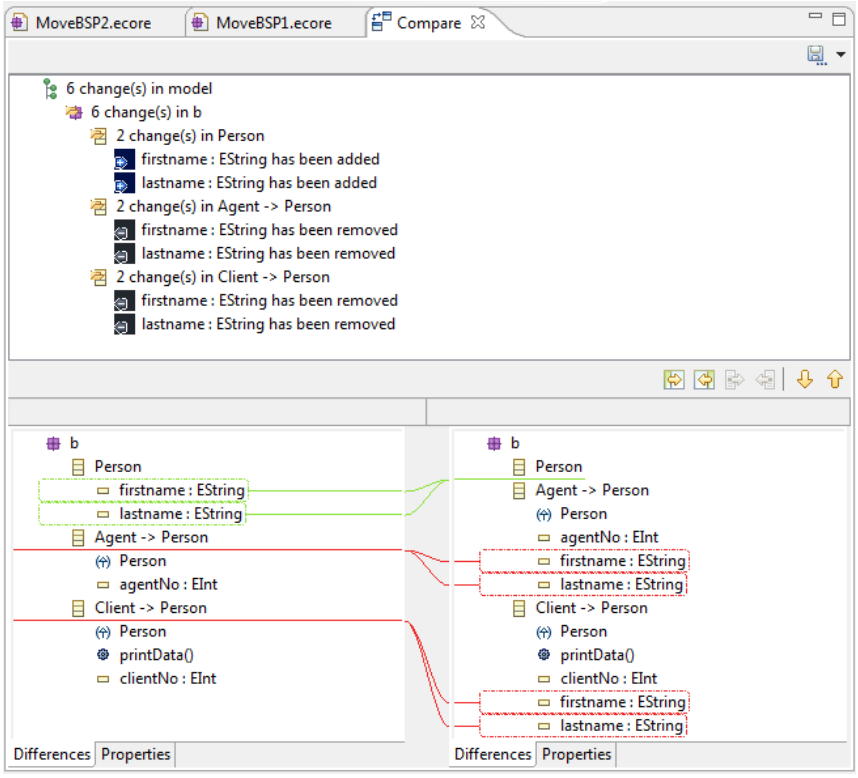

Abbildung 4.33: Differenzbericht des Vergleichs mit dem ECVerfahren ohne nachträgliches Erkennen von Verschiebungen

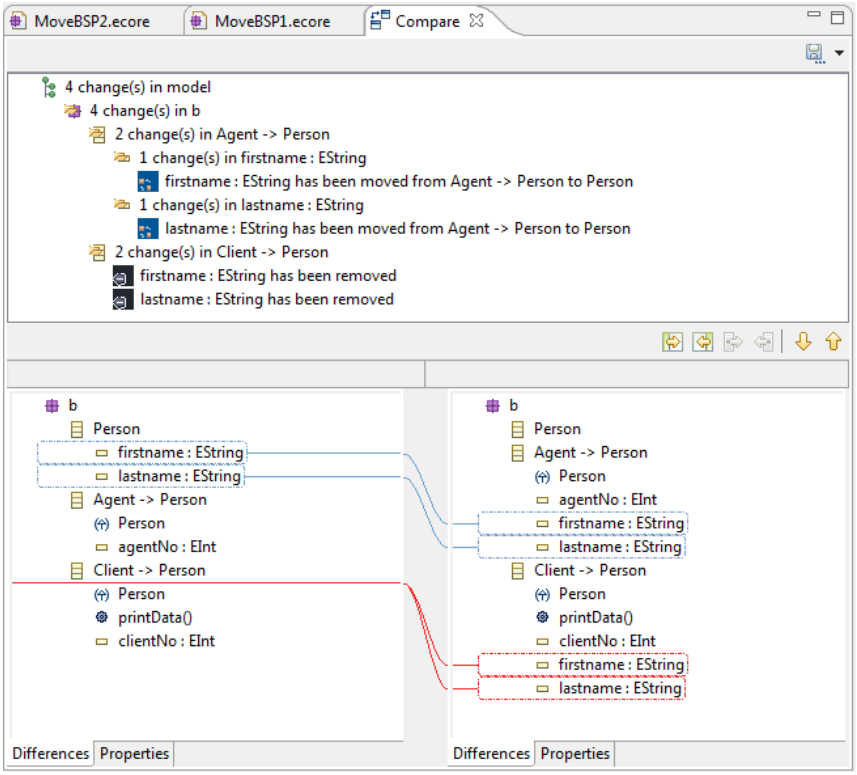

Abbildung 4.34: Differenzbericht des Vergleichs mit dem ECVerfahren mit nachträglichem Erkennen von Verschiebungen

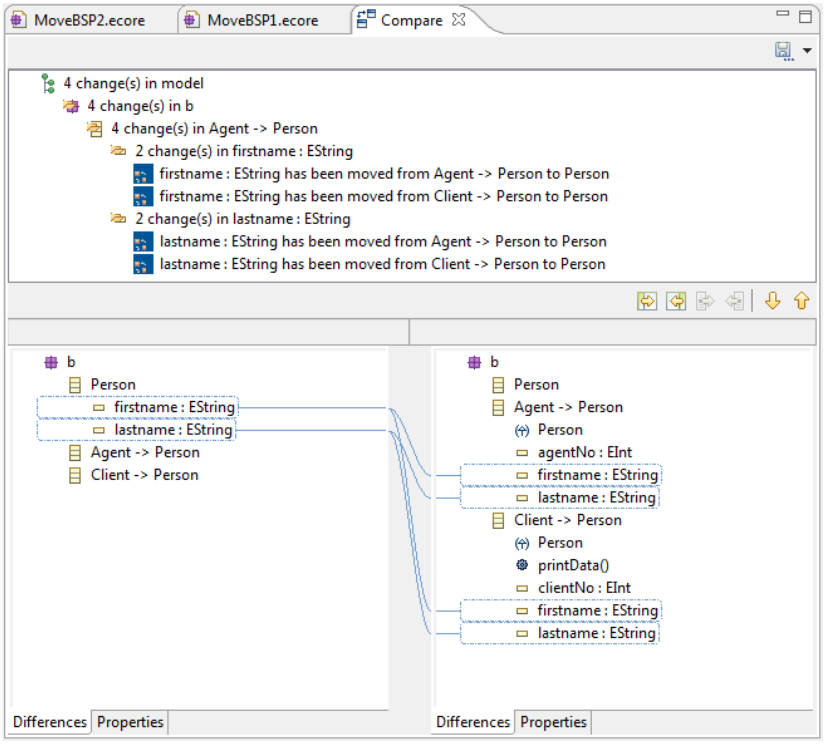

Abbildung 4.35: Differenzbericht des Vergleichs mit dem ECVerfahren mit nachträglichem Erkennen von Mehrfachverschiebungen

und eine Löschoperation zu deuten. Auch das Erkennen von Mehrfachzuordnungen ist in dem Nachbearbeitungsschritt in begrenztem Umfang möglich, auch wenn im verwendeten Korrespondenzberechnungsverfahren eine eindeutige Zuordnung bestimmt wurde. Abbildung 4.35 zeigt das Ergebnis einer kleinen Codeänderung in der Nachverarbeitungsphase: nachdem zu einem Element aus der Kandidatenliste ein korrespondierendes Element gefunden wurde, wird dennoch nach weiteren Korrespondenzpartnern gesucht. Das Erzeugen mehrerer Korrespondenzobjekte zu einem Element ist im verwendeten Korrespondenzmodell von EMF Compare (siehe Abschnitt 5.3) möglich. Dieses erweiterte Verschiebeszenario wird in dieser Arbeit jedoch nicht berücksichtigt. Um komplexere Änderungsoperationen wie Kopieren oder Mehrfachverschiebungen gut zu erkennen, wird meines Erachtens ein Verfahren benötigt, das Mehrfachzuordnungen von Elementen nativ berücksichtigt.

4.5 Ein Korrespondenzberechnungsverfahren für die Paketebene

In den bisherigen Ansätzen wurde angenommen, dass alle Klassen flach in einem Paket liegen, das als Grundpaket bezeichnet wird, und die Grundpakete der beiden Modelle korrespondieren. Um Klassendiagramme vergleichen zu können, deren Klassen sich in

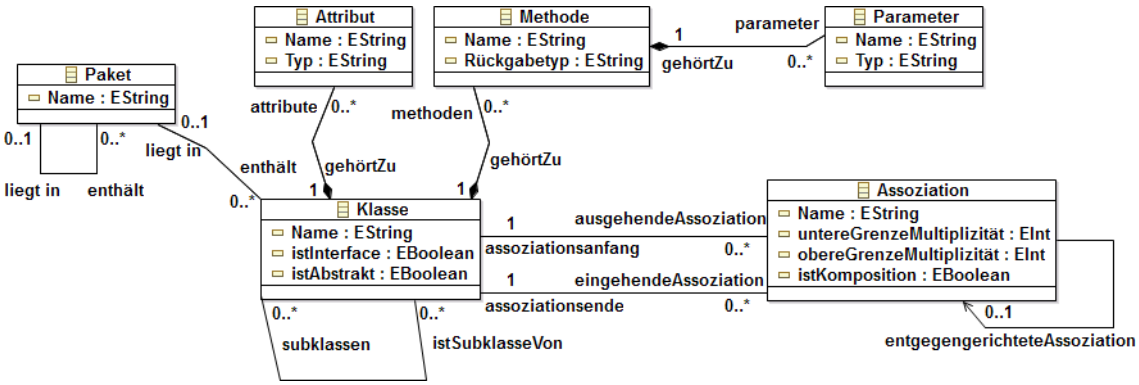

Abbildung 4.36: Erweitertes Datenmodell mit Paketen

verschiedenen Ebenen einer Pakethierarchie befinden, wird das Datenmodell um Pakete erweitert (vgl. Abbildung 4.36). Pakete können andere Pakete sowie Klassen enthalten. Jedes Paket und jede Klasse liegt in höchstens einem Paket. Pakete und Klassen, die in keinem anderen Paket liegen, liegen in dem Grundpaket des Modells. Die Grundpakete der zu vergleichenden Modelle werden weiterhin als korrespondierend betrachtet.

In den beiden vorgestellten Ansätzen zum Vergleich von Elementen auf Klassenebene gelten Unterelemente von gelöschten oder eingefügten Klassen aufgrund der vollständigen Teilbaumzuordnung ebenfalls als gelöscht bzw. eingefügt. In diesem Aspekt stimmen die vorgestellten Ansätze mit dem Vergleichsverfahren von EMF Compare überein. Während diese Einschränkung für den Vergleich von Klassen und deren Unterelementen akzeptabel ist, zeigen sich beim Vergleich von Paketen deutliche Nachteile, wie in den nächsten zwei Beispielen zu sehen ist.

Beispiel 16 (Paketevergleich mit EMF Compare)**.** Um die Auswirkungen der vollständigen Teilbaumlöschung auf Paketebene zu demonstrieren, werden nun zwei Modelle mit dem Vergleichsverfahren von EMF Compare verglichen. Das Modell B in diesem Beispiel ist identisch zu dem Modell aus Abbildung 4.25 auf Seite 116. Bei Modell A handelt es sich um eine zweite Version des Modells, die sich von der anderen lediglich darin unterscheidet, dass die Klassen `Client`, `Agent` und `Person` nicht im Grundpaket, sondern in einem Unterpaket `Persons` liegen. In Abbildung 4.37, die den Differenzbericht des Vergleichs zeigt, ist zu sehen, dass EMF Compare die drei Unterklassen des Pakets `Persons` als gelöscht und die gleichen Klassen im Grundpaket als neu eingefügt identifiziert. Aufgrund der nicht erkannten Korrespondenzen werden zwei weitere Unterschiede, Änderungen des Zieltyps zweier Assoziationen, identifiziert.

Dieses Problem im EMF Compare Verfahren zeigt sich nicht nur beim Verschieben von Klassen aus einem gelöschten Paket, sondern kann auch dann auftreten, wenn ein Paket und eine darin enthaltene Klasse an zwei verschiedene Stellen verschoben werden, wie im folgenden Beispiel gezeigt wird.

Beispiel 17 (Paketevergleich mit EMF Compare (2.Teil))**.** Die Abbildung 4.38 zeigt das Vergleichsergebnis zweier Diagramme mittels EMF Compare. Im linken Diagramm liegt

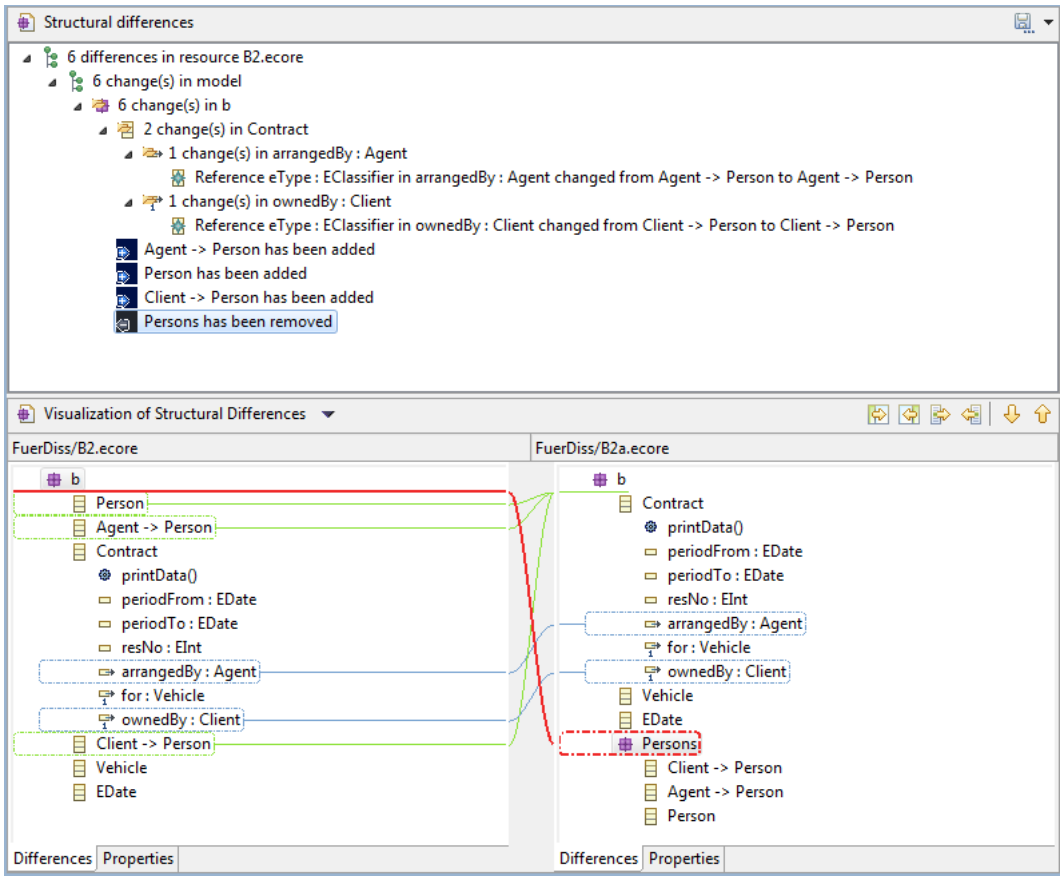

Abbildung 4.37: Differenzbericht von EMF Compare zum Vergleich der Klassendiagramme aus Beispiel 16: Die Verschiebung der Klassen `Client`, `Agent` und `Person` in ein Unterpaket `Persons` wird nicht erkannt. Da die Klassen `Agent` und `Client` nicht als korrespondierend erkannt wurden, wird außerdem berichtet, dass sich die Referenzziele der Assoziationen `arrangedBy` und `ownedBy` geändert hätten.

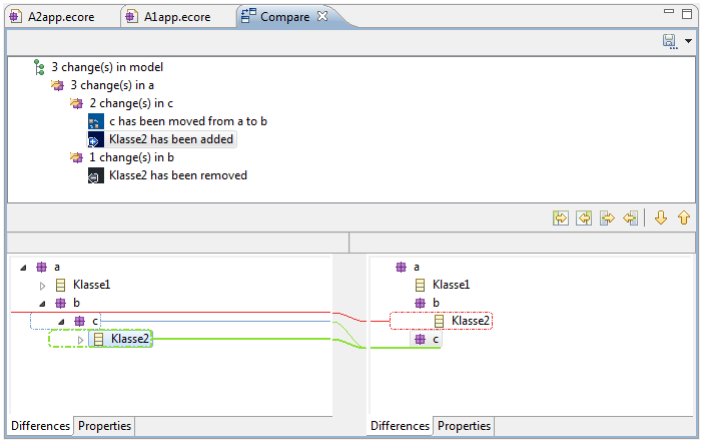

Abbildung 4.38: Differenzbericht von EMF Compare: gleichzeitiges Verschieben der Klasse `Klasse2` und des Pakets `c` wird nicht erkannt.

die Klasse `Klasse2` im Paket `c`, das ein Unterpaket des Pakets `b` darstellt. Im rechten Diagramm wurden im Vergleich dazu sowohl das Paket `c` als auch die Klasse `Klasse2` um eine Stufe in der Pakethierarchie nach oben verschoben. Das Paket `c` liegt damit auf gleicher Ebene wie das Paket `b`, die Klasse `Klasse2` liegt im Paket `b`. Das EMF Compare Verfahren erkennt zwar die Verschiebung des Pakets, nicht jedoch die Verschiebung der Klasse, die stattdessen als gelöscht und an anderer Stelle wieder eingefügt erkannt wird.

Wie die Beispiele 16 und 17 zeigen, erkennt EMF Compare nur einen Teil der Verschiebeoperationen. Da das Verschieben von Klassen in andere Pakete sowie das Einfügen und Löschen von Paketen zu den gängigen Änderungen in Klassendiagrammen zählen, ist es wünschenswert, diese Änderungen auch zuverlässig als solche zu erkennen. Um dies zu gewährleisten, werden im eigenen Ansatz alle Klassen des Klassendiagramms sowie alle Pakete unabhängig von deren Lage in Paketen miteinander verglichen, woraus sich ein zweistufiger Ansatz ergibt.

Bei dem zweistufigen Ansatz werden zunächst alle Klassen miteinander verglichen, danach erfolgt auf Basis der Klassenkorrespondenzen eine Zuordnung der Pakete. Dabei werden die Pakete des Modells A paarweise mit den Paketen des Modells B verglichen, unabhängig von der hierarchischen Struktur der Pakete. Dieser flache Vergleich der Pakete ermöglicht es, auf Paketebene Verschiebungen von Klassen und von Paketen in andere Pakete nativ zu erkennen. Die Verschiebungen werden dabei wie folgt gedeutet: Bei der Verschiebung einer Klasse wird der vollständige Teilbaum der Kompositionsstruktur mit der Klasse als Wurzel verschoben. Wird ein Paket in ein anderes Paket verschoben, werden die enthaltenen Elemente, d. h. Klassen und Pakete, nicht zwangsweise auch in das neue Paket verschoben, sondern bleiben gegebenenfalls Elemente des ehemaligen Vaterpakets des verschobenen Pakets. Im EMF Compare UI Editor wird die Visualisierung so umgesetzt, dass die Unterelemente gelöschter bzw. eingefügter Pakete weiterhin ebenfalls als gelöscht bzw. eingefügt gelten, sofern sie nicht als verschoben gekennzeichnet sind. Weiterhin gelten die Unterelemente verschobener Pakete als gemeinsam mit dem Paket verschoben (siehe Abbildung 4.39), sofern sie nicht anderweitig als verschoben,

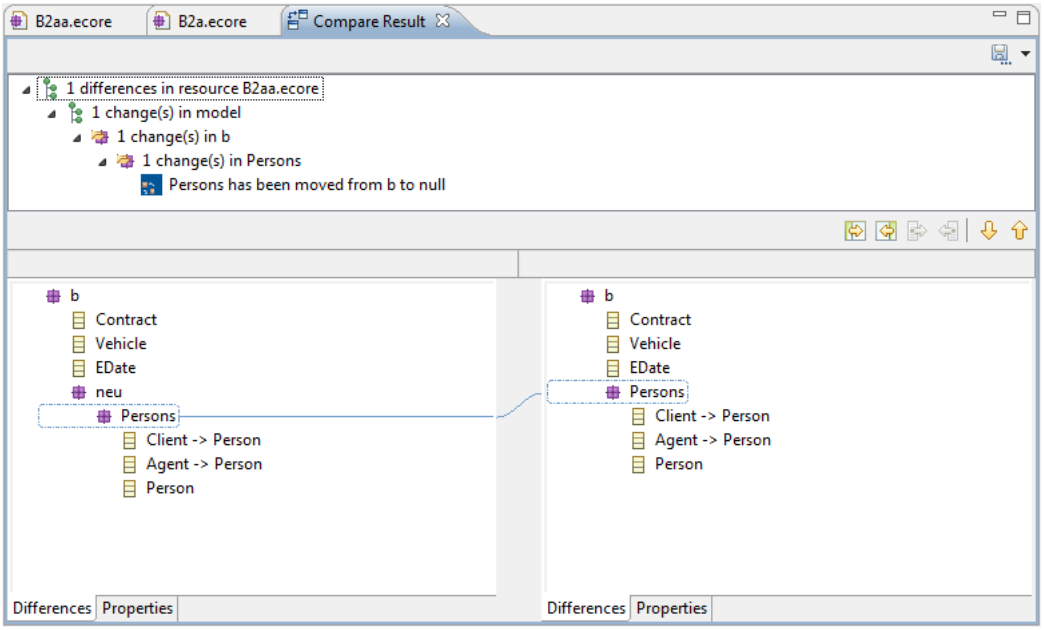

Abbildung 4.39: Verschiebung eines Pakets mit Klassen im Differenzbericht des eigenentwickelten Verfahrens. Die fehlende Referenz (`null`) in der textuellen Beschreibung der Verschiebungen ist darauf zurückzuführen, dass das Paket `neu` in dem Diagramm auf der rechten Seite nicht existiert.

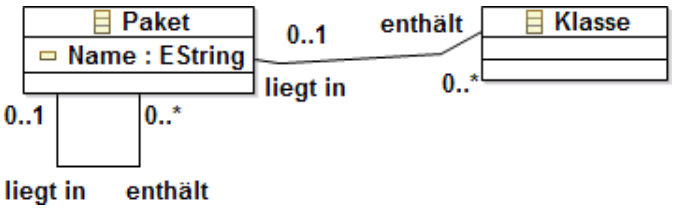

Abbildung 4.40: Der für das ähnlichkeitsbasierte Korrespondenzberechnungsverfahren auf Paketebene relevante Ausschnitt des Modells aus Abbildung 4.36

gelöscht oder eingefügt markiert sind.

Anpassung des ähnlichkeitsbasierten Verfahrens

Zur Korrespondenzberechnung auf Paketebene wird das ähnlichkeitsbasierte Verfahren aus Abschnitt 4.3 verwendet, wobei lediglich die Ähnlichkeitsberechnungen für die neuen Elemente angepasst werden. Da die Korrespondenzen für die Klassen und deren Unterelemente bereits vorliegen, ist für die Korrespondenzberechnung auf Paketebene nur ein kleiner Ausschnitt des Datenmodells aus Abbildung 4.36 relevant (siehe Abbildung 4.40). Die Ähnlichkeit von Paketen berechnet sich, wie in Formel 4.30 angegeben, zu gleichen Teilen aus der Ähnlichkeit des Paketnamens ($sim_n(a,b)$), der Ähnlichkeit der darin enthaltenen Klassen ($sim_k(a,b)$), der Ähnlichkeit der darin enthaltenen Pakete ($sim_{subP}(a,b)$) sowie der Ähnlichkeit der Vaterpakete ($sim_{superP}(a,b)$).

$$sim(a,b) = (sim_n(a,b) + sim_k(a,b) + sim_{subP}(a,b) + sim_{superP}(a,b))/4 \qquad (4.30)$$

Wie die Formel 4.30 zeigt, hängt die Berechnung der Ähnlichkeitswerte zweier Pakete a und b von den Ähnlichkeiten ihrer Vater- und Kindpakete ab. Um diese Abhängigkeiten aufzulösen, werden für die Berechnung der Werte sim_{subP} und sim_{superP} lediglich kontextfreie Eigenschaften der Pakete berücksichtigt. In der ersten Implementierung des Verfahrens werden nur die Namen der Pakete verglichen. Es bietet sich jedoch an, zusätzlich zu den Paketnamen auch die Ähnlichkeitswerte der darin enthaltenen Klassen zu berücksichtigen.

Die Ähnlichkeit der Namen kann wie für das Klassenebenenverfahren über verschiedene Methoden zum Vergleich von Zeichenketten ermittelt werden (siehe dazu Abschnitt 4.3.1). Die Ähnlichkeit zweier Klassen beträgt 1, falls die beiden Klassen korrespondieren, ansonsten 0. Die Ähnlichkeit zweier Pakete wird über den Vergleich der Namen berechnet. Analog zum ähnlichkeitsbasierten Verfahren auf Klassenebene werden beim Vergleich zweier Elementmengen die Einzelähnlichkeiten der Zuordnungen summiert und durch die Zahl der Elemente der größeren der beiden Elementmengen geteilt, damit der Ähnlichkeitswert im Intervall $[0, 1]$ liegt. Auch hier ist eine unterschiedliche Gewichtung der Einzelkriterien möglich. Als Auswahlverfahren wird das in Abschnitt 4.3.2 beschriebene Maxwertverfahren verwendet, wobei auf eine Anpassung der Ähnlichkeiten im 1-Kontext verzichtet wird. Die Abbildungen 4.41 und 4.42 zeigen die Ergebnisse, wenn die Unterschiede der Modelle aus den Beispielen 16 und 17 jeweils mit diesem Verfahren berechnet werden.

Zusammenfassung der Vorgehensweise

Der Ablauf der Korrespondenzberechnung für Klassendiagramme mit Paketen lässt sich wie folgt zusammenfassen:

1. **Korrespondenzberechnung auf Klassenebene:**
   Die korrespondierenden Klassen werden unabhängig davon, in welchem Paket sie sich befinden, mit einem Klassenebenenverfahren, wie in Abschnitt 4.2 oder 4.3 beschrieben, berechnet.

2. **Korrespondenzberechnung auf Paketebene:**
   Unter Einbeziehung der korrespondierenden Klassen in die Ähnlichkeitsberechnung der Pakete werden die korrespondierenden Pakete mit dem in diesem Abschnitt beschriebenen Verfahren bestimmt. Alternativ können aber auch andere Verfahren entwickelt und in dieser Stufe eingesetzt werden.

3. **Zusammenführung der Ergebnisse**
   Die getrennt berechneten Korrespondenzen der Klassen und der Pakete müssen abschließend in einer gemeinsamen Zuordnung zusammengeführt werden. Dieser Schritt wird in Abschnitt 5.3.2 nach Einführung des verwendeten Korrespondenzmodells beschrieben.

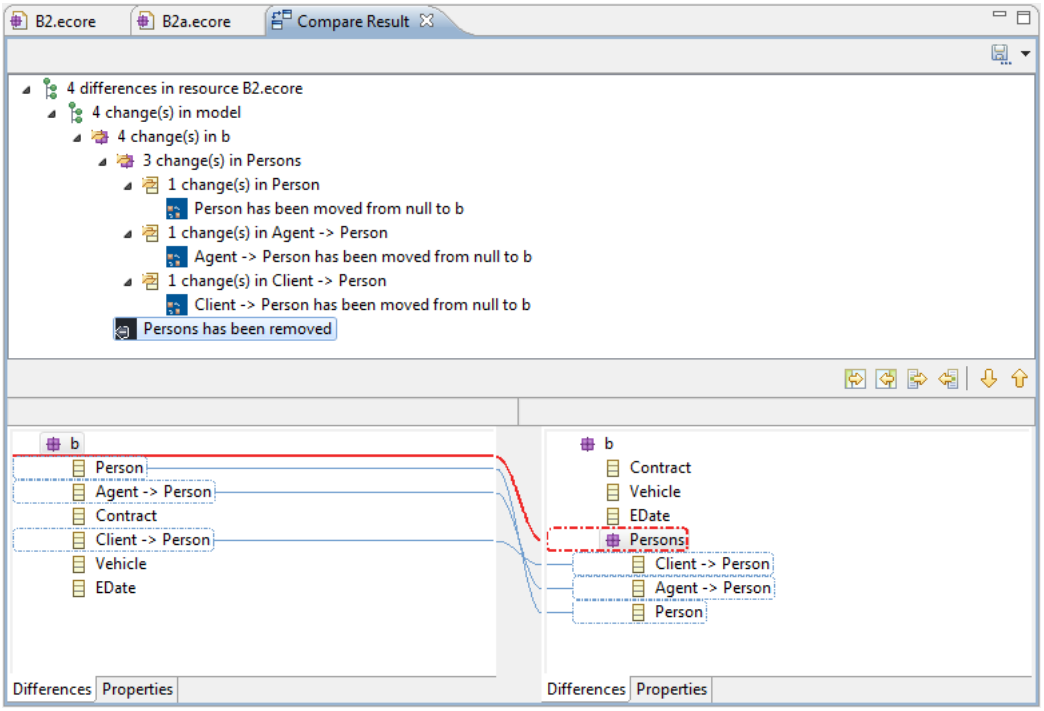

Abbildung 4.41: Differenzbericht des zweistufigen Verfahrens (ECVerfahren + ähnlichkeitsbasiertes Paketebenenverfahren) für den Vergleich der Modelle aus Beispiel 16. Die Verschiebungen der Klassen `Person`, `Client` und `Agent` werden erkannt. Die fehlenden Referenzen (`null`) in der textuellen Beschreibung der Verschiebungen sind darauf zurückzuführen, dass das Paket `Persons` gelöscht ist.

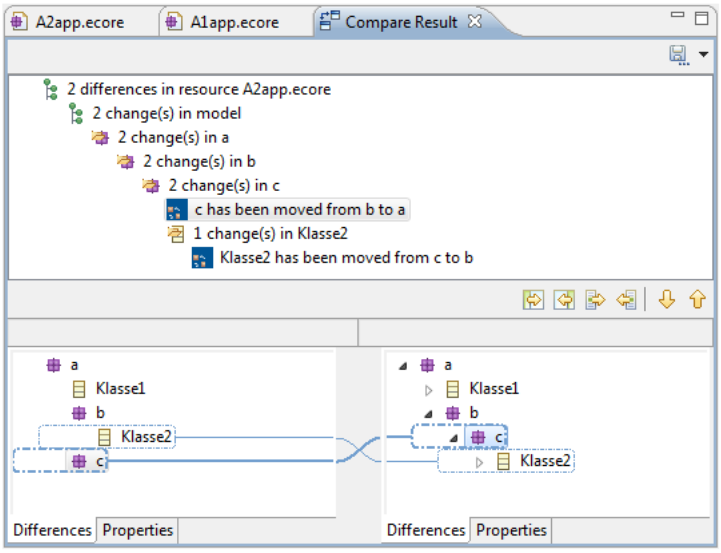

Abbildung 4.42: Differenzbericht des eigenen Paketebenenverfahrens zu Beispiel 17 bei doppelter Namensgewichtung

5 Implementierung im EMF-Rahmenwerk

Um die verschiedenen Ansätze vergleichen zu können, ist ein Rahmenwerk hilfreich, in dem Klassendiagramme erstellt und verglichen sowie die Ergebnisse des Vergleichs visualisiert werden können. Damit können die Verfahren auf die gleichen Beispiele angewandt werden und die unterschiedlichen Ergebnisse betrachtet werden. Bei der Wahl der Umgebung fiel die Entscheidung aus folgenden Gründen auf Eclipse und das Eclipse Modeling Framework [SBPM09].

Open-Source-Projekt mit großer Community Bei der Entwicklungsumgebung Eclipse für Java handelt es sich um ein Open-Source-Projekt. Dies bedeutet, dass der Quellcode der Umgebung frei zugänglich ist und die Mitarbeit am Projekt ausdrücklich gewünscht ist. So wird Eclipse nicht nur von vielen Anwendern genutzt, sondern es hat sich auch eine sehr aktive Community gebildet, die Eclipse pflegt und in verschiedenen Projekten Komponenten mit neuer Funktionalität entwickelt. Da Eclipse sich mit seiner Vielzahl an Komponenten zu einem Alleskönner zu entwickeln scheint, ist davon auszugehen, dass Eclipse auch in Zukunft eine bedeutende Rolle als Entwicklungsumgebung einnehmen wird. Eine Übersicht über die aktuellen Projekte findet sich unter http://www.eclipse.org.

Baukastensystem Eclipse besteht aus einem Kernpaket, das, basierend auf dem OSGi Framework Equinox, weitere Komponenten, sogenannte Plugins, einbinden kann. So wird die Eclipse Plattform durch die beiden Plugins *Java Development Tooling (JDT)* und *Plug-in Developer Environment (PDE)* zur Entwicklungsumgebung für Java mit Unterstützung zur Plugin-Entwicklung erweitert (siehe Abbildung 5.1). Weitere auch eigene Komponenten lassen sich an vordefinierten Anknüpfungspunkten (extension points) hinzufügen ([SBPM09, Kapitel 1]), so dass sich der Umfang und die Funktionalität gut anpassen lassen.

Nutzen vorhandener Komponenten Ein Teil der Komponenten, die für das Rahmenwerk benötigt werden, sind dank des Eclipse Modeling Projekts bereits als Eclipse-Plugins vorhanden. Dazu zählen verschiedene Editoren für Ecore-Dateien sowie Teile des Vergleichsverfahrens EMF Compare (siehe Abschnitt 3.2.2). Das EMF Compare Projekt startete 2007 als Teil der EMFT (Eclipse Modeling Framework Technology) [Bru] und wurde im Juni 2009 in das EMF Projekt aufgenommen. Seitdem wird EMF Compare

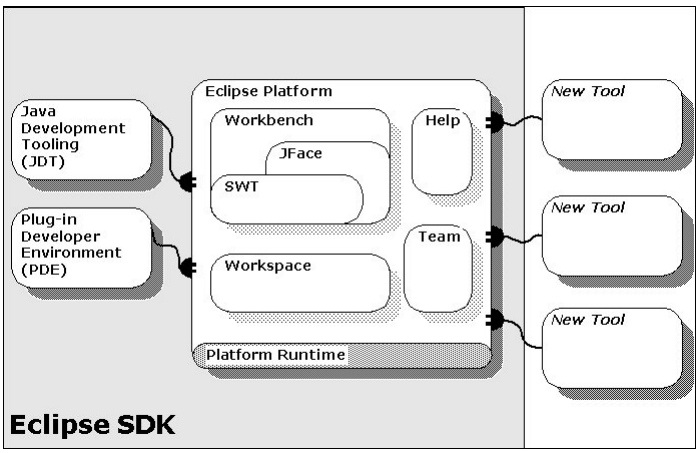

Abbildung 5.1: Architektur der Eclipse Plattform [Ecl09, Platform Plug-in Developer Guide, Programmer's Guide, Platform architecture]

zusammen mit den Eclipse Modeling Releases ausgeliefert. Durch das Verwenden bestehender Komponenten ist es möglich, sich auf die Kernfunktion, das Berechnen der Korrespondenzen zu konzentrieren. Darüber hinaus ist es interessant, das generische Verfahren von EMF Compare mit den eigenentwickelten Verfahren zu vergleichen. Welche Komponenten für das Rahmenwerk genutzt werden, wird in Abschnitt 5.1 beschrieben.

Erweiterbarkeit für andere Diagrammarten Bei einer zukünftigen Erweiterung der Vergleichsverfahren auf andere Diagrammarten kann es von Vorteil sein, dass in Eclipse für eine Vielzahl an Diagrammarten bereits eine Editor-Unterstützung vorhanden ist. Darüber hinaus können mit EMF und der Metamodellsprache Ecore leicht Modelle für eigene Diagrammarten erstellt und dazugehörige Baumeditoren generiert werden (siehe [Rau08]). Unter Verwendung der GMF-Werkzeuge des Graphical Modeling Projects [GEF] lassen sich zu den erstellten Modellen mit begrenztem Aufwand graphische Editoren bauen [Gro09].

Nach einem Überblick über den Aufbau des Rahmenwerks werden im Folgenden die beteiligten Komponenten und deren Zusammenspiel genauer beschrieben. Der Schwerpunkt liegt dabei auf den verschiedenen Modellen, die im Laufe des Vergleichprozesses entstehen. Abbildung 5.2 zeigt die Transformationskette der extern und intern verwendeten Modelle: Die beiden zu vergleichenden Klassendiagramme liegen im Ecore-Datenmodell vor, das für die Durchführung der Korrespondenzberechnungsverfahren in ein internes Datenmodell `DDiagramm` überführt wird (Abschnitt 5.2). Zwischen den Korrespondenzberechnungsverfahren der Klassen- und Paketebene wird ein Objekt der Klasse `Zuordnung` als internes Austauschformat für die berechneten korrespondierenden Elemente verwendet. Schließlich wird das Gesamtergebnis als EMF-Korrespondenzmodell (Abschnitt 5.3) exportiert und daraus ein EMF-Differenzmodell (Abschnitt 5.4) abgeleitet.

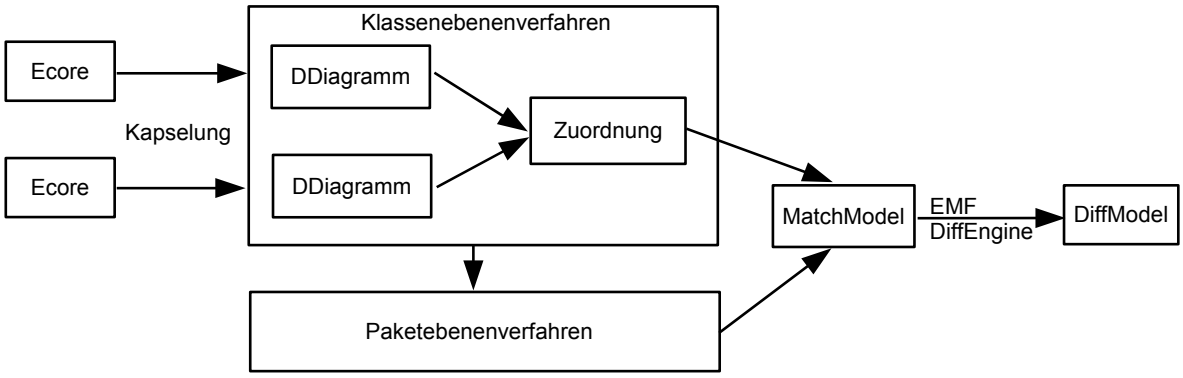

Abbildung 5.2: Transformationskette der Modelle

5.1 Aufbau des Rahmenwerks

Die in Kapitel 4 beschriebenen Korrespondenzberechnungsverfahren wurden als Plug-in implementiert, das die Entwicklungsumgebung Eclipse über den Anknüpfungspunkt `org.eclipse.ui.views` um eine Sicht erweitert. Die neue Sicht stellt eine Benutzerschnittstelle zur Verfügung, über die die zu vergleichenden Klassendiagramme sowie das zu verwendende Korrespondenzberechnungsverfahren ausgewählt werden können. Für das Rahmenwerk werden verschiedene Eclipse-Plugins von EMF und EMF Compare benutzt. Das betrifft folgende Bereiche:

- beliebige Ecore-Editoren zur Modellierung der Klassendiagramme
- die Datenstruktur `MatchModel` zur Speicherung eines Korrespondenzmodells
- den 2-Wegevergleich von EMF Compare sowie das Differenzmodell `DiffModel`
- den EMF Compare UI Editor zur Visualisierung der Differenzen in der Baumsicht

Abbildung 5.3 zeigt einen Überblick, wie diese Komponenten mit dem eigenentwickelten Plugin ein Rahmenwerk zum Vergleich von Klassendiagrammen bilden. Wie dieses Rahmenwerk benutzt wird, wird im nächsten Abschnitt beschrieben, wobei für ausführliche Screenshots auf den Anhang A.1 verwiesen wird. Die Bezeichnungen Korrespondenzmodell sowie Differenzmodell beziehen sich in diesem Kapitel, soweit nicht anders angegeben, auf das `MatchModel` und das `DiffModel` von EMF Compare.

5.1.1 Einsatz des Rahmenwerks

Zur Modellierung der Ecore-Klassendiagramme können beliebige Ecore-Editoren genutzt werden. Als Beispiele werden hier der Ecore Tree Editor, der mit einer Baumsicht auf der Kompositionsstruktur der Ecore-Datei arbeitet, und der graphische Ecore Diagram Editor von Ecore Tools[1] genannt. Für den Vergleich zweier Diagramme wird die Sicht

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

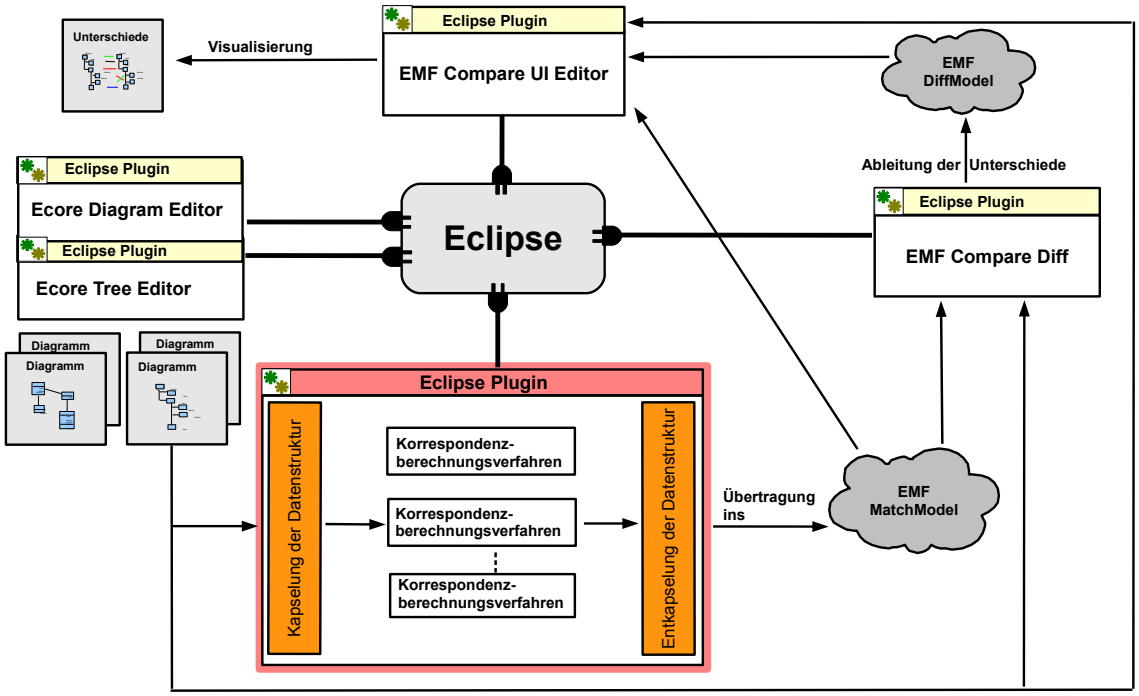

Abbildung 5.3: Aufbau des Rahmenwerks zum Vergleich von Korrespondenzberechnungsverfahren

des eigenentwickelten Plugins benötigt. Falls diese Sicht in Eclipse noch nicht offen ist, kann sie über das Menü *Windows* → *Show view* geöffnet werden. Über diese Benutzerschnittstelle, die in Abbildung 5.4 zu sehen ist, kann der Anwender die beiden zu vergleichenden Diagramme auswählen und entscheiden, mit welchem Verfahren die Korrespondenzen berechnet werden sollen.

Die verfügbaren Verfahren sind in die Kategorien „Einstufige Ansätze“ und „Mehrstufige Ansätze“ unterteilt. Als einstufiger Ansatz wurde das Korrespondenzberechnungsverfahren von EMF Compare angebunden. Bei dem mehrstufigen Ansatz sind jeweils ein Korrespondenzberechnungsverfahren auf Klassenebene und das anschließende Paketebenenverfahren auszuwählen. Als Klassenebenenverfahren wurden das edierkostenbasierte ECVerfahren, dessen Brute-Force-Variante BFVerfahren und das ähnlichkeitsbasierte Verfahren aus Kapitel 4.3 implementiert. Als Paketebenenverfahren steht bislang die ähnlichkeitsbasierte Variante aus Abschnitt 4.5 zur Auswahl. Je nach Verfahren existieren zusätzliche Optionen. So kann z. B. beim ECVerfahren das nachträgliche Erkennen von Verschiebungen auf Klassenebene und/oder ein Schwellenwert aktiviert werden (vgl. Abschnitte 4.4 und 4.2.7). Für alle mehrstufigen Verfahren kann festgelegt werden, mit welchem Verfahren Zeichenketten verglichen werden.

Löst der Anwender einen Diagrammvergleich aus, werden die Ecore-Klassendiagramme gegebenenfalls[2] in ein von dem ausgewählten Verfahren benötigtes internes Modell überführt (siehe Abschnitt 5.2.2). Im Anschluss daran werden die korrespondierenden Ele-

[2]Bei dem einstufigen Ansatz EMF Compare wird dies nicht benötigt.

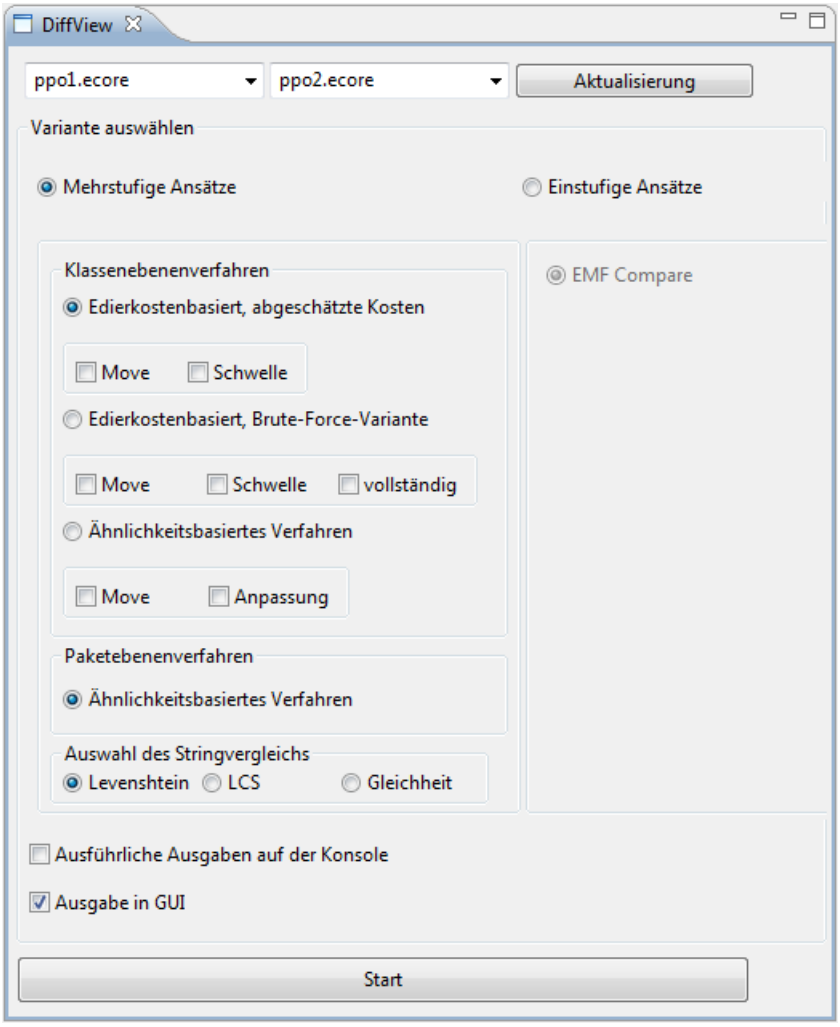

Abbildung 5.4: View in Eclipse zur Auswahl des Korrespondenzberechnungsverfahrens.

mente der beiden Diagramme durch das gewählte Verfahren bestimmt und in das Korrespondenzmodell von EMF, das `MatchModel` (siehe 5.3) übertragen. Das so erstellte Modell dient zusammen mit den beiden Diagrammen als Eingabe für die `GenericDiffEngine` von EMF Compare, die die Differenzen ableitet. Unabhängig vom ausgewählten Verfahren zur Berechnung der Korrespondenzen wird zur Ableitung der Differenzen aus dem Korrespondenzmodell das gleiche Differenzberechnungsverfahren verwendet. Dies stellt die Vergleichbarkeit der erzielten Ergebnisse sicher. Stimmen zwei Korrespondenzmodelle überein, sind auch deren Differenzmodelle identisch. Umgekehrt steht fest, dass Unterschiede in Differenzmodellen auch auf Unterschiede in den Korrespondenzmodellen zurückzuführen sind. Als Ergebnis der Differenzberechnung entsteht ein Differenzmodell, das `DiffModel` von EMF.

Beide entstandenen Modelle, das Korrespondenzmodell und das Differenzmodell, werden dem EMF Compare UI Editor übergeben (für den Aufruf siehe Listing 5.6, S. 176) und die berechneten Unterschiede in einer Baumsicht angezeigt (siehe Abbildung 5.24,

S. 178). Eine graphische Visualisierung der Unterschiede ist bereits in Entwicklung und für nachfolgende Eclipse-Versionen angekündigt ([Jäh08, S. 55]). Die berechnete Differenzdatei, die das `MatchModel` und das `DiffModel` enthält, lässt sich über einen Save-Button als `*.emfdiff` nach XML serialisieren.

5.1.2 Aufbau des Match-Plugins

Nach Beschreibung der Funktionsweise des Rahmenwerks soll nun ein Überblick über den Aufbau des Korrespondenzberechnungs-Plugins gegeben werden. Die Implementierung gliedert sich in drei Pakete: in die für die Schnittstelle nach außen relevanten Klassen, die verwendete Datenstruktur und die Verfahren (siehe Abbildung 5.5).

Das Paket **view** stellt die Schnittstelle des Plugins zur Außenwelt dar. Über die Aktivator-Klasse des Plugins und die Eclipse-Sichten in den Klassen `DiffView`, `ECParameterView` und `SimParameterView` wird das Plugin an die Eclipse Plattform angebunden. Die `DiffView` stellt die Benutzerschnittstelle zur Verfügung, in der die Vergleiche ausgelöst werden. Die Klassen `ECParameterView` und `SimParameterView` repräsentieren weitere Sichten, über die Änderungen am Kostenmodell des ECVerfahrens bzw. an der Gewichtung der Ähnlichkeitswerte des Ähnlichkeitsverfahrens vorgenommen werden können. Der Zugriff auf die Ressourcen der zu vergleichenden Diagramme und das Erstellen der `DDiagramm`-Objekte wurde in die Hilfsklasse `Diagrammfinder` ausgelagert. In der Klasse `DiffView` werden die Diagramme dem ausgewählten Verfahren übergeben. Die Implementierungen der Verfahren befinden sich im Paket **verfahren**, was im nachfolgenden Abschnitt beschrieben wird. Letztendlich wird zu dem berechneten `MatchModel` ein Objekt der Klasse `DiffAnzeige` erstellt, das für das Erstellen des `DiffModels` aus dem `MatchModel` sowie für das Öffnen des EMF Compare UI Editors verantwortlich ist. Das Paket **datenstruktur** enthält die Klassen der intern verwendeten Datenstruktur, die im Abschnitt 5.2.2 behandelt wird.

Strukturierung der Verfahren im Paket de.uni.bayreuth.uhrig.ecorediff.verfahren

Das Paket **verfahren** beinhaltet die verschiedenen Korrespondenzberechnungsverfahren, die sich in `Klassenebenenverfahren`, `Paketebenenverfahren` und `Einstufiges-Verfahren` unterteilen lassen. Ein Klassenebenenverfahren berechnet eine Zuordnung für den Vergleich von Klassendiagrammen, die lediglich ein Grundpaket besitzen, in dem alle Klassen liegen. Diese Verfahren können eigenständig genutzt werden, die Ergebnisse in ein Korrespondenzmodell übertragen und visualisiert werden. Darüber hinaus können die Klassenebenenverfahren auch als Vorstufe zu einem Paketebenenverfahren eingesetzt werden. Dabei werden alle Klassen unabhängig davon, in welchem Paket sie sich befinden, miteinander verglichen. Ein Paketebenenverfahren ordnet unter Vorgabe einer Zuordnung der Klassen die Pakete zweier Diagramme zu, wobei korrespondierende Klassen sich nicht im gleichen Paket befinden müssen (vgl. Abschnitt 4.5). Als dritte Variante sind Anbindungen von einstufigen Verfahren möglich, die das Berechnen der Korrespondenzen für Klassen und Pakete in einem Schritt behandeln.

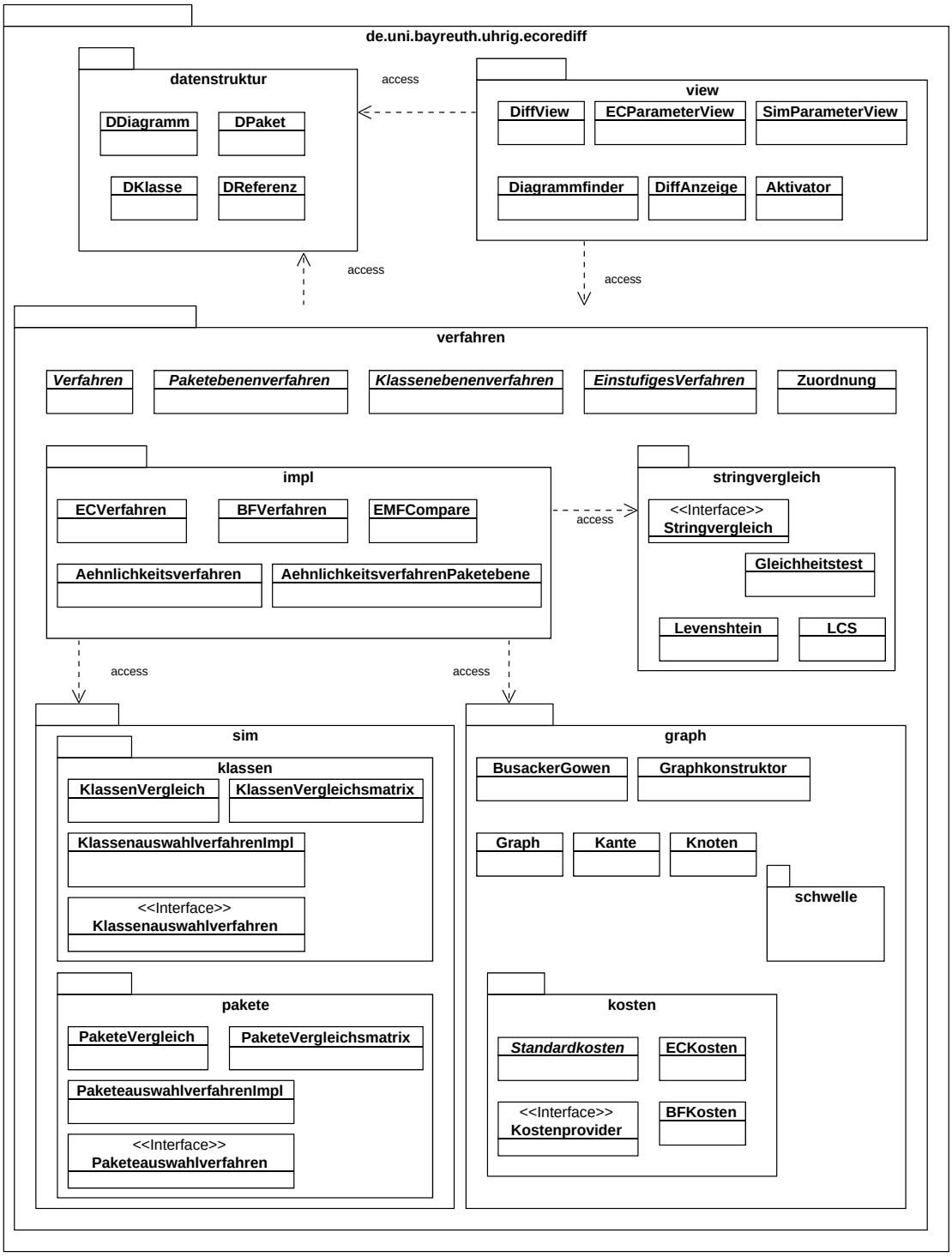

Abbildung 5.5: Übersicht über die Pakete des Plugins mit den wichtigsten import-Anweisungen.

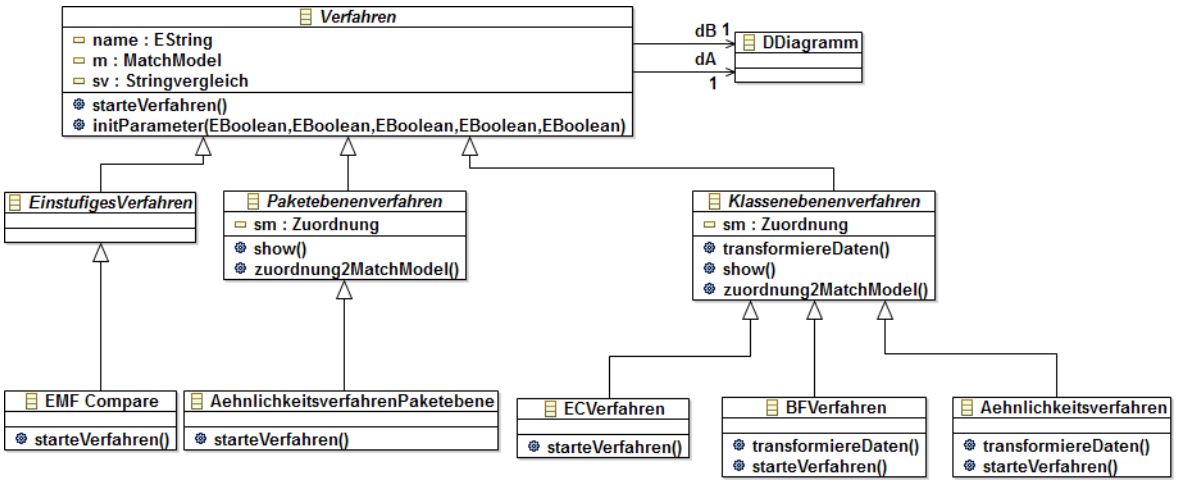

Abbildung 5.6: Vererbungshierarchie der Verfahren

Gemeinsame Eigenschaften dieser Verfahren, wie der Name des Verfahrens, die beiden Diagramme und die Signatur der Startmethode wurden in einer abstrakten Oberklasse `Verfahren` zusammengefasst. Einen Überblick über die Vererbungshierarchie der Verfahren zeigt die Abbildung 5.6. In der Klasse `Klassenebenenverfahren` wird in der Methode `transformiereDaten()` die Umwandlung des Ecore-Datenmodells in das von allen Klassenebenenverfahren verwendete interne Datenmodell realisiert. Falls zusätzliche verfahrensspezifische Vorbereitungen nötig sind, werden diese in der redefinierten Methode der Unterklasse nach einem `super()`-Aufruf umgesetzt. Die Datenstruktur `Zuordnung` dient als Schnittstelle zur Weitergabe der Zuordnung der Klassen an das nachfolgende `Paketebenenverfahren`. Während sich die abstrakten Klassen der Verfahren direkt im Paket befinden, liegen die Implementierungen der verschiedenen Verfahren im Unterpaket **verfahren.impl**. Die beiden weiteren Unterpakete **graph** und **sim** werden jeweils nur von den edierkostenbasierten bzw. ähnlichkeitsbasierten Verfahren verwendet. Im Unterpaket **verfahren.stringvergleich** liegen gemeinsam genutzte Klassen zum Vergleich von Zeichenketten. Dies beinhaltet ein Interface `Stringvergleich` und die Klassen `LCS`, `Levenshtein` und `Gleichheitstest`, die das Interface mit den in Abschnitt 4.2.1 beschriebenen Verfahren zum Vergleich von Zeichenketten implementieren. Das Interface beinhaltet eine Methode zur Berechnung eines Ähnlichkeitswerts, der von den ähnlichkeitsbasierten Verfahren genutzt wird, sowie eine Methode zur Berechnung der Edierkosten, die von den edierkostenbezogenen Verfahren verwendet wird (siehe Abbildung 5.7).

Die Klassen des Unterpakets de.uni.bayreuth.uhrig.ecorediff.verfahren.graph

Sowohl das ECVerfahren als auch das BFVerfahren arbeiten mit dem in Kapitel 4.2.4 beschriebenen Netzwerkgraphen. Die dabei verwendeten Klassen `Graph`, `Knoten` und `Kante` wurden bereits in Abbildung 4.24 auf Seite 111 dargestellt. Die Graphen der beiden Verfahren unterscheiden sich im Wesentlichen durch unterschiedliche Kostenbewertungen der Kanten. Der Aufbau der Vergleichsgraphen wird daher von einer Instanz

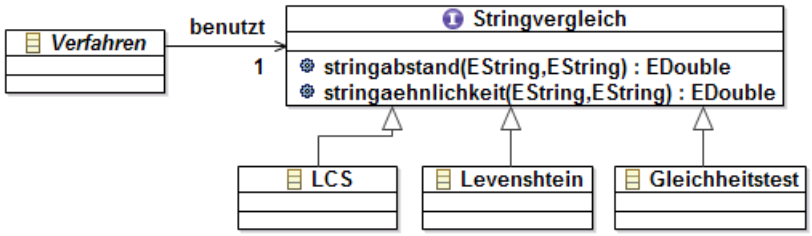

Abbildung 5.7: Zusammenhang zwischen den Verfahren und den verschiedenen Methoden des Zeichenkettenvergleichs

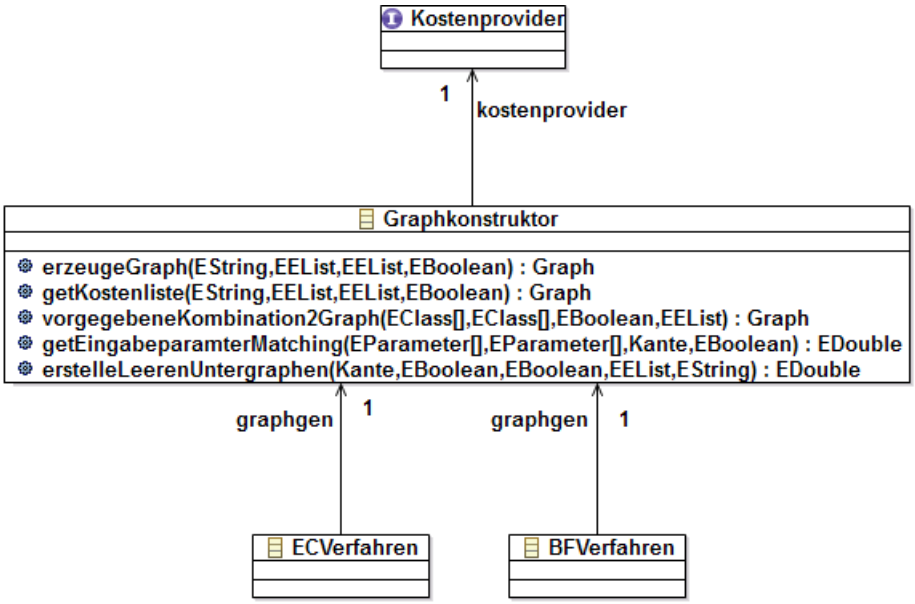

Abbildung 5.8: Graphkonstruktor

der Klasse `Graphkonstruktor` übernommen, der die Berechnung der Kantenkosten an einen `Kostenprovider` delegiert.

Abbildung 5.8 zeigt die verschiedenen öffentlichen Methoden der Klasse `Graphkonstruktor` zum Aufbau verschiedener Graphen. Dies beinhaltet das Erstellen eines Vergleichsgraphen zu zwei Objektlisten, die Erstellung eines Untergraphen für eine Kante zu zwei gegebenen Objektlisten, die Zuordnung der Übergabeparameter für zwei Arrays von Parametern und das Erstellen eines leeren Untergraphen für die Unterobjekte eines gelöschten Objekts. Für die Kostenkontrolle der berechneten Zuordnung bzw. das BFVerfahren wird noch eine weitere Methode angeboten, die zu einer vorgegebenen Zuordnung von Objekten einen Netzwerkgraphen aufbaut, um die Kosten dieser Zuordnung zu berechnen.

Für die unterschiedlichen Kosten der Verfahren existieren im Unterpaket **graph.kosten** zwei verschiedene Implementierungen des Interfaces `Kostenprovider`. `ECKosten` und `BFKosten` erben gemeinsame Methoden, wie die Berechnung der kontextfreien Kosten, von der gemeinsamen abstrakten Oberklasse `Standardkosten` (vgl. Abbildung 5.9). Da

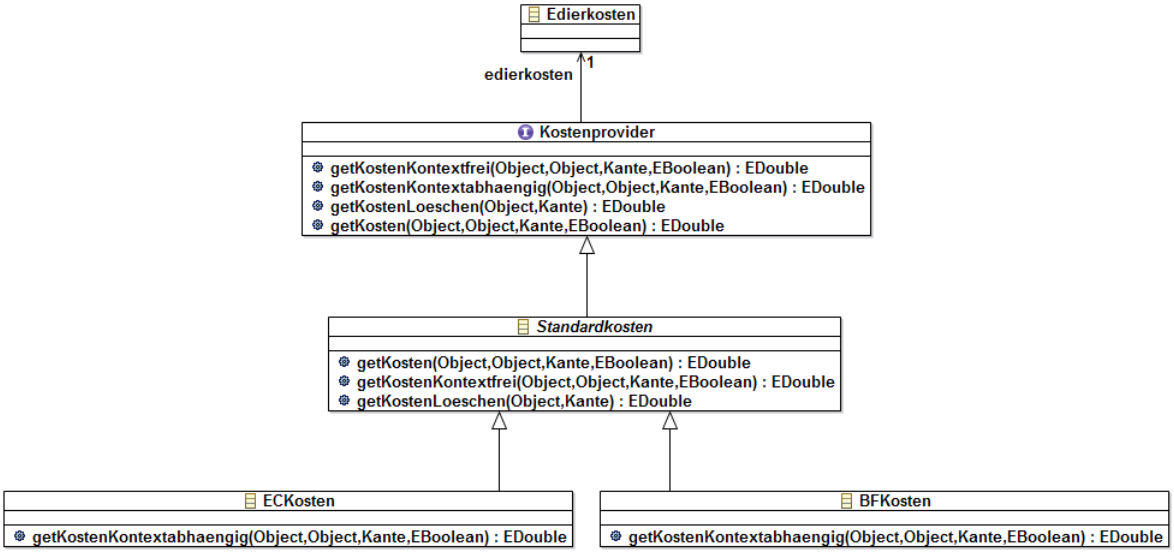

Abbildung 5.9: Kostenprovider

die kontextabhängigen Kosten bei beiden Verfahren unterschiedlich berechnet werden, wird dafür jeweils eine eigene Implementierung benötigt. Die Methoden rechnen jedoch nicht mit explizit in der Implementierung vorgegebenen Kosten, z. B. für das Löschen einer Klasse, sondern beziehen sich auf die Kosten, die von einem Objekt der Klasse `Edierkosten` vorgegeben werden. Neue Konfigurationen der Edierkosten lassen sich so einfach über ein anderes Objekt der Klasse `Edierkosten` in den `Kostenprovider` einbinden. Die Einstellung der Parameter kann auch zur Laufzeit über die in Abbildung 5.10 dargestellte Sicht vorgenommen werden

Außerdem enthält das Paket **graph.kosten** die Klasse `BusackerGowen`, die eine Implementierung des in Abschnitt 4.2.5 beschriebenen Verfahrens zur Lösung des Netzwerkproblems darstellt. In dem Unterpaket **graph.schwelle**, das in Abbildung 5.5 vereinfacht dargestellt ist, liegen die Klassen zur Realisierung von Schwellenwerten.

Die Klassen des Unterpakets de.uni.bayreuth.uhrig.ecorediff.verfahren.sim

Das Paket **sim** enthält die für die ähnlichkeitsbasierten Verfahren benötigten Klassen und Interfaces, differenziert nach Klassenebenenverfahren und Paketebenenverfahren in zwei verschiedenen Unterpaketen. Auf oberster Ebene des Pakets befinden sich lediglich zwei abstrakte Hilfsklassen, die von den Verfahren gemeinsam genutzte Methoden zur Verfügung stellen. Die Abbildungen 5.12 und 5.11 zeigen jeweils die Verbindungen der Klassen in den Unterpaketen **sim.klassen** und **sim.pakete**. In den Klassen `Klassenvergleich` bzw. `Paketevergleich` sind die Einzelvergleiche zweier Klassen bzw. Pakete als nichtöffentliche Vergleichsmethoden realisiert. Die Gewichtung der Ähnlichkeitswerte kann dabei über eine Sicht (vgl. Abbildung 4.27, Seite 123) angepasst werden. Objekte der Klassen `Klassenvergleichsmatrix` bzw. `Paketevergleichsmatrix` fassen die Einzelvergleiche in einer Matrix zusammen und binden ein geeignetes Auswahlverfahren vom Typ `KlassenAuswahlverfahren` bzw. `PaketeAuswahlverfahren` ein. Das

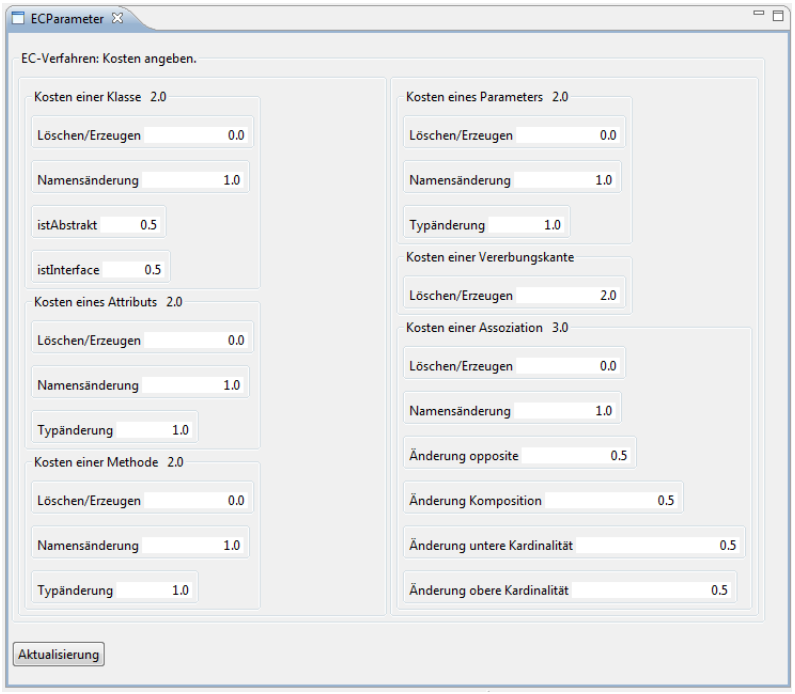

Abbildung 5.10: Sicht zur Konfiguration der Kosten des ECVerfahrens und der Brute-Force-Variante

Auswahlverfahren legt fest, wie aus der Vergleichsmatrix mit den berechneten Ähnlichkeitswerten Paare von korrespondierenden Elementen ausgewählt werden. Da für die Berechnung einer Zuordnung der Klassen auch eine iterative Variante vorgesehen ist (vgl. Abschnitt 4.3), enthalten die Klassen `Klassenvergleich` bzw. `Klassenvergleichsmatrix` entsprechende weitere Methoden zum Aktualisieren der Zwischenergebnisse zwischen den Iterationsschritten sowie eine Methode zum Speichern der temporären Teilzuordnungen.

5.2 Das interne Datenmodell

Wie bereits in Abschnitt 5.1 erwähnt, ist es für die Durchführung der Klassenebenenverfahren nötig, die Klassendiagramme, die verglichen werden sollen, in ein internes Datenmodell zu überführen. Bevor die Gründe und die Umsetzung in Abschnitt 5.2.2 beschrieben werden, soll im nächsten Abschnitt zunächst der Zusammenhang zwischen den Ecore-Klassendiagrammen und dem Ecore-Metamodell dargestellt werden.

5.2.1 Das Ecore-Metamodell als Modell für Klassendiagramme

Modelle des Eclipse Modeling Frameworks sind Instanzen von Ecore-Modellen, die mit der Sprache Ecore modelliert werden. Das Ecore-Metamodell, das die Modellierungssprache Ecore beschreibt (vgl. Abbildung 5.13), wurde ursprünglich auf dem Meta Object

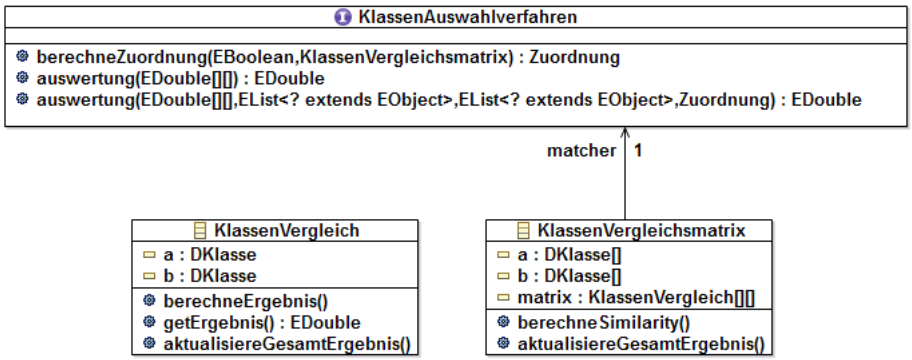

Abbildung 5.11: Klassenvergleich

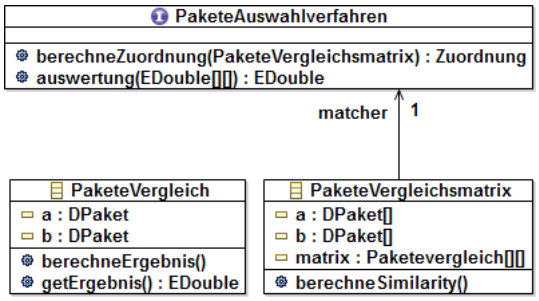

Abbildung 5.12: Paketevergleich

Facility-Standard der Object Management Group OMG [Obj06] aufgebaut und ist in etwa mit Essential MOF (EMOF) vergleichbar ([Gro09, S. 663]). Da Klassendiagramme den Ecore-Modellen hinreichend ähnlich sind, wurde in diesem Ansatz darauf verzichtet, mit der Sprache der Metaebene ein Ecore-Modell für Klassendiagramme zu erstellen. Als Metamodell für die Klassendiagramme wird im Vergleichsrahmenwerk stattdessen eine Teilmenge des Ecore-Metamodells verwendet (siehe Abbildung 5.14). Diese Teilmenge ist in Abbildung 5.15 dargestellt, wobei folgende Zuordnung zwischen den Ecore-Metamodellelementen und den Elementen eines Klassendiagramms stattfindet:

- **EPackage**. Modellelemente vom Typ `EPackage` entsprechen Paketen. Diese besitzen einen Namen und sind über die bidirektionale Assoziation `eSubPackages/ eSuperPackages` mit den Unterpaketen und dem Oberpaket verbunden. Außerdem kann ein Paket Elemente vom Typ `EClass` enthalten (bidirektionale Assoziation `eClassifiers/ ePackage` über `EClassifier` zu `EClass`).

- **EClass**. Ein Element vom Typ `EClass` besitzt einen Namen und stellt eine konkrete Klasse dar, falls die Attribute `abstract` und `interface` false sind. Falls `abstract` true ist, handelt es sich um eine abstrakte Klasse. Falls das Attribut `interface` true ist, liegt ein Interface vor. Außerdem verweist eine unidirektionale Assoziation `eSuperTypes` gegebenenfalls auf weitere Elemente vom Typ `EClass`. Ist das Ziel der Assoziation eine (abstrakte) Klasse, wird die Beziehung als Vererbungskante zur Oberklasse interpretiert. Im Fall eines Interfaces handelt es sich um

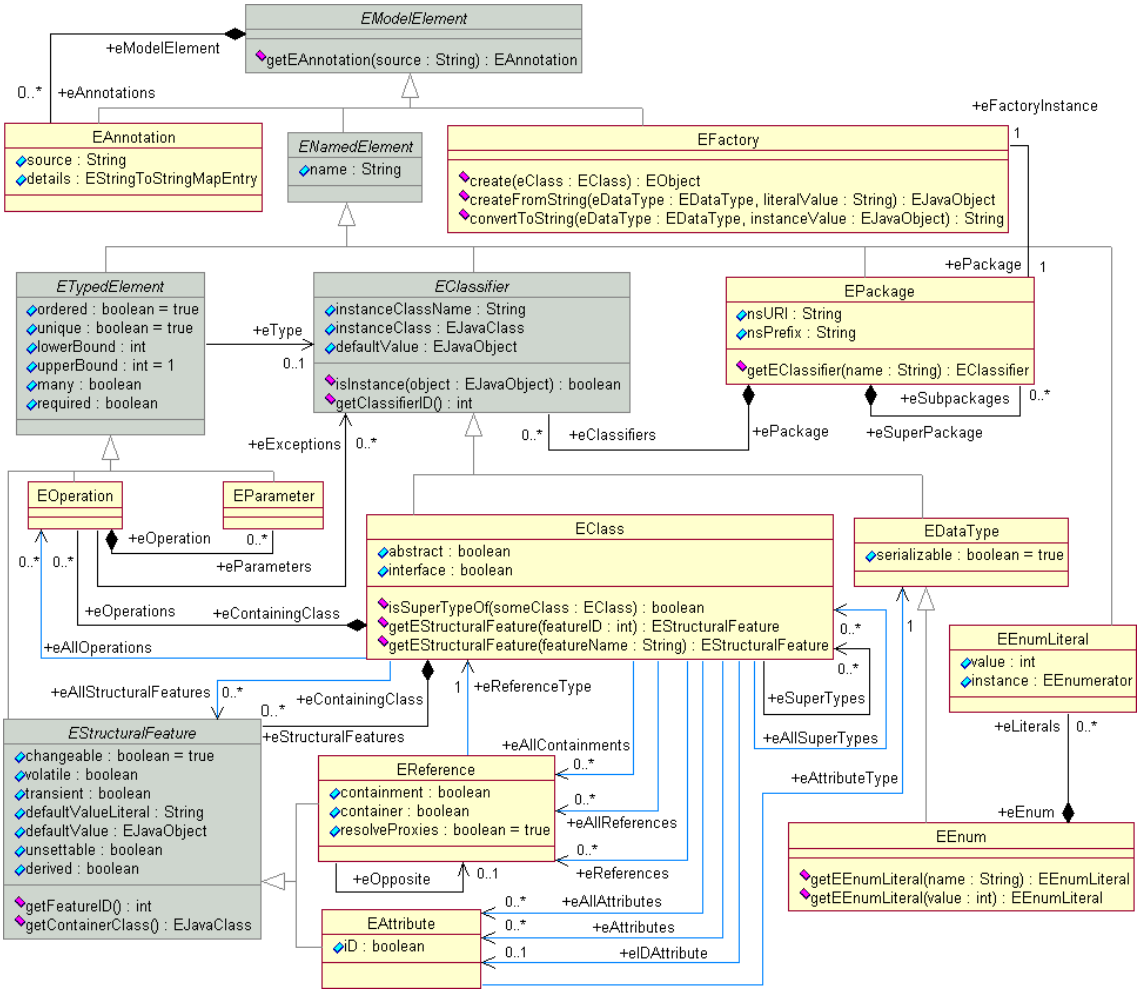

Abbildung 5.13: Das Ecore-Metamodell [Ecl09]

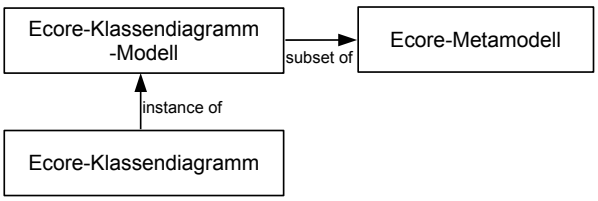

Abbildung 5.14: Die Metamodellebenen des Vergleichrahmenwerks

eine implements-Beziehung. In der Liste der `eSuperTypes` eines `EClass`-Elements können sich daher mehrere Elemente befinden, allerdings wird Einfachvererbung bei Klassen unterstellt. Ein `EClass`-Element kann außerdem Attribute (unidirektionale Assoziation `eAttributes` zu `EAttribute`) und Methoden (bidirektionale Assoziation `eOperations/ eContainingClass` zu `EOperation`) besitzen. Unidirektionale Assoziationen sind von der Klasse bzw. dem Interface über die unidirektionale Assoziation `eReferences` zu `EReference` navigierbar. Für die Attribute und unidirektionale Assoziationen ist die entsprechende Rückrichtung über die Assoziation `eContainingClass` navigierbar, die von `EStructuralFeature` geerbt wird.

- **EAttribute**. Ein `EAttribute`-Element entspricht einem Attribut einer Klasse oder eines Interfaces. Ein Attribut besitzt einen Namen und einen Typ (`eType`). Die Eigenschaften `lowerBound` und `upperBound` des `ETypedElements` werden für Attribute nicht berücksichtigt.

- **EOperation**. Methoden von Klassen oder Interfaces werden durch Objekte der Klasse `EOperation` dargestellt. Eine Methode besitzt einen Namen, einen Rückgabetyp (`eType`) sowie möglicherweise Übergabeparameter vom Typ `EParameter`. Zu den Übergabeparametern besteht eine in beide Richtungen navigierbare Assoziation `eOperation/eParameters`. Auch für Methoden werden die Eigenschaften `lowerBound` und `upperBound` nicht berücksichtigt.

- **EParameter**. Dieses Element besitzt lediglich einen Namen und einen Typ (`eType`) und entspricht einem Übergabeparameter einer Methode.

- **EReference**. Eine `EReference` wird auf eine unidirektionale Assoziation abgebildet. Die Referenz `eReferenceType`[3] verweist auf das Assoziationsende, das eine Klasse oder ein Interface sein kann. Eine bidirektionale Assoziation entspricht in dem Modell zwei entgegengerichteten unidirektionalen Assoziationen, deren `opposite`-Link auf die jeweils andere Assoziation gesetzt ist. Bei einem `EReference`-Element entsprechen die Eigenschaften `lowerBound` und `UpperBound` den Kardinalitäten der Assoziation. Eine Assoziation kann eine Kompositionsbeziehung darstellen (`containment`).

Beim Vergleich der Diagramme wird jedoch nur ein Teil der Elemente und Eigenschaften des Ecore-Metamodells berücksichtigt. Das Ecore-Metamodell enthält eine Reihe abgeleiteter Eigenschaften und Assoziationen, wie `eAllAttributes` oder `eAllSuperTypes`. In diesem Fall werden nicht nur die in der Klasse angegebenen Attribute zurückgeliefert, sondern auch Attribute, die von Oberklassen geerbt werden. Analog liefert `eAllSuperTypes` nicht nur die direkten Obertypen, sondern auch alle Typen, zu denen indirekt eine Vererbungsbeziehung besteht. Da sich der Vergleich nur auf in den Elementen gespeicherten Informationen bezieht, werden die abgeleiteten Eigenschaften

[3] ein Verweis auf das `eType`-Element von `ETypedElement`, allerdings als `EClass` gecastet (vgl. [SBPM09])

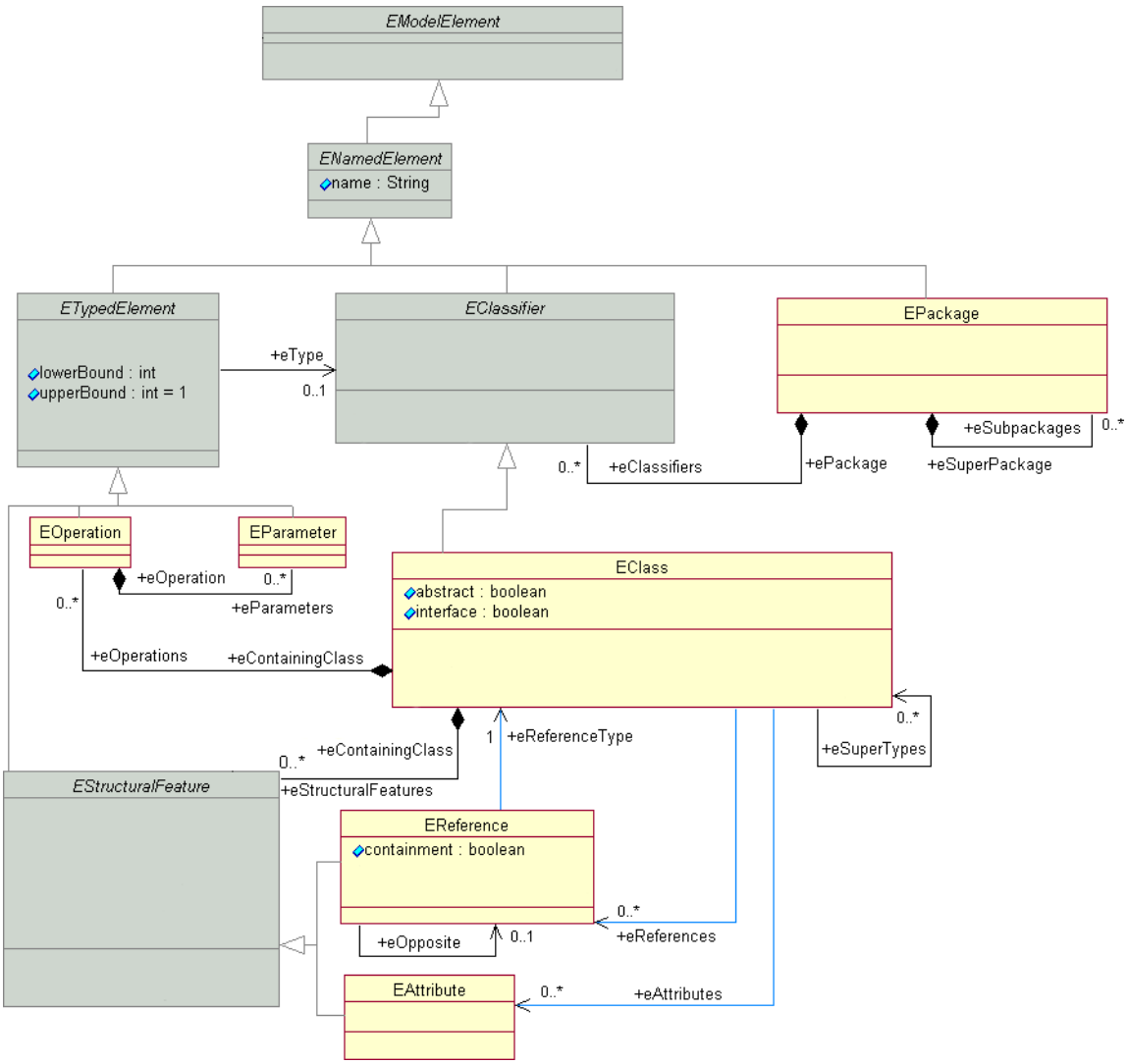

Abbildung 5.15: Das reduzierte Ecore-Metamodell

nicht verwendet. Darüber hinaus wurden weitere Elemente wie Annotationen und Datentypen nicht in den Ansatz miteinbezogen, um die Komplexität der Implementierung zu reduzieren. Die wesentlichen Bestandteile von Klassendiagrammen wurden umgesetzt, wobei bestimmte Eigenschaften exemplarisch herausgegriffen wurden. Der Umfang des gewählten Modellausschnitts lässt sich jedoch bei Bedarf erweitern, insbesondere ist es leicht, weitere kontextunabhängige Eigenschaften in das Edierkostenmodell sowie in die Ähnlichkeitsberechnung der ähnlichkeitsbasierten Verfahren miteinzubeziehen. Andere noch fehlende Elemente von Klassendiagrammen lassen sich durch das Ecore-Metamodell nicht intuitiv darstellen. Beispielsweise ist das Modellieren abstrakter Methoden oder von Sichtbarkeiten nur über Annotationen möglich.

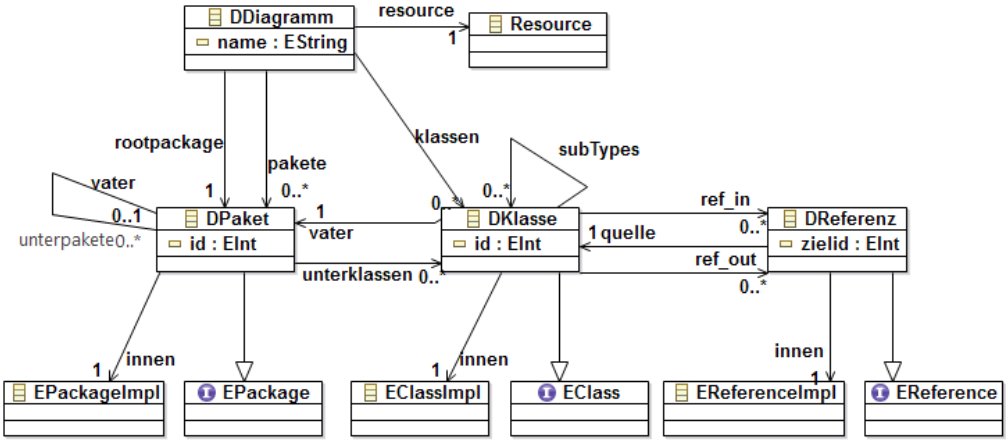

Abbildung 5.16: Das interne Datenmodell

5.2.2 Kapselung des Ecore-Modells in eine interne Datenstruktur

Da die Assoziationen der Kompositionsstruktur im Ecore-Metamodell bidirektional realisiert sind, kennt ein Element nicht nur seine Unterelemente, sondern es kann auch auf sein Oberelement zugreifen. Der Zugriff auf das sogenannte Parentelement wird zum Beispiel bei der Erstellung des Differenzmodells verwendet (siehe Abschnitt 5.4.2). Andere Assoziationen wie `eSuperTypes` und `eReferenceType` sind unidirektional umgesetzt.

Sofern diese Referenzen nur in eine Richtung verwendet werden, handelt es sich dabei um eine effiziente Lösung. Da wir beim Vergleich auch eingehende Assoziationen und Unterklassen einer Klasse in die Ähnlichkeitsberechnung miteinbeziehen, entsteht an dieser Stelle jedoch zusätzlicher Aufwand: Vor dem eigentlichen Aufruf des Vergleichsverfahrens werden die beiden Ecore-Modelle jeweils in ein internes Modell übertragen, das zusätzliche Informationen enthält, die für die Berechnung der Korrespondenzen benötigt werden. Dabei werden alle Elemente vom Typ `EClass`, `EPackage` und `EReference` jeweils in einem neuen eigenen Typ gekapselt. Dieser enthält neben den zusätzlichen Werten als Objektfeld auch eine Referenz auf das ursprüngliche Ecore-Modellelement und erfüllt über eine implements-Vereinbarung die Schnittstelle der ursprünglichen Klasse. Damit sind die Objekte des neuen Typs überall dort einsetzbar, wo ein Objekt der Ecore-Modellelementklasse erwartet wird. Abbildung 5.16 zeigt die Zusammenhänge der neuen Typen `DKlasse`, `DPaket` und `DReferenz` im Paket `de.uni.bayreuth.uhrig.ecorediff.datenstruktur`.

Für jedes der beiden zu vergleichenden Diagramme wird das interne Modell in einem `DDiagramm`-Objekt gespeichert. Dieses speichert den Namen des Diagramms und eine Referenz auf die zugehörige Ressource im Eclipse Ressourcenmodell sowie Listen aller im Diagramm enthaltenen Pakete und Klassen. Das Grundpaket ist in der Liste der Pakete nicht enthalten, sondern wird als Wurzelelement des internen Modells extra gespeichert. Die neuen Typen `DPaket`, `DKlasse` und `DReferenz` besitzen neben weiteren Assoziationen auch eindeutige `ids` und `matchids` zum schnelleren Objektvergleich und zum leichteren Identifizieren von Korrespondenzpartnern, sowie andere technische Hilfsfelder

für den Aufbau des Korrespondenzmodells, die in der Abbildung 5.16 nicht dargestellt sind. Nachdem die Korrespondenzen berechnet wurden, wird das ursprüngliche Modell wieder entkapselt, so dass die originalen Modellelemente in das Korrespondenzmodell übertragen werden.

Die nachfolgenden Ausführungen über das EMF `MatchModel` und `DiffModel` beziehen sich auf den Quellcode der verwendeten EMF Compare Version 1.0.0. Entsprechende Dokumentationen zu EMF Compare stehen leider nicht zur Verfügung.

5.3 Das Korrespondenzmodell

Die ermittelten Korrespondenzen werden in einer EMF Compare eigenen Datenstruktur, dem `MatchModel` gespeichert. Dieses verknüpft die korrespondierenden Elemente zweier Diagramme paarweise miteinander. Nach [BP08] handelt es sich bei dem Korrespondenzmodell von EMF um eine Sonderform des Weaving Modells. Dabei werden keine Kopien der Elemente erzeugt, sondern mit Verweisen auf die Modellelemente gearbeitet. Dies liefert ein schlankes Modell, dessen Aussage nur in Verbindung mit den beiden zu vergleichenden Diagrammen vollständig erfasst werden kann. Das Korrespondenzmodell ist unabhängig von dem verwendeten Metamodell und wird im Rahmen des EMF Compare Verfahrens für alle Ecore-basierten Modelle eingesetzt. Die Abbildung 5.17 zeigt eine leicht vereinfachte Darstellung des Metamodells für Korrespondenzmodelle aus dem Paket `org.eclipse.emf.compare.match.metamodel`, das im folgenden Absatz genauer beschrieben wird.

5.3.1 Aufbau des Korrespondenzmodells

Ein Korrespondenzmodell enthält neben den Ressourcen der beiden verglichenen Modelle eine Liste der Korrespondenzen und eine Liste der Einzelelemente auf oberster Ebene. Für die Verknüpfung der korrespondierenden Elemente wird ein Element vom Typ `Match2Elements`[4] eingeführt. Ein Objekt dieser Klasse stellt eine Implementierung einer Korrespondenz dar und wird im Folgenden auch als Korrespondenzobjekt bezeichnet. Ein Korrespondenzobjekt besitzt Referenzen auf die beiden korrespondierenden Modellelemente aus dem linken und aus dem rechten Klassendiagramm. Die Kompositionsstruktur der Diagramme wird auch im Korrespondenzmodell nachgebildet. Dazu verweisen die Korrespondenzobjekte über eine Referenz `getSubmatchedElements` auf die entsprechenden Korrespondenzobjekte ihrer Unterelemente. Einzelelemente werden in einer Liste von Objekten des Typs `UnmatchElement` verwaltet, die im Folgenden auch als Einzelobjekte bezeichnet werden. Ein Einzelobjekt verweist auf ein nichtkorrespondierendes Modellelement aus dem rechten oder linken Klassendiagramm. Das Attribut `Side` gibt mit `LEFT` bzw. `RIGHT` an, ob das Einzelelement aus dem linken oder rechten Modell stammt.

[4] Im Gegensatz zur Klasse `Match3Elements` zur Realisierung einer Korrespondenz aus dem 3-Wege-Vergleich

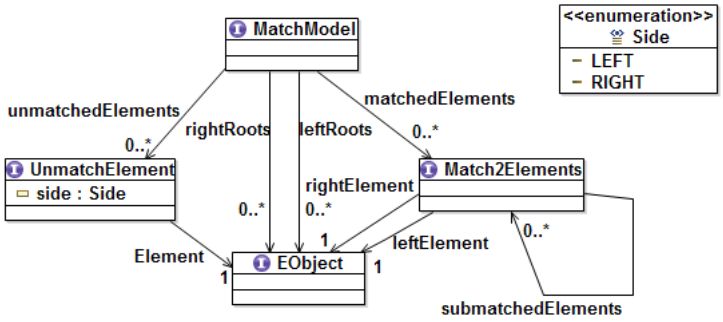

Abbildung 5.17: Aufbau des Korrespondenzmodells

Das nachstehende Listing 5.1 beschreibt die initialen Schritte bei der Erstellung eines Korrespondenzmodells. Nachdem ein `MatchModel` über die `MatchFactory` erzeugt wurde, werden die Ressourcen der beiden Diagramme festgelegt. Danach wird für die Wurzelelemente der beiden Diagramme ein Korrespondenzobjekt erstellt. Beim Vergleich von Klassendiagrammen entsprechen die Wurzelelemente den beiden Grundpaketen, von denen angenommen wird, dass diese auf jeden Fall korrespondieren. Die Referenzen auf das linke und rechte Element werden dementsprechend auf die beiden Grundpakete gesetzt. Das so erstellte Korrespondenzobjekt wird über den `add`-Befehl der `EFactory` ins Korrespondenzmodell eingetragen. Alle nachfolgenden Korrespondenzobjekte werden als Unterelemente dieses ersten Korrespondenzobjekts hinzugefügt.

Listing 5.1: Erzeugung und Initialisierung eines `MatchModels`

```
MatchModel m = MatchFactory.eINSTANCE.createMatchModel();
m.getLeftRoots().addAll(res1.getContents());
m.getRightRoots().addAll(res2.getContents());
Match2Elements e = MatchFactory.eINSTANCE.createMatch2Elements();
e.setLeftElement(package1);
e.setRightElement(package2);
EFactory.eAdd(m, "matchedElements", e);
```

Beim Einfügen korrespondierender Elemente in das Korrespondenzmodell muss die Kompositionsstruktur des rechten Diagramms berücksichtigt werden. Dabei ergeben sich folgende Besonderheiten: Unterelemente von gelöschten Elementen, die ebenfalls gelöscht werden, werden dabei nicht mehr separat als gelöschte Elemente aufgeführt, da EMF Compare das Löschen eines Elements als Löschen inklusiv des darunter liegenden Teilbaums deutet. Falls Unterelemente von gelöschten Elementen jedoch nicht gelöscht werden, sondern mit Unterelementen eines anderen Elements korrespondieren, wird für sie an der entsprechenden Stelle des Korrespondenzmodells ein Korrespondenzobjekt eingefügt. In diesem Fall handelt es sich um verschobene Elemente, die auch daran zu erkennen sind, dass die jeweiligen Vaterelemente der korrespondierenden Elemente nicht korrespondieren.

5.3.2 Übertragung der Korrespondenzen ins Korrespondenzmodell

Für den mehrstufigen Ansatz dienen Objekte der Klasse `Zuordnung` als Austauschformat zwischen dem Klassenebenenverfahren und dem Paketebenenverfahren. Bei den ähnlichkeitsbasierten Verfahren werden die ermittelten Korrespondenzen direkt in das Austauschformat eingetragen. In den edierkostenbasierten Klassenebenenverfahren ist hingegen noch ein Zwischenschritt nötig, der im nächsten Abschnitt beschrieben wird. Im Anschluss daran wird erklärt, wie die Ergebnisse des Klassenebenenverfahrens und des Paketebenenverfahrens zu einer hierarchischen Zuordnung zusammengesetzt werden.

Auslesen der Zuordnung aus dem hierarchischen Vergleichsgraphen

Nach Durchführung des `ECVerfahrens` oder des `BFVerfahrens` liegen die Informationen über die ermittelten Korrespondenzen in Form des Netzwerkflusses im Vergleichsgraphen vor und müssen in das interne Austauschformat `Zuordnung` exportiert werden. Dies enthält jeweils eine Liste der Korrespondenzobjekte auf Klassenebene und eine Liste der Einzelobjekte aus beiden Modellen. Die im Folgenden beschriebene Übertragung findet in der Objektmethode `public Zuordnung GraphtoSubMatch()` der Klasse `Graph` statt.

Wie in Abschnitt 4.2.5 beschrieben, legt jede innere Kante, die eine Einheit des Netzwerkflusses transportiert, eine Korrespondenz zwischen zwei Elementen fest oder identifiziert ein Element als Einzelelement. Beginnend auf der Klassenebene des Graphen wird daher zu jeder inneren Kante mit Flussstärke 1 entweder ein Korrespondenzobjekt oder ein Einzelobjekt des Korrespondenzmodells erstellt.

Falls die Kante den Epsilonknoten des Netzwerkgraphen inzidiert, wird ein Einzelobjekt erzeugt, das auf die Klasse des anderen inzidenten Knotens verweist. Für das Setzen des Side-Attributs, das die Zugehörigkeit zum linken oder rechten Modell ausdrückt, ist es relevant, ob die Kante vom Epsilonknoten ausgeht oder in den Epsilonknoten eingeht. Dabei muss auch der Drehsinn des Graphen berücksichtigt werden, der angibt, ob die Diagramme zu Beginn des Verfahrens vertauscht wurden. In einem nicht gedrehten Graphen entsteht aus einer Kante $a \rightarrow \epsilon$ ein linkes Einzelelement. Falls der Graph gedreht ist, führt die gleiche Kante zu einem rechten Einzelelement. Eine vollständige Fallunterscheidung ist in Abbildung 5.18 aufgestellt. Die erstellten Einzelelemente der Klassenebene werden in der Liste der Einzelobjekte der Zuordnung abgespeichert. Da nur die Wurzelelemente gelöschter Teilbäume im Korrespondenzmodell berücksichtigt sind, werden die Untergraphen der Kante nicht durchlaufen.[5]

Zu einer Kante, die zwei normale Knoten verbindet, wird ein Korrespondenzobjekt erzeugt, wobei bei der Festlegung des rechten und linken Elements die Orientierung des Graphen berücksichtigt wird. Um für die Erstellung des Korrespondenzmodells die Navigation durch die korrespondierenden Klassen zu erleichtern, werden die beiden Klassen gegenseitig mit ihrem Korrespondenzpartner verlinkt. Für jedes Korrespondenzobjekt erfolgt ein rekursiver Abstieg in die Untergraphen der Kante. Dabei wird der Unter-

[5] Anmerkung: Dadurch lassen sich aus einem gelöschten Teilbaum verschobene Elemente im nachträglichen Verfahren zur Identifikation von Verschiebungen nicht erkennen. Siehe Beispiel am Ende dieses Kapitels.

innere Kante mit 1-Fluss	ungedreht	gedreht
a —1→ ε	UnmatchElement element: a Side: LEFT	UnmatchElement element: a Side: RIGHT
ε —1→ a	UnmatchElement element: a Side: RIGHT	UnmatchElement element: a Side: LEFT
a —1→ b	Match2Elements leftElement: a rightElement: b	Match2Elements leftElement: b rightElement: a

Abbildung 5.18: Zusammenhang zwischen den inneren Kanten des Netzwerkgraphen auf Klassenebene und den Elementen des Korrespondenzmodells

Klassenebene: 1.Unterebene :	ungedreht ungedreht	gedreht gedreht	gedreht ungedreht	ungedreht gedreht
a —1→ ε	UnmatchElement element: a Side: LEFT		UnmatchElement element: a Side: RIGHT	
ε —1→ a	UnmatchElement element: a Side: RIGHT		UnmatchElement element: a Side: LEFT	
a —1→ b	Match2Elements leftElement: a rightElement: b		Match2Elements leftElement: b rightElement: a	

Abbildung 5.19: Zusammenhang zwischen den inneren Kanten des Netzwerkgraphen auf der ersten Unterebene und den Elementen des Korrespondenzmodells

graph für eingehende Referenzen jedoch nicht berücksichtigt. Da die Assoziationen sowohl in einem Untergraphen der ausgehenden Klasse als auch in einem Untergraphen der Zielklasse enthalten sind, wird so verhindert, dass die Assoziationen zweimal ins Korrespondenzmodell übertragen werden. Im Gegensatz zu den anderen Elementen werden die Referenzen an dieser Stelle bereits entkapselt, da die in der Kapselung enthaltenen Zusatzinformationen im weiteren Verlauf des Verfahrens nicht mehr benötigt werden. Korrespondenzobjekte, die in tieferen Ebenen entstehen, werden nach dem Rücksprung aus der Rekursion als Unterelement an das Korrespondenzobjekt der aufrufenden Ebene angehängt. Einzelobjekte aller Ebenen werden direkt in die Zuordnung übertragen.

Aufgrund des verwendeten Modells ergibt sich eine maximale Rekursionstiefe von 2 über den Abstieg Klassenebene → Methodenebene → Ebene der Methodeneingabeparameter. Beim Auslesen der Korrespondenzen aus dem Vergleichsgraphen der ersten Unterebene muss die Orientierung des Graphen berücksichtigt werden. Eine Vertauschung der Seiten ist dann nötig, falls der Graph auf Klassenebene und der Graph der Unterebene entgegengesetzt orientiert sind (siehe Abbildung 5.19). Der Vergleichsgraph der zweiten Unterebene für Übergabeparameter besitzt den Orientierungssinn seines direkten Obergraphen, da der Vergleich zwischen den Übergabeparametern positionsweise ohne Drehung erfolgt.

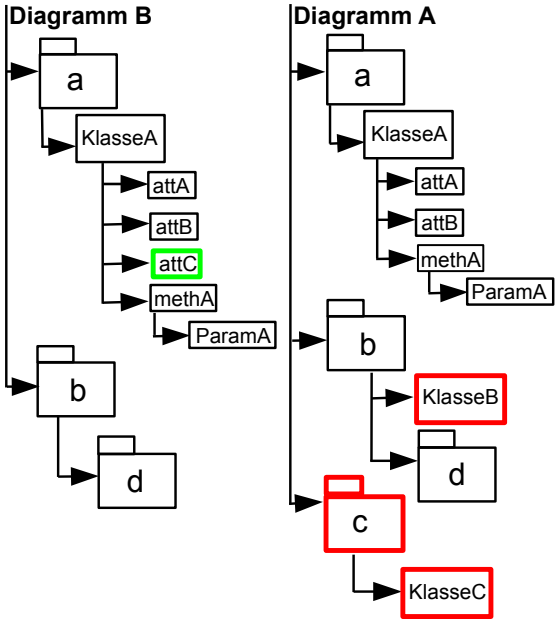

Abbildung 5.20: Zwei Klassendiagramme in der Baumdarstellung: Korrespondierende Elemente werden mit identischen Namen bezeichnet. Das Modell B entsteht aus dem Modell A durch Löschen der Klassen `KlasseB`, `KlasseC` und des Pakets `c` sowie durch Einfügen des Attributs `attC`.

Zusammensetzen der hierarchischen Zuordnung

Unabhängig vom eingesetzten Klassenebenenverfahren wird das Ergebnis der Zuordnung der Klassen in Form einer Zuordnung an das nachfolgende Paketebenenverfahren weitergereicht. In diesem Abschnitt wird das Verfahren beschrieben, das entwickelt wurde, um die Zuordnung der Pakete und die Zuordnung der Klassen zu einer gesamten Zuordnung zusammenzusetzen, die in das bereits beschriebene Korrespondenzmodell überführt wird. Dies wird zunächst an einem Beispiel motiviert, bevor die allgemeine Vorgehensweise erläutert wird.

Abbildung 5.21 zeigt die Ergebnisse der Zuordnung auf Klassen- und Paketebene für den Vergleich zweier Klassendiagramme, die in Abbildung 5.20 dargestellt sind, sowie die zusammengesetzte Zuordnung. Alle drei Zuordnungen sind als Objekte der Klasse Zuordnung realisiert und bestehen jeweils aus einer Liste der Korrespondenzobjekte und Einzelobjekte. Die Korrespondenzobjekte der Klassenzuordnung sind bereits hierarchisch organisiert, während die Korrespondenzobjekte in der Paketezuordnung flach ohne Berücksichtigung der Kompositionsstruktur vorliegen. Das Korrespondenzobjekt für die korrespondierenden Pakete `(d,d)` liegt ebenso wie das Korrespondenzobjekt der korrespondierenden Oberpakete `(b,b)` auf oberster Ebene. In der zusammengesetzten Zuordnung wird der Modellstruktur entsprechend das Korrespondenzobjekt `(d,d)` als Unterelement des Korrespondenzobjekts `(b,b)` aufgeführt. Außerdem ist dort das Korrespondenzobjekt für die `(KlasseA,KlasseA)` als Unterelement des Korrespondenzobjekts `(a,a)` eingefügt. Die Liste der Einzelelemente der Gesamtzuordnung entspricht

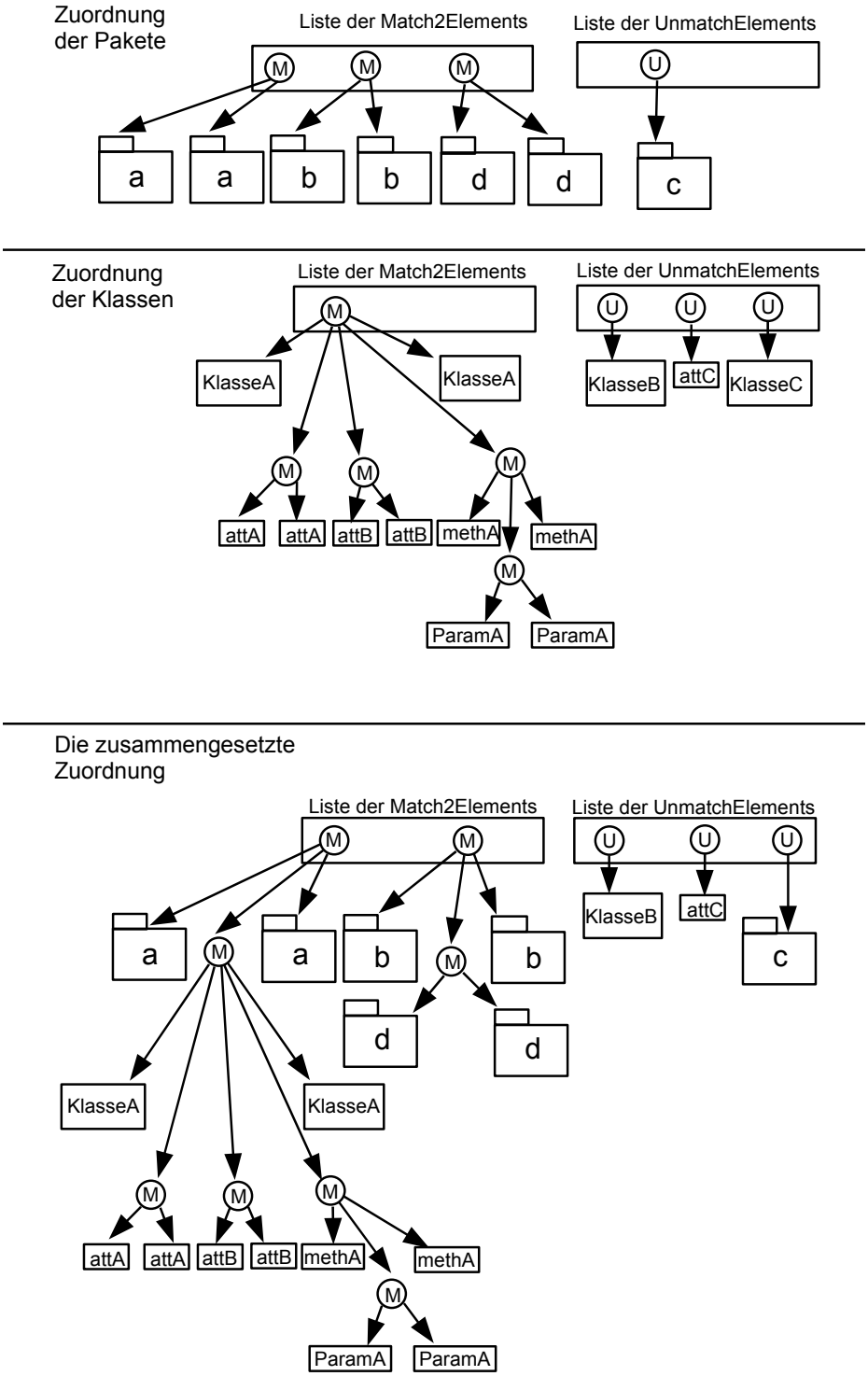

Abbildung 5.21: Ergebnisse der Paketezuordnung (oben) und der Klassenzuordnung (Mitte) für das Beispiel aus Abbildung 5.20 und die entsprechende zusammengesetzte hierarchische Zuordnung (unten)

der Zusammenführung der Einzelelemente der Klassen- und der Paketezuordnung, wobei das Einzelobjekt der Klasse `KlasseC` nicht übernommen wurde, da es als gelöschtes Unterelement eines gelöschten Elements nicht ins Korrespondenzmodell aufgenommen wird.

Allgemein wird durch das Zusammensetzen der einzelnen Zuordnungen die Kompositionsstruktur der Pakete über die Korrespondenzobjekte und deren Unterelemente nachgebaut und die Korrespondenzobjekte der Klassen an den entsprechenden Stellen der Pakethierarchie eingefügt. Die Korrespondenzobjekte aus der Klassenzuordnung, die verschobene Klassen oder verschobene Klassenunterelemente wie Attribute repräsentieren, werden auf die oberste Ebene der Korrespondenzobjekte der Gesamtzuordnung übernommen. Die Einzelobjekte der beiden einzelnen Zuordnungen werden in einer Liste vereinigt, wobei die Einzelobjekte von Klassen in nichtkorrespondierenden Paketen sowie die Einzelobjekte nichtkorrespondierender Unterpakete von nichtkorrespondierenden Paketen entfernt werden.

Algorithmus 3 zeigt den Ablauf des Verfahrens: Im **ersten Schritt** werden die Einzelobjekte aus der Klassenzuordnung in die Liste der Einzelobjekte der Gesamtzuordnung übertragen. Die Einzelobjekte können gelöschte Klassen, Attribute, Methoden, Übergabeparameter oder Assoziationen repräsentieren. Die Einzelobjekte für Klassen werden nur dann in die Gesamtzuordnung übernommen, falls das Vaterpaket einen Korrespondenzpartner besitzt. Die anderen Einzelobjekte können ohne weitere Kontrolle übertragen werden, da durch Aufbaukriterien der Klassenzuordnung bereits sichergestellt wurde, dass gelöschte Klassenunterlemente von gelöschten Klassen nicht in der Klassenzuordnung enthalten sind. Im **zweiten Schritt** werden die Einzelobjekte der Paketezuordnung durchlaufen. Alle Einzelobjekte beziehen sich auf Pakete und werden nur dann in die Gesamtzuordnung übernommen, falls das Vaterpaket einen Korrespondenzpartner besitzt.

Schritt 3 baut die Kompositionsstruktur der Korrespondenzobjekte auf. Dazu werden alle Pakete des rechten Modells mittels einem Tiefensucheverfahren beginnend im Grundpaket bearbeitet. Falls das aktuelle Paket einen Korrespondenzpartner besitzt, wird das zugehörige Korrespondenzobjekt in die Gesamtzuordnung übertragen. Falls mindestens eines der korrespondierenden Pakete im Grundpaket liegt oder die Vaterelemente der korrespondierenden Paketelemente nicht oder zu unterschiedlichen Elementen korrespondieren, wird das Korrespondenzobjekt auf der obersten Ebene der Korrespondenzobjekte der Gesamtzuordnung eingetragen. Andernfalls wird das Korrespondenzobjekt als Unterelement des Korrespondenzobjekts der Vaterpakete eingetragen. Bei einem korrespondierenden Paket werden die Unterklassen des Pakets betrachtet. Für korrespondierende Unterklassen werden die entsprechenden Korrespondenzobjekte in die Gesamtzuordnung übertragen: Falls die Vaterpakete der korrespondierenden Klassen korrespondieren, wird das Korrespondenzobjekt als Unterelement des Korrespondenzobjekts der Vaterpakete eingetragen. Ansonsten wird es auf oberster Ebene der Korrespondenzobjekte der Gesamtzuordnung eingefügt. Für jedes korrespondierende und nichtkorrespondierende Paket erfolgt der rekursive Abstieg in die Unterpakete, wobei für jedes Unterpaket ebenfalls Schritt 3 ausgeführt wird. Da auch ein nichtkorrespondierendes Paket ein korrespondierendes, verschobenes Unterpaket enthalten kann, das wiederum

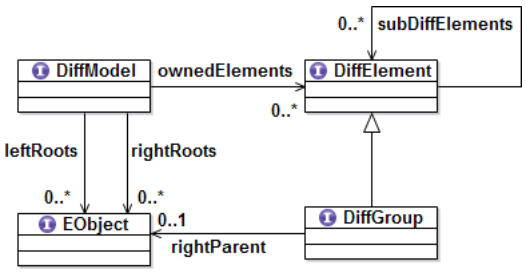

Abbildung 5.22: Aufbau des Differenzmodells

korrespondierende Unterpakete beinhalten kann, wird durch den rekursiven Durchlauf aller Pakete sichergestellt, dass auch die Kompositionsstruktur verschobener Teilbäume in der Struktur der Korrespondenzobjekte abgebildet wird.

In **Schritt 4** werden all diejenigen Korrespondenzobjekte aus der Klassenzuordnung in die Gesamtzuordnung übertragen, die noch nicht bearbeitet wurden. Damit werden verschobene Attribute und Methoden übernommen, die bereits für die Klassenzuordnung ermittelt wurden. Auch Korrespondenzobjekte, deren rechte Klassen im Grundpaket p_{rb} liegen, werden somit abschließend auf die oberste Ebene der Korrespondenzobjekte der Gesamtzuordnung übernommen.

Diese Vorgehensweise liefert eine hierarchische Gesamtzuordnung, die sich in ein korrektes Korrespondenzmodell überführen lässt, wobei die in Kapitel 4.5 vorgestellte Interpretation von Verschiebungen auf Paketebene umgesetzt wird. Dies lässt zu, dass die Elemente gelöschter Pakete nicht im Sinne eines Teilbaumlöschens mitgelöscht werden, sondern dennoch mit anderen Paketen oder Klassen korrespondieren können. Daher werden auch korrespondierende Klassen gelöschter Pakete ins Korrespondenzmodell übertragen.

Bei der Zusammensetzung der Zuordnung wird noch mit den gekapselten Elementen des internen Datenmodells gearbeitet. Dies ermöglicht die Verwendung von Markierungen, die angeben, ob eine Klasse bereits bearbeitet worden ist, sowie eine erleichterte Navigation zum Vaterelement, zum Korrespondenzpartner und zum zugehörigen Korrespondenzobjekt. Die Elemente werden nach dem Aufbau der hierarchischen Zuordnung durch einen rekursiven Durchlauf aller Elemente und deren Unterelemente entkapselt, bevor sie schließlich ins Korrespondenzmodell übertragen werden. Dazu wird ein `MatchModel` wie in Abschnitt 5.3.1 erzeugt, die Liste der Korrespondenzobjekte als Unterelemente des Korrespondenzobjekts der Grundpakete eingefügt und die Liste der Einzelobjekte übernommen.

5.4 Das Differenzmodell

Liegen die Korrespondenzen in Form eines EMF-Korrespondenzmodells vor, können die Unterschiede mit der `GenericDiffEngine` aus `org.eclipse.emf.compare.diff.engine` bestimmt und in ein Differenzmodell übertragen werden. Ein EMF Compare DiffModel (siehe Abbildung 5.22) enthält neben Verweisen auf die Diagrammelemente der ver-

Algorithmus 3 Das Zusammensetzen der Klassen- und der Paketezuordnung

Verwendete Notationen:

U_K, U_P, U_G: Menge der Einzelobjekte der Klassenzuordnung, der Paketezuordnung bzw. der Gesamtzuordnung; M_K, M_P, M_G: Menge der Korrespondenzobjekte der Klassenzuordnung, der Paketezuordnung bzw. der Gesamtzuordnung; $M(a,b)$: das Korrespondenzobjekt des Korrespondenzpaares (a,b); $S(m)$: Menge der SubmatchElements des Korrespondenzobjekts m; $U(a)$: das Einzelobjekt des Einzelelements a; $v(a)$: das Vaterelement (Paket) von a; $mp(a)$: das korrespondierende Element zu a ($mp(a)= \emptyset$ falls a Einzelelement); p_i, p_j: Pakete; p_{ra}, p_{rb}: das Grundpaket des linken bzw. rechten Modells

Schritt 1: Übertragung der Einzelobjekte aus der Klassenzuordnung

$\forall\ U(e) \in U_K$:
Falls (e !instanceof DKlasse **oder** $mp(v(e)) \neq \emptyset$) **setze** $U_G := U_G \cup \{U(e)\}$

Schritt 2: Übertragung der Einzelobjekte aus der Paketezuordnung

$\forall\ U(e) \in U_P$:
Falls $mp(v(e)) \neq \emptyset$ **setze** $U_G := U_G \cup \{U(e)\}$

Schritt 3: Übertragung der Korrespondenzobjekte des rechten Grundpakets

Setze $p_i := p_{rb}$
Falls ($mp(p_i) \neq \emptyset$ **und** $p_i \neq p_{rb}$)
Falls ($v(mp(p_i)) = p_{ra}$ **oder** $v(p_i) = p_{rb}$ **oder** $mp(v(mp(p_i))) = \emptyset$
oder $mp(v(p_i)) = \emptyset$ **oder** $v(mp(p_i)) \neq mp(v(p_i))$)
setze $M_G := M_G \cup \{M(mp(p_i), p_i)\}$
andernfalls
setze $S(M(v(mp(p_i)), v(p_i))) := S(M(v(mp(p_i)), v(p_i))) \cup \{M(mp(p_i), p_i)\}$

$\forall\ \{k \mid k$ instanceof DKlasse **und** $v(k) = p_i$ **und** $mp(k) \neq \emptyset\}$
Markiere $mp(k)$ und k als bearbeitet.
Falls ($mp(v(k)) \neq \emptyset$ **und** $mp(v(k)) = v(mp(k))$)
setze $S(M(mp(p_i), p_i) := S(M(mp(p_i), p_i) \cup \{M(mp(k), k)\}$
andernfalls setze $M_G := M_G \cup \{M(mp(k), k)\}$

$\forall\ \{p_j \mid p_j$ instanceof DPaket **und** $v(p_j) = p_i\}$:
wiederhole Schritt 3 mit $p_i := p_j$

Schritt 4: Übertragung der unerreichten Einzelobjekte

$\forall\ M(a,b)\ \in M_K$:
Falls(a !instanceof DKlasse **oder** a, b noch nicht abgearbeitet)
setze $M_G := M_G \cup \{M(a,b)\}$

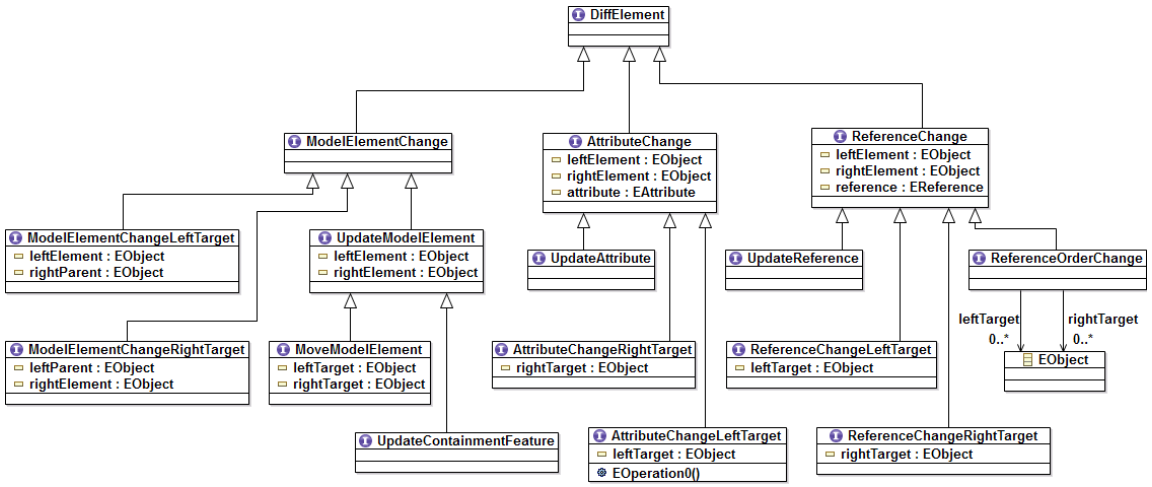

Abbildung 5.23: Arten von Differenzen

glichenen Diagramme nur die Änderungen in Form von `DiffElement`-Objekten. Diese werden im Folgenden als Differenzobjekte bezeichnet. Differenzobjekte, die sich auf das gleiche Modellelement beziehen, werden in Differenzgruppen, vom Typ `DiffGroup`, zusammengefasst, die hierarchisch geschachtelt werden. Diese kompakte Darstellungsform besitzt den Vorteil, dass vorhandene Informationen auch hier nicht vervielfältigt werden [BP08]. Falls keine Differenzen vorliegen, enthält das Modell auch keine Differenzobjekte. Falls die Abhängigkeiten von den Vergleichsmodellen jedoch unerwünscht sind, lassen sich das EMF `MatchModel` und `DiffModel` z. B. mit dem Ansatz aus [Kön09] auch in modellunabhängige Korrespondenz- und Differenzmodelle überführen.

Bevor auf die Vorgehensweise der `GenericDiffEngine` eingegangen wird, die zum Ableiten der Differenzen verwendet wird, soll zunächst durch einen Blick ins Paket `org.eclipse.emf.compare.diff.metamodel` geklärt werden, welche Arten von Differenzen von der `GenericDiffEngine` erkannt werden.

5.4.1 Arten von Differenzen

Die Differenzen werden als Objekte vom Typ `DiffElement` realisiert und lassen sich, wie in der Vererbungshierarchie in Abbildung 5.23 zu sehen ist, in Bezug auf die betroffenen Elemente in die drei Kategorien `ModelElementChange`, `AttributeChange` und `ReferenceChange` unterteilen. In unserem Modell zählen Pakete, Klassen, Attribute und Methoden zu den Modellelementen. Eigenschaften von Klassen, Attributen, Methoden und Referenzen werden als Attribute behandelt. Vererbungsbeziehungskanten, Assoziationskanten (und `eType`-Verweise) stellen Referenzen dar. Der Vollständigkeit wegen werden alle Arten von Differenzobjekten angegeben, auch wenn nicht alle der im Folgenden beschriebenen Differenzen beim Vergleich von Ecore-Klassendiagrammen eingesetzt werden.

ModelElementChange Als Differenzen auf Modellelementen werden das Einfügen, Löschen und Verschieben von Modellelementen betrachtet. Ist ein Modellelement nur im linken Diagramm vorhanden, gilt es als eingefügt. Dazu wird ein Differenzobjekt vom Typ `ModelElementChangeLeftTarget` erstellt, das Referenzen auf das neu eingefügte Element (`leftElement`) im linken Modell und auf das Modellelement im rechten Modell enthält, als dessen Kind das neue Element eingefügt wird (`rightParent`). Falls ein Modellelement nur im rechten Diagramm enthalten ist, gilt es als gelöscht und es wird ein Differenzobjekt vom Typ `ModelElementChangeRightTarget` erstellt. In diesem Fall werden Referenzen auf das gelöschte Element (`rightElement`) im rechten Diagramm und das Element im linken Diagramm gehalten, dem das entsprechende Unterelement fehlt (`leftParent`). Falls ein Modellelement zwar in beiden Diagrammen vorhanden ist, aber die Vaterelemente nicht korrespondieren, handelt es sich um ein verschobenes Element. In solchen Fällen wird ein Differenzobjekt vom Typ `MoveModelElement` erzeugt. Dies enthält Referenzen auf die korrespondierenden Modellelemente im linken und rechten Diagramm (`leftElement` bzw. `rightElement`), sowie Referenzen auf die korrespondierenden Elemente ihrer Vaterelemente im jeweils anderen Diagramm (`leftTarget` bzw. `rightTarget`). Ein Beispiel für ein `MoveModelElement` zeigt Listing 5.2 in Form eines Ausschnitts aus dem serialisierten Differenzmodell.

Listing 5.2: XML Darstellung eines `MoveModelElement`s:
Das Attribut `att3` wurde von `Klasse1` nach `Klasse2` verschoben.

```
<subDiffElements xsi:type="diff:MoveModelElement">
<rightElement href="platform:/resource/../A1.ecore#//Klasse2/att3"/>
<leftElement href="platform:/resource/../A2.ecore#//Klasse1/att3"/>
<leftTarget href="platform:/resource/../A2.ecore#//Klasse2"/>
<rightTarget href="platform:/resource/../A1.ecore#//Klasse1"/>
</subDiffElements>
```

AttributeChange Bei den Differenzen auf Attributen wird unterschieden, ob es sich um ein Attribut handelt, das genau einen Wert besitzt oder mehrere Werte besitzen kann. Im Fall eines einfachen Attributes führen Änderungen zur Erzeugung eines Differenzobjekts vom Typ `UpdateAttribute`. Dies enthält eine Referenz auf das Attribut, das sich geändert hat, und Referenzen auf das linke und rechte Modellelement, die das betroffene Attribut besitzen. Ein solches Differenzobjekt wird zum Beispiel dann erzeugt, wenn der Klassenname zweier korrespondierender Klassen nicht übereinstimmt (siehe Listing A.2, S. 245, Zeile 6-10). Bei mehrwertigen Attributen wird ein Attributwert entweder eingefügt (`AttributeChangeLeftTarget`) oder gelöscht (`AttributeChangeRightTarget`). Diese Differenzobjekte besitzen neben den Referenzen auf das betroffene Attribut und das linke und rechte Modellelement noch eine weitere Referenz auf den eingefügten Wert (`leftTarget`) bzw. auf den gelöschten Wert (`rightTarget`). Für eine Reihenfolgeänderung der Werte eines mehrwertigen Attributs gibt es kein Differenzobjekt. Bei allen in den Ecore-Klassendiagrammen relevanten Eigenschaften handelt es sich um einwertige Attribute, so dass die Differenzobjekte für mehrwertige Attribute hier keine Rolle spielen.

ReferenceChange Als dritte Variante werden Differenzen betrachtet, die sich auf Referenzen beziehen. Eine `UpdateReference` tritt dann auf, wenn zwei korrespondierende Referenzen auf verschiedene Modellelemente zeigen. Die Objektfelder `rightElement` und `leftElement` des Differenzobjekts verweisen auf die Elemente im rechten bzw. linken Diagramm, die die betroffene Referenz besitzen, auf die selbst mit `reference` verlinkt wird. Die Referenzen `leftTarget` und `rightTarget` zeigen auf die beiden unterschiedlichen, nicht korrespondierenden Ziele der Referenzen im linken bzw. rechten Diagramm. Eine solche Differenz tritt bei einwertigen Referenzen auf, wenn zum Beispiel der `eType` einer Assoziation geändert wird, wie in Listing 5.3, oder das `eOpposite`-Element einer Assoziation geändert oder entfernt wird.

Listing 5.3: XML Darstellung einer `UpdateReference`:
Die Referenz `ass` zeigt nun nicht mehr auf `Klasse1`, sondern auf `Klasse2`.

```
<subDiffElements xsi:type="diff:UpdateReference">
<reference href="http://www.eclipse.org/emf/2002/Ecore#//
    ETypedElement/eType"/>
<rightElement href="platform:/resource/../A1.ecore#//Klasse1/ass"/>
<leftElement href="platform:/resource/../A2.ecore#//Klasse1/ass"/>
<leftTarget href="platform:/resource/../A2.ecore#//Klasse1"/>
<rightTarget href="platform:/resource/../A1.ecore#//Klasse2"/>
</subDiffElements>
```

Bei mehrwertigen Referenzen, die nicht nur auf ein Modellelement verweisen, wird zwischen dem Einfügen eines Elements und dem Löschen eines Elements unterschieden. Die dazugehörigen Differenzobjekte `ReferenceChangeLeftTarget` und `ReferenceChangeRightTarget` sind symmetrisch und enthalten als Objektfeld neben dem Verweis auf die betroffene Referenz (`reference`) und die Modellelemente im linken und rechten Diagramm (`leftElement` bzw. `rightElement`) eine Referenz auf das neu eingefügte Referenzziel (`leftTarget`) bzw. das gelöschte Referenzziel (`rightTarget`). Mehrwertige Referenzen kommen zum Beispiel bei Vererbungsbeziehungen vor, da eine Klasse mehrere Interfaces implementieren kann.

Ein Differenzobjekt vom Typ `ReferenceOrderChanged` wird erzeugt, falls sich die Reihenfolge der Zielelemente einer mehrwertigen Referenz geändert hat. Dies ist beispielsweise dann der Fall, falls die Elemente der `eSuperTypes` einer Klasse in der Reihenfolge vertauscht sind (siehe Listing 5.4), aber auch dann, wenn sich die Reihenfolge der Übergabeparameter einer Methode ändert[6].

Listing 5.4: XML Darstellung einer `ReferenceOrderChanged`-Differenz:
Die Reihenfolge der Klassen `K1` und `K2` in der Liste der `eSuperTypes` der Klasse `K3` hat sich geändert.

```
<subDiffElements xsi:type="diff:ReferenceOrderChange">
 <reference href="http://www.eclipse.org/emf/2002/Ecore#//EClass/
     eSuperTypes"/>
```

[6]Erst ab Version 1.0.1. Bei der verwendeten Version 1.0.0. wurde eine geänderte Reihenfolge der Übergabeparameter noch nicht erkannt.

```
<rightElement href="platform:/resource/../My2.ecore#//K3" />
<leftElement href="platform:/resource/../My.ecore#//K3" />
<leftTarget href="platform:/resource/../My.ecore#//K2" />
<leftTarget href="platform:/resource/../My.ecore#//K1" />
<rightTarget href="platform:/resource/../My2.ecore#//K1" />
<rightTarget href="platform:/resource/../My2.ecore#//K2" />
</subDiffElements>
```

Außerdem existiert ein Differenzobjekt vom Typ `UpdateContainmentFeature`, dessen Verwendungszweck an dieser Stelle unklar bleibt, da bei der Änderung der `containment`-Eigenschaft einer Referenz in den Versionen 1.0.0 und 1.0.1 ein Differenzobjekt vom Typ `UpdateAttribute` erzeugt wird.

5.4.2 Ableitung der Differenzen aus dem Korrespondenzmodell

Aus dem selbst erstellten Korrespondenzmodell sollen mit Hilfe der `GenericDiffEngine` die Differenzen abgeleitet werden. Dazu ist der Aufruf einer `DiffEngine` erforderlich, für den es zwei Möglichkeiten gibt (siehe Listing 5.5). Ist man sich unschlüssig, welche `DiffEngine` geeignet ist, wendet man sich am besten an den `DiffService`. Dieser entscheidet im Fall mehrerer zur Auswahl stehenden `DiffEngines` anhand der Ressource, welche `DiffEngine` verwendet wird. Der boolesche Übergabeparameter der Methode `doDiff()` bestimmt, ob eine 3-Wege-Differenzberechnung stattfinden soll. In unserem Rahmenwerk wird gezielt die `GenericDiffEngine`, die für den Vergleich von Ecore-Modellen verwendet wird, mit dem Standard-2-Wege-Differenzberechnungsverfahren aufgerufen.

Listing 5.5: Zwei Alternativen für den programmatischen Aufruf der `DiffEngine`

```
DiffModel d1 = DiffService.doDiff(matchmodel, false);
DiffModel d2 = new GenericDiffEngine().doDiff(matchmodel);
```

Bei der Ermittlung der Differenzen aus dem Korrespondenzmodell werden in einem ersten Schritt die Korrespondenzobjekte und im zweiten Schritt die Einzelobjekte verarbeitet. Die Korrespondenzobjekte werden beginnend bei der Wurzel des Korrespondenzmodells nach dem Tiefensucheverfahren durchlaufen. Jedes Korrespondenzobjekt wird auf Differenzen in den Kategorien Attributänderungen, Referenzänderungen, Modellelementverschiebung (und ContainmentUpdate) überprüft. Falls die Vaterelemente der korrespondierenden Elemente zum Beispiel nicht korrespondieren, wird an dieser Stelle eine Verschiebung identifiziert. Die entsprechenden Differenzobjekte werden erzeugt und zusammen in einer Differenzgruppe am Korrespondenzobjekt des rechten Modellelements verankert.

Nachdem alle Korrespondenzobjekte des Korrespondenzmodells abgearbeitet sind, wird die Liste der Einzelobjekte durchlaufen. Dabei werden die Einzelobjekte je nach Seitenangabe als gelöscht oder erzeugt identifiziert. Besitzt das Attribut `Side` den Wert `RIGHT`, wird über die `DiffFactory` ein Differenzobjekt vom Typ `ModelElementChangeRightTarget` erzeugt. Ansonsten stammt das Modellelement aus dem linken Diagramm und es wird ein `ModelElementChangeLeftTarget` erzeugt. Um das Differenzobjekt in

der Hierarchie der Differenzgruppen richtig einzuordnen, werden die erstellten Differenzgruppen nach der entsprechenden Gruppe durchsucht. Im Fall eines linken Modellelements ist der `Owner` der Differenzgruppe das `ContainerElement`, im Fall eines rechten Modellelements muss nach dem Korrespondenzpartner seines `ContainerElements` gesucht werden. Eine Ausnahme stellt ein gelöschtes bzw. erzeugtes Wurzelelement dar, da hier kein `ContainerElement` vorhanden ist. In diesem Fall wird das Differenzobjekt auf oberster Ebene eingeordnet. Im Differenzmodell werden gelöschte Unterelemente eines gelöschten Elements nicht einzeln als Unterschiede dargestellt. Dies reduziert die Anzahl der gemeldeten Differenzen. Beim Aufbau des Korrespondenzmodells muss jedoch beachtet werden, dass nur die Wurzel eines gelöschten Teilbaums als Einzelobjekt einzufügen ist. Da innerhalb des gelöschten Teilbaums das entsprechende korrespondierende `ContainerElement` nicht zur Verfügung steht, käme es ansonsten zu einem Fehler im Differenzberechnungsprozess.

Das hier vorgestellte Differenzenmodell stellt für die Ecore-Metamodelelemente Basisdifferenzen zur Verfügung, die sich mit zwei Ausnahmen, dem `MoveModelElement` und dem `ReferenceOrderChanged`, auf atomare Operationen beziehen. Bei Bedarf lässt sich dieses generische Differenzmodell erweitern. Einerseits ist es möglich, die Differenzen auf spezifische Modelle anzupassen. Andererseits lassen sich auch mehrere Basisdifferenzen zu zusammengesetzten Differenzen zusammenfassen, wie es zum Beispiel für die Darstellung von Refactoringmaßnahmen sinnvoll ist. Alternativ kann das erstellte Differenzmodell auch mit Model-to-Model-Transformationen nachverarbeitet werden. Darüber hinaus können Model-to-Text-Transformationen eingesetzt werden, um aus dem Differenzmodell spezifische Differenzberichte z. B. im Html-Format (siehe [MSW09b]) zu generieren.

5.4.3 Darstellung der Differenzen

Zur Visualisierung der Differenzen wird der EMF Compare UI Editor aus dem Paket `org.eclipse.emf.compare.ui.editor` verwendet. Um das Differenzmodell eines Vergleichs in diesem Editor zu öffnen, muss zunächst ein `ComparisonResourceSnapshot` erzeugt werden, der das `MatchModel` und das `DiffModel` enthält. Aus diesem `Snapshot` wird dann ein `ModelCompareInput`-Objekt erzeugt, mit dem der Editor schließlich geöffnet werden kann (siehe Listing 5.6). Dies erfolgt in unserem Rahmenwerk in der Klasse `DiffAnzeige` des Pakets **view**.

Listing 5.6: Aufruf des Compare UI Editors

```
1 ComparisonResourceSnapshot snapshot = DiffFactory.eINSTANCE.
      createComparisonResourceSnapshot();
2 snapshot.setMatch(matchmodel);
3 snapshot.setDiff(diffmodel);
4 ModelCompareEditorInput e = new ModelCompareEditorInput(
      snapshot);
5 e.setTitle(titel);
6 CompareUI.openCompareEditor(e);
```

Der Bildschirmausschnitt in Abbildung 5.24 zeigt den zweigeteilten EMF Compare UI Editor. Im oberen Teil wird eine textuelle Zusammenfassung der strukturellen Unterschiede geliefert, während im unteren Teil die beiden verglichenen Diagramme in einer Baumsicht gegenübergestellt werden. Die Beschreibung der strukturellen Unterschiede bezieht sich zwar auf Änderungsoperationen, diese sind jedoch nicht immer elementar. So wird zum Beispiel das Löschen einer Klasse als ein struktureller Unterschied angegeben, dieser kann jedoch auch das Löschen der Attribute dieser Klasse einschließen. In beiden Fällen werden die Differenzen gerichtet dargestellt, wobei das rechte Diagramm als Ausgangsdiagramm und das linke Diagramm als dessen neue Version behandelt wird. Die identifizierten Unterschiede werden in der graphischen Ansicht mittels farbiger Verbindungslinien repräsentiert, wobei eine grüne Linie für das Erzeugen und eine rote Linie für das Löschen eines Elements steht. Blaue Verbindungslinien zeigen, dass zwei Elemente als korrespondierend betrachtet werden, diese jedoch nicht in allen Eigenschaften identisch sind. In diesem Fall spricht man auch von einer Modifikation oder Änderung. Was geändert wurde, erschließt sich am besten über die Properties-Seite, die alternativ zur Baumsicht angezeigt werden kann und die Wertebelegungen aller Eigenschaften der markierten Modellelemente auflistet.

Abbildung 5.24 greift nochmal das Vergleichsergebnis mit dem `ECVerfahren` für das Beispiel aus Abschnitt 1.2.1 auf, für das auch das dazugehörige serialisierte `MatchModel` im Anhang A.2 abgedruckt ist. Die Klasse `Address` wurde in `USAddress` umbenannt und um ein neues Attribut (`String state`) erweitert. Die Klasse `USAddress` wurde gelöscht. In diesem Fall wäre es nicht möglich, mit dem Nachbearbeitungsschritt aus Abschnitt 4.4 eine Verschiebung des Attributs `state` zu erkennen, da die Elemente des gelöschten Teilbaums nicht als Einzelobjekte in das Korrespondenzmodell aufgenommen werden.

Die textuelle Darstellung der Differenzen im oberen Teil des EMF Compare Editors folgt dem hierarchischen Aufbau der Differenzgruppen, die sich auch im serialisierten Differenzmodell (siehe Listing im Anhang A.2) zeigt. Die Differenzgruppe, die am Wurzelelement des rechten Diagramms verankert ist, enthält ein Differenzobjekt für das Löschen der Klasse `USAddress` und eine untergeordnete Differenzgruppe, die an der Klasse `Address` verankert ist. Diese Gruppierung enthält die Differenzobjekte für die Namensänderung sowie das Einfügen des Attributs. Klickt man auf ein Differenzobjekt der textuellen Darstellung, wird das entsprechende Element in der Baumdarstellung markiert.

5.5 Integration eines neuen Verfahrens

Das Rahmenwerk wurde mit dem Ziel konzipiert, dass sich weitere Verfahren gut einbinden lassen. Je nach Art des zu integrierenden Verfahrens wird eine Unterklasse der Klasse `EinstufigesVerfahren`, `Klassenebenenverfahren` oder `Paketebenenverfahren` erstellt (siehe Abbildung 5.25). Dabei ist die abstrakte Methode `starteVerfahren()` der Klasse `Verfahren` zu implementieren.

Für die Implementierung von Klassenebenenverfahren und Paketebenenverfahren, die flexibel mit anderen Verfahren im mehrstufigen Ansatz eingesetzt werden sollen, muss die

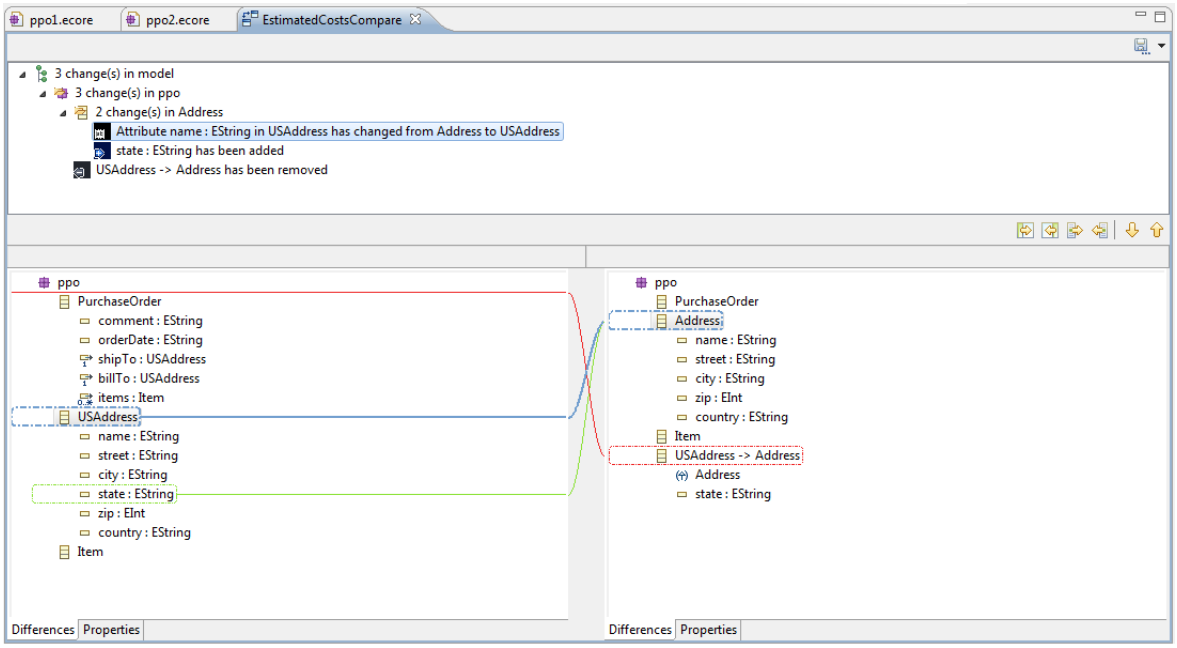

Abbildung 5.24: Screenshot des EMF Compare UI Editors zur Visualisierung der Unterschiede in der Baumsicht

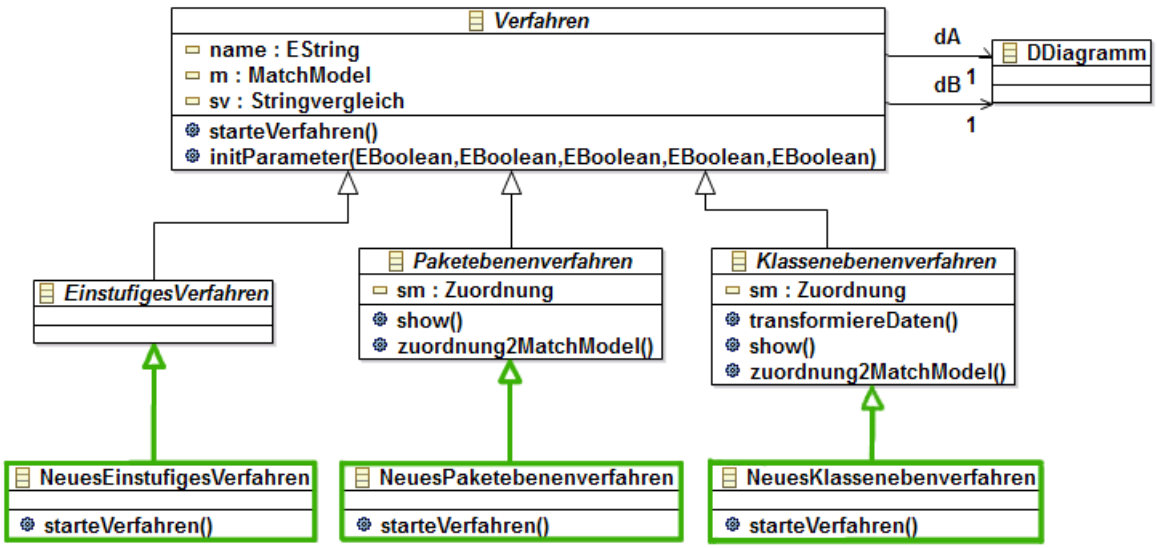

Abbildung 5.25: Einbettung neuer Verfahren in das Rahmenwerk

in Abschnitt 5.2.2 vorgestellte interne Datenstruktur `DDiagramm (DPaket, DKlasse)` verwendet werden. Diese Anforderung ist darauf zurückzuführen, dass das Paketebenenverfahren als Ergebnis des Klassenebenenverfahrens eine `Zuordnung` mit den noch gekapselten Diagrammelementen erwartet. Diese Klassenzuordnung kann entweder direkt ins Korrespondenzmodell übertragen oder an das nachfolgende Paketebenenverfahren weitergereicht werden. Der Export der Klassenzuordnung direkt ins Korrespondenzmodell erfolgt über die Methode `zuordnung2MatchModel()` der Klasse `Klassenebenenverfahren`. Falls an das Klassenebenenverfahren noch ein Paketebenenverfahren anschließt, erstellt die Methode `zuordnung2MatchModel()` der Klasse Paketebenenverfahren die Gesamtzuordnung und führt sie in ein Korrespondenzmodell über. In beiden Fällen wird die Ableitung des Differenzmodells aus dem Korrespondenzmodell sowie das Öffnen des EMF Compare UI Editors in der Methode `show()` der Klasse Klassenebenenverfahren bzw. Paketebenenverfahren durchgeführt, die ein entsprechendes Objekt der Klasse `DiffAnzeige` erstellt.

Für ein einstufiges Verfahren gibt es keine Vorgaben, wie zu den ausgewählten Ecore-Dateien ein EMF-Korrespondenzmodell zu erstellen ist. Zum Ableiten der Differenzen und Öffnen des EMF Compare UI Editors kann eine Instanz der Klasse `DiffAnzeige` verwendet werden. Für jedes neue Verfahren muss außerdem die graphische Oberfläche der `DiffView` um entsprechende Auswahlmöglichkeiten erweitert werden.

5.6 Zusammenfassung

Die beiden eigenentwickelten Verfahren, das edierkostenbasierte und das ähnlichkeitsbasierte Verfahren aus Kapitel 4, und ihre Varianten wurden in einem Eclipse-Plugin implementiert, das zusammen mit weiteren Plugins des Eclipse Modeling Frameworks ein Rahmenwerk zum Vergleich von Ecore-Klassendiagrammen bildet. Das Rahmenwerk wurde so konzipiert, dass sich neue Verfahren gut einbinden lassen. Insbesondere bietet es die Möglichkeit, verschiedene Klassenebenen- und Paketebenenverfahren miteinander zu kombinieren. Darüber hinaus werden auch Variationspunkte innerhalb eines Verfahrens als Auswahlmöglichkeit an die Oberfläche weitergereicht, wie zum Beispiel die mögliche Aktivierung von Schwellenwerten im Rahmen des ECVerfahrens. Eine Feineinstellung des Kostenmodells des edierkostenbasierten Verfahrens oder eine Anpassung der Gewichtung der Ähnlichkeitswerte des ähnlichkeitsbasierten Verfahrens sind über weitere Sichten möglich.

Die Integration der Verfahren in die Entwicklungsumgebung Eclipse erleichtert das flüssige Arbeiten, da zum Vergleichen der Modelle innerhalb der Entwicklungsumgebung lediglich eine neue Sicht geöffnet werden muss. Das Rahmenwerk kann zum Evaluieren von verschiedenen Korrespondenzberechnungsverfahren verwendet werden. Da im Anschluss das gleiche Differenzberechnungsverfahren, die EMF Compare `DiffEngine`, verwendet wird, können die Auswirkungen verschiedener Zuordnungen auf die Differenzberechnung gut untersucht werden. Die graphische Darstellung des Vergleichsergebnisses über die Baumsicht des EMF Compare UI Editors erleichtert das Erkennen und Verstehen der Unterschiede. Die Nutzung der `DiffEngine` und des `DiffModels` von EMF

Compare ist mit kleineren Abstrichen in Hinblick auf die Visualisierung verbunden. So entspricht die umgesetzte Interpretation der Reihenfolgeänderung bei Übergabeparametern und der Referenzen in der Liste der `eSuperTypes` nicht ganz den Erwartungen. Im Austausch für die Vergleichbarkeit der Ergebnisse, da nicht für jedes Verfahren ein eigenes angepasstes Differenzmodell und Differenzberechnungsverfahren verwendet wird, sind diese Einschränkungen jedoch akzeptabel. Im Bedarfsfall lässt sich die Differenzberechnung und Visualisierung auch gut austauschen. Eine graphische Visualisierung der Unterschiede ist von EMF Compare bereits in Entwicklung und kann ergänzt werden.

6 Evaluation

In diesem Kapitel werden die Ergebnisse der durchgeführten Evaluation vorgestellt. Trotz der eingeschränkten Verfügbarkeit von echten Fallstudien und Testpersonen soll dabei ein Eindruck über die Arbeitsweise der Verfahren vermittelt werden, wobei ausgewählte Beispiele die Besonderheiten der Verfahren verdeutlichen. Neben den Laufzeiten der Verfahren und der qualitativen Evaluation der Ergebnisse, wurde das edierkostenbasierte Verfahren auch in Hinblick auf die Optimalität der Lösung untersucht. Dabei wurden die Fragestellungen betrachtet, wie gut das hinreichende Optimalitätskriterium ist, in wievielen Fällen die optimale Lösung erreicht wird und wie weit die Kosten der gefundenen Lösungen von den optimalen Kosten abweichen. Für das ähnlichkeitsbasierte Verfahren wurde weiterhin getestet, in wievielen Fällen die Ergebnisse der Variante mit Anpassung der Ähnlichkeitswerte, die in Abschnitt 4.3.2 beschrieben wird, von den Ergebnissen der Standardvariante abweichen.

Bei der Evaluation dieses Kapitels stand der Test der eigenentwickelten Verfahren im Vordergrund. Die gezogenen Vergleiche zu dem generischen Vergleichsverfahren von EMF Compare sollen dem Leser lediglich veranschaulichen, wie die bisherige Werkzeugunterstützung zum Vergleichen von Klassendiagrammen in Eclipse aussieht, ohne dass spezielle Verfahren installiert wurden. Für einen fairen Vergleich des generischen EMF Compare Verfahrens mit den eigenentwickelten domänenspezifischen Verfahren müssten auch mögliche Anpassungen der Parameter in Betracht gezogen werden, die sich zwar nicht über die Benutzerschnittstelle, aber im frei zugänglichen Quellcode des Verfahrens einstellen lassen.

6.1 Methodisches Vorgehen

Die Evaluation basiert auf sechs verschiedenen Testmengen, deren Eckdaten in Tabelle 6.1 zusammengefasst sind. Die Klassendiagramme der Gruppen A, B und C enthalten jeweils ein Grundpaket, in dem alle Klassen des Modells liegen. Die Klassendiagramme dieser Testgruppen bestehen aus jeweils bis zu 11 Klassen, so dass die Testfälle auch mit der Brute-Force-Variante des edierkostenbasierten Verfahrens noch in vertretbarer Zeit gerechnet werden können. Die Testfälle der Mengen A, B und C stellen jeweils Varianten eines Ausgangsmodells dar, wobei die drei Ausgangsmodelle unterschiedliche Schwerpunkte besitzen. Die Klassendiagramme der Testmenge A modellieren das Gesellschaftsspiel Ludo und enthalten vergleichsweise viele Methoden. In den Beispielen der Testmenge C wird ein Workflow modelliert, wobei die Klassen durch vergleichsweise viele Assoziationen miteinander verbunden sind. Die Beispiele der Testmenge B, die das in dieser Arbeit schon mehrfach verwendete Thema des Autovermietungsystems behandeln,

Tabelle 6.1: Die Testmengen der Evaluation

Name	Diagramme	Vergleiche	Klassen	Pakete	Thema
A	75	2775	5-10 (⌀ 8)	1	Ludo
B	100	4950	4-10 (⌀ 6,1)	1	Autovermietung
C	70	2415	5-11 (⌀ 8,47)	1	Workflow
E	12	66	6-9 (⌀ 7,42)	6	FSWorkspaceManager
F	15	105	17-20 (⌀ 18,6)	5	FSListener
M	15	105	65-80 (⌀ 70,73)	24-30	MOD2-SCM

sind in Hinblick auf die Verteilung der kontextunabhängigen und kontextabhängigen Klasseneigenschaften eher ausgewogen. Während die 245 Beispiele der Testmengen A, B und C manuell konstruiert wurden, handelt es sich bei den weiteren drei Testmengen E, F und M um echte Fallstudien des Lehrstuhls. Die größte der drei Fallstudien, die Testmenge M, besteht aus 15 verschiedenen Versionen, die 65 bis 80 Klassen verteilt auf 24 bis 30 Pakete enthalten. Innerhalb einer Testmenge werden alle verschiedenen Modelle paarweise miteinander verglichen, so dass sich für eine Testmenge mit n Elementen $\binom{n}{2}$ Testfälle ergeben.

Bis kurz vor Ende dieser Arbeit standen im Rahmen des Projekts MOD2-SCM für die Testmenge M nur Klassendiagramme zur Verfügung, in denen keine Namensänderungen durchgeführt wurden. Um die Effekte der Verfahren bei Umbenennungen zu testen, erschien es daher sinnvoll, zusätzlich eigene Testbeispiele in den Testmengen A, B und C zu erzeugen. Auf die zufällige Generierung von Klassendiagrammen wurde verzichtet, um die Option offen zu halten, dass eine Versuchsperson beim Betrachten des Vergleichsergebnisses entscheiden kann, ob die berechnete Zuordnung sinnvoll ist. Die Testbeispiele der Mengen A, B und C wurden daher per Hand konstruiert. Dabei wurden, ausgehend von einem Klassendiagramm, verschiedene Teilmengen an Klassen gelöscht und durch geringfügige Eingriffe sichergestellt, dass es sich bei dem Ergebnis um ein Klassendiagramm ohne unvollständige Assoziationen handelt, denen ein Assoziationsende fehlt. Aus diesen Klassendiagrammen wurden weitere Klassendiagramme erzeugt, indem wieder neue andere Klassen hinzugefügt wurden. Weitere Beispiele der Testmenge sind dadurch entstanden, dass vorhandene Klassendiagramme kopiert und durch eine beliebige Auswahl an Edieroperationen verändert wurden. Dabei wurden Änderungen der Namen, der Klassentypen, der Attribute, der Methoden, der Assoziationen und/oder Vererbungsbeziehungen durchgeführt sowie Attribute, Methoden, Assoziationen und/oder Vererbungsbeziehungen gelöscht oder eingefügt. Alle Klassendiagramme wurden zu einer Menge zusammengefasst; die Anzahl der darin enthaltenen Diagramme ergab sich mehr oder weniger zufällig.

Vor den Ergebnissen werden zunächst die Laufzeiten der Verfahren betrachtet.

6.2 Laufzeitverhalten der Verfahren

Die eigenentwickelten Verfahren wurden bislang nicht auf eine möglichst kurze Laufzeit optimiert, da im Fall einzelner Vergleiche weniger die Laufzeit als die Qualität des Ergebnisses im Vordergrund steht. Anders stellt sich die Situation jedoch dar, wenn Massentests durchgeführt werden sollen. An dieser Stelle werden die Laufzeiten jedoch eher in Hinblick darauf untersucht, ob der Benutzer mit den Verfahren gut arbeiten kann, ohne dass er zu lange auf das Ergebnis warten muss.

Die Laufzeiten wurden auf einem Rechner mit einem Intel Core 2 Extreme Prozessor Q9300 mit 2,53 GHz und 4 GB RAM ermittelt, wobei für die Berechnung nur ein Prozessorkern genutzt wurde. Für die Testläufe wurde die Testmenge M verwendet, da diese die größten Klassendiagramme aller Testmengen enthält. Um Schwankungen auszugleichen, wurden die Laufzeittests je dreimal durchgeführt. Die gemessenen Zeiten beinhalten bei den eigenentwickelten Verfahren auch den Zeitaufwand für die Transformation der Ecore-Klassendiagramme in das interne Format. Die Zeitmessung wurde für alle Klassenebenenverfahren beendet, sobald die Berechnung der Zuordnung auf Klassenebene abgeschlossen war. Der Aufwand für das Ableiten der Differenzen aus dem Korrespondenzmodell, das Öffnen des EMF Compare UI Editors und die Anzeige der Differenzen hängt nicht von dem verwendeten Verfahren, sondern von den berechneten Korrespondenzen und der Größe der zu vergleichenden Klassendiagramme ab und ist mit ca. 1,5 Sekunden anzusetzen.

Die Tabelle 6.2 enthält die Ergebnisse für die Laufzeittests der Klassenebenenverfahren. Aus den Zahlen wird ersichtlich, dass das ähnlichkeitsbasierte Klassenebenenverfahren weniger als ein Siebtel der Zeit des edierkostenbasierten Verfahrens benötigt. Das edierkostenbasierte Verfahren mit Schwellenwertregel benötigt weniger als zwei Drittel der Laufzeit der ursprünglichen Variante und zeigt damit ein deutlich besseres Laufzeitverhalten. Die Zeit ist aber dennoch mehr als viermal so hoch wie die Zeit des ähnlichkeitsbasierten Verfahrens.

Wie in Tabelle 6.3 zu sehen ist, benötigt das darauf aufbauende Paketebenenverfahren unabhängig von dem darunterliegenden Klassenebenenverfahren durchschnittlich nur 6 Millisekunden für die Berechnung der Zuordnung der Pakete und der Kombination der Ergebnisse auf Klassen- und Paketebene zu einer gemeinsamen Zuordnung. Die Zeit für das Öffnen des EMF Compare UI Editors mit der Anzeige des Ergebnisses wurde auch hier in den Messungen nicht berücksichtigt. Tabelle 6.4 zeigt im Vergleich dazu die ermittelte Laufzeit des einstufigen Verfahrens von EMF Compare. Auch hier wurde nur die Zeit für die Berechnung und nicht die Zeit für die Anzeige des Ergebnisses berücksichtigt. Es zeigt sich, dass das einstufige EMF Compare Verfahren bei der ausgeprägten Pakethierarchie der Testgruppe M sehr unterschiedliche Laufzeiten von 0,05 bis 3,6 Sekunden aufweist. Im Durchschnitt liegt das EMF Compare Verfahren mit 0,99 Sekunden vor dem Edierkostenverfahren mit anschließenden Paketebenenverfahren. Das zweistufige ähnlichkeitsbasierte Verfahren benötigt im Durchschnitt jedoch nur 0,288 Sekunden für einen Vergleich aus dieser Testmenge. Um die Laufzeit auch auf flachen Klassendiagrammen zu vergleichen, wurde eine Testmenge M' erstellt, indem in den Modellen jeweils alle Klassen in das Grundpaket verschoben und die anderen Pakete

Tabelle 6.2: Laufzeiten der verschiedenen Verfahren auf Klassenebene (in Sekunden)

Klassenebenenverfahren	Testgruppe	Minimum	Maximum	∅
Edierkostenverfahren (EC)	M	1,803	2,750	2,179
Edierkostenverfahren mit Schwelle (ECTH)	M	1,024	1,508	1,327
Ähnlichkeitsverfahren	M	0,202	0,493	0,282

Tabelle 6.3: Laufzeiten des Paketebenenverfahrens (in Sekunden)

Paketebenenverfahren	Testgruppe	Minimum	Maximum	∅
Ähnlichkeitsverfahren	M	0,004	0,008	0,006

Tabelle 6.4: Laufzeiten des EMF Compare Verfahrens (in Sekunden)

Gesamtverfahren	Testgruppe	Minimum	Maximum	∅
EMF Compare (Version 1.0.0)	M	0,05	3,6	0,99
EMF Compare (Version 1.0.0)	M'	0,254	30,470	4,744

gelöscht wurden. Während diese Änderung für die eigenentwickelten Klassenebenenverfahren keine messbare Auswirkung auf die Laufzeiten besitzt und das Paketebenenverfahren sogar etwas schneller arbeitet, da keine Pakete verglichen werden müssen, zeigt das EMF Compare Verfahren auf den flachen Klassendiagrammen deutlich schlechtere Ergebnisse in der Laufzeit. Dies ist darauf zurückzuführen, dass EMF Compare in einem Top-down-Verfahren im Wesentlichen nur die Klassen bereits korrepondierender Pakete miteinander vergleicht. Liegen alle Klassen in dem gleichen Paket, müssen alle Klassenkombinationen berücksichtigt werden, was bei den großen Diagrammen die Laufzeit auf bis zu 30,47 Sekunden erhöht. Im Durchschnitt benötigt der Vergleich zweier Klassendiagramme der Testmenge M' 4,744 Sekunden. Dies entspricht mehr als der doppelten durchschnittlichen Laufzeit des ECVerfahrens für die gleiche Testmenge.

Da in den Gesamtaufwand der eigenentwickelten Verfahren auch der Aufwand für das Lösen der kleineren Teilprobleme, wie die Zuordnung der Attribute und Methoden, einfließt, ist die Laufzeit nicht nur von der Anzahl der Klassen, sondern auch von der Anzahl der Klassenunterelemente und deren Verteilung abhängig. Um das Laufzeitverhalten des ECVerfahrens und des Ähnlichkeitsverfahrens in Abhängigkeit von der Zahl der Klassen, die in den zu vergleichenden Modellen enthalten sind, zu visualisieren, wurde eine Testmenge L konstruiert, bei der der Vergleich zweier Klassen als konstant betrachtet werden kann. Dies wird dadurch erreicht, dass jede Klasse genau drei Attribute und drei Methoden enthält. Es wurden für verschiedene Klassenzahlen Klassendiagramme konstruiert, wobei für die Laufzeittests je zwei identische Klassendiagramme miteinander verglichen wurden. Die Laufzeiten ergeben sich aus dem Mittel von drei Testläufen. Die Diagramme 6.1 und 6.2 zeigen das Laufzeitverhalten des edierkostenbasierten bzw. des ähnlichkeitsbasierten Verfahrens im Einzelnen. Das Diagramm 6.3 stellt die Laufzeiten der beiden Verfahren gegenüber, wobei die Laufzeit des edierkostenbasierten Verfahrens mit zunehmender Klassenzahl deutlich schneller als die Laufzeiten des ähnlichkeitsba-

Tabelle 6.5: Laufzeiten des BFVerfahrens

Anzahl Klassen	5	6	7	8	9	10
Laufzeit (Sekunden)	0,077	0,473	3,686	33,952	358,394	3865,354

sierten Verfahrens ansteigt.

Für die Brute-Force-Variante wurden ebenfalls Laufzeitstest durchgeführt, aufgrund der hohen Komplexität des Verfahrens jedoch in einem weniger ausführlichen Umfang. Für die 66 Testfälle der Testmenge E benötigt die Brute-Force-Variante insgesamt ungefähr 38 Minuten, während das ECVerfahren das Ergebnis bereits nach ca. 0,25 Sekunden liefert. Die Tabelle 6.5 gibt die Laufzeiten für den Vergleich zweier Klassendiagramme der Testmenge L wieder. Während der Brute-Force-Vergleich zweier Klassendiagramme mit je 9 Klassen ca. 6 Minuten benötigt, wird für je 10 Klassen schon mehr als eine Stunde benötigt. Aufgrund der Komplexität des Aufwands für das Kombinieren der möglichen Klassenkorrespondenzen ist zu erwarten, dass die Laufzeit des Vergleichs zweier Klassendiagramme mit n Klassen n-mal länger dauert als der Vergleich zweier Klassendiagramme mit $n-1$ Klassen.

Die Brute-Force-Variante mit Schwellenwertregel ist deutlich schneller als die Standardvariante. Bei einem Vergleich zweier Klassendiagramme mit je 11 Klassen, der in der Brute-Force-Variante ca. 160 Minuten dauert, konnten beispielsweise 52 Kanten aus dem Vergleichsgraphen entfernt werden, so dass die Berechnung nur noch ca. 9 Minuten benötigte. Durch das Entfernen einzelner Kanten aus dem Vergleichsgraphen auf Klassenebene reduziert sich die Komplexität im Allgemeinen jedoch nicht so stark, dass die Schwellenwertvariante des Brute-Force-Verfahrens damit für die großen Testfälle der Menge M mit vertretbarem Aufwand einsetzbar wäre.

Wie die Laufzeittests zeigen, liegen die Rechenzeiten des ähnlichkeitsbasierten Verfahrens und des edierkostenbasierten Verfahrens mit Schwellenwertregel für die betrachteten Modellgrößen unter der Zeit, die für das Öffnen des EMF Compare UI Editors benötigt wird. Daher ist davon auszugehen, dass ein Anwender, der in Eclipse arbeitet und einen Einzelvergleich manuell auslöst, die Wartezeit als unproblematisch einstuft. Wenn das edierkostenbasierte Verfahren die besseren Ergebnisse liefert, ist sicherlich auch eine Wartezeit von wenigen Sekunden für den Vergleich zweier Klassendiagramme mittlerer Größe zu vertreten.

6.3 Evaluation des edierkostenbasierten Verfahrens unter dem Aspekt der Optimalität

Bei dem edierkostenbasierten Verfahren handelt es sich im Gegensatz zu dem ähnlichkeitsbasierten Verfahren um kein Greedy-Verfahren, bei dem eine Reihe lokal getroffener Entscheidungen zur Lösung führt. Stattdessen ist es in jedem Schritt möglich, vorherige Entscheidungen wieder zu revidieren, um schließlich eine global optimale Lösung des relaxierten Problems zu erhalten. Um zu untersuchen, ob diese theoretisch möglichen

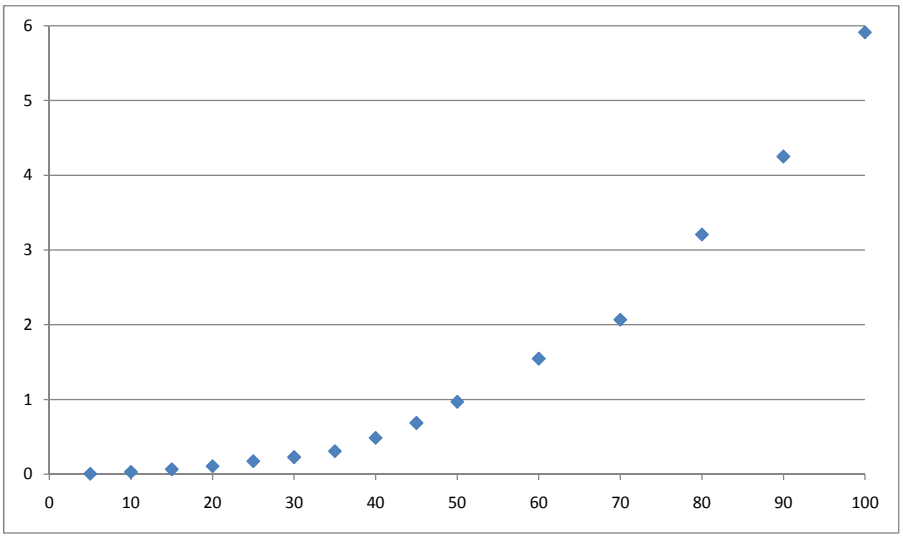

Abbildung 6.1: Laufzeit des ECVerfahrens (Sekunden)

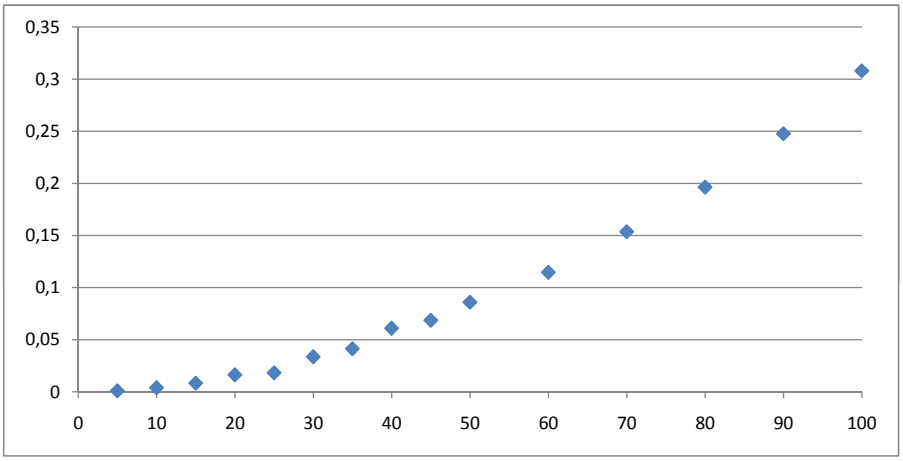

Abbildung 6.2: Laufzeit des Ähnlichkeitsverfahrens (Sekunden)

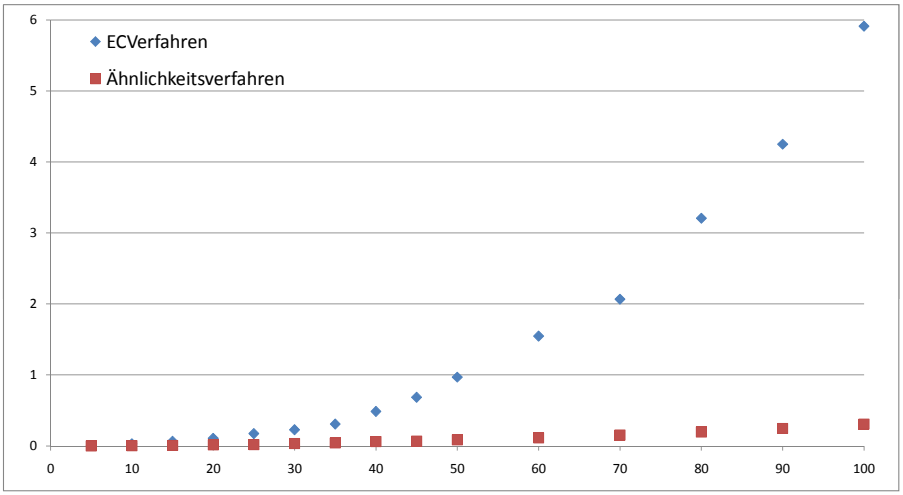

Abbildung 6.3: Laufzeit der Verfahren im Vergleich (Sekunden)

Revisionen der Zwischenlösungen auch in der Praxis durchgeführt werden, wurden die Testfälle der Menge A auch dahingehend untersucht, ob im Busacker-Gowen-Verfahren bereits ausgewählte Kanten des Netzwerkflussproblems in einem späteren Schritt als Rückwärtskanten durchlaufen werden. Bei den 2775 Vergleichen der Testmenge A finden in 1328 Fällen eine oder mehrere Revisionen der Entscheidungen auf Klassenebene statt, was einem Anteil von 47,86% entspricht. Bezieht man die darunterliegenden Ebenen der Attribute, Methoden etc. ein, wird sogar in jedem Testfall der Menge A mindestens einmal eine Flussvergrößerung über Rückwärtskanten ausgeführt. Da das edierkostenbasierte Verfahren eine optimale Lösung für das relaxierte Verfahren berechnet, werden im Folgenden die Ergebnisse des Verfahrens in Hinblick auf die Optimalität für das ursprüngliche Optimierungsproblem untersucht.

In Kapitel 4 wurde für das edierkostenbasierte Verfahren, das mit den abgeschätzten Kosten des relaxierten Problems arbeitet, ein hinreichendes Optimalitätskriterium angegeben. Falls die tatsächlichen Kosten des relaxierten Problems mit den abgeschätzten Kosten identisch sind, ist die optimale Lösung des relaxierten Problems auch für das ursprüngliche Optimierungsproblem optimal. Die zweite Spalte der Tabelle 6.6 enthält für die verschiedenen Testmengen die Prozentsätze, in wie vielen Fällen das hinreichende Optimalitätskriterium erfüllt ist, wobei die sehr unterschiedlichen Ergebnisse von 12,92% bis zu 100% reichen.

Da das Optimalitätskriterium zwar hinreichend aber nicht notwendig ist und auch in Fällen, wenn die tatsächlichen Kosten die abgeschätzten Kosten übersteigen, eine optimale Lösung gefunden worden sein kann, wurden mit Hilfe der Brute-Force-Variante die Kosten der optimalen Lösungen ermittelt und mit den Kosten der Lösungen des abgeschätzten Verfahrens verglichen. Die Ergebnisse dieses Vergleichs in der dritten Spalte von Tabelle 6.6 zeigen, dass die optimale Lösung in deutlich mehr Fällen erreicht wurde. Für die Testmenge A wird das hinreichende Optimalitätskriterium nur in 17,77% der Fälle erfüllt, die optimale Lösung jedoch dennoch in 91,96% erreicht. Folglich stellt das hinreichende Optimalitätskriterium im Allgemeinen einen relativ schlechten Indikator für die Optimalität der Ergebnisse dar.

Darüber hinaus fällt auf, dass für die Testmenge C die optimale Lösung nur in 21,24% der Fälle erreicht wird, während die ermittelten Trefferquoten für die anderen Testmengen bei 89,25% bis 100% liegen. Da die Klassendiagramme in Testmenge C im Gegensatz zu den Klassendiagrammen der Testmengen A und B einen größeren Anteil an kontextabhängigen Informationen enthalten, ist dies ein Hinweis darauf, dass das Verfahren dann besser abschneidet, wenn viele kontextunabhängige Informationen in den Modellen zur Verfügung stehen.

Mit den Ergebnissen der Brute-Force-Variante lässt sich weiterhin untersuchen, wie stark die Kosten des ECVerfahrens von den optimalen Kosten abweichen. Die Spalten 4 und 5 der Tabelle 6.6 zeigen die Gesamtkosten aller Ergebnisse des Brute-Force-Verfahrens bzw. des Verfahrens mit abgeschätzten Kosten.[1] Die letzte Spalte der Tabelle

[1] Die Brute-Force-Variante lässt sich leider nur bis zu Modellgrößen von 11 Klassen einsetzen. Die Höhe der Gesamtkosten der Brute-Force-Variante entspricht im Fall der Menge F, da das hinreichende Optimalitätskriterium zu 100% erfüllt ist, den Gesamtkosten des ECVerfahrens.

Tabelle 6.6: Evaluation des edierkostenbasierten Verfahrens in Hinblick auf die Optimalität der Ergebnisse

Menge	Untere Schranke erreicht?	Optimale Lösung erreicht?	Gesamtkosten BF	Gesamtkosten EC	Verhältnis der Gesamtkosten
A	17,77%	91,96%	265415,04	271191,97	102,18%
B	64,69%	89,25%	192671,36	193879,95	100,63%
C	12,92%	21,24%	157165,22	166970,04	106,24%
E	100%	100%	1539,62	1539,62	100%
F	100%	100%	(885,25)	885,25	100%
M	36,19%	?	?	18706,44	?

enthält das Verhältnis der Gesamtkosten beider Verfahren. Mit einem Verhältnis von 100% bis 106,24% ist die Abweichung der Kosten der Lösung des ECVerfahrens von den optimalen Kosten im Schnitt sehr gering, wobei auch hier für die Testmenge C die größte Abweichung erreicht wird.

In Kapitel 4 wurde bereits ein Beispiel angegeben, bei dem das hinreichende Optimalitätskriterium erfüllt ist (Beispiel 10). Es folgen nun zwei weitere Beispiele für die anderen beiden Situationen. Beispiel 18 demonstriert, dass das Optimalitätskriterium nur hinreichend, aber nicht notwendig ist. Beispiel 19 zeigt im Anschluss mit einem Fall, in dem die kostenoptimale Lösung des ursprünglichen Problems nicht gefunden wird, die Grenzen des ECVerfahrens.

Beispiel 18 (Hinreichendes Optimalitätskriterium wird nicht erfüllt, die Lösung ist dennoch optimal)**.** Der Unterschied der beiden Klassendiagramme aus Abbildung 6.4 besteht in der gerichteten Assoziation `reservedBy` der Klasse `Vehicle`, die in einem Fall auf die Klasse `ReservationContract` und in dem anderen Fall auf die Klasse `Client` verweist. Die Listings 6.1 und 6.2 zeigen die relevanten Ausschnitte aus den berechneten geschätzten Kosten bzw. den im Nachhinein berechneten tatsächlichen Kosten. Die geschätzten Kosten der Korrespondenz (`ReservationContract`, `ReservationContract`) betragen 1,5, was den anteiligen Kosten des Löschens/Einfügens einer Assoziation entspricht, da nur eine der beiden Klassen eine eingehende Assoziation `reservedBy` besitzt. Analog ergeben sich die Kosten für (`Client`, `Client`) in Höhe von 1,5. Durch die optimistische Annahme, dass die beiden ausgehenden Assoziationen `reservedBy` der Klassen `Vehicle` korrespondieren, fallen für die Korrespondenz (`Vehicle`, `Vehicle`) keine weiteren Kosten an. Die geschätzten Gesamtkosten betragen somit 3. Bei der Berechnung der tatsächlichen Kosten bei bekanntem Kontext ergeben sich Kosten in Höhe von 6. Die Kosten für die Zuordnungen (`ReservationContract`, `ReservationContract`) und (`Client`, `Client`) entsprechen den tatsächlichen Kosten. Da nun bekannt ist, dass die Assoziationsenden der Assoziationen `reservedBy` nicht korrespondieren, fallen zusätzliche Kosten in Höhe von 3 für das anteilige Löschen/Einfügen zweier Assoziationen an. Damit übersteigen die tatsächlichen Kosten die abgeschätzten Kosten und das hinreichende Optimalitätskriterium ist nicht erfüllt. Das Ergebnis entspricht jedoch der

Lösung der Brute-Force-Variante und ist damit optimal. In diesem Fall entspricht die kostenoptimale Lösung auch dem Vergleichsergebnis, das der Benutzer erwartet.

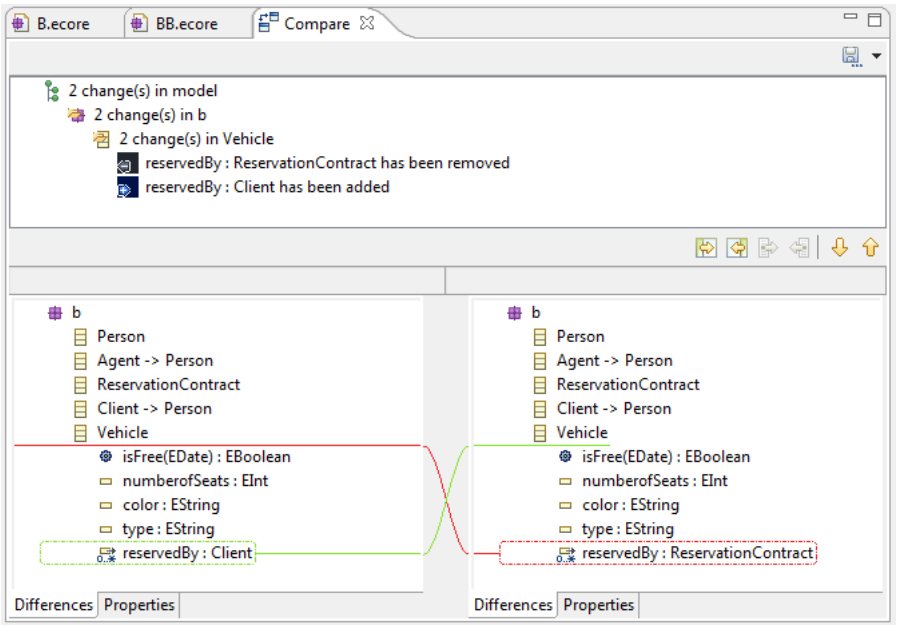

Abbildung 6.4: Differenzbericht des ECVerfahrens für zwei Klassendiagramme, die sich nur im Ziel der unidirektionalen Assoziation `reservedBy` unterscheiden. Die Assoziation wird als gelöscht und mit anderem Zieltyp neu eingefügt identifiziert.

Listing 6.1: Ausschnitt aus den geschätzten Kosten (Gesamtkosten: 3)

```
ReservationContract - ReservationContract ( 1.5 )
  In_Referenzen ( 1.5 )
    Matching:
      e - reservedBy ( 1.5 )
Client - Client ( 1.5 )
  In_Referenzen ( 1.5 )
    Matching:
      reservedBy - e ( 1.5 )
Vehicle - Vehicle ( 0.0 )
  Out_Referenzen ( 0.0 )
    Matching:
      reservedBy - reservedBy ( 0.0 )
```

Listing 6.2: Ausschnitt aus den tatsächlichen Kosten (Gesamtkosten: 6)

```
ReservationContract - ReservationContract ( 1.5 )
  In_Referenzen 3 ( 1.5 )
    Matching:
```

```
      e - reservedBy     ( 1.5 )
Client - Client     ( 1.5 )
  In_Referenzen 1 ( 1.5 )
    Matching:
      reservedBy - e     ( 1.5 )
Vehicle - Vehicle     ( 3.0 )
  Out_Referenzen 2 ( 1.5 )
    Matching:
      e - reservedBy     ( 1.5 )
  Out_Referenzen 1 ( 1.5 )
    Matching:
      reservedBy - e     ( 1.5 )
```

Beispiel 19 (Die optimale Lösung wird nicht erreicht)**.** Die Abbildungen 6.5 und 6.6 zeigen zwei Modelle, für die das ECVerfahren nicht die kostenoptimale Lösung findet. Der Differenzbericht in Abbildung 6.7 zeigt das Ergebnis des Vergleichs mit dem ECVerfahren. Die geschätzten Kosten betragen 35,126351072985884, die tatsächlichen Kosten der Lösung 37,947206687959145. Die Kosten der optimalen Lösung, die mit Hilfe der Brute-Force-Variante ermittelt wurde, betragen jedoch nur 37,328947368421055 und liegen damit knapp unter den tatsächlichen Kosten der Lösung des ECVerfahrens. Der Differenzbericht des BFVerfahrens ist in Abbildung 6.8 abgebildet. Die berechneten Korrespondenzen der beiden Lösungen unterscheiden sich in der Zuordnung der Klassen `Client` und `bus`. Während bei der Lösung des Brute-Force-Verfahrens die beiden Klassen `Client` als korrespondierend identifiziert werden und die Klasse `bus` gelöscht wird, ordnet das ECVerfahren (ϵ, `Client`) und (`Client`, `bus`) zu. Dies ist darauf zurückzuführen, dass diese Lösung aufgrund der abgeschätzten Kosten günstiger erscheint (siehe Tabelle 6.7). Die Differenz in Höhe von 2,820855614973261 zwischen den tatsächlichen Kosten und den abgeschätzten Kosten des ECVerfahrens ist wie folgt zustande gekommen: aufgrund des Löschens der Klasse `Client` fallen zusätzliche Kosten in Höhe von 2 für das Löschen der Vererbungskante `Client` → `Person` an. Ebenso ist die Korrespondenz der Klassen `Contract-ReservationContract` um 0,8208556149732623 teurer als abgeschätzt, wie die relevanten Ausschnitte aus den berechneten geschätzten Kosten bzw. den im Nachhinein berechneten tatsächlichen Kosten zeigen (Listings 6.3 und 6.4). Die vollständigen Kostenlisten sowie die Kostenliste des BFVerfahrens sind in Anhang A.3.2 abgedruckt.

Die ähnlichkeitsbasierte Variante liefert in diesem Fall das gleiche Resultat wie die Brute-Force-Variante. Unter der Einschränkung, dass eine eindeutige Zuordnung berechnet wird, ist diese Lösung auch aus Sicht des Benutzers optimal. Bei erlaubter Mehrfachzuordnung wäre es möglich, die beiden Refactoringmaßnahmen besser abzubilden. In diesem Fall könnte die Klasse `Vehicle` mit den beiden Klassen `bus` und `car` korrespondieren. Analog könnten die Attribute `firstname` und `lastname` der Klassen `Client` und `Agent` jeweils auf die Attribute `firstname` und `lastname` der Klasse `Person` abgebildet werden, was jedoch schon in Kapitel 4.4 behandelt wurde.

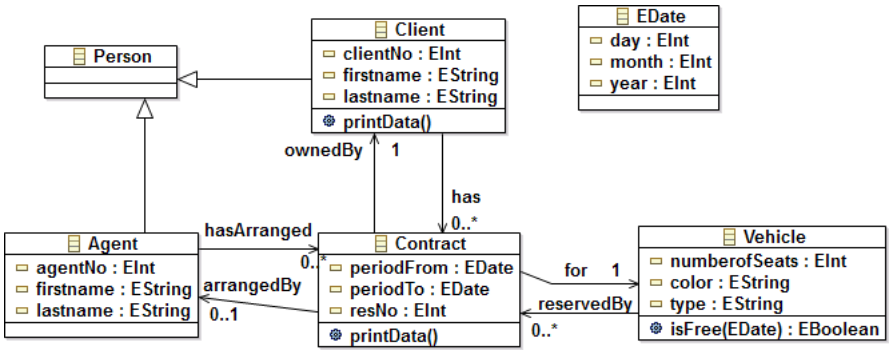

Abbildung 6.5: Modell A aus Beispiel 19

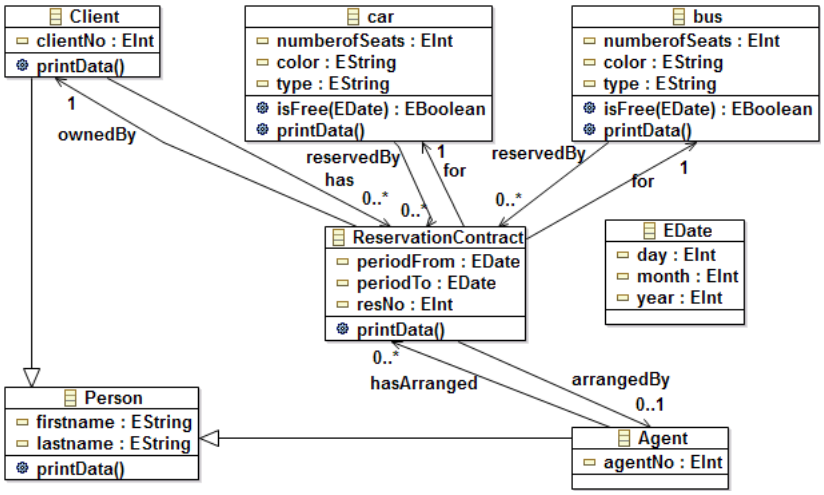

Abbildung 6.6: Modell B aus Beispiel 19

Tabelle 6.7: Werte der abgeschätzten Kosten des Vergleichsgraphen auf Klassenebene. Die abweichenden tatsächlichen Kosten sind gegebenenfalls in Klammern angegeben.

	Person	Agent	Contract	Client	Vehicle	EDate	ϵ
Person	**6(8)**	10,705	16,976	8,666	12,776	10,167	10
Agent	8,705	**4**	15,084	7,602	11,445	9,254	8
ReservationContract	22,75	17,494	**3,578(4,399)**	16,065	19,054	18,962	22
Client	10,666	7,602	13,575	4	10,875	11,337	**10**
car	17,714	10,927	16,346	8,797	**2,75**	13,723	17
EDate	8,705	8,791	15,833	10,874	11,723	**0**	8
bus	17,714	10,927	16,41	**8,797**	2,823	13,826	17

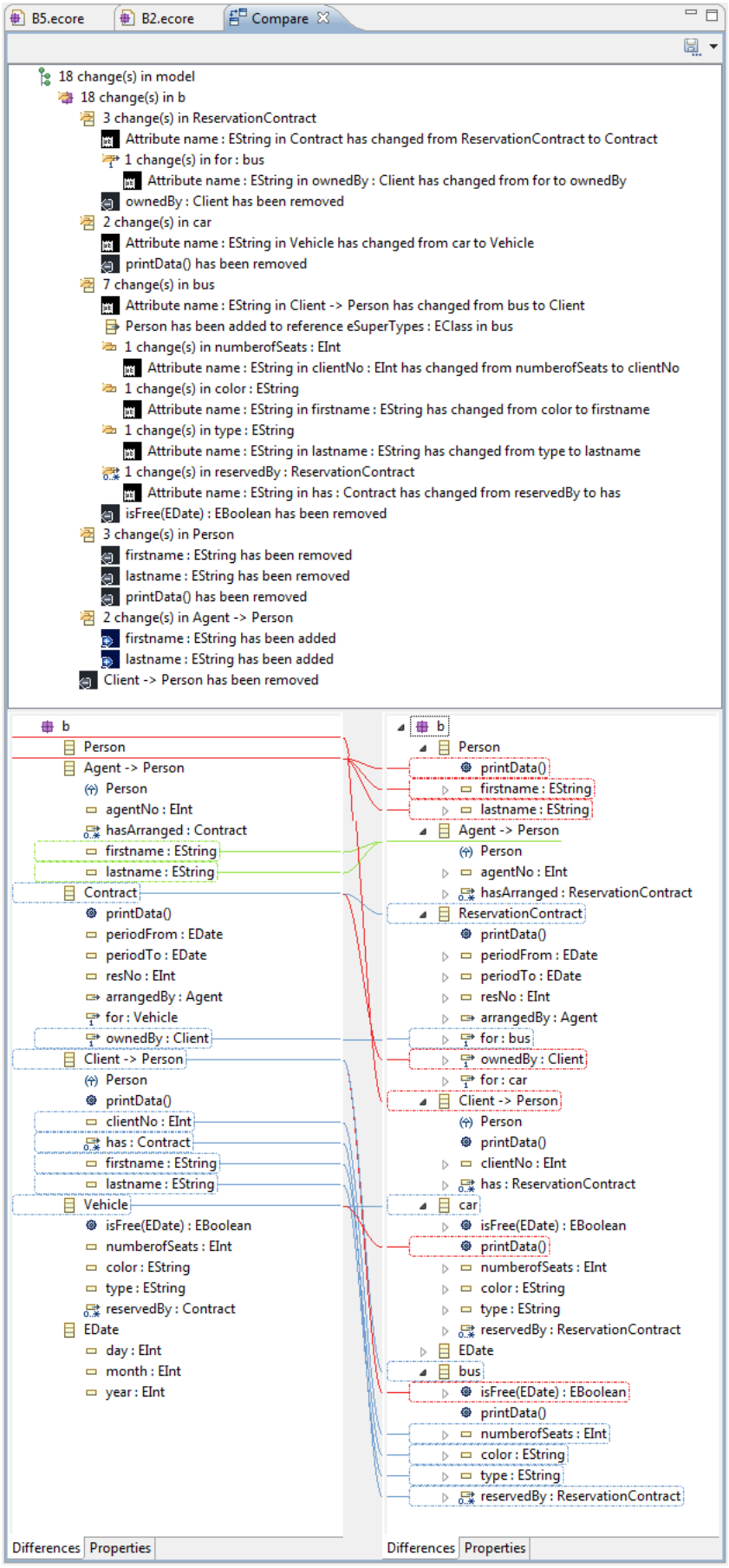

Abbildung 6.7: Differenzbericht des ECVerfahrens zu Beispiel 19

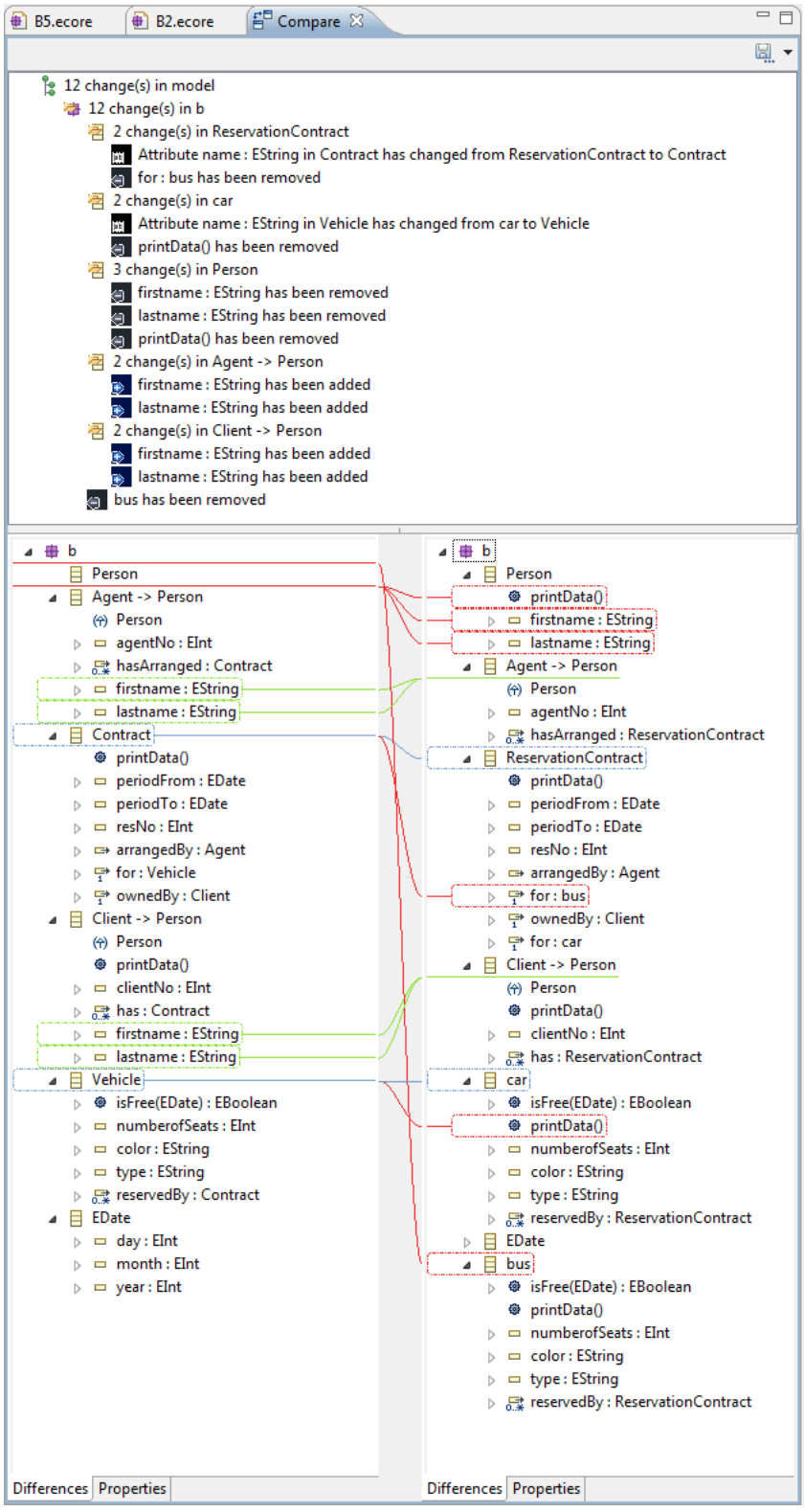

Abbildung 6.8: Differenzbericht des BFVerfahrens zu Beispiel 19

Listing 6.3: Geschätzte Kosten der Korrespondenz `Contract-ReservationContract`

```
Contract - ReservationContract ( 3.5789473684210527 )
    In_Referenzen ( 1.5 )
      Matching:
        hasArranged - hasArranged ( 0.0 )
        has - has ( 0.0 )
        reservedBy - reservedBy ( 0.0 )
        reservedBy - e ( 1.5 )
    Out_Referenzen ( 1.5 )
      Matching:
        arrangedBy - arrangedBy ( 0.0 )
        for - for ( 0.0 )
        ownedBy - ownedBy ( 0.0 )
        for - e ( 1.5 )
```

Listing 6.4: Tatsächliche Kosten der Korrespondenz `Contract-ReservationContract`

```
Contract - ReservationContract ( 4.399802983394315 )
    In_Referenzen 1 ( 1.5 )
      Matching:
        has - e ( 1.5 )
    In_Referenzen ( 0.0 )
      Matching:
        reservedBy - reservedBy ( 0.0 )
    In_Referenzen ( 0.4090909090909091 )
      Matching:
        reservedBy - has ( 0.4090909090909091 )
    In_Referenzen ( 0.0 )
      Matching:
        hasArranged - hasArranged ( 0.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        arrangedBy - arrangedBy ( 0.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        for - for ( 0.0 )
    Out_Referenzen ( 0.4117647058823529 )
      Matching:
        for - ownedBy ( 0.4117647058823529 )
    Out_Referenzen 1 ( 1.5 )
      Matching:
        ownedBy - e ( 1.5 )
```

Auch wenn die Kosten der nicht optimalen Lösung des ECVerfahrens für das Beispiel 19 nur 101,66% der optimalen Kosten betragen, enthält das Ergebnis bei 6 erwarteten Zuordnungen auf Klassenebene eine falsche Zuordnung (`Client`, `bus`) und eine daraus resultierende fehlende Zuordnung (`Client`, `Client`). Bei dem edierkostenbasierten An-

Tabelle 6.8: Evaluation des edierkostenbasierten Verfahrens mit Schwellenwert (EC-S) in Hinblick auf die Optimalität der Ergebnisse

Menge	Optimale Lösung erreicht?	Gesamtkosten BF	Gesamtkosten EC-S	Verhältnis der Gesamtkosten
A	13,77%	265415,04	409965,00	154,46%
B	34,16%	192671,36	259191,59	134,53%
C	10,02%	157165,22	219928,20	139,93%

satz zeigt sich allgemein die Problematik, dass eine optimale Lösung von einer Lösung mit fast optimalen Kosten in den Korrespondenzen sehr stark abweichen kann. Weiterhin ist es möglich, dass verschiedene Lösungen mit optimalen Kosten existieren, die aus Sicht des Benutzers nicht alle gleich gut sind.

Evaluation des edierkostenbasierten Verfahrens mit Schwellenwert unter dem Aspekt der Optimalität

Die Fehlzuordnung (`Client`, `bus`) des ECVerfahrens in Beispiel 19 lässt sich mit der Einführung einer Schwellenwertregel verhindern, so dass auch das ECVerfahren die optimale Lösung berechnet. Im Fall von Beispiel 19 verbessert sich die Lösung durch die Schwellenwertregel nicht nur aus Sicht des Nutzers, sondern auch in Hinblick auf die Kosten. Die berechnete Lösung stimmt sogar mit der kostenminimalen Lösung des ursprünglichen Problems überein. Dieser Effekt ist im Allgemeinen jedoch nicht zu erwarten. Da nur ein Teil des Lösungsraums durchsucht wird und damit bestimmte Kombinationen von Klassen ausgeschlossen werden, liegen die Kosten der gefundenen Lösung meist über den Kosten der Brute-Force Variante. Die Werte in Tabelle 6.8 belegen, dass das edierkostenbasierte Verfahren mit Schwellenwertregel deutlich seltener die kostenoptimale Lösung berechnet. Während das ECVerfahren in der Standardvariante für die drei Testmengen A, B und C durchschnittlich in 67,48% der Fälle die optimale Lösung berechnet, wird die kostenoptimale Lösung vom ECVerfahren mit Schwellenwertregel nur in 19,32% der Fälle gefunden. Auch die durchschnittlichen Gesamtkosten der berechneten Lösungen in Höhe von 142,97% übersteigen die Gesamtkosten des ECVerfahrens in der Standardvariante in Höhe von 103,02% deutlich. Für das ECVerfahren mit Schwellenwertregel sind im Gegensatz zu dem Verfahren in der Standardvariante zwei Einschränkungen zu berücksichtigen, die für die Praxis jedoch nicht relevant sind. Die Übereinstimmung der geschätzten Kosten mit den tatsächlichen Kosten ist in diesem Fall kein hinreichendes Kriterium, ob die Lösung des relaxierten Problems auch für das ursprüngliche Problem optimal ist. Weiterhin wird durch die Schwellenwertregel in das Kostenmodell eingegriffen, so dass nicht mehr gewährleistet ist, dass die Kosten des Vergleichsergebnisses einem Abstand zwischen den beiden Modellen entsprechen, der die Kriterien einer Metrik erfüllt.

Für die Beispiele 18 und 19 liefert das Brute-Force-Verfahren ebenso wie das ähnlichkeitsbasierte Verfahren eine identische und aus Sicht des Benutzers korrekte Lösung. Jedoch bedeutet das Erreichen der kostenoptimalen Lösung nicht immer, dass diese Lösung

Tabelle 6.9: Vergleich der Ergebnisse des ähnlichkeitsbasierten Verfahrens mit (A) und ohne Anpassung (O) der kontextabhängigen Ähnlichkeiten sowie des edierkostenbasierten Verfahrens (EC) auf identische Korrespondenzmodelle

Testmenge (Größe)	O = A	O = EC	A = EC
A (2775)	2580 (92,97%)	670 (24,14%)	696 (25,08%)
B (4950)	4852 (98,02%)	2185 (44,14%)	2243 (45,31%)
C (2415)	2135 (88,41%)	413 (17,10%)	439 (18,18%)

Tabelle 6.10: Ergebnisse des ähnlichkeitsbasierten Verfahrens mit (A) und ohne (O) Anpassung in Relation zu den Kosten des Brute-Force Verfahrens

Testmenge (Größe)	Kosten O (Verhältnis BF-Kosten)	Kosten A (Verhältnis BF-Kosten)
A (2775)	324595,50 (122,30%)	319660,03 (120,44%)
B (4950)	215659,19 (111,93%)	214027,34 (111,08%)
C (2415)	191266,64 (121,70%)	185481,22 (118,02%)

auch den Erwartungen des Benutzers entspricht. Insgesamt ist eine Untersuchung der Optimalität der Ergebnisse nicht ausreichend, um die Qualität der berechneten Zuordnungen beurteilen zu können. Daher wird in Abschnitt 6.5 der Benutzer zur Bewertung der Ergebnisse beider eigenentwickelten Verfahren miteinbezogen. Doch zunächst wird im folgenden Abschnitt das ähnlichkeitsbasierte Verfahren dahingehend überprüft, ob sich die Version mit der iterativen Anpassung der Ähnlichkeiten von der Grundversion unterscheidet.

6.4 Evaluation des ähnlichkeitsbasierten Verfahrens mit Anpassung der Ähnlichkeiten auf Klassenebene

Um zu untersuchen, inwieweit sich die Ergebnisse durch die in Abschnitt 4.3.2 vorgestellte Anpassung der Ähnlichkeitswerte im 1-Kontext der Klassen unterscheiden, wurden zwei Hilfsmethoden implementiert: eine Methode, die überprüft, ob zwei Korrespondenzmodelle identisch sind sowie eine Methode, die ein vorgegebenes Korrespondenzmodell mit Kosten bewertet, die dem Kostenmodell des edierkostenbasierten Verfahrens entsprechen. Auch wenn das ähnlichkeitsbasierte Verfahren Korrespondenzen nicht unter Berücksichtigung der Edierkosten, sondern nach Ähnlichkeitswerten bildet, liefert diese Kennzahl dennoch einen Eindruck über die Unterschiedlichkeit der Ergebnisse.

Beim Vergleich der ähnlichkeitsbasierten Verfahren mit und ohne Anpassung stimmen die berechneten Korrespondenzmodelle für die Testmengen A, B und C im Schnitt in 93,13 % der Fälle überein (vgl. Tabelle 6.9). Die Korrespondenzmodelle des ähnlichkeitsbasierten Verfahrens ohne Anpassung und des edierkostenbasierten Verfahrens stimmen hingegen nur in durchschnittlich 28,46 % der Fälle überein. Der Prozentsatz

der übereinstimmenden Korrespondenzmodelle erhöht sich leicht auf 29,52%, wenn das ähnlichkeitsbasierte Verfahren mit Anpassung verwendet wird (vgl. Spalte 4 der Tabelle 6.9). Insgesamt zeigt sich jedoch, dass sich die Ergebnisse des ähnlichkeitsbasierten Verfahrens mit und ohne Anpassung relativ wenig voneinander unterscheiden, sich aber deutlich von denen des edierkostenbasierten Verfahrens abgrenzen. Dieser Eindruck spiegelt sich auch wieder, wenn die Ergebnisse der Verfahren in Hinblick auf die Kosten betrachtet werden. Tabelle 6.10 zeigt die Gesamtkosten der berechneten Zuordnungen aller Testfälle der Mengen A, B bzw. C. In jeder Testgruppe sinken durch Anpassung der Ähnlichkeitswerte die Gesamtkosten gegenüber den Gesamtkosten des Standardverfahrens ohne Anpassung leicht. Geringere Kosten einer Zuordnung deuten darauf hin, dass mehr Korrespondenzen gebildet wurden. Vergleicht man das durchschnittliche Verhältnis der Gesamtkosten zu den Gesamtkosten des Brute-Force-Verfahrens, liegt das ähnlichkeitsbasierte Verfahren ohne Anpassung mit 118,64% knapp hinter dem ähnlichkeitsbasierten Verfahren mit Anpassung (116,51%). Der Durchschnitt über die Verhältnisse der Gesamtkosten der gleichen Testmengen liegt für das edierkostenbasierte Verfahren bei 103,02% (vgl. Tabelle 6.6, S. 188).

Wie der Tabelle zu entnehmen ist, ändert sich das Ergebnis durch die Anpassung der Ähnlichkeitswerte nur in wenigen Fällen. Wie in Beispiel 14 auf Seite 125 demonstriert wurde, kann die Anpassung der Ähnlichkeiten das erzielte Ergebnis verbessern, indem zusätzliche Korrespondenzen gebildet werden. Das folgende Beispiel behandelt jedoch den anderen Fall, in dem durch Anpassung der Ähnlichkeiten weniger Korrespondenzen gebildet werden und diese aus Sicht des Benutzers sogar ungünstiger sind.

Beispiel 20 (Anpassung verschlechtert das Ergebnis aus Sicht des Benutzers)**.** Die Abbildungen 6.9 und 6.10 zeigen zwei Klassendiagramme der Testmenge C, die in diesem Beispiel verglichen werden. Das ähnlichkeitsbasierte Verfahren ohne Anpassung berechnet, wie in Tabelle 6.11 zu sehen ist, die Korrespondenzen (`Edge`, `Edge`), (`Task`, `Task`), (`Workflow`, `Workflow`), (`InputPort`, `Input`), (`WorkflowElement`, `WorkflowElement`), (`WorkflowNode`, `WorkflowNode`), (`OutputPort`, `Output`) und (`Choice`, `Comment`) in dieser Reihenfolge. Die Klassen `Transformation` und `ConditionalOutputPort` aus Modell A werden als Einzelelemente identifiziert. Dies entspricht bis auf eine Ausnahme, die Zuordnung der Klassen `Choice` und `Comment`, den Erwartungen des Benutzers. Der Ähnlichkeitswert von (`Choice`, `Comment`) in Höhe von 0,6839 ist darauf zurückzuführen, dass der Klassentyp übereinstimmt, beide Klassen weder Attribute noch Methoden noch Unterklassen besitzen, die Klassennamen in Höhe des Werts 0,444 ähnlich sind und jeweils eine Vererbungsbeziehung zu einer Oberklasse mit einem Ähnlichkeitswert von 0,3181 besitzen.[2]

Auch das ähnlichkeitsbasierte Verfahren mit Anpassung der kontextabhängigen Ähnlichkeiten im 1-Kontext identifiziert die Klassen und `Edge` und `Edge` als erste Korrespondenz. Nach der Korrespondenzbildung erhöhen sich die Ähnlichkeitswerte der Paare {`Workflow`, `InputPort`, `OutputPort`} × {`Workflow`, `Input`, `Output`} aufgrund von

[2] An dieser Stelle zeigt sich eventuell eine Verbesserungsmöglichkeit in Form einer dynamischen Gewichtung der einzelnen Ähnlichkeitswerte. So könnte der Einfluss der Ähnlichkeiten der Attribute und Methoden reduziert werden, falls keine derartigen Elemente existieren.

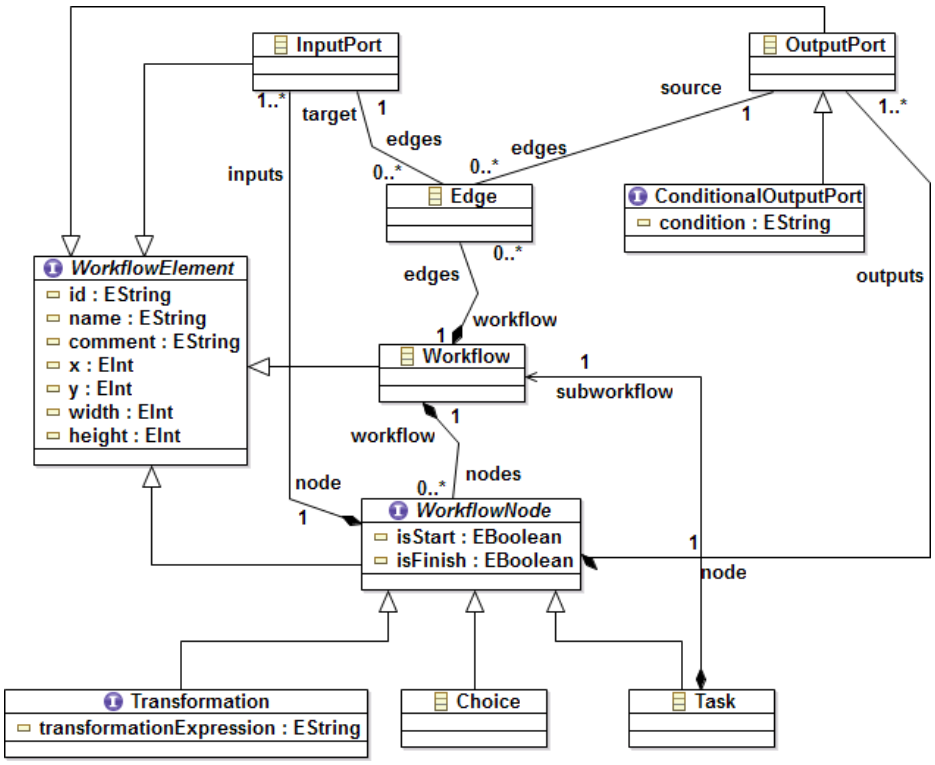

Abbildung 6.9: Modell A zu Beispiel 20

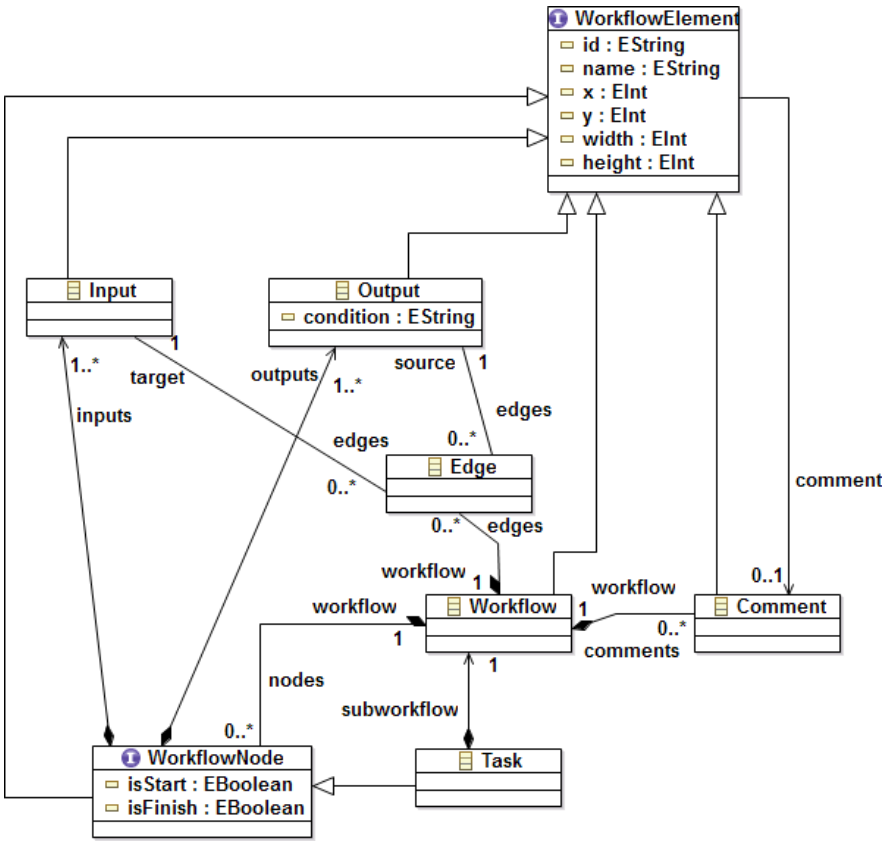

Abbildung 6.10: Modell B zu Beispiel 20

Tabelle 6.11: Beispiel 20: Ähnlichkeitsverfahren ohne Anpassung

	Workflow-Element	Workflow-Node	Comment	Workflow	Edge	Task	Input	Output
Workflow-Element (5)	**0,6941**	0,295	0,2291	0,2555	0,2812	0,2752	0,196	0,2165
Workflow-Node (6)	0,3273	**0,6416**	0,2989	0,392	0,292	0,2419	0,2817	0,3036
Transformation	0,496	0,425	0,3219	0,332	0,2708	0,4152	0,3141	0,4176
Conditional-OutputPort	0,4936	0,4114	0,3024	0,3007	0,2685	0,3657	0,2966	0,476
Workflow (3)	0,2756	0,3554	0,7335	**0,8743**	0,675	0,6639	0,7263	0,5659
Edge (1)	0,292	0,2627	0,6834	0,6806	**0,9555**	0,6568	0,6687	0,4949
Task (2)	0,3216	0,2419	0,7006	0,6609	0,6568	**0,9333**	0,6855	0,5129
Choice (8)	0,2892	0,2408	**0,6839**	0,6553	0,6388	0,75	0,6588	0,4987
InputPort (4)	0,2318	0,3095	0,7397	0,7461	0,6705	0,6626	**0,8009**	0,605
OutputPort (7)	0,24	0,3168	0,6523	0,6587	0,5834	0,5767	0,6775	**0,5583**

Assoziationen (vgl. Tabelle 6.12). Als nächste Korrespondenz wird (`Task`, `Task`) identifiziert. Dies führt zu einer Anpassung der Ähnlichkeitswerte für (`Workflow`, `Workflow`), da jeweils eine unidirektionale Assoziation zwischen `Task` und `Workflow` existiert (vgl. Tabelle 6.13). Außerdem wird der Ähnlichkeitswert für (`WorkflowNode`, `WorkflowNode`) aufgrund der gemeinsamen Unterklasse `Task` angepasst, was sich hier im Gesamtähnlichkeitswert jedoch nicht auswirkt. Im dritten Schritt wird die Korrespondenz (`Workflow`, `Workflow`) gebildet. Bei der Anpassung der Ähnlichkeitswerte erhöhen sich die Werte für {`WorkflowNode`, `Comment`} × {`WorkflowNode`} aufgrund von Assoziationen (vgl. Tabelle 6.14). Die Klassen `Task` und `Edge` werden in die Anpassung nicht mehr miteinbezogen, da sie bereits als korrespondierend identifiziert sind. Der Ähnlichkeitswert für (`WorkflowElement`, `WorkflowElement`) wird ebenfalls angepasst, was den Wert jedoch nicht mehr erhöht. Als nächste Korrespondenz wird (`InputPort`, `Input`) gebildet. Dies verursacht, wie in Tabelle 6.15 zu sehen ist, eine Erhöhung der Ähnlichkeitswerte für (`WorkflowNode`, `WorkflowNode`) wegen einer Assoziation und (`WorkflowElement`, `WorkflowElement`) wegen einer Vererbungsbeziehung. Das Verfahren identifiziert die Klassen (`WorkflowElement`, `WorkflowElement`) als nächstes Korrespondenzpaar. Die Ähnlichkeitswerte für deren noch nicht korrespondierende Unterklassen {`WorkflowNode`, `OutputPort`} × {`WorklowNode`, `Comment`, `Output`} werden in Tabelle 6.16 erhöht. Als nächstes wird die Korrespondenz (`WorkflowNode`, `WorkflowNode`) gebildet. Dies steigert den Ähnlichkeitswert von (`OutputPort`, `Output`) von 0,6166 auf 0,625 (vgl. Tabel-

Tabelle 6.12: Beispiel 20: Ähnlichkeitsverfahren mit Anpassung nach der Korrespondenzbildung von (`Edge`, `Edge`)

	Workflow-Element	Workflow-Node	Comment	Workflow	Edge	Task	Input	Output
WorkflowElement	0,6941	0,295	0,2291	0,2555		0,2752	0,196	0,2165
WorkflowNode	0,3273	0,6416	0,2989	0,392		0,2419	0,2817	0,3036
Transformation	0,496	0,425	0,3219	0,332		0,4152	0,3141	0,4176
Conditional-OutputPort	0,4936	0,4114	0,3024	0,3007		0,3657	0,2966	0,476
Workflow	0,2756	0,3554	0,7335	*0,884*		0,6639	*0,7402*	*0,5798*
Edge (1)					**0,9555**			
Task	0,3216	0,2419	0,7006	0,6609		0,9333	0,6855	0,5129
Choice	0,2892	0,2408	0,6839	0,6553		0,75	0,6588	0,4987
InputPort	0,2318	0,3095	0,7397	*0,7558*		0,6626	*0,8175*	*0,6217*
OutputPort	0,24	0,3168	0,6523	*0,6684*		0,5767	*0,6942*	*0,575*

Tabelle 6.13: Beispiel 20: Ähnlichkeitsverfahren mit Anpassung nach der Korrespondenzbildung von (`Task`, `Task`)

	Workflow-Element	Workflow-Node	Comment	Workflow	Edge	Task	Input	Output
WorkflowElement	0,6941	0,295	0,2291	0,2555			0,196	0,2165
WorkflowNode	0,3273	*0,6416*	0,2989	0,392			0,2817	0,3036
Transformation	0,496	0,425	0,3219	0,332			0,3141	0,4176
Conditional-OutputPort	0,4936	0,4114	0,3024	0,3007			0,2966	0,476
Workflow	0,2756	0,3554	0,7335	*0,8881*			0,7402	0,5798
Edge (1)					**0,9555**			
Task (2)						**0,9333**		
Choice	0,2892	0,2408	0,6839	0,6553			0,6588	0,4987
InputPort	0,2318	0,3095	0,7397	0,7558			0,8175	0,6217
OutputPort	0,24	0,3168	0,6523	0,6684			0,6942	0,575

Tabelle 6.14: Beispiel 20: Ähnlichkeitsverfahren mit Anpassung nach der Korrespondenzbildung von (`Workflow`, `Workflow`)

	Workflow-Element	Workflow-Node	Comment	Workflow	Edge	Task	Input	Output
WorkflowElement	*0,6941*	0,295	0,2291				0,196	0,2165
WorkflowNode	0,3273	*0,6527*	*0,3101*				0,2817	0,3036
Transformation	0,496	0,425	0,3219				0,3141	0,4176
Conditional-OutputPort	0,4936	0,4114	0,3024				0,2966	0,476
Workflow (3)				**0,8881**				
Edge (1)					**0,9555**			
Task (2)						**0,9333**		
Choice	0,2892	0,2408	0,6839				0,6588	0,4987
InputPort	0,2318	0,3095	0,7397				0,8175	0,6217
OutputPort	0,24	0,3168	0,6523				0,6942	0,575

Tabelle 6.15: Beispiel 20: Ähnlichkeitsverfahren mit Anpassung nach der Korrespondenzbildung von (`InputPort`, `Input`)

	Workflow-Element	Workflow-Node	Comment	Workflow	Edge	Task	Input	Output
Workflow-Element	*0,6978*	0,295	0,2291					0,2165
Workflow-Node	0,3273	*0,6583*	0,3101					0,3036
Transformation	0,496	0,425	0,3219					0,4176
Conditional-OutputPort	0,4936	0,4114	0,3024					0,476
Workflow (3)				**0,8881**				
Edge (1)					**0,9555**			
Task (2)						**0,9333**		
Choice	0,2892	0,2408	0,6839					0,4987
InputPort (4)							**0,8175**	
OutputPort	0,24	0,3168	0,6523					0,575

Tabelle 6.16: Beispiel 20: Ähnlichkeitsverfahren mit Anpassung nach der Korrespondenzbildung von (`WorkflowElement`, `WorkflowElement`)

	Workflow-Element	Workflow-Node	Comment	Workflow	Edge	Task	Input	Output
Workflow-Element(5)	**0,6978**							
Workflow-Node		*0,7*	*0,3517*					*0,3452*
Transformation		0,425	0,3219					0,4176
Conditional-OutputPort		0,4114	0,3024					0,476
Workflow (3)				**0,8881**				
Edge (1)					**0,9555**			
Task (2)						**0,9333**		
Choice		0,2408	0,6839					0,4987
InputPort (4)							**0,8175**	
OutputPort		*0,3585*	*0,694*					*0,6166*

le 6.17). Dies kann jedoch nicht verhindern, dass im nächsten Schritt die Korrespondenz (`Comment`, `Output`) mit dem Ähnlichkeitswert 0,694 ausgewählt wird. Eine Anpassung wird nicht durchgeführt und da die restlichen Ähnlichkeitswerte unterhalb des Schwellenwerts von 0,5 liegen, ist das Verfahren damit beendet. Das Ergebnis unterscheidet sich wie folgt von dem Ergebnis des Verfahrens ohne Anpassung: `OutputPort` bildet mit `Comment` eine Korrespondenz und `Output` und `Choice` werden stattdessen als weitere Einzelelemente identifiziert. Dies stellt eine Verschlechterung des Ergebnisses dar, da neben der Fehlzuordnung (`OutputPort`,`Comment`) die erwartete Korrespondenz (`OutputPort`, `Output`) fehlt. Der Unterschied zu der Lösung des ähnlichkeitsbasierten Verfahrens ohne Anpassung ist darauf zurückzuführen, dass durch die Anpassung der Ähnlichkeitswert von (`Comment`, `Output`) größer als der Ähnlichkeitswert von (`Comment`, `Choice`) geworden ist. Betrachtet man die Edierkosten der beiden Lösungen, liegen die Kosten der Variante mit Anpassung der Ähnlichkeitswerte in Höhe von 59,64 über den Kosten der Variante ohne Anpassung in Höhe von 38,4. In beiden Fällen wurde nicht das vom Benutzer erwartete Ergebnis berechnet, das lediglich aus den Korrespondenzen (`Edge`, `Edge`), (`Task`, `Task`), (`Workflow`, `Workflow`), (`InputPort`, `Input`), (`WorkflowElement`, `WorkflowElement`), (`WorkflowNode`, `WorkflowNode`) und (`OutputPort`, `Output`) besteht.

Für das Beispiel 20 ist die Variante mit Anpassung in dieser Form nicht sinnvoll. Die Anpassung der Ähnlichkeitswerte kann jedoch für andere Beispiele zu einer Verbesserung der Ergebnisse führen. In den Fällen, in denen durch die Anpassung der Ähnlichkeits-

Tabelle 6.17: Beispiel 20: Ähnlichkeitsverfahren mit Anpassung nach der Korrespondenzbildung von (`WorkflowNode`, `WorkflowNode`)

	Workflow-Element	Workflow-Node	Comment	Workflow	Edge	Task	Input	Output
Workflow-Element(5)	**0,6978**							
Workflow-Node (6)		**0,7**						
Transformation			0,3219					0,4176
Conditional-OutputPort			0,3024					0,476
Workflow (3)				**0,8881**				
Edge (1)					**0,9555**			
Task (2)						**0,9333**		
Choice			0,6839					0,4987
InputPort (4)							**0,8175**	
OutputPort			0,694					*0,625*

werte zusätzliche Referenzen gebildet werden können, kann als Alternative auch eine Absenkung des Schwellenwerts in Betracht gezogen werden. So ließe sich in Beispiel 14 das Ergebnis der Variante mit Anpassung auch dadurch erreichen, dass der Schwellenwert für die Zuordnung von Klassen von 0,5 auf 0,47 gesenkt wird.

6.5 Evaluation der beiden Verfahren in Hinblick auf die Qualität der Ergebnisse

Die Evaluation durch den Benutzer wurde auf Testfällen durchgeführt, die aus einem versionierten Softwareentwicklungsprojekt für ein modulares Softwarekonfigurationsmanagementsystem [BDW08] des Lehrstuhls stammen, das mit dem CASE-Werkzeug Fujaba [Zün01] entwickelt wurde. Die modellierten Fujaba-Klassendiagramme wurden mit einem Fujaba-Plugin [GBD07], das ebenfalls am Lehrstuhl entwickelt wurde, als Ecore-Klassendiagramme exportiert. In Anlehnung an die Evaluation in [KKK+07] wird die Menge der berechneten Korrespondenzen (K) mit der Menge der vorgegebenen Korrespondenzen (K^*) verglichen, wobei folgende zwei Arten von Fehlern unterschieden werden: Eine Korrespondenz, die in K, aber nicht in K^* existiert, wird als α-Fehler (engl. false positive) bezeichnet. In diesem Fall wurden zwei Klassen, die manuell nicht zugeordnet wurden, von dem Verfahren fälschlicherweise als korrespondierend identifiziert. Als β-Fehler (engl. false negative) werden diejenigen Korrespondenzen gezählt, die in

Tabelle 6.18: Anzahl der Klassen in den Klassendiagrammen aus M

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
65	65	65	67	69	72	72	72	80	73	78	71	70	71	71

K^*, aber nicht in K enthalten sind. Eine Klasse, die bei der manuellen Zuordnung einen Korrespondenzpartner besitzt, wurde in diesem Fall vom Verfahren als Einzelelement identifiziert.

Die qualitative Evaluation der eigenentwickelten Verfahren besteht aus zwei Teilen. Während im ersten Teil die α- und β-Fehler programmatisch ermittelt werden konnten, da die Testfälle keine Umbenennungen enthalten, wurden die Ergebnisse im zweiten Teil manuell evaluiert. Die Testfälle wurden in beiden Fällen jeweils mit dem edierkostenbasierten Verfahren, dem ähnlichkeitsbasierten Verfahren sowie mit EMF Compare verglichen.

Teil 1: Fallbeispiele ohne Umbenennung der Klassennamen Die 15 Klassendiagramme der Testmenge M sind verschiedene aufeinanderfolgende Versionen des MOD2-SCM Projekts, die 65 bis 80 Klassen in 24 bis 30 Paketen enthalten. Tabelle 6.18 zeigt die mengenmäßige Entwicklung der Klassen des Gesamtmodells im Verlauf der verschiedenen Versionen. Da nach Aussage der Entwickler des Projekts in den Klassendiagrammen der Testgruppe M keine Klassennamen geändert wurden, wird im Rahmen der Benutzerevaluation vereinfacht untersucht, ob Klassen mit unterschiedlichen Namen zugeordnet werden oder Klassen als Einzelelemente identifiziert werden, zu denen eine gleichnamige Klasse im anderen Klassendiagramm existiert. Auf diese Weise lassen sich die α- und β-Fehler programmatisch auswerten.

Das edierkostenbasierte Verfahren wurde zunächst mit der in Kapitel 4 angegebenen Standardkonfiguration ohne Schwellenwertregel eingesetzt. Tabelle 6.19 zeigt die Ergebnisse des Verfahrens, das für die Teilmenge (M_1,...,M_9) zwar fehlerfreie Zuordnungen berechnet, in den anderen Vergleichen jedoch teilweise mit bis zu 10 α-Fehler abschneidet, was im Verhältnis zu der Klassenzahl eine sehr hohe Fehlerquote darstellt. Es treten dabei deutlich mehr α- als β-Fehler auf. Das Verfahren identifiziert die maximale Anzahl an Korrespondenzen. Dies entspricht in vielen Fällen sehr gut den Erwartungen des Anwenders. Zum Beispiel, wenn das Klassendiagramm A aus dem Klassendiagramm B durch das Modifizieren von Elementen hervorgeht, wie im Fall der Umbenennung aller Bezeichner des Klassendiagramms (vgl. Beispiel 21). Auch funktioniert es sehr gut, wenn das eine Modell durch das Löschen von Elementen aus dem anderen Modell entsteht, wobei die ungelöschten Elemente dabei ebenfalls modifiziert werden dürfen. Analog gilt die Aussage auch für den symmetrischen Fall, wenn ein Klassendiagramm durch das Einfügen weiterer Elemente aus dem anderen Klassendiagramm entstanden ist. Problematisch wird es dann, wenn als Änderungsoperationen sowohl Einfügungen als auch Löschungen von Elementen des gleichen Typs (z. B. Klasse oder Attribute) vorgenommen wurden. Diese Einfüge- und Löschoperationen werden von dem Verfahren aus Sicht der Minimierung der Edierkosten zu einer Änderungsoperation zusammengefasst, was

Tabelle 6.19: Edierkostenbasiertes Verfahren: Anzahl der α/β-Fehler der Testfälle in M

	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
M_1	0	0	0	0	0	0	0	0	0	0	8/0	9/0	9/0	9/0
M_2	-	0	0	0	0	0	0	0	0	0	8/0	9/0	9/0	9/0
M_3	-	-	0	0	0	0	0	0	0	0	8/0	9/0	9/0	9/0
M_4	-	-	-	0	0	0	0	0	0	0	8/0	9/0	9/0	9/0
M_5	-	-	-	-	0	0	0	0	0	0	8/0	10/1	10/1	10/1
M_6	-	-	-	-	-	0	0	0	0	0	7/0	7/0	8/0	8/0
M_7	-	-	-	-	-	-	0	0	0	0	7/0	7/0	8/0	8/0
M_8	-	-	-	-	-	-	-	0	0	0	9/2	7/0	8/0	8/0
M_9	-	-	-	-	-	-	-	-	2/1	6/0	10/2	8/1	9/1	9/1
M_{10}	-	-	-	-	-	-	-	-	-	0	9/3	7/1	8/1	8/1
M_{11}	-	-	-	-	-	-	-	-	-	-	3/2	5/4	6/4	6/4
M_{12}	-	-	-	-	-	-	-	-	-	-	-	1/1	2/1	2/1
M_{13}	-	-	-	-	-	-	-	-	-	-	-	-	0	0
M_{14}	-	-	-	-	-	-	-	-	-	-	-	-	-	0

für den Benutzer teilweise unintuitive Zuordnungen zur Folge haben kann. Da die Kosten für das Löschen bzw. Einfügen von Elementen auch die Kosten für das Umbenennen des Klassennamens umfassen, ändert auch eine Erhöhung der Kosten für die Umbenennung nichts daran, dass es immer günstiger ist, zwei Klassen als korrespondierend zu identifizieren als beide Klassen als Einzelelemente zu identifizieren. Mit einer einfachen Anpassung des Edierkostenmodells durch eine andere Gewichtung der Edieroperationen lässt sich die Zuordnung der maximalen Anzahl an Klassen daher nicht verhindern.

Dieselbe Testmenge wird nun auch mit dem edierkostenbasierten Verfahren in der Schwellenwertvariante verglichen. Durch den Einsatz einer Schwellenwertregel, die bei unterschiedlichen Klassennamen zwei Klassen nur dann als Korrespondenzpartner in Betracht zieht, wenn die Transformationskosten maximal 25% der Lösch/-Einfügekosten der kleineren der beiden Klassen beträgt, lassen sich verbesserte Ergebnisse erzielen (siehe Tabelle 6.20). In nur 12 von 105 Fällen treten jeweils ein α- und ein β-Fehler auf, die restlichen Ergebnisse sind fehlerfrei. Mit der Schwellenwertregel werden die sieben α-Fehler des Vergleichs M_8 vs. M_{13} auf einen α- und einen β-Fehler reduziert. Diese beiden Fehler sind auf die Zuordnung der Klassen (`ComplexVersionedItemFactory`, `SingleVersionedItemFactory`) zurückzuführen, die die Klasse `SingleVersionedItemFactory` als Einzelelement übrig lässt. Stattdessen fehlt die erwartete Zuordnung (`SingleVersionedItemFactory`, `SingleVersionedItemFactory`). Der Unterschied zwischen den Klassen `ComplexVersionedItemFactory` und `SingleVersionedItemFactory` aufgrund des unterschiedlichen Namens ist geringer als der Unterschied zwischen den beiden gleichnamigen Klassen (vgl. Abbildungen 6.11 und 6.12), die sich um eine Assoziation `factory` unterscheiden. Bei diesen sehr ähnlichen Factory-Klassen zeigt sich die Schwierigkeit der Konfiguration des Verfahrens. Das optimale Ergebnis entspricht in dieser Fallstudie der Zuordnung aller Elemente, die einen identischen Namen

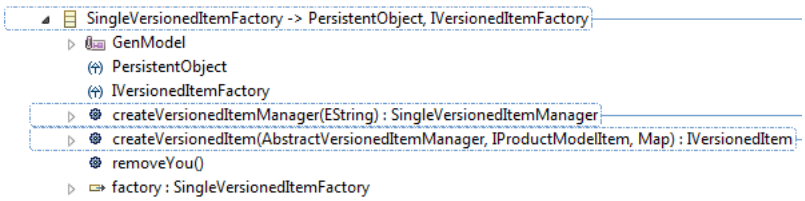

Abbildung 6.11: Die Klasse `SingleVersionedItemFactory` des Klassendiagramms M_{13}

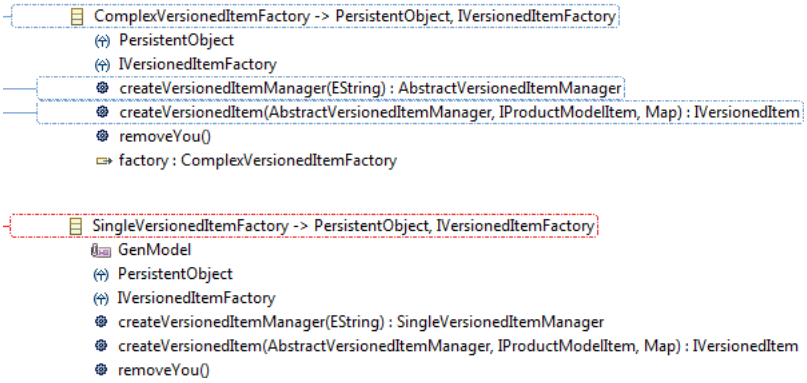

Abbildung 6.12: Die Klassen `ComplexVersionedItemFactory` und `SingleVersionedItemFactory` des Klassendiagramms M_8

besitzen. Es lässt sich die Schwellenwertregel zwar derart anpassen, dass nur gleichnamige Klassen korrespondieren sollen, dies entspricht jedoch einer Zuordnung, die sich auf eindeutige Namen anstelle von eindeutigen Objektbezeichnern verlässt. Eine solche Korrespondenzbildung ist im Allgemeinen jedoch nicht gewünscht und lässt sich darüber hinaus auch mit deutlich weniger aufwändigen Verfahren erreichen.

Mit der in vorherigen Kapiteln verwendeten Standardeinstellung liefert das ähnlichkeitsbasierte Verfahren α- und β-Fehler, wie sie in Tabelle 6.21 angegeben sind. Die α- und β-Fehler konzentrieren sich auch hier in etwa auf die gleichen Testfälle wie bei dem edierkostenbasierten Verfahren, liegen aber zahlenmäßig knapp unter den Werten des edierkostenbasierten Verfahrens. Nach einer Anpassung der Namensgewichtung auf $\frac{4}{9}$ und einer Schwelle von 0,73 liefert das ähnlichkeitsbasierte Verfahren fehlerfreie Ergebnisse.

Zum Vergleich wurden die Testfälle auch mit dem Verfahren von EMF Compare verglichen, die α- und β-Fehler bestimmt und in Tabelle 6.22 angegeben. α-Fehler treten bei den Ergebnissen des EMF Compare Verfahrens kaum auf. Dies überrascht nicht, da in den Klassendiagrammen keine Umbenennungen von Klassen auftreten und EMF Compare korrespondierende Elemente primär über die Namen zuordnet. Auffallend ist jedoch die hohe Anzahl von β-Fehlern, die darauf zurückzuführen sind, dass die Klassen gelöschter/eingefügter Pakete nicht auf mögliche Korrespondenzen untersucht werden. Insbesondere fallen die Vergleiche mit dem Klassendiagramm M_3 durch eine sehr hohe Zahl von β-Fehlern auf. Dies ist darauf zurückzuführen, dass in M_3 im Gegensatz zu den anderen Klassendiagrammen eine andere Paketstruktur realisiert ist. Die Klassen und

Tabelle 6.20: Edierkostenbasiertes Verfahren mit Schwelle: Anzahl der α/β-Fehler der Testfälle in M

	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
M_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M_2	-	0	0	0	0	0	0	0	0	0	0	0	0	0
M_3	-	-	0	0	0	0	0	0	0	0	0	0	0	0
M_4	-	-	-	0	0	0	0	0	0	0	0	0	0	0
M_5	-	-	-	-	0	0	0	0	0	0	0	0	0	0
M_6	-	-	-	-	-	0	0	0	0	0	0	0	0	0
M_7	-	-	-	-	-	-	0	0	0	0	0	0	0	0
M_8	-	-	-	-	-	-	-	0	0	0	0	1/1	1/1	1/1
M_9	-	-	-	-	-	-	-	-	0	0	0	1/1	1/1	1/1
M_{10}	-	-	-	-	-	-	-	-	-	0	0	1/1	1/1	1/1
M_{11}	-	-	-	-	-	-	-	-	-	-	0	1/1	1/1	1/1
M_{12}	-	-	-	-	-	-	-	-	-	-	-	0	0	0
M_{13}	-	-	-	-	-	-	-	-	-	-	-	-	0	0
M_{14}	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Tabelle 6.21: Ähnlichkeitsbasiertes Verfahren (ohne Anpassung): Anzahl der α/β-Fehler der Testfälle in M

	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
M_1	0	0	0	0	0	0	0	0	0	0	7/1	7/0	7/0	7/0
M_2	-	0	0	0	0	0	0	0	0	0	7/1	7/0	7/0	7/0
M_3	-	-	0	0	0	0	0	0	0	0	7/1	7/0	7/0	7/0
M_4	-	-	-	0	0	0	0	0	0	0	6/0	6/0	7/0	7/0
M_5	-	-	-	-	0	0	0	0	0	0	6/0	6/0	7/0	7/0
M_6	-	-	-	-	-	0	0	0	0	0	6/0	6/0	7/0	7/0
M_7	-	-	-	-	-	-	0	0	0	0	6/0	6/0	7/0	7/0
M_8	-	-	-	-	-	-	-	0	0	0	8/1	8/1	9/1	9/1
M_9	-	-	-	-	-	-	-	-	1/0	5/0	7/1	7/1	8/1	8/1
M_{10}	-	-	-	-	-	-	-	-	-	1/1	7/2	6/0	7/1	7/0
M_{11}	-	-	-	-	-	-	-	-	-	-	2/1	2/1	3/1	3/1
M_{12}	-	-	-	-	-	-	-	-	-	-	-	0	0	0
M_{13}	-	-	-	-	-	-	-	-	-	-	-	-	0	0
M_{14}	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Tabelle 6.22: EMF Compare: Anzahl der α/β-Fehler der Testfälle in M

	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
M_1	0	0/59	0	0	0	0	0	0	0	0	0/7	0	0	0/7
M_2	-	0/59	0	0	0	0	0	0	0	0	0/7	0	0	0/7
M_3	-	-	0/59	0/59	0/59	0/59	0/59	0/59	0/59	0/59	0/51	0/50	0/50	0/50
M_4	-	-	-	0	0	0	0	0	0	0	0/7	0/7	0/7	0/7
M_5	-	-	-	-	0	0	0	0	0	0	0/7	0/7	0/7	0/7
M_6	-	-	-	-	-	0	0	0	0	0	0/7	0/7	0/7	0/7
M_7	-	-	-	-	-	-	0	0	0	0	0/7	0/7	0/7	0/7
M_8	-	-	-	-	-	-	-	0	0	1/0	2/7	3/7	1/7	1/7
M_9	-	-	-	-	-	-	-	-	0	0	0/7	0/7	0/7	0/7
M_{10}	-	-	-	-	-	-	-	-	-	0	2/7	3/7	0/7	0/7
M_{11}	-	-	-	-	-	-	-	-	-	-	0/7	0/7	0/7	0/7
M_{12}	-	-	-	-	-	-	-	-	-	-	-	0	0	0
M_{13}	-	-	-	-	-	-	-	-	-	-	-	-	0	0
M_{14}	-	-	-	-	-	-	-	-	-	-	-	-	-	0

Pakete des Projekts liegen hier direkt im Grundpaket anstatt in `de.ubt.ai1`.[3]

Teil 2: Fallbeispiele mit Refactoring Zur qualitativen Beurteilung der Verfahren durch den Benutzer wurden drei Paare verschiedener späterer Versionen des Projekts durch den Entwickler ausgewählt, die markante Änderungen aufweisen. Die ersten beiden Klassendiagramme M_{ref_1} und M_{ref_2} enthalten je 123 bzw. 133 Klassen und unterscheiden sich im Wesentlichen darin, dass ein Kommunikationsinterface eingeführt wurde. Dabei wurden drei Klassen umbenannt und 10 Klassen hinzugefügt. Das Klassendiagramm M_{ref_4} (139 Klassen) entstand aus dem Klassendiagramm M_{ref_3} (134 Klassen) durch eine größere Refactoringmaßnahme, bei der nicht nur Klassen umbenannt, sondern auch in verschiedenen Paketen je eine Factory- und eine Manager-Klasse durch eine Module-Klasse ersetzt wurden. Die beiden letzten Versionen M_{ref_5} und M_{ref_6} enthalten je 160 Klassen und unterscheiden sich im Wesentlichen durch Änderungen in der Pakethierarchie.

Für die ausgewählten Modell-Paare wurden mit dem edierkostenbasierten und dem ähnlichkeitsbasierten Korrespondenzberechnungsverfahren, beide in der Standardkonfiguration, sowie mit EMF Compare die Unterschiede ermittelt. Die Differenzberichte wurden dem Entwickler des Systems vorgelegt, wobei hauptsächlich die Zuordnungen der Klassen untersucht wurden. Tabelle 6.23 gibt einen Überblick über die identifizierten α- und β-Fehler der Vergleichsergebnisse. Das edierkostenbasierte Verfahren berechnet in dem ersten und in dem dritten Beispiel eine korrekte Zuordnung der Klassen. Das Ergebnis des zweiten Beispiels enthält 6 falsch zugeordnete Korrespondenzen zwischen Factory- und Modul-Klassen, aus denen 6 fehlende erwartete Zuordnungen resultieren. Das ähn-

[3] Vermutlich wurde bei der Generierung der Ecore-Dateien nicht das Grundverzeichnis, sondern ein tieferliegendes Verzeichnis ausgewählt.

Tabelle 6.23: Vergleichsergebnisse der Fallbeispiele: α- und β-Fehler auf Klassenebene

	M_{ref_1} vs. M_{ref_2}	M_{ref_3} vs. M_{ref_4}	M_{ref_5} vs. M_{ref_6}
ECVerfahren	0	6/6	0
Ähnlichkeitsverfahren	0	0	0
EMF Compare	0/5	0/9	0/13
EMF Compare (bereinigt)	0/1	0/7	0/0

lichkeitsbasierte Verfahren liefert in allen drei Testfällen eine aus Sicht des Entwicklers korrekte Zuordnung der Klassen. Im ersten Fall unterscheidet sie sich von der Zuordnung des edierkostenbasierten Verfahrens insoweit, dass die Klasse `IWorkspaceManager` auf `IWorkspaceModule` zugeordnet wurde und nicht auf `AbstractWorkspaceManager`. Es sind nach Angabe des Entwicklers jedoch beide Varianten korrekt, da hier zwei Klassen zu einer zusammengeführt wurden. In den Differenzberichten des EMF Compare Verfahrens wurden sehr viele β-Fehler identifiziert, die nicht nur auf die Änderungen an der Pakethierarchie zurückzuführen sind. Die letzte Zeile der Tabelle 6.23 enthält die bereinigten Fehlerzahlen des EMF Compare Verfahrens, bei denen fehlende Klassen-Zuordnungen aufgrund von nichtzugeordneten Paketen herausgerechnet wurden. Die fehlenden Zuordnungen beziehen sich dabei alle auf Klassenpaare mit unterschiedlichen Namen.

Der befragte Entwickler bewertet die Konfigurationsmöglichkeit der eigenentwickelten Korrespondenzberechnungsverfahren positiv, da so das Vergleichsverfahren an Spezialfälle angepasst werden kann. Die Zuordnung der Übergabeparameter von Methoden anhand ihrer Reihenfolge überzeugt hingegen in den Fällen nicht, wenn ein zusätzlicher Parameter an erster oder an einer mittleren Position eingefügt wird, da für alle nachfolgenden Parameter damit Änderungsoperationen entstehen. Da das Problem, weshalb diese Umgehungslösung eingebaut wurde, in den neueren EMF Compare Versionen behoben wurde, wäre es sinnvoll, die Implementierung der Korrespondenzberechnungsverfahren dahingehend umzustellen. Da Factory-Klassen leichter zu unterscheiden sind, wenn der Name ihres Pakets miteinbezogen wird, könnte es für diese Testfälle sinnvoll sein, die strikte Trennung der Klassenebene von der Paketebene aufzuweichen und den Paketnamen bereits bei der Zuordnung der Klassen zu berücksichtigen. Dies würde jedoch vermutlich in anderen Fällen, wenn Klassen in verschiedene Pakete verschoben werden, wieder zu ungünstigen Effekten führen.

Der Benutzer bewertete auch die Zuordnung der Pakete im Testfall M_{ref_5} vs. M_{ref_6}. Da beide eigenentwickelten Verfahren für das Beispiel eine identische Zuordnung auf Klassenebene berechnen und das gleiche ähnlichkeitsbasierte Paketebenenverfahren verwenden, stimmt auch die Zuordnung der Pakete für beide Verfahren in diesem Fall überein. Aus dem Differenzbericht in Abbildung 6.13 ist abzulesen, dass das Paket `history` in `server` umbenannt und ein neues Oberpaket `history` eingeführt worden ist. Der Benutzer erwartet an dieser Stelle jedoch eine andere Darstellung der Differenz, nämlich dass ein neues Unterpaket `server` eingefügt wurde und alle Klassen aus `history` in dieses neue Unterpaket verschoben wurden, wie in Abbildung 6.14 zu sehen ist. Die Differenz

aus Abbildung 6.13 entspricht jedoch der tatsächlichen Edierfolge, wie das Unterpaket in Fujaba eingefügt worden ist, so dass möglichst wenig Änderungen durchzuführen waren. Die angezeigte Differenz ist für den Benutzer damit nachvollziehbar, auch wenn die andere Darstellung aus Abbildung 6.14 intuitiver ist. Dieser verbesserte Differenzbericht lässt sich mit einer anderen Gewichtung der Ähnlichkeitswerte im Paketebenenverfahren erreichen, indem der Name des Pakets im Vergleich zu den anderen Kriterien (Ähnlichkeit des Vaterpakets, Ähnlichkeit der Unterpakete und Ähnlichkeit der Unterklassen) dreifach gewichtet wird. Abbildung 6.15 zeigt den entsprechenden Ausschnitt aus dem Differenzbericht von EMF Compare. EMF Compare identifiziert das Paket `server` inklusive Unterklassen als neu eingefügt und die drei Klassen aus dem `history` Paket als gelöscht. Dies ist darauf zurückzuführen, dass die Klassen, die in nicht korrespondierenden Paketen enthalten sind, nicht weiter auf mögliche Korrespondenzen untersucht werden. Leider ist es in dieser Darstellungsform für den Benutzer nicht ersichtlich, ob die eingefügten und gelöschten drei Klassen identisch sind oder darin ebenfalls Änderungen enthalten sind.

Die von den eigenen Verfahren berechnete Paketezuordnung in der Standardvariante entspricht daher nicht ganz den Erwartungen des Benutzers, schneidet aber auch ohne Anpassung der Gewichte deutlich besser als das Ergebnis von EMF Compare ab, da auch die Klassen gelöschter oder eingefügter Pakete verglichen werden und dadurch nicht so viele Informationen verloren gehen.

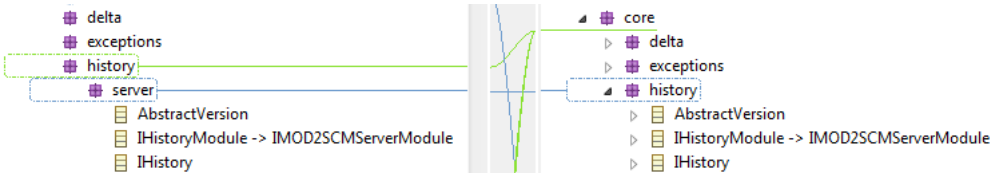

Abbildung 6.13: Ausschnitt aus dem Differenzbericht des eigenentwickelten Paketebenenverfahrens für den Vergleich von M_{ref_5} und M_{ref_6}

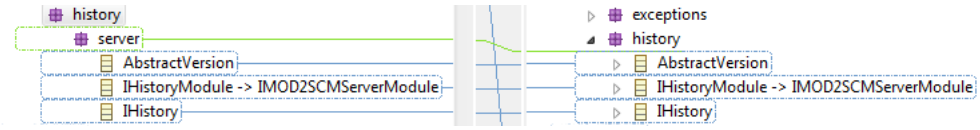

Abbildung 6.14: Ausschnitt aus dem Differenzbericht des Paketebenenverfahrens nach Anpassung der Gewichtungen für den Vergleich von M_{ref_5} und M_{ref_6}

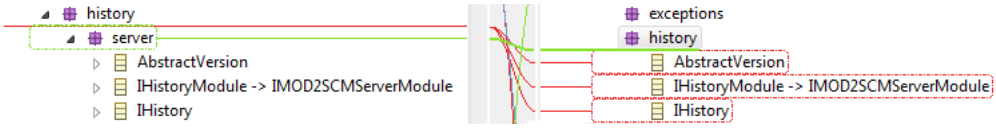

Abbildung 6.15: Ausschnitt aus dem Differenzbericht von EMF Compare für den Vergleich von M_{ref_5} und M_{ref_6}

Um den Vorteil der geringen Sensitivität der eigenentwickelten Verfahren gegenüber Namensänderungen zu demonstrieren, wird erneut das Fremdsprachenbeispiel aufgegriffen, bei dem ein Modell vom Englischen ins Französische übersetzt wurde.

Beispiel 21 (Geringe Sensitivität gegenüber Namensänderungen)**.** Die Modelle der Abbildungen 6.16 und 6.17 sind bis auf Umbenennungen der Bezeichner identisch. Wie bereits in Kapitel 3 gezeigt wurde, erkennen EMF Compare und das SiDiff-Plugin für Fujaba die korrespondierenden umbenannten Klassen nicht vollständig. Wie die Abbildung 3.8 auf Seite 74 zeigt, werden von EMF Compare nur diejenigen 4 von 8 Klassen richtig zugeordnet, deren Klassennamen und Namen der Unterelemente trotz Umbenennung hinreichend ähnlich sind (`EDate`, `EDate`),(`Client`, `client`), (`Agent`, `agent`), (`Bus`, `autobus`). Das SiDiff-Plugin identifiziert sogar nur 1 Klasse als korrespondierend, die Klasse `EDate`, deren Namen nicht geändert wurde (vgl. Abbildung 3.6, Seite 69). Beide eigenentwickelten Verfahren, sowohl das edierkostenbasierte als auch das ähnlichkeitsbasierte Verfahren, erkennen alle korrespondierenden Elemente trotz Umbenennungen korrekt. Die in Abbildung 6.18 angezeigten 33 Änderungen beziehen sich ausschließlich auf Namensänderungen.

6.6 Parametrisierung

Der im Interview mit dem Entwickler genannte Vorteil der Konfigurierbarkeit kann sich auch als Nachteil erweisen, wenn das Verfahren für jedes Paar an Klassendiagrammen neu zu konfigurieren ist. In der qualitativen Evaluation der Testmenge M konnten die Ergebnisse für beide Verfahren durch eine Nachjustierung der Konfiguration über einen Schwellenwert bzw. über die Gewichtung der Ähnlichkeitswerte deutlich verbessert werden. Dabei stellt sich die Frage, wie sich die geänderte Konfiguration auf andere Testfälle auswirkt. Ausführliche Tests in Hinblick auf die Sensitivität der Verfahren gegenüber Parameteränderungen wurden noch nicht durchgeführt. Stattdessen werden die Beispiele dieses Kapitels und des Kapitels 4 nochmal aufgegriffen und mit den Konfigurationen aus dem ersten Teil der qualitativen Evaluation verglichen. Das edierkostenbasierte Verfahren wird hier mit der verschärften Schwellenwertregel von 0,25 bei Namensungleichheit, das ähnlichkeitsbasierte Verfahren ohne Anpassung der Ähnlichkeitswerte mit einer Namensgewichtung von $\frac{4}{9}$ und einem Schwellenwert von 0,73 auf Klassenebene eingesetzt. Aus den Fehlerwerten des Vergleichsverfahrens EMF Compare wurden diejenigen Fehler herausgerechnet, die darauf zurückzuführen sind, dass die Klassen gelöschter bzw. eingefügter Pakete nicht auf mögliche Korrespondenzen untersucht wurden.

Wie in Tabelle 6.24 zu sehen ist, liefern alle Verfahrensvarianten in denjenigen Fällen die korrekte Lösung, wenn die Testfälle keine Umbenennungen enthalten und nicht in beiden Modellen Einzelelemente existieren, die unzugeordnet bleiben sollen. Die Tabelle 6.25 zeigt für die kleineren Beispiele jeweils die Menge der Korrespondenzen, die der Benutzer erwartet, die nachfolgenden Tabellen 6.26 bis 6.31 enthalten die jeweiligen Fehlzuordnungen und fehlenden Korrespondenzen für die einzelnen Verfahrensvarianten.

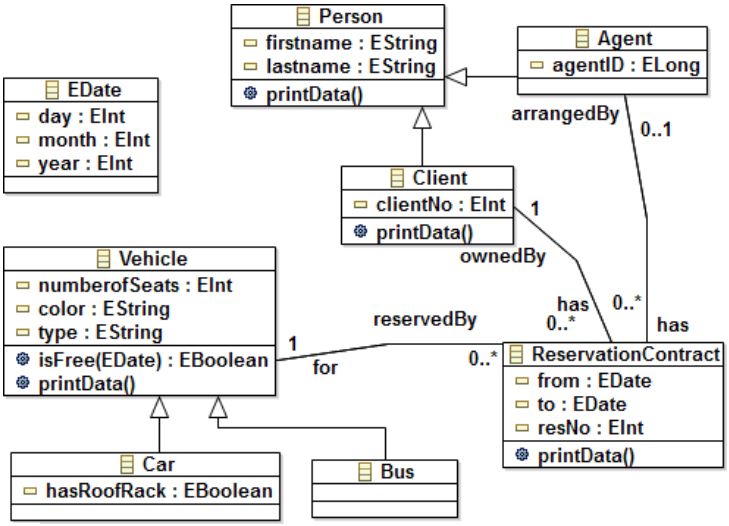

Abbildung 6.16: Modell A aus Beispiel 21

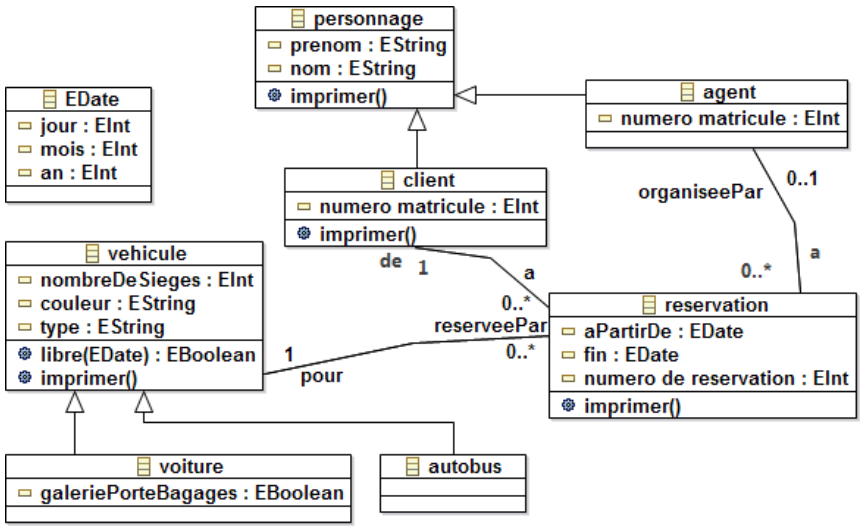

Abbildung 6.17: Modell B aus Beispiel 21

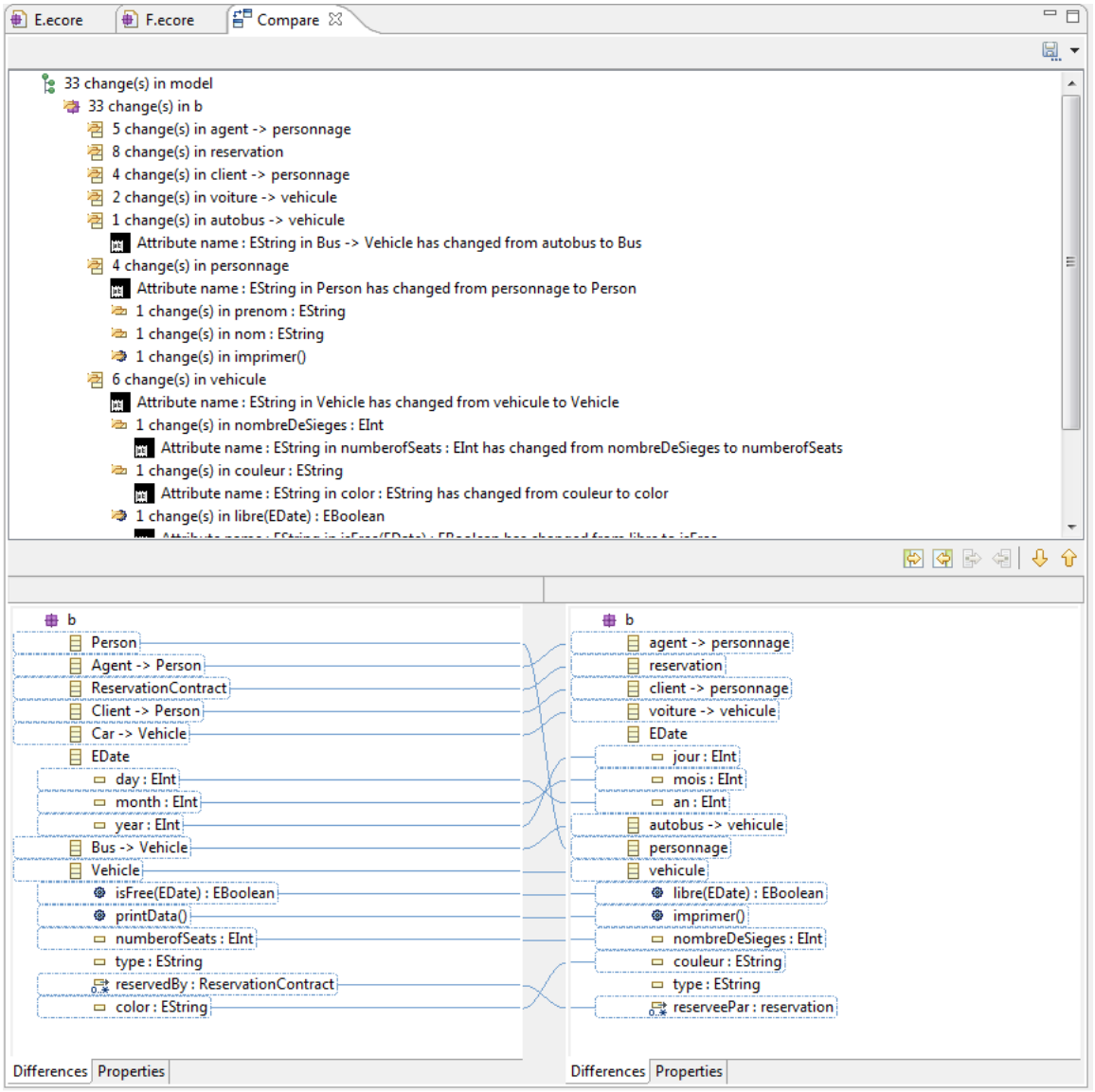

Abbildung 6.18: Differenzbericht des Vergleichs der Modelle A und B aus Beispiel 21 mit dem ECVerfahren (Levenshtein)

Tabelle 6.24: α- und β-Fehler der verschiedenen Verfahren im Vergleich

	Umbenennungen	Einzelelemente in beiden Modellen	ECVerfahren	BFVerfahren	ECVerfahren =BFVerfahren	ECVerfahren mit Schwellenwertregel	Ähnlichkeitsverfahren	Ähnlichkeitsverfahren modifiziert	EMF Compare
Beispiel 9	x	-	0	0	j	0	0	0	1/1
Beispiel 11	x	x	1/0	1/0	j	1/1	2/1	0/1	1/1
Beispiel 14	x	-	0	0	j	0/4	0/1	0/5	0/3
Beispiel 15	-	-	0	0	j	0	0	0	0
Beispiel 16	-	-	0	0	j	0	0	0	0
Beispiel 18	-	-	0	0	j	0	0	0	0
Beispiel 19	x	-	1/1	0	n	0	0	0/2	0/1
Beispiel 20	x	x	2/1	1/0	n	0/2	1/0	0/2	0/1
Beispiel 21	x	-	0	0	j	0/3	0	0/4	0/4
M_{ref_1} vs. M_{ref_2}	x	-	0	?	?	0/1	0	0	0/1
M_{ref_3} vs. M_{ref_4}	x	x	6/6	?	?	0/18	0	0/12	0/7
M_{ref_4} vs. M_{ref_5}	-	-	0	?	?	0	0	0	0
$\sum$			10/8			1/29	3/2	0/26	2/19

Das ECVerfahren ohne Schwellenwertregel liefert in allen Beispielen, die Einzelelemente auf beiden Seiten besitzen, α-Fehler, da stets die maximale Anzahl an Korrespondenzen gebildet wird. Der Einsatz der verschärften Schwellenwertregel, die die Korrespondenzbildung derjenigen Klassen erschwert, die verschiedene Namen besitzen, reduziert zwar in drei der vier Fälle die Zahl der α-Fehler, führt bei den Beispielen mit Umbenennungen jedoch in fünf von acht Fällen zu einer Verschlechterung und nur in zwei Fällen zu einer Verbesserung des Gesamtergebnisses. Auch die Modifizierung des Ähnlichkeitsverfahrens mit einer stärkeren Gewichtung der Namen verschlechtert die Fehleranzahl in fünf von acht Fällen der Beispiele, die Umbenennungen enthalten.

Das BFVerfahren liefert bis auf Beispiel 19 und 20 für die kleinen Beispiele die gleiche Lösung wie das ECVerfahren, was bedeutet, dass das ECVerfahren eine kostenoptimale Lösung bestimmt hat. Im Fall von Beispiel 19 berechnet das BFVerfahren im Gegensatz zum ECVerfahren die korrekte Lösung, im Fall von Beispiel 20 wird nur eine Fehlzuordnung, (`Choice`, `Comment`), berechnet. Damit schneidet das BFVerfahren für die kleinen Beispiele mit dem besten Ergebnis der aufgeführten Verfahren ab. Für die drei größeren Beispiele ist es jedoch aus Gründen des Aufwands nicht einsetzbar.

In allen betrachteten Fällen, in denen Klassen umbenannt wurden, enthält die Lösung des EMF Compare Verfahrens einen oder mehrere β-Fehler. Eine falsche Zuordnung in Form eines α-Fehlers wird vom EMF Compare Verfahren hingegen nur in zwei Fällen bestimmt. In Beispiel 9 wird die Umbenennung der Klasse `Address` nach `USAddress`

nicht erkannt und stattdessen die Klasse `USAddress` als korrespondierend zu `USAddress` identifiziert. Eine analoge Fehlzuordnung ergibt sich für das Beispiel 11 mit (`Contract`, `ReservationContract`). Betrachtet man die Gesamtfehlerzahlen der letzten Zeile in Tabelle 6.24, zeigt sich, dass das ähnlichkeitsbasierte Verfahren in der Standardvariante unter denjenigen Verfahren, die auch für größere Testfälle einsetzbar sind, für die ausgewählten Beispiele mit Abstand die besten Lösungen berechnet.

Die Ergebnisse bestätigen das, was zu vermuten war. Die verschärfte Schwellenwertregel des ECVerfahrens sowie die Konfiguration des ähnlichkeitsbasierten Verfahrens mit erhöhter Gewichtung des Namens in Verbindung mit einem höheren Schwellenwert führt für die betrachteten Beispiele, in denen Klassennamen geändert wurden, eher zu einer Verschlechterung als zu einer Verbesserung der Ergebnisse. Die Schwierigkeit besteht darin, eine Einstellung der Parameter zu finden, die für möglichst viele Vergleiche zu möglichst guten Lösungen führt. Darüber hinaus zeigen sich auch die Grenzen der automatischen Korrespondenzerkennung, ohne die Bedeutung der Namen mit einzubeziehen. Die ähnliche Bedeutung zweier Bezeichner, die der Anwender bei der Bestimmung korrespondierender Elemente intuitiv berücksichtigt, lässt sich nicht ausschließlich über Vergleichsmethoden erfassen, die den Edierabstand zwischen zwei Zeichenketten berechnen. Dies ist nur möglich, wenn Wörterbücher für Synonyme miteinbezogen werden.

6.7 Abschließende Zusammenfassung der Evaluation

6.7.1 Ergebnisse der Laufzeittests

Vertretbare Laufzeiten des ECVerfahrens und des ähnlichkeitsbasierten Verfahrens Für den interaktiven Einsatz kleiner bis mittlerer Beispiele sind sowohl das edierkostenbasierte Verfahren als auch das ähnlichkeitsbasierte Verfahren gut einsetzbar. Für größere Klassendiagramme oder Massentests empfiehlt es sich, bei dem ECVerfahren durch eine Schwellenwertregel bestimmte Kombinationen im Voraus auszuschließen, um die Laufzeit zu beschleunigen. Das ähnlichkeitsbasierte Verfahren hingegen ist auch für größere Modelle gut einsetzbar und skaliert deutlich besser als das edierkostenbasierte Verfahren.

Zeitlicher Vorteil der eigenentwickelten Verfahren gegenüber EMF Compare nur bei Klassendiagrammen mit flacher Paketstruktur Der vollständige Vergleich aller Klassen unabhängig von deren Lage in der Pakethierarchie ist etwas zeitaufwändiger als der Teilvergleich in EMF Compare, bei dem nur die Unterelemente jeweils korrespondierender Elemente berücksichtigt werden. Bei flachen Klassendiagrammen wandelt sich der Vorteil für das auf tiefe Pakethierarchien spezialisierte Verfahren sogar in einen Nachteil um. In diesem Fall zeigen die eigenentwickelten Verfahren bessere Laufzeiten.

Tabelle 6.25: Menge der als korrekt vorgegebenen Korrespondenzen für die kleinen Beispiele aus Tabelle 6.24

Beispiel 9	S. 107	(PurchaseOrder, PurchaseOrder), (Item, Item), (Address, USAddress)
Beispiel 11	S. 115	(Agent, Agent), (Person, Person), (Client, Client), (Vehicle, Vehicle), (Contract, ReservationContract)
Beispiel 14	S. 125	(TokenPool, TokenPool), (StartingSpace, Start), (GameBoard, GameBoard), (Ludo, Ludo), (GoalSpace, Goal), (Token, Token), (Player, Team), (Place, Space)
Beispiel 15	S. 137	(Agent, Agent), (Person, Person), (Client, Client)
Beispiel 16	S. 140	(Agent, Agent), (Person, Person), (Client, Client), (Vehicle, Vehicle), (Contract, Contract), (EDate, EDate)
Beispiel 18	S. 188	(Agent, Agent), (Person, Person), (Client, Client), (Vehicle, Vehicle), (ReservationContract, Reservation-Contract)
Beispiel 19	S. 190	(Agent, Agent), (Person, Person), (Client, Client), (EDate, EDate), (Contract, ReservationContract), (Vehicle, car)
Beispiel 20	S. 197	(Edge, Edge), (Task, Task), (Workflow, Workflow), (InputPort, Input), (WorkflowElement, WorkflowElement), (WorkflowNode, WorkflowNode), (OutputPort, Output)
Beispiel 21	S. 211	(EDate, EDate), (Client, client), (Agent, agent), (Bus, autobus), (Car, voiture), (Person, personnage), (ReservationContract, reservation), (Vehicle, vehicule)

Tabelle 6.26: Fehler des ECVerfahrens für die kleinen Beispiele aus Tabelle 6.24

Beispiel	Seite	α-Fehler	β-Fehler
11	115	(EDate, Contract)	
19	190	(bus, Client)	(Client, Client)
20	197	(ConditionalOutputPort, Output), (OutputPort, Comment)	(OutputPort, Output)

Tabelle 6.27: Fehler des BFVerfahrens für die kleinen Beispiele aus Tabelle 6.24

Beispiel	Seite	α-Fehler	β-Fehler
11	115	(EDate, Contract)	
20	197	(Choice, Comment)	

Tabelle 6.28: Fehler des ECVerfahrens mit Schwellenwert für die kleinen Beispiele aus Tabelle 6.24

Beispiel	Seite	α-Fehler	β-Fehler
11	115	(Contract, Contract)	(Contract, ReservationContract)
14	125		(StartingSpace, Start), (Place, Space), (Player, Team), (GoalSpace, Goal)
20	197		(OutputPort, Output),(InputPort, Input)
21	211		(Agent, agent), (ReservationContract, reservation), (Car, voiture)

Tabelle 6.29: Fehler des Ähnlichkeitsverfahrens für die kleinen Beispiele aus Tabelle 6.24

Beispiel	Seite	α-Fehler	β-Fehler
11	115	(Person, Bus), (EDate, Car)	(Person, Person)
14	125		(Place, Space)
20	197	(Choice, Comment)	

Tabelle 6.30: Fehler des Ähnlichkeitsverfahrens mit erhöhter Namensgewichtung für die kleinen Beispiele aus Tabelle 6.24

Beispiel	Seite	α-Fehler	β-Fehler
11	115		(Contract, ReservationContract)
14	125		(StartingSpace, Start), (GoalSpace, Goal), (Token, Token), (Player, Team), (Place, Space)
19	190		(Contract, ReservationContract), (Vehicle, car)
20	197		(InputPort, Input),(OutputPort, Output)
21	211		(Bus, autobus), (Car, voiture), (Person, personnage), (ReservationContract, reservation)

Tabelle 6.31: Fehler von EMF Compare für die kleinen Beispiele aus Tabelle 6.24

Beispiel	Seite	α-Fehler	β-Fehler
9	107	(USAddress, USAddress)	(Address, USAddress)
11	115	(Contract, Contract)	(Contract, ReservationContract)
14	125		(GoalSpace, Goal), (Player, Team), (Place, Space)
19	190		(Vehicle, car)
20	197		(OutputPort, Output)
21	211		(Car, voiture), (Person, personnage), (ReservationContract, reservation), (Vehicle, vehicule)

6.7.2 Erkenntnisse bei der Untersuchung der Ergebnisse des edierkostenbasierten Verfahrens

Geringe Aussagekraft des nicht erfüllten hinreichenden Optimalitätskriteriums Das hinreichende Optimalitätskriterium der Übereinstimmung der geschätzten Kosten mit den tatsächlichen Kosten stellt einen relativ schlechten Indikator für die Optimalität der Ergebnisse dar, da das edierkostenbasierte Verfahren in vielen Fällen eine optimale Lösung für das ursprüngliche Problem berechnet, auch wenn das hinreichende Optimalitätskriterium nicht erfüllt ist.

Geringe Kostenabweichung vom Optimum bei ausreichendem Anteil an kontextunabhängigen Informationen Die Gesamtabweichung von den optimalen Kosten ist durchschnittlich sehr gering, wobei die Ergebnisse des Verfahrens für Testfälle, in denen ausreichend viele kontextunabhängige Informationen vorliegen, besser sind. In Hinblick auf die Optimalität der Kosten ist das Verfahren mit Kostenabschätzung jedoch weniger geeignet, wenn Modelle überwiegend kontextabhängige Informationen enthalten.

Die Höhe der Kosten ist als alleiniges Kriterium zur Beurteilung der Qualität einer Zuordnung ungeeignet. Die Evaluation veranschaulicht an Beispielen, dass die Kostenoptimalität nicht das eine entscheidende Kriterium für die Qualität der Vergleichsergebnisse ist. Zum einen kann eine Zuordnung mit fast optimalen Kosten trotzdem deutlich von der Lösung mit optimalen Kosten abweichen. Zum anderen zeigt die Betrachtung der Vergleichsergebnisse, dass eine kostenminimale Zuordnung nicht immer den Erwartungen des Nutzers entspricht.

Begrenzte Anpassbarkeit des Kostenmodells Ein Kostenmodell, das die Kriterien einer Metrik erfüllen soll, ist nicht flexibel genug anpassbar, um zu verhindern, dass das edierkostenbasierte Verfahren die maximal mögliche Anzahl an Korrespondenzen identifiziert.

Bewertung der Schwellenwertregel Die Bildung der maximalen Zahl an Korrespondenzen kann in vielen Fällen jedoch über eine Schwellenwertregel verhindert werden. Da die Schwellenwertregel den Lösungsraum verkleinert, bestimmt das ECVerfahren mit Schwellenwertregel seltener eine optimale Lösung, das Ergebnis kann dennoch aus Sicht des Benutzers besser sein.

6.7.3 Erkenntnisse bei der Untersuchung der Ergebnisse des ähnlichkeitsbasierten Verfahrens

Geringe Unterschiede der Varianten mit und ohne Anpassung der Ähnlichkeiten im 1-Kontext Die Ergebnisse des ähnlichkeitsbasierten Verfahrens mit und ohne Anpassung stimmen in den meisten Fällen überein. Falls beide Varianten unterschiedliche

Ergebnisse liefern, zeigt das aufwändigere Verfahren mit Anpassung nicht eindeutig bessere Ergebnisse.

Vor- und Nachteil der Parametrisierung Das konservativere Identifizieren von Korrespondenzen, das vom Nutzer besser akzeptiert wird, lässt sich leicht mit dem ähnlichkeitsbasierten Verfahren realisieren, da die Gewichte der Ähnlichkeitskriterien sowie die Grenze des Ähnlichkeitswertes für die Korrespondenzbildung frei wählbar sind. Allerdings führen auf eine bestimmte Menge an Testfällen optimierte Parametereinstellungen bei anderen Gruppen von Testfällen unter Umständen zu sehr schlechten Ergebnissen. Letztlich stellt sich, wie bei allen konfigurierbaren Verfahren, die Frage nach der Feinjustierung der Parameter, so dass das Verfahren in möglichst vielen verschiedenen Situationen möglichst gute Ergebnisse berechnet.

6.7.4 Stärken beider eigenentwickelten Verfahren

Geringe Sensitivität gegenüber Namensänderungen Die strukturbasierte Arbeitsweise der eigenentwickelten Verfahren, die den Schwerpunkt nicht auf die Bezeichner der Modellelemente setzt, ist vor allem dann von Vorteil, wenn viele Elemente umbenannt wurden, so dass für andere Verfahren kaum Orientierungspunkte zum Identifizieren weiterer Elemente zur Verfügung stehen.

Vollständiger Klassenvergleich Der Vergleich aller Klassen, unabhängig von deren Lage in der Paketstruktur, liefert aus Sicht des Benutzers wertvolle Informationen über Änderungen an Klassen in gelöschten oder eingefügten Paketen.

6.7.5 Abschließendes Fazit

Die Vorteile des ähnlichkeitsbasierten Verfahrens gegenüber dem edierkostenbasierten Verfahren mit abgeschätzten Kosten überwiegen. Für die betrachteten Beispiele besitzt die Berechnung einer kostenminimalen Zuordnung in der abgeschätzten Variante keinen entscheidenden Vorteil, der den höheren Aufwand im Vergleich zu dem ähnlichkeitsbasierten Verfahren rechtfertigt. Auf Grundlage der durchgeführten Evaluation ist das ähnlichkeitsbasierte Verfahren dem edierkostenbasierten Verfahren vorzuziehen, da es nicht nur schnellere, sondern aus Sicht des Anwenders auch bessere Ergebnisse liefert.

Allgemeine Bewertung von edierkostenbasierten Differenzen Da das ECVerfahren nicht in allen Fällen die kostenoptimale Lösung erreicht, kann über die Betrachtung der Ergebnisse des ECVerfahrens keine allgemeine Aussage über die möglichen Ergebnisse eines exakten edierkostenbasierten Verfahrens, wie das BFVerfahren, getroffen werden. Zur sicheren Beurteilung, ob die optimalen edierkostenbasierten Zuordnungen oder die ähnlichkeitsbasierten Zuordnungen aus Sicht des Nutzers besser abschneiden, fehlt derzeit noch die Datenbasis, da von einer manuellen Prüfung der Ergebnisse der über 10000

Testfälle aus A, B und C für die verschiedenen Verfahren aufgrund des damit verbundenen Aufwands Abstand genommen wurde. Die Evaluation lässt jedoch folgende Schlüsse zu:

- Das metrik-konforme Kostenmodell eines edierkostenbasierten Verfahrens, das eine Bildung der maximal möglichen Korrespondenzen erzwingt, entspricht nicht den Erwartungen des Anwenders, der die Existenz von Einzelelementen in beiden Modellen der Korrespondenz zweier recht verschiedener Elemente vorzieht.
- Der Lösungsraum der Zuordnungen ist in dem Sinne schwierig, dass kostenmäßig fast optimale Lösungen dennoch keine akzeptablen Lösungen aus Sicht des Benutzers darstellen können. Dies deutet darauf hin, dass approximative Verfahren, die die Lösung annähern, nicht hilfreich sind.

7 Zusammenfassung

Für die weitere Verbreitung und Akzeptanz der modellgetriebenen Softwareentwicklung sind Werkzeuge, die ein komfortables Arbeiten mit Klassendiagrammen ermöglichen, eine Grundvoraussetzung. Insbesondere bei verteilter Entwicklung sind gute Vergleichsverfahren entscheidend, um Änderungen zwischen verschiedenen Versionen von Klassendiagrammen erkennen und visualisieren zu können. Diese Unterschiede bilden auch die Grundlage für das Verschmelzen zweier Klassendiagramme.

Die aus der Literatur bekannten Vergleichsverfahren für Klassendiagramme verwenden entweder eindeutige Objektbezeichner oder Ähnlichkeitsheuristiken für die Identifizierung der korrespondierenden Elemente. Die Verfahren der ersten Kategorie sind jedoch nur eingeschränkt einsetzbar und liefern Ergebnisse, die auf der Ediergeschichte der Klassendiagramme basieren. Die anderen Verfahren setzen auf Ähnlichkeitsheuristiken und verwenden sehr ähnliche Elemente als Orientierungspunkte für weitere mögliche Korrespondenzen. Die Verfahren verlassen sich dabei meines Erachtens zu sehr auf die Namen der Elemente und berücksichtigen teilweise auch nicht alle möglichen Korrespondenzen.

Inspiriert von Vergleichsverfahren auf Bäumen, wurde in dieser Arbeit ein edierkostenbasiertes Korrespondenzberechnungsverfahren für Klassendiagramme konzipiert und implementiert, das anstelle von Ähnlichkeitsheuristiken mit einem Edierabstand zwischen Klassendiagrammen arbeitet. Dazu werden die Klassendiagramme als Bäume betrachtet, wobei jedoch die Edierkosten der Querverbindungen über eine Abschätzung berücksichtigt werden. Das Verfahren berechnet nicht in allen Fällen die optimale Lösung des Problems, im Gegensatz zu den von vornherein heuristischen Verfahren existiert jedoch ein leicht überprüfbares, hinreichendes Optimalitätskriterium. Im Rahmen der durchgeführten Evaluation zeigte sich, dass die berechneten Edierkosten relativ nahe an den optimalen Kosten liegen.

Als Vergleichsmöglichkeit wurde ein weiteres ähnlichkeitsbasiertes Verfahren entwickelt. Die beiden Verfahren und deren Varianten wurden in ein Rahmenwerk zum Vergleich von Ecore-Klassendiagrammen integriert. Dieses basiert auf verschiedenen Plugins des Eclipse Modeling Frameworks und ermöglicht es, in der Entwicklungsumgebung Eclipse Ecore-Klassendiagramme zu modellieren und mit den verschiedenen Vergleichsverfahren zu vergleichen. Mit Hilfe dieses Rahmenwerks wurde auch die Evaluation durchgeführt. Diese zeigte als wesentliches Ergebnis, dass Differenzberichte mit niedrigeren Edierkosten im Vergleich zu anderen Differenzberichten mit höheren Edierkosten vom Benutzer nicht immer als besser, sondern teilweise auch als schlechter eingestuft werden. Dies ist ein Hinweis darauf, dass der Edierabstand nicht als das alleinige Kriterium für die Güte von Zuordnungen betrachtet werden kann. Insgesamt lieferte das einfachere, ähnlichkeitsbasierte Verfahren in Bezug auf die gewählten Testfälle nicht nur schnellere, sondern aus Sicht des Benutzers auch bessere Ergebnisse als das edierkosten-

basierte Verfahren mit den abgeschätzten Kosten.

Gemessen an den in Abschnitt 1.2.3 vorgestellten Kriterien für Vergleichsverfahren, arbeiten beide Verfahren auf Modellebene (A2) mit einem Schwerpunkt auf der Korrektheit (A1a) und Güte (A1b) der Ergebnisse. Als Umsetzung des Kriteriums (A1b) wurden für beide Verfahren ein vollständiger Vergleich der Klassen sowie eine geringe Sensitivität gegenüber Namensänderungen realisiert. Im edierkostenbasierten Ansatz wird eine Minimierung der Edierkosten als weiteres Kriterium verwendet, im ähnlichkeitsbasierten Ansatz eine Maximierung der Ähnlichkeitswerte. Auf die Umsetzung der Domainenunabhängigkeit (A3) wurde verzichtet, um klassendiagrammspezifische Besonderheiten auszunutzen. Eine mögliche Verallgemeinerung der Ansätze bzw. deren Übertragbarkeit auf andere Diagrammarten wird noch anschließend in Abschnitt 7.1 diskutiert. Die Unabhängigkeit von spezifischen Werkzeugen (A4) wurde weitestgehend umgesetzt, da beliebige in Ecore modellierte Klassendiagramme verglichen werden können, ohne dass spezielle Editoren verwendet werden müssen. Des Weiteren verzichten beide Verfahren auf die Verwendung von eindeutigen Objektbezeichnern und vergleichen die Klassendiagramme stattdessen unabhängig von der Ediergeschichte (A5). Das Kriterium der Effizienz (A6) wurde insoweit umgesetzt, dass beide Ansätze interaktiv in der Entwicklungsumgebung Eclipse eingesetzt werden können. Die Ergebnisse der Differenzberechnung werden dem Benutzer nicht nur textuell, sondern auch graphisch in der Baumsicht angezeigt (A7), eine zusätzliche Verbesserung der Darstellung wird sich ergeben, sobald das Rahmenwerk um eine Visualisierungsmöglichkeit der Unterschiede in einer Diagrammsicht ergänzt wird. Beide Ansätze sind flexibel in das Rahmenwerk integriert, so dass einzelne Komponenten leicht ausgetauscht werden können (A8).

7.1 Generischer Ansatz

Bei beiden eigenentwickelten Ansätzen handelt es sich um generische Verfahren, die grundsätzlich auch auf andere Diagrammarten übertragbar sind. Die Abbildung 7.1 vermittelt einen Überblick über die nötigen Schritte zur Durchführung des Modellvergleichs am Beispiel des edierkostenbasierten Ansatzes, wobei zwischen den drei verschiedenen Ebenen der Metamodell-Ebene, Modell-Ebene und Graph-Ebene (vgl auch [UW10]) unterschieden wird. Die Metamodell-Ebene der Werkzeugentwicklung beinhaltet die Schritte, die lediglich einmal zur Übertragung des Ansatzes auf eine neue Diagrammart durchgeführt werden müssen. Diese Schritte sind abhängig vom Metamodell und auch von dem Speicherformat der Modelle zu entwickeln. In dem realisierten Ansatz für Klassendiagramme werden die Klassendiagramme im Rahmen der Vorverarbeitung in das interne Datenmodell übertragen, das in Abschnitt 5.2 beschrieben wurde. Außerdem wird hier festgelegt, wie der Vergleichsgraph konstruiert werden soll. Auf den Vergleichsgraphen wird ein generischer Graphalgorithmus zur Bestimmung eines kostenoptimalen maximalen Netzwerkflusses angewendet, der die Lösung des Optimierungsproblems darstellt. Die Rückabbildung transformiert den Netzwerkfluss, der auf der Graph-Ebene berechnet wurde, in eine Zuordnung der Modellelemente auf Modell-Ebene. Damit ist bekannt, welche Modellelemente in A und B korrespondieren. Die Rückübertragung beinhaltet in

diesem Fall auch die Entkapselung der Daten und die Übertragung der Korrespondenzen in das Korrespondenzmodell von EMF Compare. Aus der Zuordnung wird schließlich die Modell-Differenz berechnet. Auf die Entwicklung eines Algorithmus zur Differenzberechnung wurde in dieser Arbeit verzichtet, da auf das Plugin von EMF Compare zur Ableitung der Differenzen zurückgegriffen werden kann.

Eine analoge Darstellung für den ähnlichkeitsbasierten Ansatz würde sich von der Darstellung in Abbildung 7.1 nur in wenigen Punkten unterscheiden. In den ersten Schritt der Werkzeugentwicklung fließt nicht ein Kostenmodell, sondern ein Ähnlichkeitsmodell ein. Statt einem Graphen wird in der bisherigen Umsetzung des ähnlichkeitsbasierten Verfahrens lediglich eine Matrix von Ähnlichkeitswerten erzeugt, die jedoch ebenfalls einen bipartiten Graphen repräsentiert (vgl. Abbildung 4.28, Seite 124). In der bisherigen Realisierung des ähnlichkeitsbasierten Verfahrens wird die Zuordnung aus der Ähnlichkeitsmatrix nicht über ein optimales, sondern über ein heuristisches Auswahlverfahren bestimmt.

Während der generische Graphalgorithmus zur Lösung des Optimierungsproblems nicht von dem Metamodell abhängt, auf dem die zu vergleichenden Modelle basieren, sind die auf der Modellebene angeordneten Schritte des Verfahrens dagegen vom Metamodell abhängig. Die dafür benötigten Algorithmen sind für jedes Metamodell spezifisch zu entwickeln, wobei der kritische Schritt die Konstruktion des Vergleichsgraphen ist. In beiden vorgestellten Ansätzen zum Vergleich von Klassendiagrammen wurde ein hierarchischer Vergleichsgraph bzw. eine hierarchische Ähnlichkeitsmatrix konstruiert, wobei die in Abschnitt 4.1 genannten Annahmen, wie z. B. die vollständige Teilbaumzuordnung, getroffen wurden. Dadurch ließen sich die kontextabhängigen Kosten bzw. Ähnlichkeiten weitgehend entkoppeln und das Problem auf mehrere Ebenen aufteilen.

Die Übertragung eines der Korrespondenzberechnungsverfahren auf eine andere Diagrammart ist daher mit Entwicklungsarbeit in erheblichem Umfang verbunden. Da beispielsweise für die 13 verschiedenen Diagrammarten der Unified Modeling Language (UML) [Obj07] ein sehr hoher Gesamtaufwand entsteht, wenn das Verfahren für jede Diagrammart einzeln angepasst wird, ist auch eine Übertragung des Ansatzes auf eine allgemeinere Ebene in Betracht zu ziehen. So wäre eine metamodellunabhängige Umsetzung des Verfahrens von Vorteil, da so mit dem gleichen Verfahren alle Modellinstanzen miteinander verglichen werden können, deren Metamodelle mit Hilfe des gleichen Meta-Metamodells, z. B. Ecore oder MOF [Obj06], beschrieben wurden. Es ist zu erwarten, dass die domainenspezifische, angepasste Variante eines Vergleichsverfahrens etwas bessere Ergebnisse als die allgemeine, metamodellunabhängige Variante berechnet. Dennoch wäre es interessant, die Ergebnisse beider Varianten für die gleiche Diagrammart miteinander zu vergleichen, um zu untersuchen, inwieweit sich der Mehraufwand für die Anpassung des Verfahrens auf diese Diagrammvariante lohnt. Im Folgenden wird die Übertragbarkeit für jedes der beiden Verfahren einzeln diskutiert.

Übertragbarkeit des abstandsbasierten Ansatzes Bei der Evaluation des abstandsbasierten Ansatzes in Abschnitt 6.3 zeigte sich, dass bei Klassendiagrammen, in denen zu wenig kontextunabhängige Informationen vorliegen, schlechtere Ergebnisse in Be-

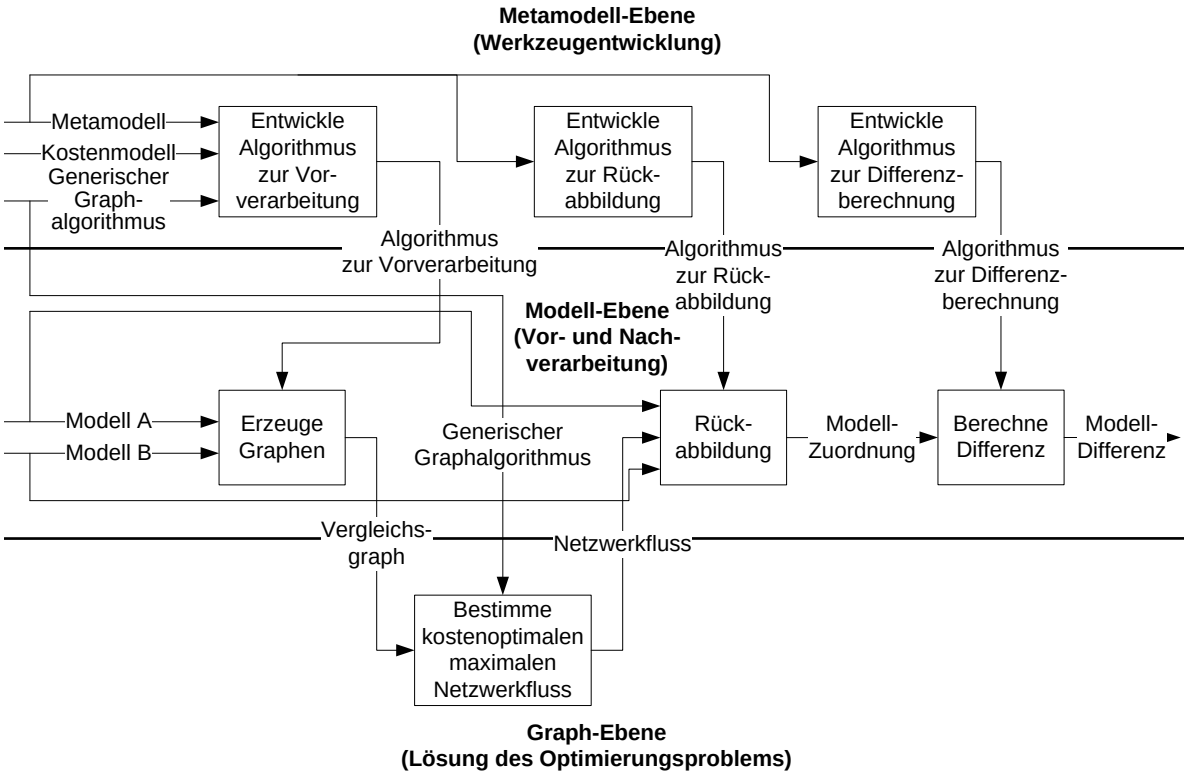

Abbildung 7.1: Graphbasierter Ansatz zum Vergleich von Modellversionen

zug auf die Optimalität der Ergebnisse erzielt wurden. Da die allgemeinste Form eines Modells ein Graph ohne Knoten- und Kantenbeschriftungen ist und bei diesem überwiegend kontextabhängige Informationen vorliegen, kann meines Erachtens nicht für beliebige Modelle sichergestellt werden, dass die Resultate des edierkostenbasierten Verfahrens in Hinblick auf die Optimalität überzeugen. Für Diagrammarten, die ähnlich wie Klassendiagramme grundsätzlich hierarchisch aufgebaut sind, aber zusätzliche Querverbindungen besitzen, kann eine ähnliche Transformation auf den Vergleichsgraphen verwendet werden. Für Diagrammarten, bei denen hingegen die wesentlichen Informationen in der Reihenfolge der Elemente liegen, wie zum Beispiel bei Sequenzdiagrammen, ist es meines Erachtens erforderlich, ein anderes Konzept für die Abbildung der Elemente auf den Vergleichsgraphen zu entwickeln. Zusammenfassend sind bei der Verallgemeinerung des abstandsbasierten Verfahrens auf Instanzen beliebiger Metamodelle vor allem in Hinblick auf die Optimalität keine guten Ergebnisse zu erwarten, während ich die Chancen für die Anpassung des abstandsbasierten Ansatzes auf andere hierarchische Diagrammarten als gut beurteile.

Übertragbarkeit des ähnlichkeitsbasierten Ansatzes Das ähnlichkeitsbasierte Verfahren ist leichtgewichtiger als das edierkostenbasierte Verfahren und daher auch leichter auf andere Diagramme übertragbar. So wurde das ähnlichkeitsbasierte Verfahren im Rahmen der zweistufigen Korrespondenzberechnung mit Trennung der Klassen- und der

Paketebene, wie in Abschnitt 4.5 beschrieben, bereits für den Vergleich von Paketen angepasst. Da die Ähnlichkeiten der Elemente im Gegensatz zu Edierkosten beliebig berechnet werden können, da sie nicht auf Edieroperationen basieren und auch keine Abstandsmetrik berücksichtigen müssen, ist der Ansatz flexibler. So ist beispielsweise vorstellbar, das ähnlichkeitsbasierte Verfahren generisch zu entwickeln, indem die Berechnung der Ähnlichkeitswerte in Abhängigkeit der Elementeigenschaften durchgeführt werden, die durch das Metamodell vorgegeben werden. Dies setzt voraus, dass das Metamodell verfügbar ist und als Eingabe für die Generierung eines Korrespondenzberechnungsverfahrens verwendet werden kann. Bei der Umsetzung sind zwei Varianten denkbar, die sich darin unterscheiden, wie stark das Metamodell die Berechnung der Ähnlichkeitswerte beeinflusst. Eine Möglichkeit besteht darin, alle Eigenschaften und Referenzen, die durch das Metamodell vorgegeben sind, in die Berechnung der Ähnlichkeitswerte einzubeziehen. Alternativ könnte sich das Verfahren die tatsächlich verwendeten Modellelementeigenschaften und -referenzen, die nicht auf Standardwerte gesetzt sind, aus den beiden zu vergleichenden Modellen auslesen und nur diese Eigenschaften und Referenzen berücksichtigen. Diese Filterung kann dann von Vorteil sein, wenn in den Modellen viele Eigenschaften und Referenzen auf Standardwerte gesetzt sind, da die Berechnung der Ähnlichkeiten auf diese Weise sowohl schlanker ist als auch stärker die Unterschiede der Modellelemente betont. Mit beiden vorgeschlagenen Varianten ließen sich dann Instanzen beliebiger Ecore-Modelle vergleichen.

7.2 Ausblick

Laufende Arbeiten betreffen einerseits die Laufzeitverbesserung des edierkostenbasierten Verfahrens und andererseits das Auswahlverfahren des ähnlichkeitsbasierten Verfahrens. Wie in Abschnitt 4.2.6 angegeben, kann der Aufwand des Busacker-Gowen Verfahrens auf Klassenebene auf $O(n^3)$ reduziert werden, indem ein anderes Kürzeste-Wege-Verfahren eingesetzt wird. Wird in dem Netzwerkflussproblem auf die Kanten vom ϵ-Knoten zu den Knoten der rechten Seite des bipartiten Graphen verzichtet, können noch weniger komplexe Verfahren zur Lösung des Netzwerkproblems eingesetzt werden. Dies besitzt allerdings den Nachteil, dass die Schwellenwertregel in der beschriebenen Form nicht mehr umgesetzt werden kann. Das ähnlichkeitsbasierte Verfahren, das bislang nur ein heuristisches Auswahlverfahren nutzt, wird um ein weiteres Auswahlverfahren erweitert, das aus der Ähnlichkeitsmatrix eine optimale maximale Zuordnung bestimmt. Nach Integration des neuen Auswahlverfahrens in das Rahmenwerk soll getestet werden, inwieweit sich die Lösungen des heuristischen und des optimalen Auswahlverfahrens unterscheiden. Da sich das ähnlichkeitsbasierte Verfahren bei der durchgeführten Evaluation in Hinblick auf die Laufzeit und die Beschaffenheit der Ergebnisse gegen das abstandsbasierte Verfahren durchgesetzt hat, läuft derzeit eine Arbeit zur Übertragung des ähnlichkeitsbasierten Ansatzes auf Instanzen beliebiger Ecore-Modelle.

Weitere zukünftige Arbeiten wären im Bereich der Evaluation denkbar. Für die über 10.000 Differenzberichte der Testmengen A, B und C pro Verfahrensvariante wurde noch keine Bewertung der Ergebnisse durch den Benutzer durchgeführt. Um eine größere

Datenbasis für die Beurteilung der Qualität der Vergleichsergebnisse zu erhalten, könnten die α- und β-Fehler der Vergleichsergebnisse für die verschiedenen Verfahren bestimmt werden. Insbesondere die Untersuchung der Ergebnisse des BFVerfahrens wäre für die Frage interessant, ob sich die Erwartungen von Nutzern formalisieren lassen. Jedoch wäre diese Erkenntnis leider nur von theoretischem Interesse, da das BFVerfahren aufgrund der Komplexität nicht einsetzbar ist. Auch zeigten die Ergebnisse des ECVerfahrens, dass der Lösungsraum in dem Sinne schwierig ist, dass kostenmäßig fast-optimale Lösungen dennoch keine akzeptablen Lösungen aus Sicht des Benutzers darstellen. Dies deutet darauf hin, dass approximative Verfahren, die die Lösung annähern, nicht hilfreich sind.

Diese Arbeit zeigt am Rande auch die Grenzen der Korrespondenzberechnungsverfahren auf, die den menschlichen Entscheidungsprozess, welche Elemente korrespondieren, nur bedingt nachbilden können. Vielleicht ist es sinnvoll, den Vergleich von Zeichenketten durch die Verwendung von Wörterbüchern zu verbessern und den Nutzer stärker in den Korrespondenzberechnungsprozess einzubinden. So ist ein Szenario denkbar, in dem der Benutzer nach dem berechneten Vorschlag der Zuordnung die Möglichkeit hat, gebildete Korrespondenzen zu verhindern oder andere Korrespondenzen zu erzwingen. Auf diese Weise wird ein Teil der Zuordnung bereits vorgegeben und die übrigen noch nicht festgelegten Korrespondenzen könnten erneut berechnet werden. Auch für diesen Anwendungsbereich sind die vorgestellten eigenentwickelten Verfahren mit nur wenig Anpassung einsetzbar.

A Anhang

A.1 Benutzung des Rahmenwerks in Screenshots

Dieser Abschnitt enthält eine Reihe von Bildschirmausschnitten, die zeigen, wie das Rahmenwerk zum Vergleich von Ecore-Klassendiagrammen eingesetzt werden kann. Die Ecore-Klassendiagramme werden als *.ecore-Dateien in einem Projekt verwaltet. Wie ein Eclipse-Projekt angelegt wird, zeigt die Abbildung A.1. Wie in dem Projekt eine neue Ecore-Datei erstellt wird, wird in der Abbildung A.2 dargestellt. Die Screenshots A.3 bis A.6 geben einen kurzen Eindruck, wie im Ecore-Baumeditor ein Klassendiagramm modelliert werden kann. Alternativ kann dazu auch der Ecore Diagrammeditor verwendet werden. Wie aus einer Ecore-Datei eine Ecore-Diagramm-Datei erzeugt wird und wie die graphische Modellierung funktioniert, wird aus den Abbildungen A.7 bis A.10 ersichtlich. Die Screenshots A.11 bis A.14 befassen sich schließlich mit dem Vergleich von Ecore-Klassendiagrammen. Sie zeigen, wie die DiffView geöffnet, der Vergleich ausgelöst und das Ergebnis repräsentiert wird.

Tabelle A.1: Übersicht über die Screenshots

Thema	**Abb.-Nrn**	**Seiten**
Anlegen eines Projekts	A.1	S. 228
Erstellen eines Ecore-Klassendiagramms	A.2	S. 229
Modellieren im Ecore-Baumeditor	A.3 - A.6	S. 230 - S. 233
Modellieren im Ecore-Diagrammeditor	A.7 - A.10	S. 234 - S. 237
Vergleich von Ecore-Klassendiagrammen	A.11 - A.14	S. 238 - S. 241

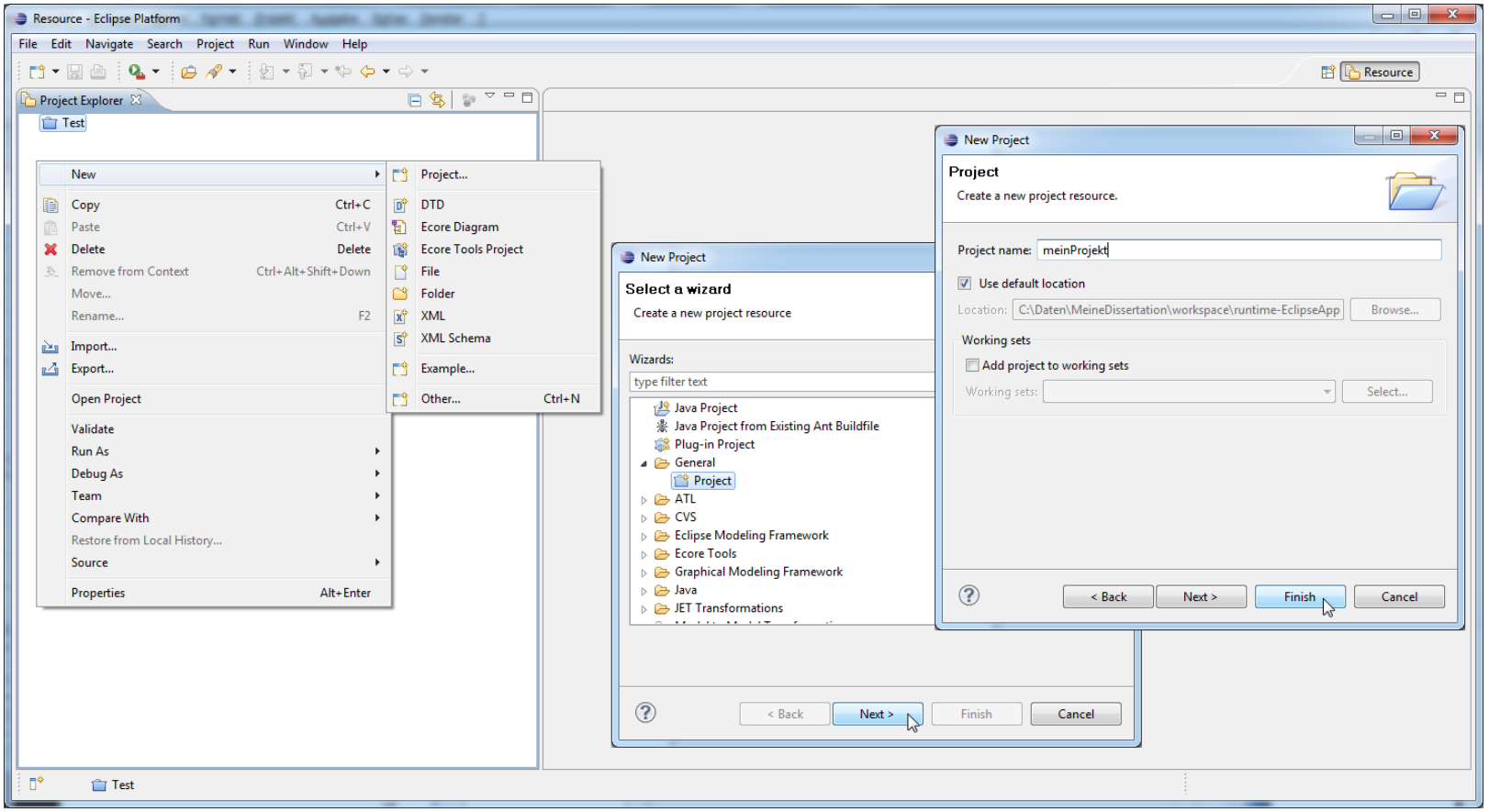

Abbildung A.1: Anlegen eines Projekts: Im Kontextmenü des Project Explorers den Project Wizard über *New* → *Project* öffnen. Im Project Wizard auf der ersten Seite als Art des Projekts *General Project* auswählen und auf der zweiten Seite den Namen des Projekts festlegen.

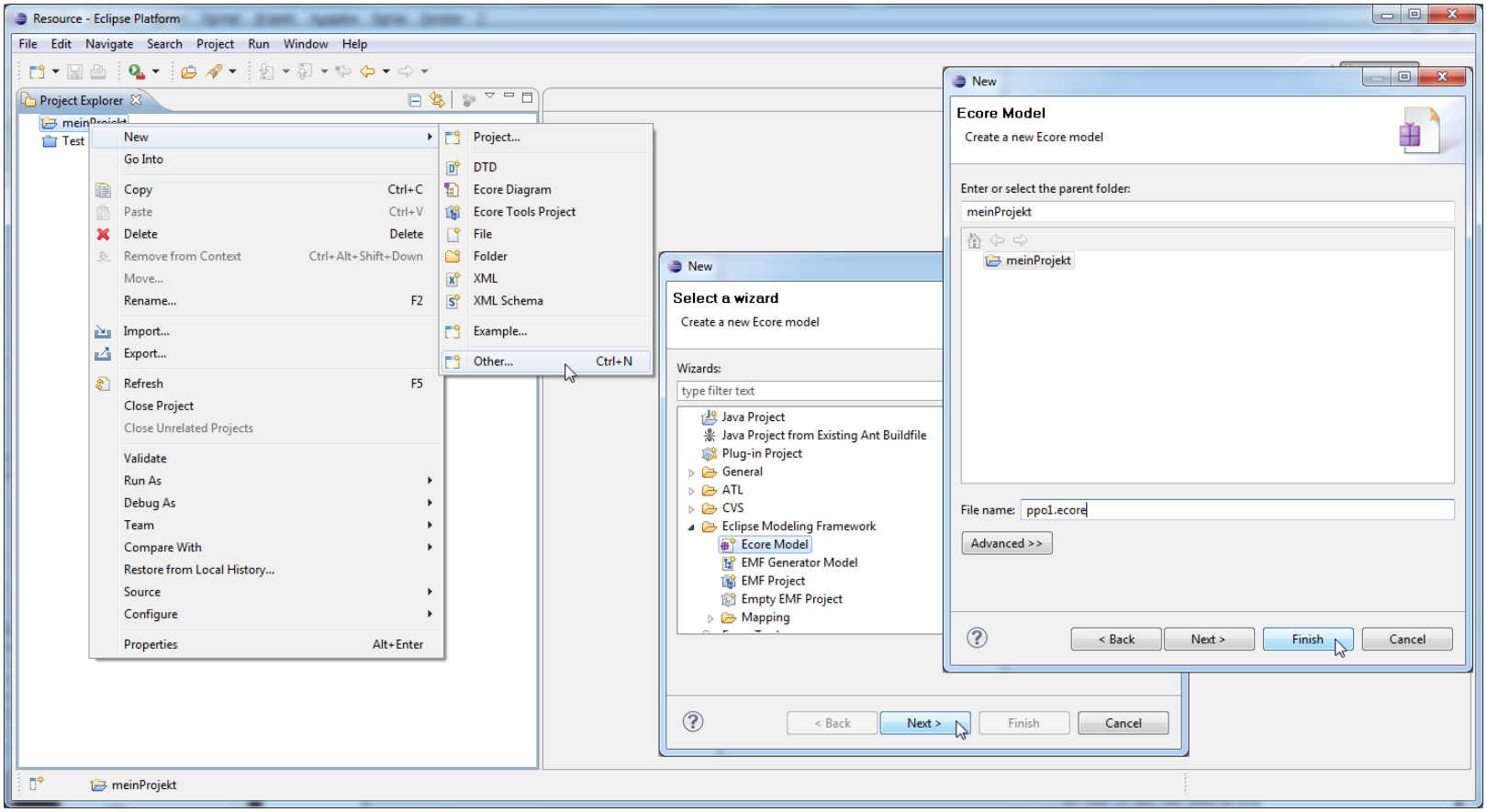

Abbildung A.2: Erstellen eines Ecore-Klassendiagramms: Im Kontextmenü eines Projekts unter *New* → *Other* den Wizard *Eclipse Modeling Framework* → *Ecore Model* auswählen und im Wizard den Ordner und den Namen für das Ecore-Modell festlegen.

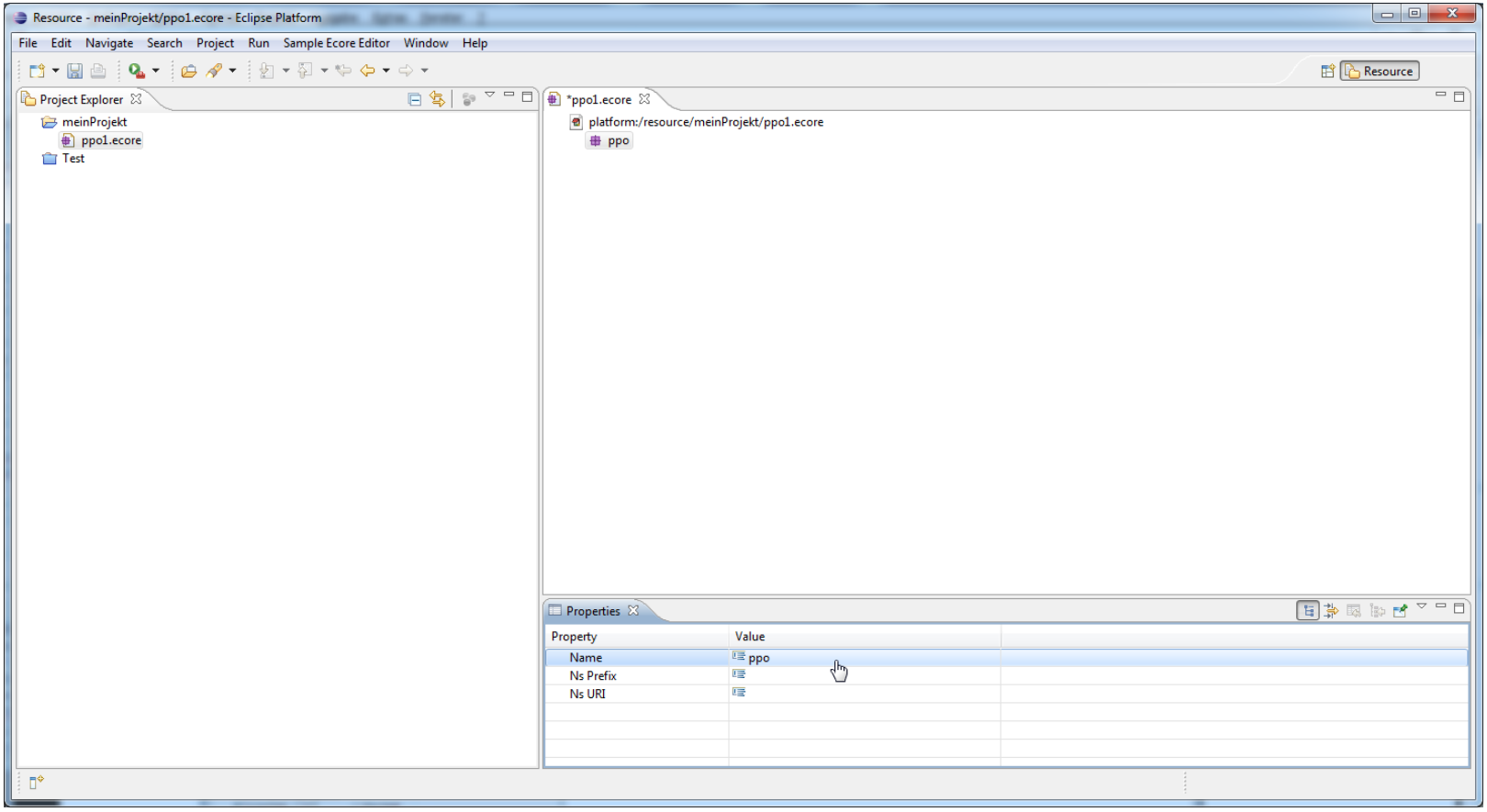

Abbildung A.3: Modellieren im Ecore-Baumeditor: Benennen des Grundpakets im Properties-Fenster

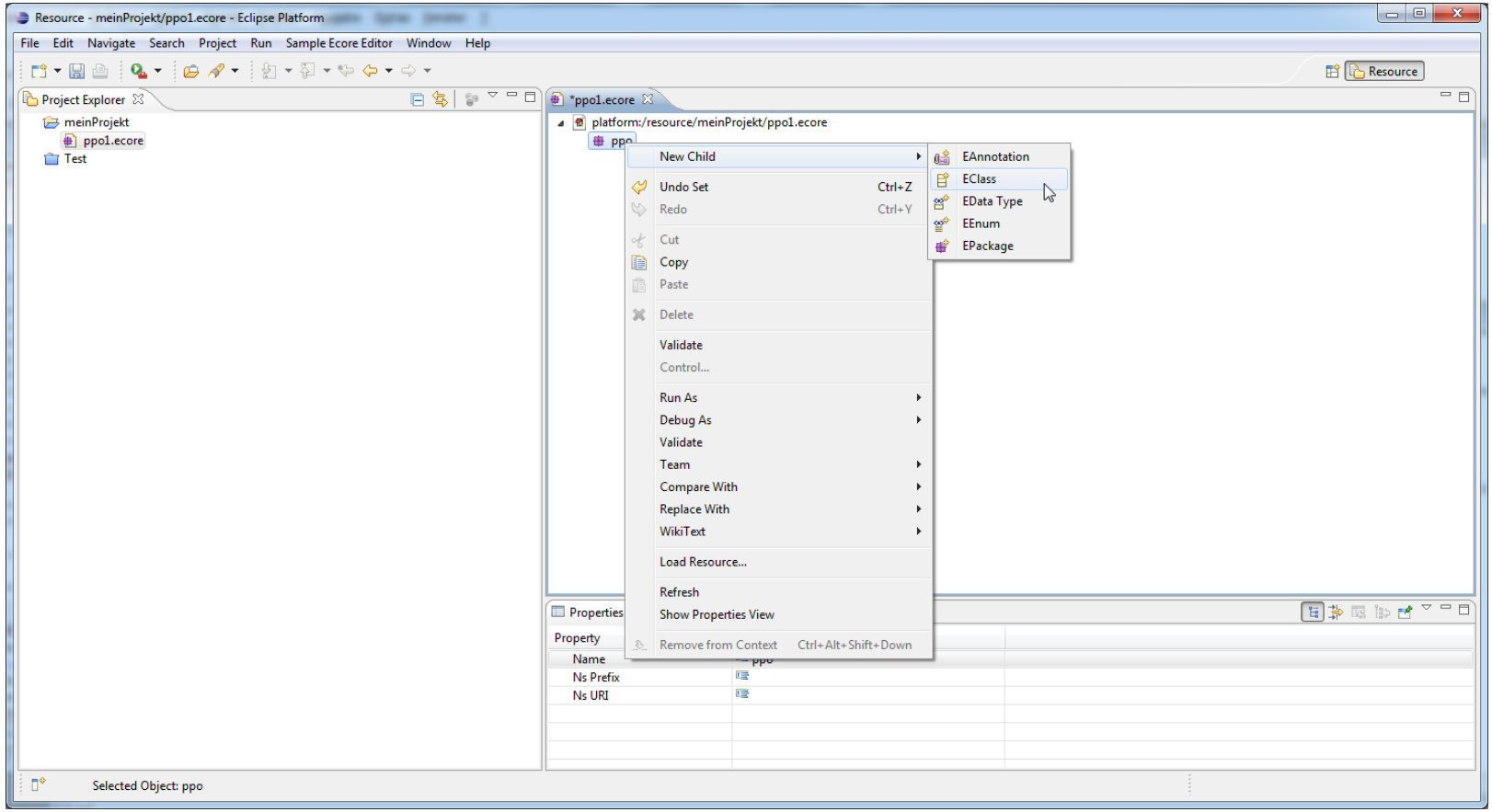

Abbildung A.4: Modellieren im Ecore-Baumeditor: Anlegen einer Klasse über das Kontextmenü *New Child* → *EClass* des Zielpakets

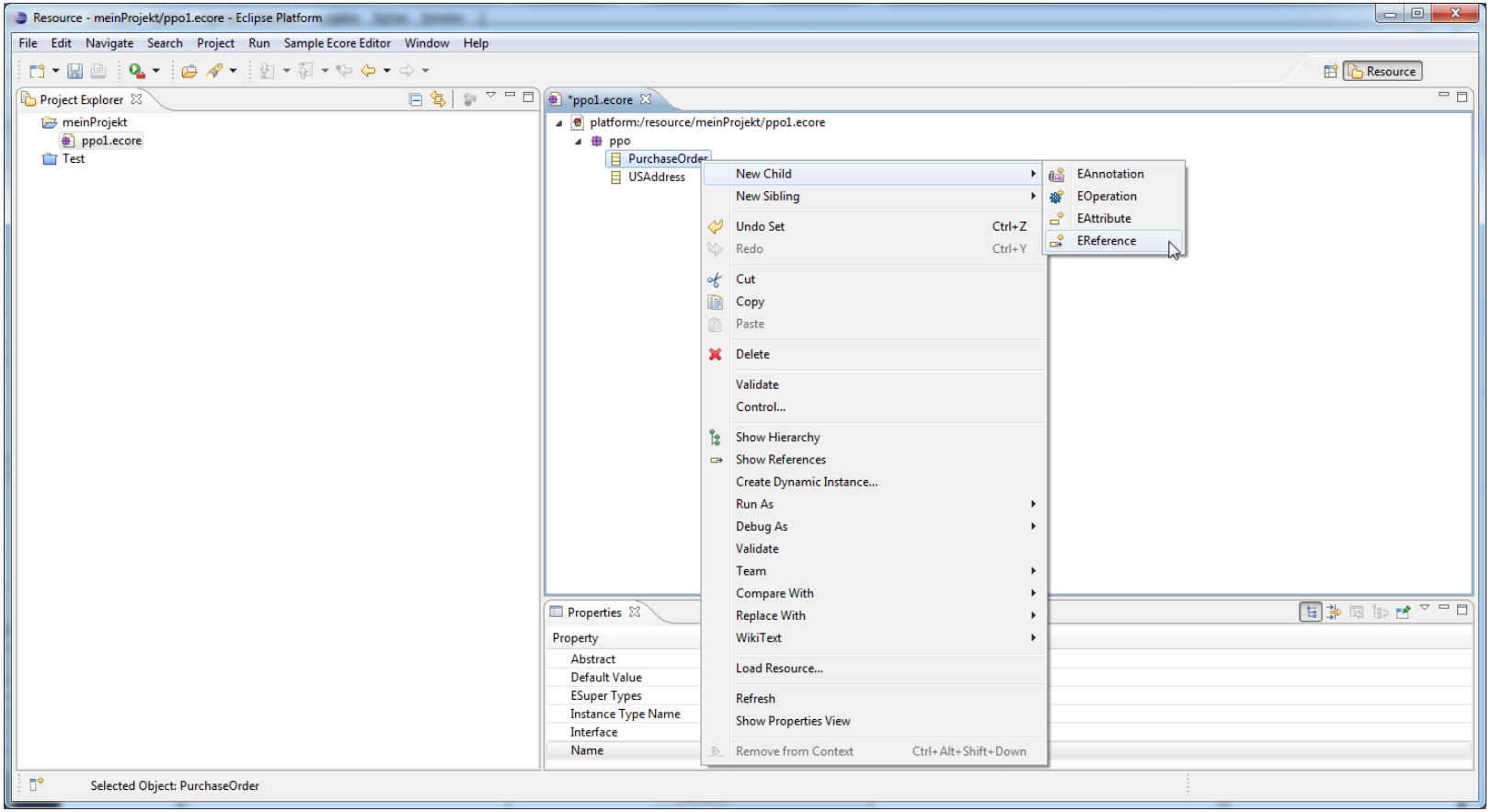

Abbildung A.5: Modellieren im Ecore-Baumeditor: Anlegen einer Assoziation über das Kontextmenü *New Child* → *EReference*

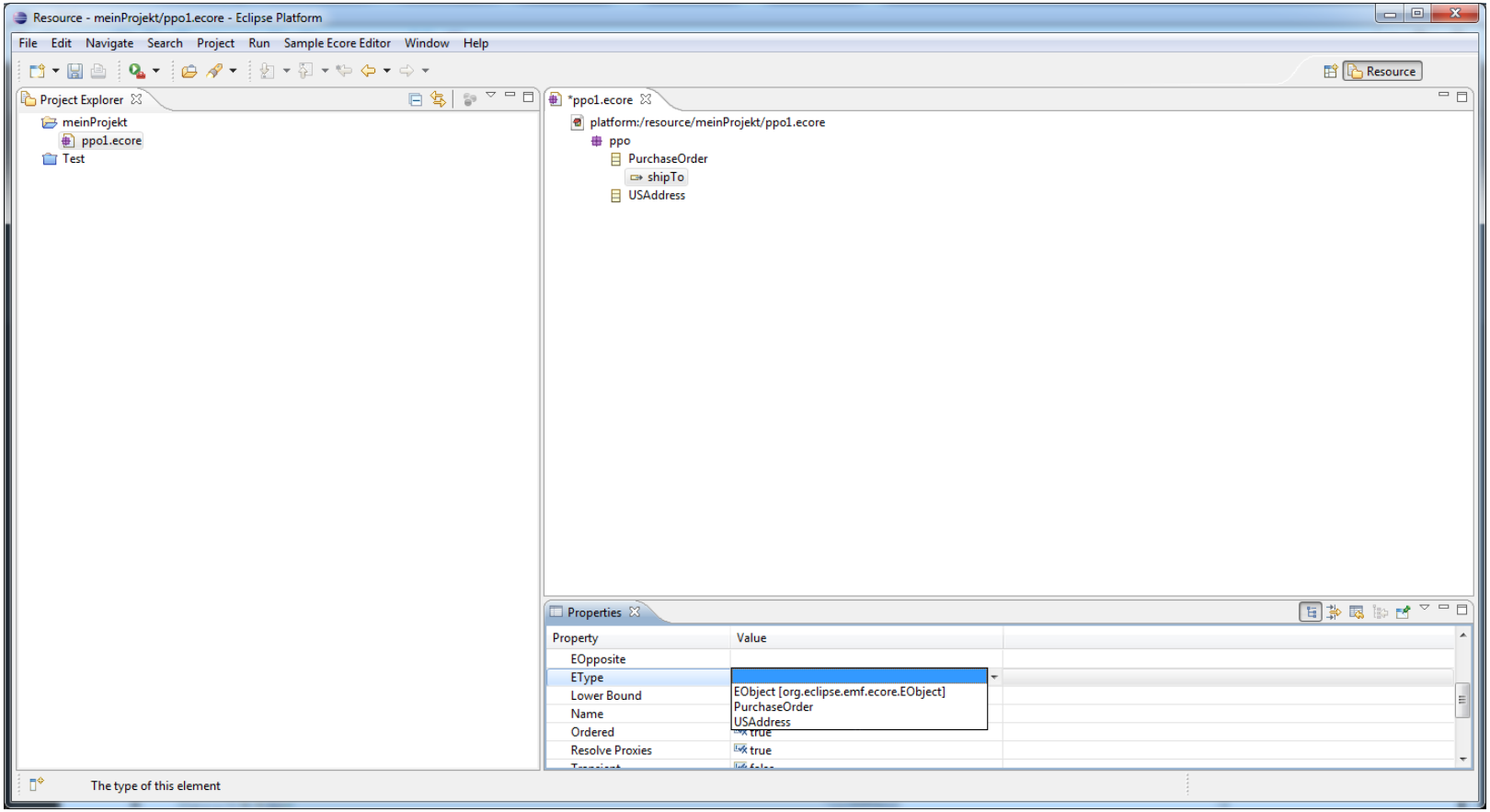

Abbildung A.6: Modellieren im Ecore-Baumeditor: Der Zieltyp der Assoziation kann im Properties-Fenster ausgewählt werden.

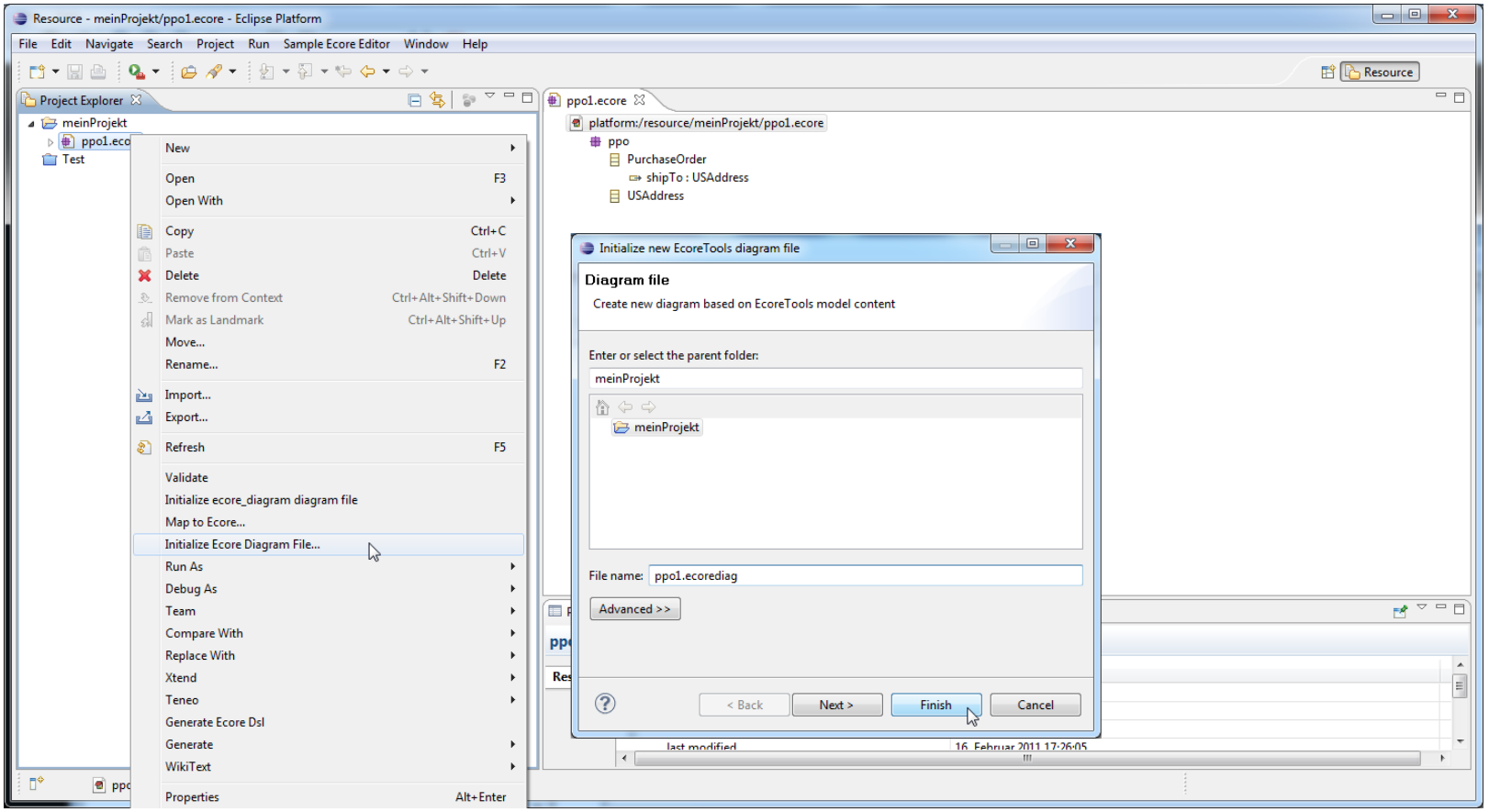

Abbildung A.7: Modellieren im Ecore-Diagrammeditor: Initialisierung eines graphischen Diagramms zu einem Ecore-Klassendiagramm über das Konextmenü *Initialize Ecore Diagram File*. Dabei muss der Ordner und der Name des Ecore-Diagramms festgelegt werden.

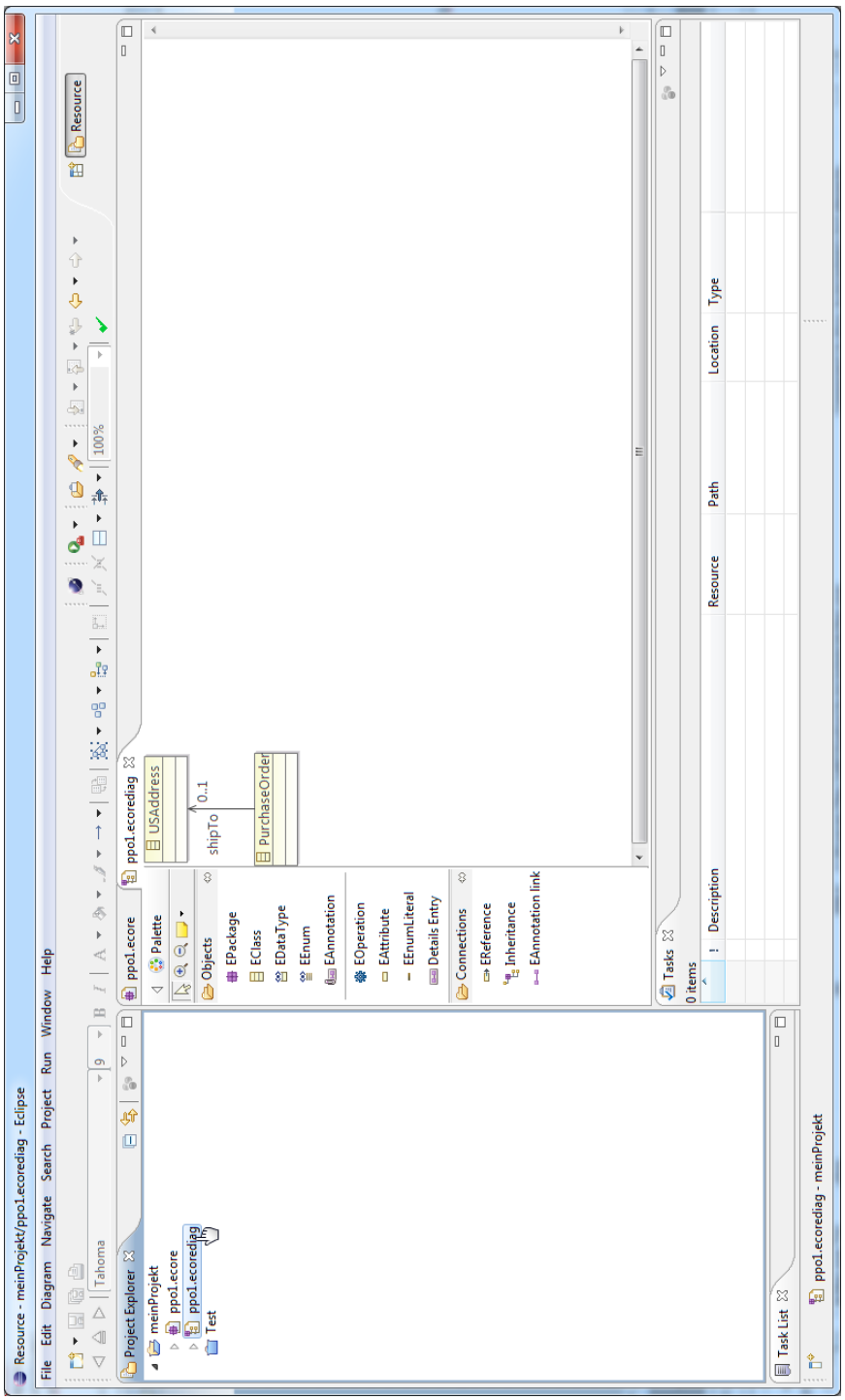

Abbildung A.8: Modellieren im Ecore-Diagrammeditor: Graphische Sicht auf das modellierte Ecore-Diagramm.

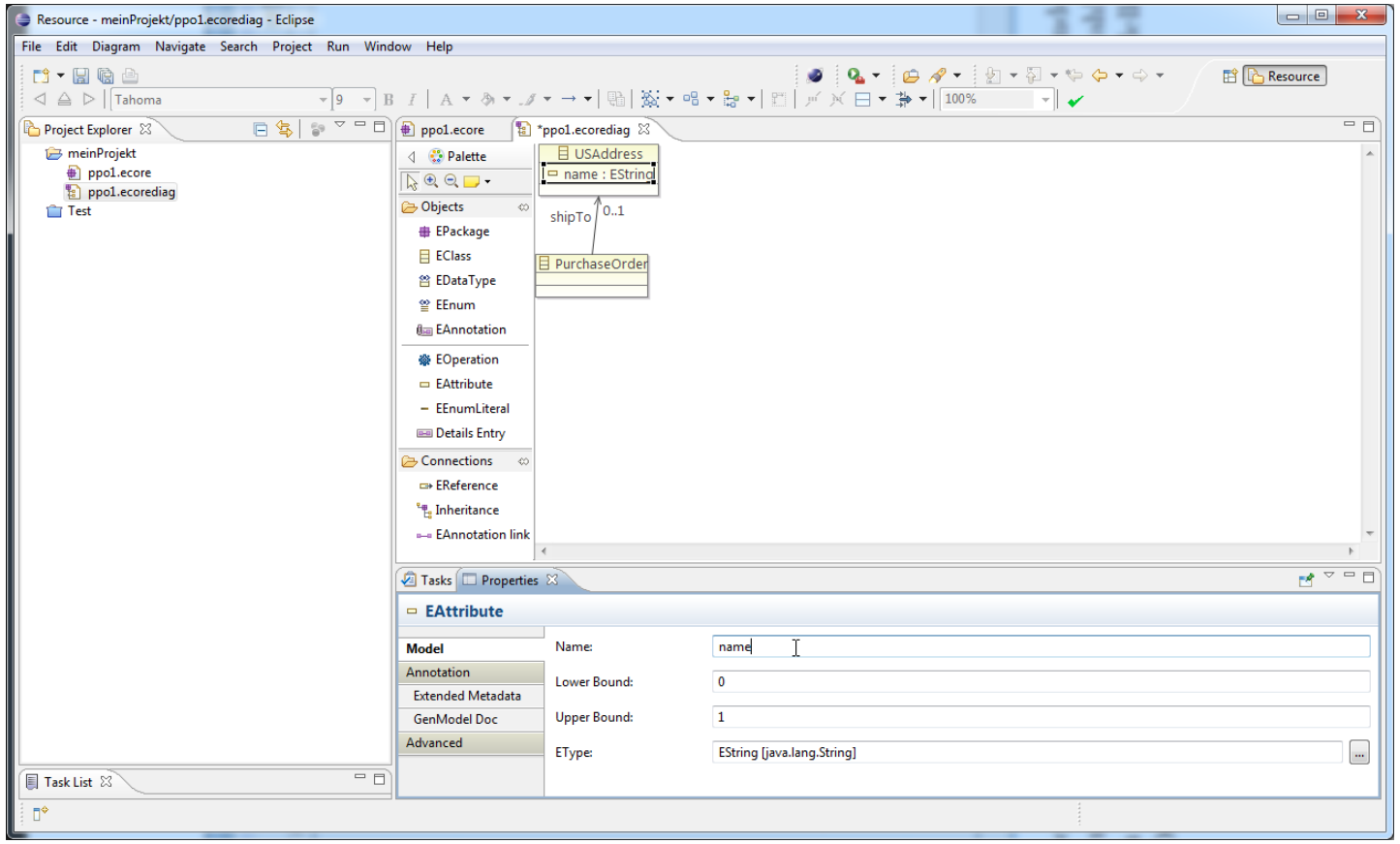

Abbildung A.9: Modellieren im Ecore-Diagrammeditor: Ein Attribut wird hinzugefügt.

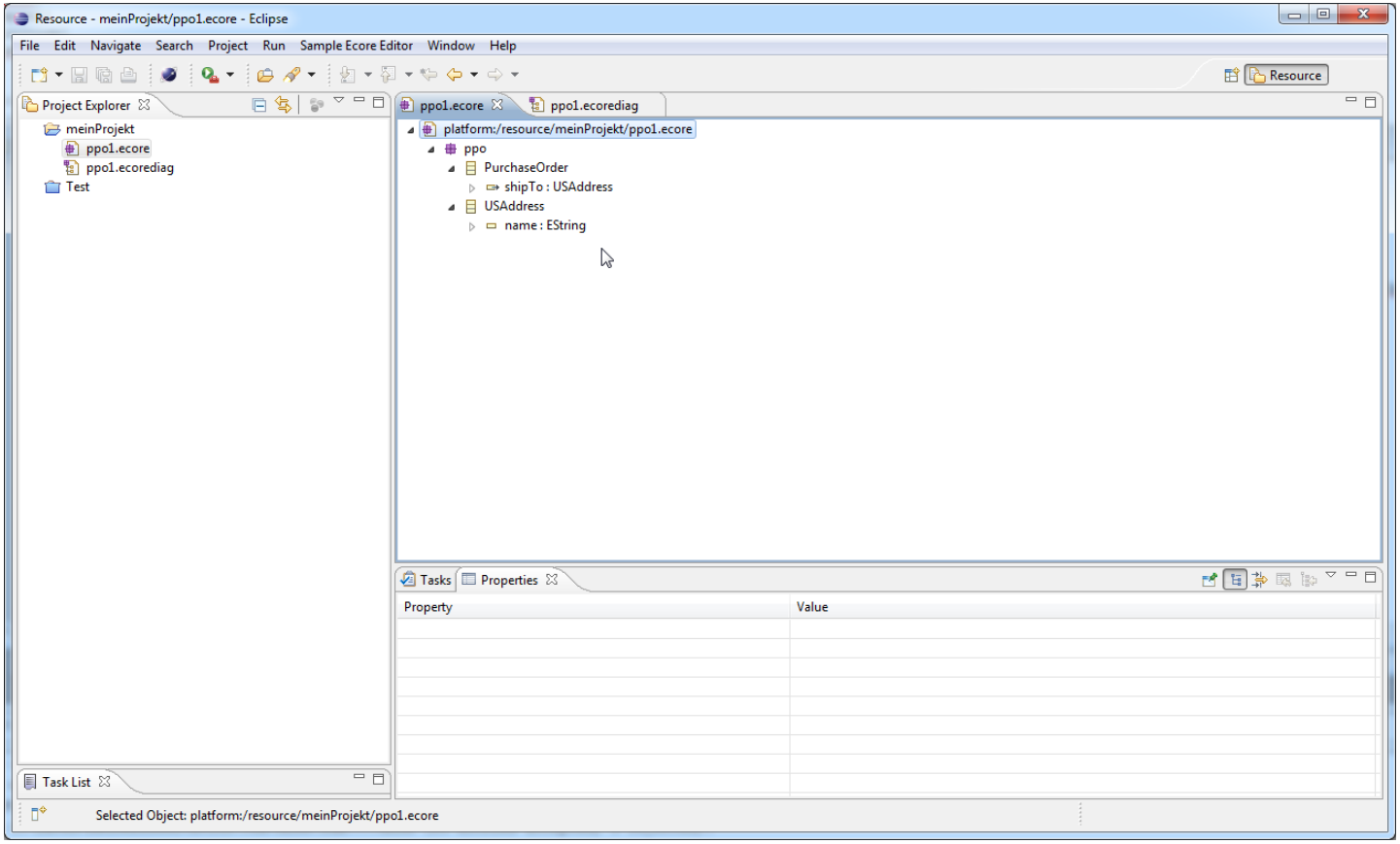

Abbildung A.10: Modellieren im Ecore-Diagrammeditor: Nach dem Speichern der Datei `*.ecorediag` wird die Änderung auch in der Datei `*.ecore` umgesetzt.

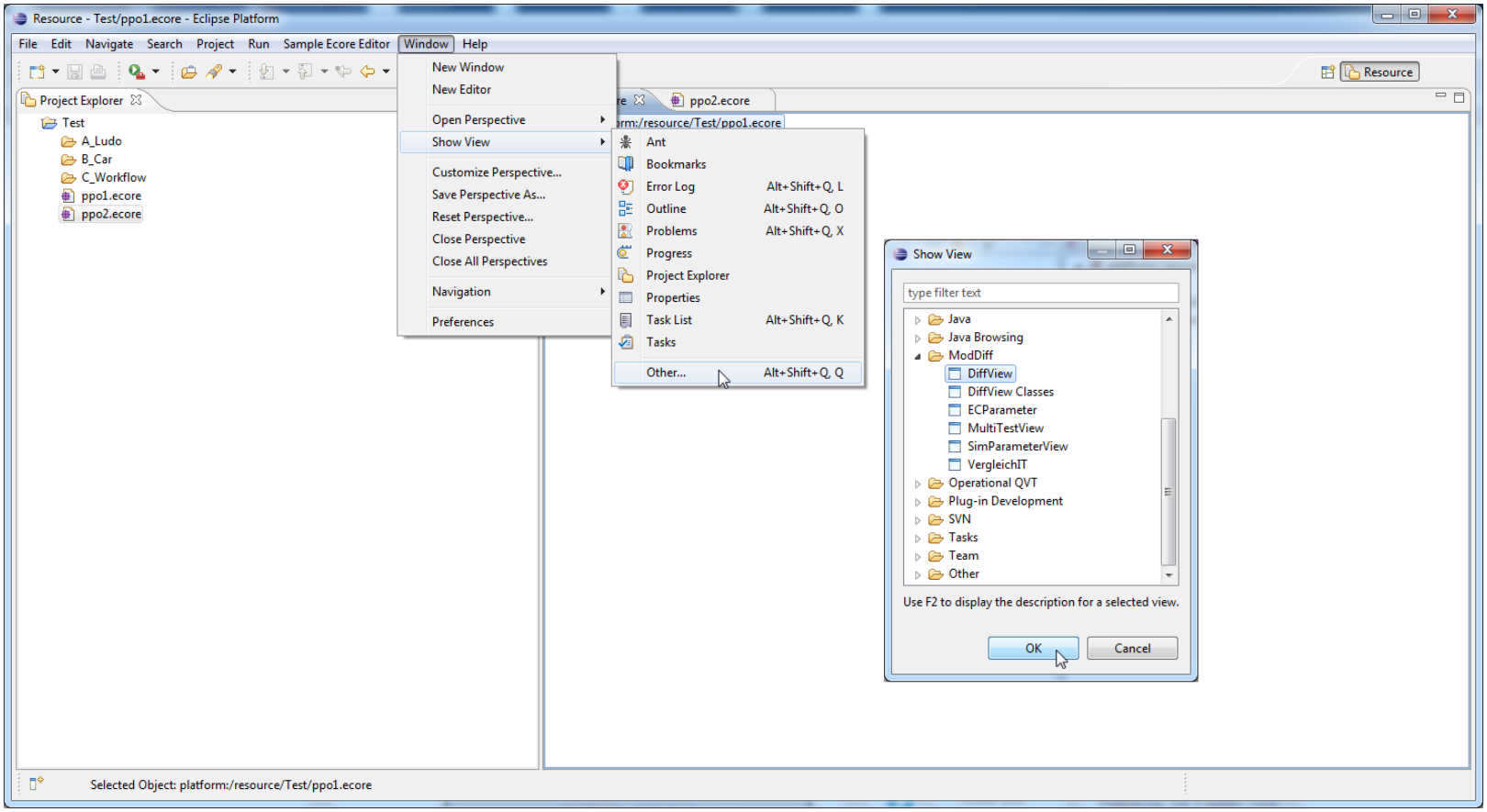

Abbildung A.11: Vergleich von Ecore-Klassendiagrammen: Die beiden geöffneten Ecore-Klassendiagramme `ppo1.ecore` und `ppo2.ecore` sollen verglichen werden. Die Ansicht zum Vergleich kann über *Window* → *Show View* → *Other* → *ModDiff* → *DiffView* geöffnet werden.

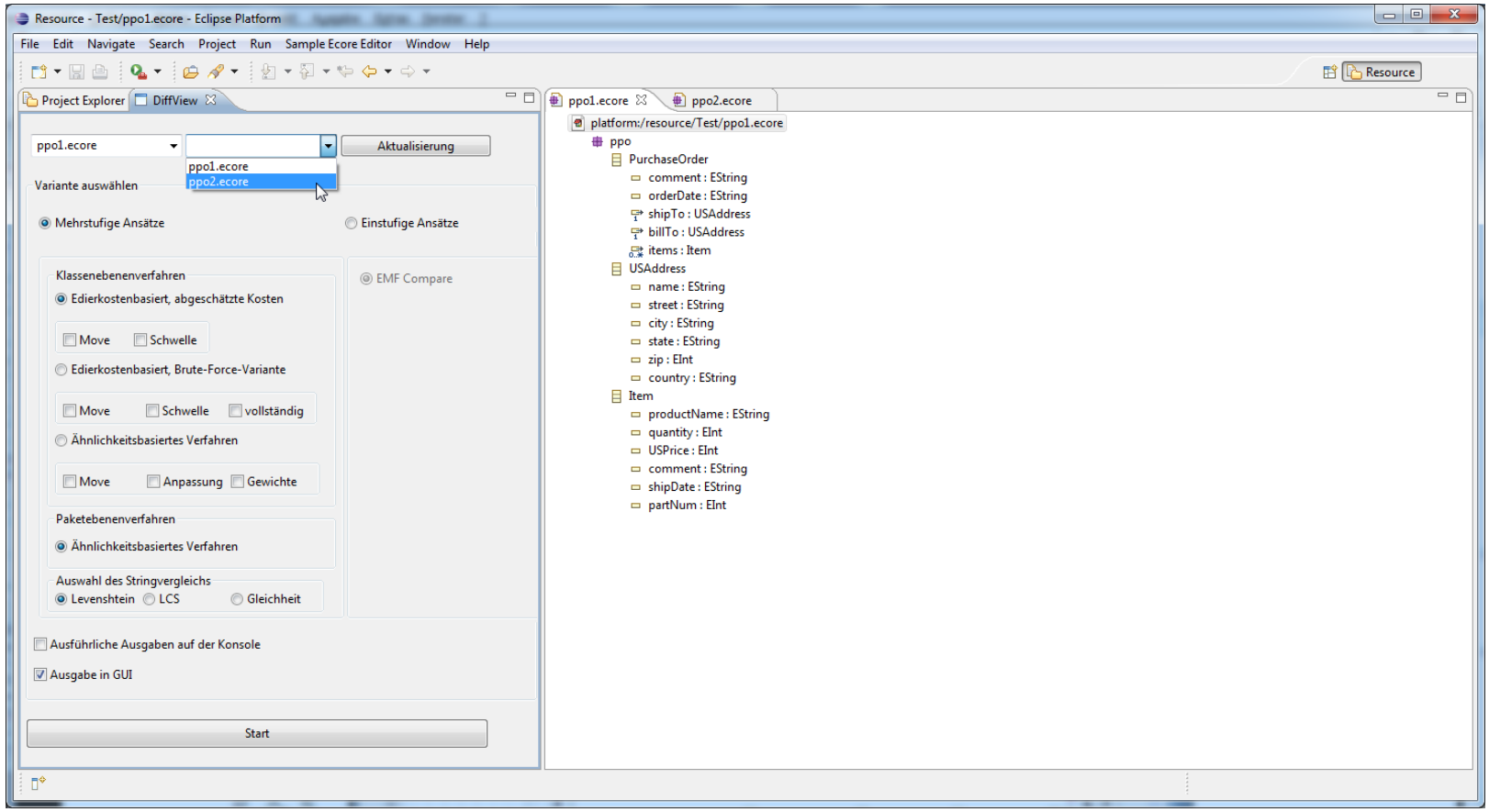

Abbildung A.12: Vergleich von Ecore-Klassendiagrammen: In der DiffView (links) werden die zu vergleichenden Klassendiagramme und das Verfahren ausgewählt. Der Aktualisierungsbutton aktualisiert die Auswahlmöglichkeiten der Dateien in den beiden Pull-down-Menüs entsprechend den im Editor geöffneten Modellen. Der Start-Button (unten) löst den Diagrammvergleich aus.

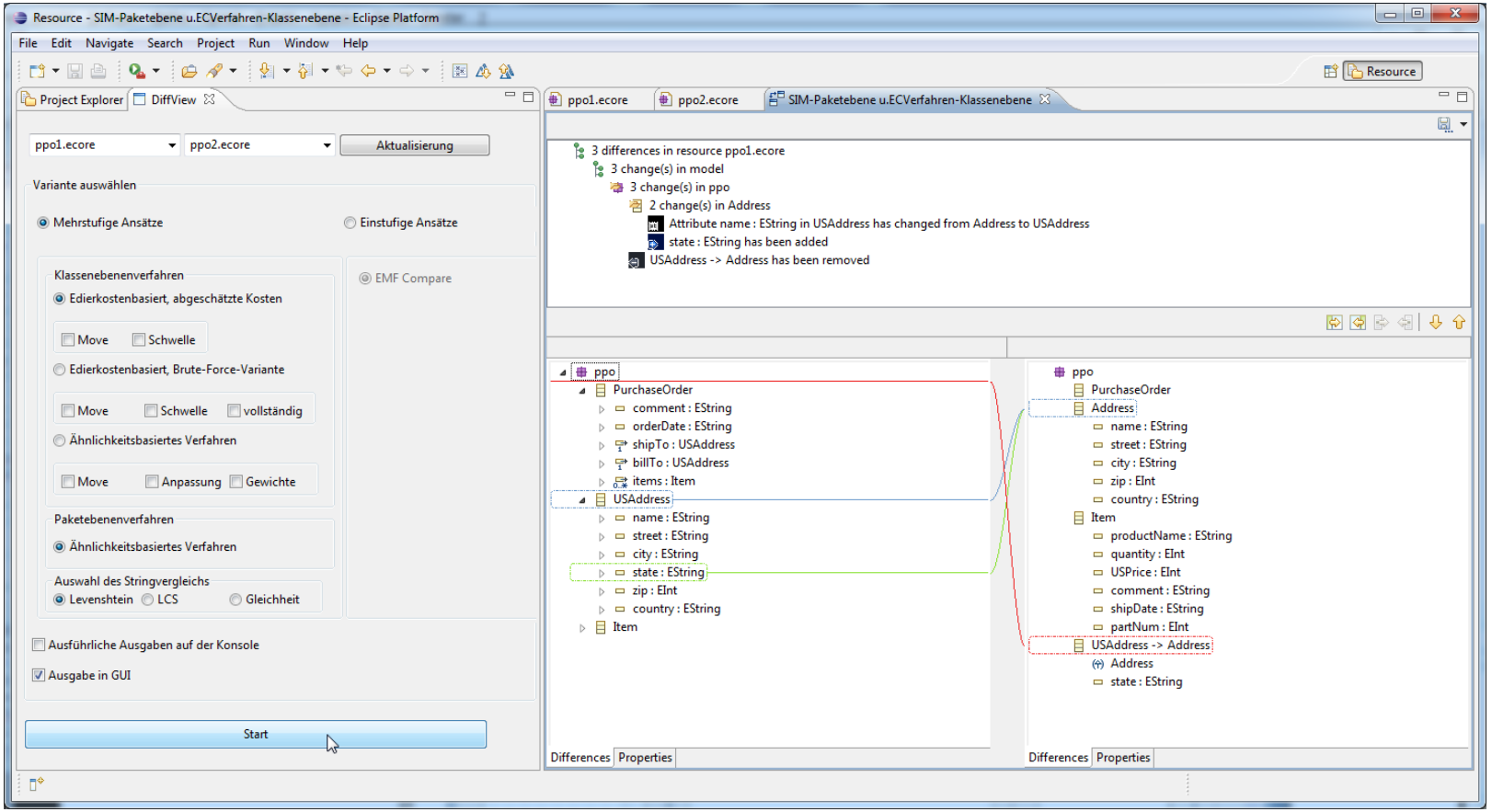

Abbildung A.13: Vergleich von Ecore-Klassendiagrammen: Der EMF Compare UI Editor öffnet sich und zeigt die Unterschiede in der Baumansicht.

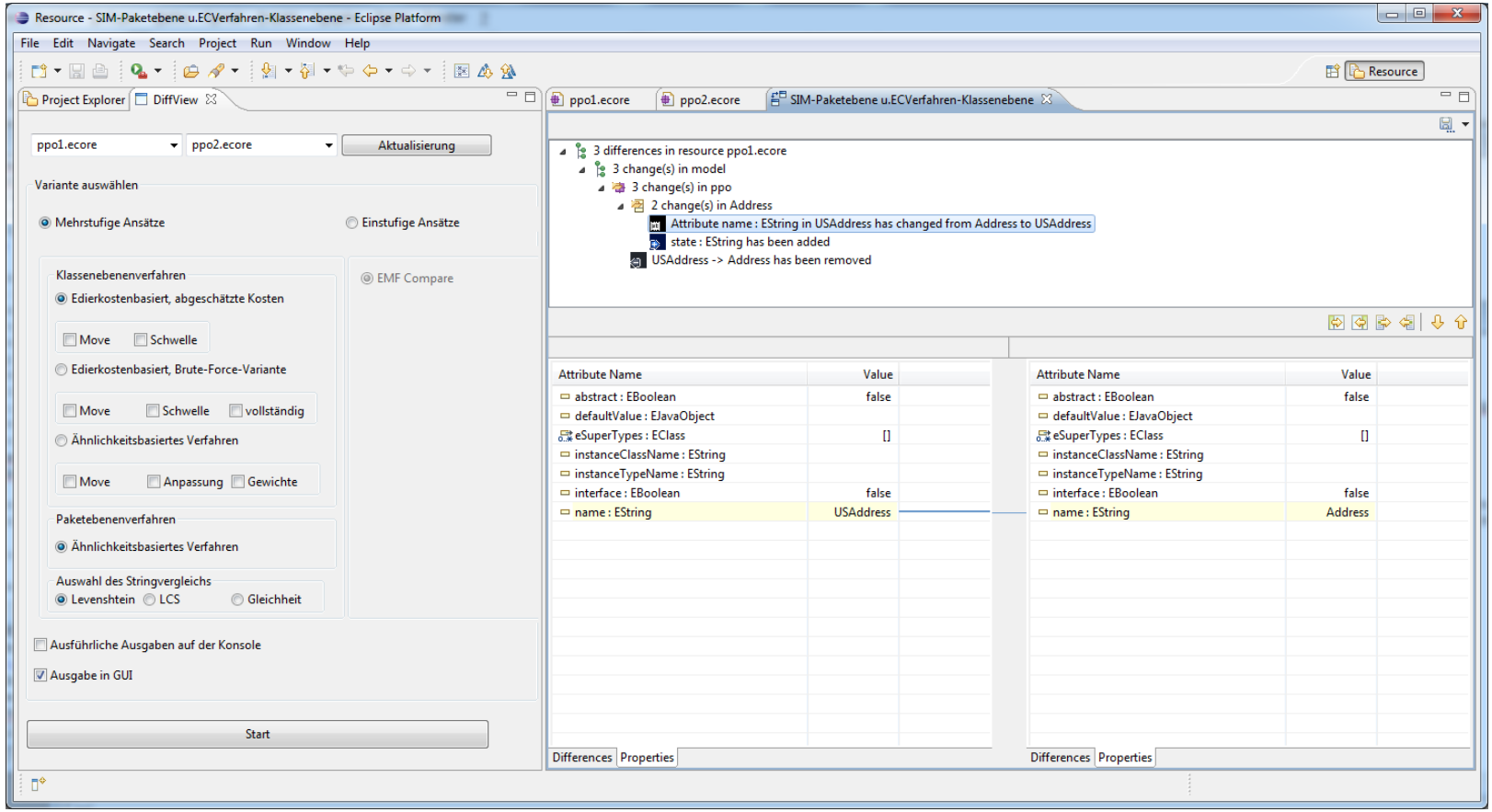

Abbildung A.14: Vergleich von Ecore-Klassendiagrammen: Details geänderter Elemente werden in der Properties-Sicht angezeigt.

A.2 Serialisierung des Korrespondenzmodells und des Differenzmodells

Listing A.1: XML-Darstellung des `MatchModels`

```
1  <?xml version="1.0" encoding="Cp1252"?>
2  <match:MatchModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/
       XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:match="http://www.eclipse.org/emf/compare/match/1.1">
3    <matchedElements xsi:type="match:Match2Elements">
4      <subMatchElements xsi:type="match:Match2Elements">
5        <subMatchElements xsi:type="match:Match2Elements">
6          <leftElement href="platform:/resource/test/ppo1.ecore#//
               PurchaseOrder/comment"/>
7          <rightElement href="platform:/resource/test/ppo2.ecore#//
               PurchaseOrder/comment"/>
8        </subMatchElements>
9        <subMatchElements xsi:type="match:Match2Elements">
10         <leftElement href="platform:/resource/test/ppo1.ecore#//
               PurchaseOrder/orderDate"/>
11         <rightElement href="platform:/resource/test/ppo2.ecore#//
               PurchaseOrder/orderDate"/>
12       </subMatchElements>
13       <subMatchElements xsi:type="match:Match2Elements">
14         <leftElement href="platform:/resource/test/ppo1.ecore#//
               PurchaseOrder/shipTo"/>
15         <rightElement href="platform:/resource/test/ppo2.ecore#//
               PurchaseOrder/shipTo"/>
16       </subMatchElements>
17       <subMatchElements xsi:type="match:Match2Elements">
18         <leftElement href="platform:/resource/test/ppo1.ecore#//
               PurchaseOrder/billTo"/>
19         <rightElement href="platform:/resource/test/ppo2.ecore#//
               PurchaseOrder/billTo"/>
20       </subMatchElements>
21       <subMatchElements xsi:type="match:Match2Elements">
22         <leftElement href="platform:/resource/test/ppo1.ecore#//
               PurchaseOrder/items"/>
23         <rightElement href="platform:/resource/test/ppo2.ecore#//
               PurchaseOrder/items"/>
24       </subMatchElements>
25       <leftElement href="platform:/resource/test/ppo1.ecore#//
             PurchaseOrder"/>
26       <rightElement href="platform:/resource/test/ppo2.ecore#//
             PurchaseOrder"/>
27     </subMatchElements>
```

```
28      <subMatchElements xsi:type="match:Match2Elements">
29        <subMatchElements xsi:type="match:Match2Elements">
30          <leftElement href="platform:/resource/test/ppo1.ecore#//
                USAddress/name"/>
31          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Address/name"/>
32        </subMatchElements>
33        <subMatchElements xsi:type="match:Match2Elements">
34          <leftElement href="platform:/resource/test/ppo1.ecore#//
                USAddress/street"/>
35          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Address/street"/>
36        </subMatchElements>
37        <subMatchElements xsi:type="match:Match2Elements">
38          <leftElement href="platform:/resource/test/ppo1.ecore#//
                USAddress/city"/>
39          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Address/city"/>
40        </subMatchElements>
41        <subMatchElements xsi:type="match:Match2Elements">
42          <leftElement href="platform:/resource/test/ppo1.ecore#//
                USAddress/zip"/>
43          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Address/zip"/>
44        </subMatchElements>
45        <subMatchElements xsi:type="match:Match2Elements">
46          <leftElement href="platform:/resource/test/ppo1.ecore#//
                USAddress/country"/>
47          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Address/country"/>
48        </subMatchElements>
49        <leftElement href="platform:/resource/test/ppo1.ecore#//
              USAddress"/>
50        <rightElement href="platform:/resource/test/ppo2.ecore#//
              Address"/>
51      </subMatchElements>
52      <subMatchElements xsi:type="match:Match2Elements">
53        <subMatchElements xsi:type="match:Match2Elements">
54          <leftElement href="platform:/resource/test/ppo1.ecore#//Item
                /productName"/>
55          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Item/productName"/>
56        </subMatchElements>
57        <subMatchElements xsi:type="match:Match2Elements">
58          <leftElement href="platform:/resource/test/ppo1.ecore#//Item
                /quantity"/>
```

```
59          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Item/quantity"/>
60        </subMatchElements>
61        <subMatchElements xsi:type="match:Match2Elements">
62          <leftElement href="platform:/resource/test/ppo1.ecore#//Item
                /USPrice"/>
63          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Item/USPrice"/>
64        </subMatchElements>
65        <subMatchElements xsi:type="match:Match2Elements">
66          <leftElement href="platform:/resource/test/ppo1.ecore#//Item
                /comment"/>
67          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Item/comment"/>
68        </subMatchElements>
69        <subMatchElements xsi:type="match:Match2Elements">
70          <leftElement href="platform:/resource/test/ppo1.ecore#//Item
                /shipDate"/>
71          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Item/shipDate"/>
72        </subMatchElements>
73        <subMatchElements xsi:type="match:Match2Elements">
74          <leftElement href="platform:/resource/test/ppo1.ecore#//Item
                /partNum"/>
75          <rightElement href="platform:/resource/test/ppo2.ecore#//
                Item/partNum"/>
76        </subMatchElements>
77        <leftElement href="platform:/resource/test/ppo1.ecore#//Item"/
              >
78        <rightElement href="platform:/resource/test/ppo2.ecore#//Item"
              />
79      </subMatchElements>
80      <leftElement href="platform:/resource/test/ppo1.ecore#/"/>
81      <rightElement href="platform:/resource/test/ppo2.ecore#/"/>
82    </matchedElements>
83    <unmatchedElements>
84      <element href="platform:/resource/test/ppo1.ecore#//USAddress/
            state"/>
85    </unmatchedElements>
86    <unmatchedElements side="Right">
87      <element href="platform:/resource/test/ppo2.ecore#//USAddress"/>
88    </unmatchedElements>
89    <leftRoots href="platform:/resource/test/ppo1.ecore#/"/>
90    <rightRoots href="platform:/resource/test/ppo2.ecore#/"/>
91  </match:MatchModel>
```

Listing A.2: XML-Darstellung des `DiffModels`

```
1  <?xml version="1.0" encoding="Cp1252"?>
2  <diff:DiffModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:diff
     ="http://www.eclipse.org/emf/compare/diff/1.1">
3  <ownedElements xsi:type="diff:DiffGroup">
4  <subDiffElements xsi:type="diff:DiffGroup">
5    <subDiffElements xsi:type="diff:DiffGroup">
6     <subDiffElements xsi:type="diff:UpdateAttribute">
7      <attribute href="http://www.eclipse.org/emf/2002/Ecore#//
           ENamedElement/name" />
8      <leftElement href="platform:/../ppo1.ecore#//USAddress" />
9      <rightElement href="platform:/../ppo2.ecore#//Address" />
10    </subDiffElements>
11    <subDiffElements xsi:type="diff:ModelElementChangeLeftTarget">
12     <rightParent href="platform:/../ppo2.ecore#//Address" />
13     <leftElement href="platform:/../ppo1.ecore#//USAddress/state" />
14    </subDiffElements>
15    <rightParent href="platform:/../ppo2.ecore#//Address" />
16   </subDiffElements>
17   <subDiffElements xsi:type="diff:ModelElementChangeRightTarget">
18    <leftParent href="platform:/../ppo1.ecore#/" />
19    <rightElement href="platform:/../ppo2.ecore#//USAddress" />
20   </subDiffElements>
21   <rightParent href="platform:/../ppo2.ecore#/" />
22  </subDiffElements>
23 </ownedElements>
24 <leftRoots href="platform:/../ppo1.ecore#/" />
25 <rightRoots href="platform:/../ppo2.ecore#/" />
26 </diff:DiffModel>
```

A.3 Ergänzungen zu Beispielen

In diesem Abschnitt werden Differenzberichte und Kostenlisten zu Beispielen ergänzt, die aus Platzgründen und aus Gründen der Lesbarkeit nicht direkt im Kapitel des Beispiels abgedruckt sind.

A.3.1 Differenzberichte zu Beispiel 11

Die Abbildungen A.15 und A.16 zeigen eine textuelle und eine graphische Beschreibung der Ergebnisse der Differenzberechnung zu Beispiel 11, wie sie im EMF Compare UI Editor visualisiert werden. Abbildung A.17 zeigt das Ergebnis, wenn die beiden Modelle stattdessen mit EMF Compare verglichen werden. EMF Compare identifiziert neben den vier Einzelelementen, die auch das ECVerfahren (mit Schwelle) identifiziert hat, zwei wei-

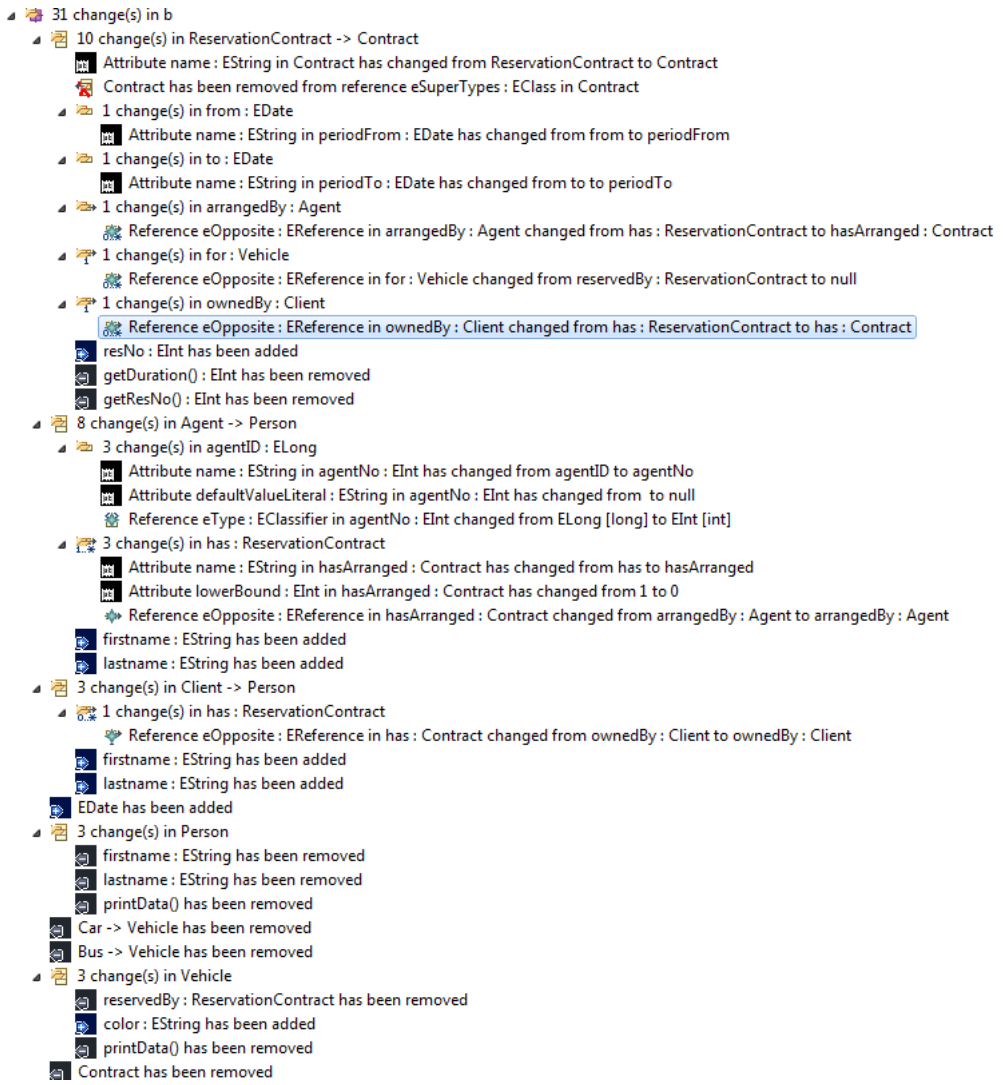

Abbildung A.15: Textuelle Beschreibung der Unterschiede im EMF Compare UI Editor: Ergebnisse zu Beispiel 11 (ECVerfahren mit Schwelle)

tere Einzelelelemente, da die Klassen `Contract` aus Modell A und `ReservationContract` aus Modell B nicht als korrespondierend betrachtet werden.

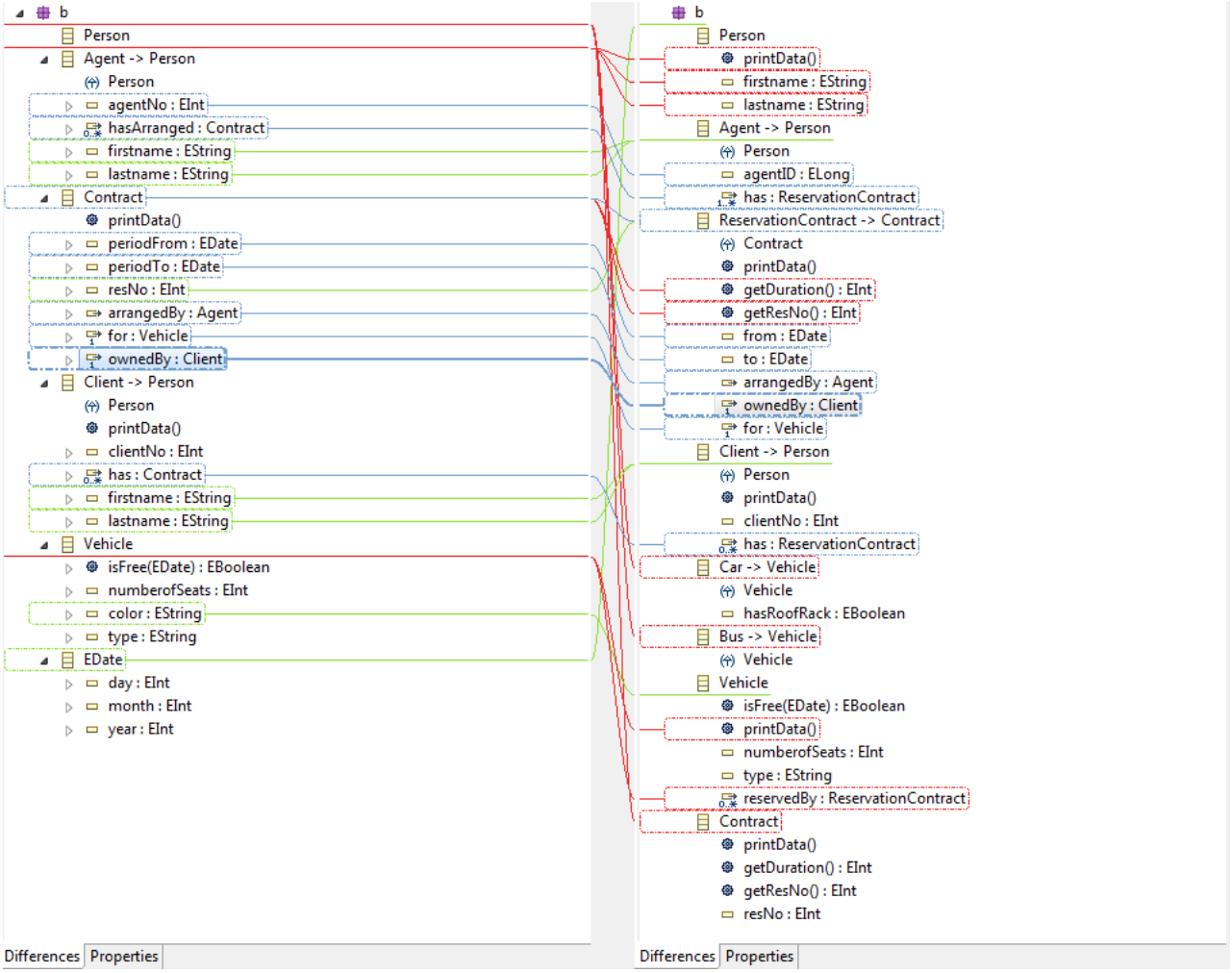

Abbildung A.16: Visualisierung der Unterschiede in der Baumsicht des EMF Compare UI Editors: Ergebnisse zu Beispiel 11 (ECVerfahren mit Schwelle)

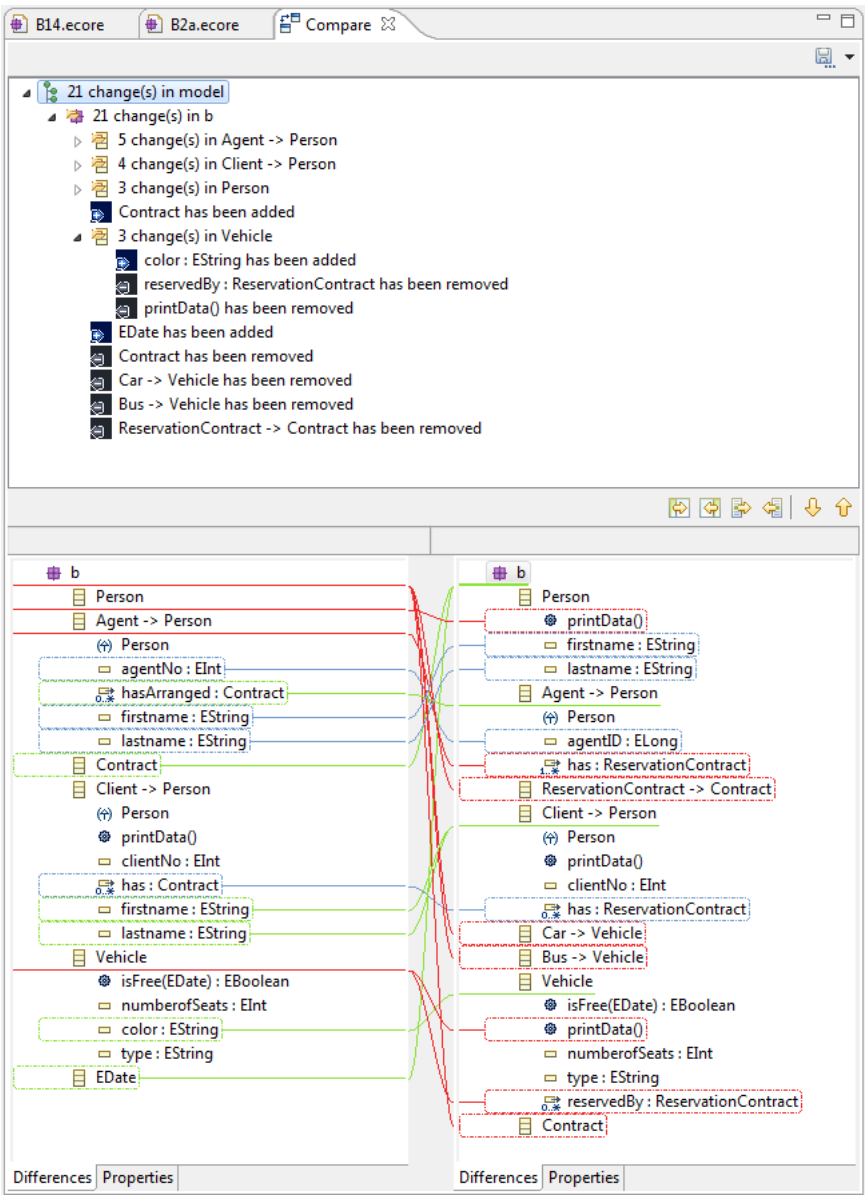

Abbildung A.17: Ergebnis des Vergleichs der Modelle aus Beispiel 11 mit EMF Compare

A.3.2 Die vollständigen Kostenlisten zu Beispiel 19

Es folgen die Kosten der aktiven Kanten des Vergleichsgraphen auf Klassenebene und der jeweiligen Untergraphen.

Listing A.3: Gesamtkosten des ECVerfahrens (geschätzte Kosten)

```
Gesamtkosten (ECVerfahren (Levenshtein)): 35.126351072985884
Matching:
  Person - Person ( 6.0 )
    Attribut ( 4.0 )
      Matching:
        firstname - e ( 2.0 )
        lastname - e ( 2.0 )
    Methoden ( 2.0 )
      Matching:
        printData - e ( 2.0 )
  Agent - Agent ( 4.0 )
    In_Referenzen ( 0.0 )
      Matching:
        arrangedBy - arrangedBy ( 0.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        hasArranged - hasArranged ( 0.0 )
    Attribut ( 4.0 )
      Matching:
        agentNo - agentNo ( 0.0 )
        e - firstname ( 2.0 )
        e - lastname ( 2.0 )
  Contract - ReservationContract ( 3.5789473684210527 )
    In_Referenzen ( 1.5 )
      Matching:
        hasArranged - hasArranged ( 0.0 )
        has - has ( 0.0 )
        reservedBy - reservedBy ( 0.0 )
        reservedBy - e ( 1.5 )
    Out_Referenzen ( 1.5 )
      Matching:
        arrangedBy - arrangedBy ( 0.0 )
        for - for ( 0.0 )
        ownedBy - ownedBy ( 0.0 )
        for - e ( 1.5 )
    Attribut ( 0.0 )
      Matching:
        periodFrom - periodFrom ( 0.0 )
        periodTo - periodTo ( 0.0 )
        resNo - resNo ( 0.0 )
    Methoden ( 0.0 )
```

```
      Matching:
        printData - printData    ( 0.0 )
          Rueckgabetyp ( 0.0 )
            Matching:
              void - void    ( 0.0 )
  e - Client    ( 10.0 )
    In_Referenzen ( 1.5 )
      Matching:
        ownedBy - e    ( 1.5 )
    Out_Referenzen ( 1.5 )
      Matching:
        has - e    ( 1.5 )
    Attribut ( 2.0 )
      Matching:
        clientNo - e    ( 2.0 )
    Methoden ( 2.0 )
      Matching:
        printData - e    ( 2.0 )
  Vehicle - car    ( 2.75 )
    In_Referenzen ( 0.0 )
      Matching:
        for - for    ( 0.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        reservedBy - reservedBy    ( 0.0 )
    Attribut ( 0.0 )
      Matching:
        numberofSeats - numberofSeats    ( 0.0 )
        color - color    ( 0.0 )
        type - type    ( 0.0 )
    Methoden ( 2.0 )
      Matching:
        isFree - isFree    ( 0.0 )
          Eingabeparameter ( 0.0 )
            Matching:
              d - d    ( 0.0 )
          Rueckgabetyp ( 0.0 )
            Matching:
              EBoolean - EBoolean    ( 0.0 )
        printData - e    ( 2.0 )
  EDate - EDate    ( 0.0 )
    Attribut ( 0.0 )
      Matching:
        day - day    ( 0.0 )
        month - month    ( 0.0 )
        year - year    ( 0.0 )
```

```
  Client - bus     ( 8.79740370456483 )
    In_Referenzen ( 0.4117647058823529 )
      Matching:
        for - ownedBy     ( 0.4117647058823529 )
    Out_Referenzen ( 0.4090909090909091 )
      Matching:
        reservedBy - has     ( 0.4090909090909091 )
    Attribut ( 2.176548089591568 )
      Matching:
        numberofSeats - clientNo     ( 0.7272727272727273 )
        color - firstname     ( 0.782608695652174 )
        type - lastname     ( 0.6666666666666666 )
    Methoden ( 4.0 )
      Matching:
        isFree - e     ( 4.0 )
        printData - printData     ( 0.0 )
          Rueckgabetyp ( 0.0 )
            Matching:
              void - void     ( 0.0 )
```

Listing A.4: Gesamtkosten des Vergleichsgraphen (tatsächliche Kosten):

```
Gesamtkosten (tatsächliche Kosten (Levenshtein)): 37.947206687959145
Matching:
  Person - Person     ( 8.0 )
    Attribut ( 4.0 )
      Matching:
        firstname - e     ( 2.0 )
        lastname - e     ( 2.0 )
    Methoden ( 2.0 )
      Matching:
        printData - e     ( 2.0 )
  Agent - Agent     ( 4.0 )
    In_Referenzen ( 0.0 )
      Matching:
        arrangedBy - arrangedBy     ( 0.0 )
    Attribut ( 4.0 )
      Matching:
        agentNo - agentNo     ( 0.0 )
        e - firstname     ( 2.0 )
        e - lastname     ( 2.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        hasArranged - hasArranged     ( 0.0 )
  Contract - ReservationContract     ( 4.399802983394315 )
    In_Referenzen ( 0.0 )
      Matching:
```

```
        reservedBy - reservedBy    ( 0.0 )
    In_Referenzen ( 0.4090909090909091 )
      Matching:
        reservedBy - has    ( 0.4090909090909091 )
    In_Referenzen 2 ( 1.5 )
      Matching:
        has - e    ( 1.5 )
    In_Referenzen ( 0.0 )
      Matching:
        hasArranged - hasArranged    ( 0.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        arrangedBy - arrangedBy    ( 0.0 )
    Attribut ( 0.0 )
      Matching:
        periodFrom - periodFrom    ( 0.0 )
        periodTo - periodTo    ( 0.0 )
        resNo - resNo    ( 0.0 )
    Out_Referenzen ( 0.4117647058823529 )
      Matching:
        for - ownedBy    ( 0.4117647058823529 )
    Out_Referenzen 2 ( 1.5 )
      Matching:
        ownedBy - e    ( 1.5 )
    Methoden ( 0.0 )
      Matching:
        printData - printData    ( 0.0 )
          Rueckgabetyp ( 0.0 )
            Matching:
              void - void    ( 0.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        for - for    ( 0.0 )
  e - Client    ( 10.0 )
    In_Referenzen ( 1.5 )
      Matching:
        ownedBy - e    ( 1.5 )
    Out_Referenzen ( 1.5 )
      Matching:
        has - e    ( 1.5 )
    Attribut ( 2.0 )
      Matching:
        clientNo - e    ( 2.0 )
    Methoden ( 2.0 )
      Matching:
        printData - e    ( 2.0 )
```

```
Vehicle - car    ( 2.75 )
  In_Referenzen ( 0.0 )
    Matching:
      for - for    ( 0.0 )
  Attribut ( 0.0 )
    Matching:
      numberofSeats - numberofSeats    ( 0.0 )
      color - color    ( 0.0 )
      type - type    ( 0.0 )
  Methoden ( 2.0 )
    Matching:
      isFree - isFree    ( 0.0 )
        Eingabeparameter ( 0.0 )
          Matching:
            d - d    ( 0.0 )
        Rueckgabetyp ( 0.0 )
          Matching:
            EBoolean - EBoolean    ( 0.0 )
      printData - e    ( 2.0 )
  Out_Referenzen ( 0.0 )
    Matching:
      reservedBy - reservedBy    ( 0.0 )
EDate - EDate    ( 0.0 )
  Attribut ( 0.0 )
    Matching:
      day - day    ( 0.0 )
      month - month    ( 0.0 )
      year - year    ( 0.0 )
Client - bus    ( 8.79740370456483 )
  In_Referenzen ( 0.4117647058823529 )
    Matching:
      for - ownedBy    ( 0.4117647058823529 )
  Attribut ( 2.176548089591568 )
    Matching:
      numberofSeats - clientNo    ( 0.7272727272727273 )
      color - firstname    ( 0.782608695652174 )
      type - lastname    ( 0.6666666666666666 )
  Methoden ( 4.0 )
    Matching:
      isFree - e    ( 4.0 )
      printData - printData    ( 0.0 )
        Rueckgabetyp ( 0.0 )
          Matching:
            void - void    ( 0.0 )
  Out_Referenzen ( 0.4090909090909091 )
    Matching:
```

```
        reservedBy - has    ( 0.4090909090909091 )
```

Listing A.5: Gesamtkosten des Vergleichsgraphen (Brute-Force)

```
Gesamtkosten Brute-Force (Levenshtein): 37.328947368421055
Matching:
  Person - Person    ( 6.0 )
    Attribut ( 4.0 )
      Matching:
        firstname - e    ( 2.0 )
        lastname - e    ( 2.0 )
    Methoden ( 2.0 )
      Matching:
        printData - e    ( 2.0 )
  Agent - Agent    ( 4.0 )
    In_Referenzen ( 0.0 )
      Matching:
        arrangedBy - arrangedBy    ( 0.0 )
    Attribut ( 4.0 )
      Matching:
        agentNo - agentNo    ( 0.0 )
        e - firstname    ( 2.0 )
        e - lastname    ( 2.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        hasArranged - hasArranged    ( 0.0 )
  Contract - ReservationContract    ( 3.5789473684210527 )
    In_Referenzen ( 0.0 )
      Matching:
        reservedBy - reservedBy    ( 0.0 )
    In_Referenzen 3 ( 1.5 )
      Matching:
        reservedBy - e    ( 1.5 )
    In_Referenzen ( 0.0 )
      Matching:
        has - has    ( 0.0 )
    In_Referenzen ( 0.0 )
      Matching:
        hasArranged - hasArranged    ( 0.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        arrangedBy - arrangedBy    ( 0.0 )
    Attribut ( 0.0 )
      Matching:
        periodFrom - periodFrom    ( 0.0 )
        periodTo - periodTo    ( 0.0 )
        resNo - resNo    ( 0.0 )
```

```
  Out_Referenzen 3 ( 1.5 )
    Matching:
      for - e ( 1.5 )
  Out_Referenzen ( 0.0 )
    Matching:
      ownedBy - ownedBy ( 0.0 )
  Methoden ( 0.0 )
    Matching:
      printData - printData ( 0.0 )
        Rueckgabetyp ( 0.0 )
          Matching:
            void - void ( 0.0 )
  Out_Referenzen ( 0.0 )
    Matching:
      for - for ( 0.0 )
Client - Client ( 4.0 )
  In_Referenzen ( 0.0 )
    Matching:
      ownedBy - ownedBy ( 0.0 )
  Attribut ( 4.0 )
    Matching:
      clientNo - clientNo ( 0.0 )
      e - firstname ( 2.0 )
      e - lastname ( 2.0 )
  Methoden ( 0.0 )
    Matching:
      printData - printData ( 0.0 )
        Rueckgabetyp ( 0.0 )
          Matching:
            void - void ( 0.0 )
  Out_Referenzen ( 0.0 )
    Matching:
      has - has ( 0.0 )
Vehicle - car ( 2.75 )
  In_Referenzen ( 0.0 )
    Matching:
      for - for ( 0.0 )
  Attribut ( 0.0 )
    Matching:
      numberofSeats - numberofSeats ( 0.0 )
      color - color ( 0.0 )
      type - type ( 0.0 )
  Methoden ( 2.0 )
    Matching:
      isFree - isFree ( 0.0 )
        Eingabeparameter ( 0.0 )
```

```
            Matching:
              d - d     ( 0.0 )
          Rueckgabetyp ( 0.0 )
            Matching:
              EBoolean - EBoolean     ( 0.0 )
        printData - e     ( 2.0 )
    Out_Referenzen ( 0.0 )
      Matching:
        reservedBy - reservedBy     ( 0.0 )
EDate - EDate     ( 0.0 )
    Attribut ( 0.0 )
      Matching:
        day - day     ( 0.0 )
        month - month     ( 0.0 )
        year - year     ( 0.0 )
e - bus     ( 17.0 )
    In_Referenzen ( 1.5 )
      Matching:
        for - e     ( 1.5 )
    Out_Referenzen ( 1.5 )
      Matching:
        reservedBy - e     ( 1.5 )
    Attribut ( 6.0 )
      Matching:
        numberofSeats - e     ( 2.0 )
        color - e     ( 2.0 )
        type - e     ( 2.0 )
    Methoden ( 6.0 )
      Matching:
        isFree - e     ( 4.0 )
        printData - e     ( 2.0 )
```

Literaturverzeichnis

[AAAN+06] Abi-Antoun, Marwan ; Aldrich, Jonathan ; Nahas, Nagi ; Schmerl, Bradley ; Garlan, David: Differencing and Merging of Architectural Views. In: *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. Washington, DC : IEEE Computer Society Press, 2006. – ISBN 0–7695–2579–2, S. 47–58

[AP03] Alanen, Marcus ; Porres, Ivan: Difference and Union of Models. In: Stevens, Perdita (Hrsg.) ; Whittle, Jon (Hrsg.) ; Booch, Grady (Hrsg.): *UML 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference* Bd. 2863. San Francisco, CA : Springer Verlag, Oktober 2003 (Lecture Notes in Computer Science), S. 2–17

[ASW09] Altmanninger, Kerstin ; Seidl, Martina ; Wimmer, Manuel: A survey on model versioning approaches. In: *International Journal of Web Information Systems (IJWIS)* 5 (2009), Nr. 3, S. 271–304

[BDW08] Buchmann, Thomas ; Dotor, Alexander ; Westfechtel, Bernhard: MOD2-SCM: Experiences with co-evolving models when designing a modular SCM system. In: Deridder, Dirk (Hrsg.) ; Gray, Jeff (Hrsg.) ; Pierantonio, Alfonso (Hrsg.) ; Schoppens, Pierre-Yves (Hrsg.): *Proceedings 1st International Workshop on Model Co-Evolution and Consistency Management (MCCM 2008)*. Toulouse, France, September 2008, S. 15–30

[BG61] Busacker, R.G. ; Gowen, P.J.: A Procedure for Determining a Family of Minimum Cost Flow Networks / John Hopkins University, Operations Research Office. Baltimore, MD, 1961 (15). – Forschungsbericht

[BHE09] Bildhauer, Daniel ; Horn, Tassilo ; Ebert, Jürgen: Similarity-driven software reuse. In: *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models (CVSM'09)*. Washington, DC : IEEE Computer Society Press, 2009. – ISBN 978–1–4244–3714–6, S. 31–36

[BK00] Bunke, H. ; Kandel, A.: Mean and maximum common subgraph of two graphs. In: *Pattern Recognition Letters* 21 (2000), Februar, Nr. 2, S. 163–168

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

[Bru] Brun, Cédric: *Comparing and Merging Models with Eclipse: an Update on EMF Compare.* `http://www.eclipsecon.org/2008/?page=sub/&id=328`. – Short Talk at the EclipseCon 2008, March 2008, California

[CAM02] Cobéna, Grégory ; Abitoul, Serge ; Marian, Amelie: Detecting Changes in XML Documents. In: *Proceedings of the 18th International Conference on Data Engineering (ICDE '02)*. Washington, DC : IEEE Computer Society Press, 2002, 41–52

[CFSV04] Conte, Donatello ; Foggia, Pasquale ; Sansone, Carlo ; Vento, Mario: Thirty Years Of Graph Matching In Pattern Recognition. In: *International Journal of Pattern Recognition and Artificial Intelligence* 18 (2004), Nr. 3, S. 265–298

[CGM97] Chawathe, Sudarshan S. ; Garcia-Molina, Hector: Meaningful change detection in structured data. In: Peckman, Joan M. (Hrsg.): *Proceedings ACM SIGMOD International Conference on Management of Data*, ACM Press, 1997. – ISBN 0–89791–911–4, 26–37

[CLRS07] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Algorithmen - eine Einführung*. 2. Auflage. München : Oldenburg Wissenschaftsverlag GmbH, 2007

[CRGMW96] Chawathe, Sudarshan S. ; Rajaraman, Anand ; Garcia-Molina, Hector ; Widom, Jennifer: Change detection in hierarchically structured information. In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*, ACM Press, 1996. – ISBN 0–89791–794–4, S. 493–504

[CW98] Conradi, Reidar ; Westfechtel, Bernhard: Version Models for Software Configuration Management. In: *ACM Computing Surveys* 30 (1998), Juni, Nr. 2, S. 232–282

[Dij59] Dijkstra, Edsger W.: A Note on Two Problems in Connection with Graphs. In: *Numerical Mathematics* 1 (1959), 269-271. `http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf`

[Ecl09] *Eclipse Documentation - Galileo Release*. The Eclipse Foundation, 2009

[ELH+05] Estublier, Jacky ; Leblang, David ; Hoek, André van d. ; Conradi, Reidar ; Clemm, Geoffrey ; Tichy, Walter ; Wiborg-Weber, Darcy: Impact of software engineering research on the practice of software configuration management. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 14 (2005), October, S. 383–430

[För07] Förtsch, Sabrina: Differenzberechnung auf Diagrammen mittels Editierabstand. In: *Softwaretechnik-Trends* 27 (2007), Mai, Nr. 2, S. 60–61

[FR07] France, Robert ; Rumpe, Bernhard: Model-driven Development of Complex Software: A Research Roadmap. In: *Future of Software Engineering 2007 (FOSE '07)*. Washington, DC : IEEE Computer Society Press, 2007. – ISBN 0–7695–2829–5, S. 37–54

[FV09] Fabro, Marcos Didonet D. ; Valduriez, Patrick: Towards the efficient development of model transformations using model weaving and matching transformations. In: *Software and System Modeling* 8 (2009), Nr. 3, S. 305–324

[FW07] Förtsch, Sabrina ; Westfechtel, Bernhard: Differencing and Merging of Software Diagrams - State of the Art and Challenges. In: Filipe, Joaquim (Hrsg.) ; Helfert, Markus (Hrsg.) ; Shishkov, Boris (Hrsg.): *Proceedings of the Second International Conference on Software and Data Technologies (ICSOFT 2007)*. Barcelona, Spain : INSTICC Press, Juli 2007, S. 90–99

[Gal87] Gal, Tomas (Hrsg.): *Grundlagen des Operations Research*. Bd. 2: *Graphen und Netzwerke - Netzplantechnik, Transportprobleme, Ganzzahlige Optimierung*. Berlin : Springer Verlag, 1987

[GBD07] Geiger, Leif ; Buchmann, Thomas ; Dotor, Alexander: EMF Code Generation with Fujaba. In: Geiger, Leif (Hrsg.) ; Giese, Holger (Hrsg.) ; Zündorf, Albert (Hrsg.): *Fujaba Days 2007*. Kassel, Germany, Oktober 2007, S. 25–29

[GEF] Graphical Modeling Project (GMP). In: *http://www.eclipse.org/modeling/gmp/*

[Gro09] Gronback, Richard C.: *Eclipse Modeling Project, A Domain-Specific Language Toolkit*. Upper Saddle River, NJ : Addison-Wesley, 2009

[HNR68] Hart, Peter ; Nilsson, Nils ; Raphael, Bertram: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *IEEE Transactions on Systems Science and Cybernetics* 4 (1968), S. 100–107

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

[Jäh08] Jähnel, Tobias: *Visualization of differences between EMF models*. Nürnberg, Germany, Georg-Simon-Ohm University of Applied Sciences Nürnberg, Masterarbeit, 2008. `http://mt.jonmedia.net`

[Jun08] Jungnickel, Dieter: *Algorithms and Computation in Mathematics*. Bd. 5: *Graphs, Networks and Algorithms*. 3. Auflage. Berlin, Germany : Springer Verlag, 2008

[JWZ94] Jiang, Tao ; Wang, Lusheng ; Zhang, Kaizhong: Alignment of Trees - An Alternative to Tree Edit. In: *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM '94)*. London, UK : Springer Verlag, 1994. – ISBN 3–540–58094–8, S. 75–86

[Kel09] Kelter, Udo: *Dokumentdifferenzen. Softwaretechnik 2, Praktische Informatik, Lehrmodul zur Vorlesung*. Universität Siegen, 2009 `http://pi.informatik.uni-siegen.de/lehre/`

[KKK+07] Kappel, Gerti ; Kargl, Horst ; Kramler, Gerhard ; Schauerhuber, Andrea ; Seidl, Martina ; Strommer, Michael ; Wimmer, Manuel: Matching Metamodels with Semantic Systems - An Experience Report. In: *Datenbanksysteme in Business, Technologie und Web (BTW 2007), Workshop Proceedings*, Verlag Mainz, 2007. – ISBN 3–86130–929–7, S. 38–52

[KN05] Krumke, Sven O. ; Noltemeier, Hartmut: *Graphentheoretische Konzepte und Algorithmen*. Wiesbaden : Teubner Verlag, 2005 (Leitfäden der Informatik)

[Kön09] Könemann, Patrick: Model-independent differences. In: *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models (CVSM'09)*. Washington, DC : IEEE Computer Society Press, 2009. – ISBN 978–1–4244–3714–6, S. 37–42

[KPP06] Kolovos, Dimitrios S. ; Paige, Richard E. ; Polack, Fiona A.: Model comparison: a foundation for model composition and model transformation testing. In: *Proceedings of the 2006 International Workshop on Global Integrated Model Management (GaMMa'06)*. Shanghai, China : ACM Press, New York, NY, 2006. – ISBN 1–59593–410–3, S. 13–20

[KWN05] Kelter, Udo ; Wehren, Jürgen ; Niere, Jörg: A Generic Difference Algorithm for UML Models. In: Liggesmeyer, Peter (Hrsg.) ; Pohl, Klaus (Hrsg.) ; Goedicke, Michael (Hrsg.): *Software Engineering 2005*, Gesellschaft für Informatik, 2005 (Lecture Notes in Informatics), S. 105–116

[Let05] Letkeman, Kim: Comparing and merging UML models in IBM Rational Software Architect. (2005). `http://www.ibm.com/developerworks/rational/library/`

[LGJ07] Lin, Y. ; Gray, J. ; Jouault, F.: DSMDiff: A Differentiation Tool for Domain-Specific Models. In: *European Journal of Information Systems* (2007), S. 349–361

[LKT06] Lindholm, Tancred ; Kangasharju, Jaakko ; Tarkoma, Sasu: Fast and simple XML tree differencing by sequence alignment. In: *Proceedings of the 2006 ACM Symposium on Document Engineering (DocEng '06)*. Amsterdam, The Netherlands : ACM Press, 2006. – ISBN 1–59593–515–0, S. 75–84

[LMB+01] Ledeczi, Akos ; Maroti, Miklos ; Bakay, Arpad ; Karsai, Gabor ; Garrett, Jason ; Thomason, Charles ; Nordstrom, Greg ; Sprinkle, Jonathan ; Volgyesi, Peter: The Generic Modeling Environment. In: *Proceedings of Workshop on Intelligent Signal Processing (WISP '01)*. Budapest, Hungary : IEEE Computer Society Press, Mai 2001

[MGH05] Mehra, Akhil ; Grundy, John C. ; Hosking, John G.: A generic approach to supporting diagram differencing and merging for collaborative design. In: Redmiles, David F. (Hrsg.) ; Ellman, Thomas (Hrsg.) ; Zisman, Andrea (Hrsg.): *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, ACM Press, 2005, 204–213

[MGMR02] Melnik, S. ; Garcia-Molina, H. ; Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In: *Proceedings 18th International Conference on Data Engineering*. San Jose, CA : IEEE Computer Society Press, Februar 2002, S. 117–128

[MKY06] Mandelin, David ; Kimelman, Doug ; Yellin, Daniel: A Bayesian approach to diagram matching with application to architectural models. In: *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*. New York, NY : ACM Press, 2006. – ISBN 1–59593–375–1, S. 222–231

[MRR10] Maoz, Shahar ; Ringert, Jan ; Rumpe, Bernhard: A Manifesto for Semantic Model Differencing. In: *Proceedings of the International Workshop on Models and Evolution (ME '10)*. Oslo, Norway, Oktober 2010

[MSW09a] Mülder, Andreas ; Schill, Holger ; Wendehals, Lothar: Modellvergleich mit EMF Compare: Funktionsweise des Frameworks. In: *Eclipse Magazin* Bd. 4. Frankfurt am Main, Germany : Software & Support Media GmbH, 2009

[MSW09b] Mülder, Andreas ; Schill, Holger ; Wendehals, Lothar: Modellvergleich mit EMF Compare (Teil 2: Ein Praxisbeispiel). In: *Eclipse Magazin* Bd. 5. Frankfurt am Main, Germany : Software & Support Media GmbH, 2009

[Mun57] Munkres, J.: Algorithms for the Assignment and Transportation Problems. In: *Journal of the Society of Industrial and Applied Mathematics* 5 (1957), S. 32–38

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

[NRB06] Neuhaus, Michel ; Riesen, Kaspar ; Bunke, Horst: Fast Suboptimal Algorithms for the Computation of Graph Edit Distance. In: Yeung, Dit-Yan (Hrsg.) ; Kwok, James T. (Hrsg.) ; Fred, Ana L. N. (Hrsg.) ; Roli, Fabio (Hrsg.) ; Ridder, Dick de (Hrsg.): *Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR)* Bd. 4109, Springer Verlag, 2006 (Lecture Notes in Computer Science), S. 163–172

[Obj06] Object Management Group (Hrsg.): *Meta Object Facility (MOF) Core Specification: OMG Available Specification Version 2.0*. formal/06-01-01. Needham, MA: Object Management Group, Januar 2006

[Obj07] Object Management Group (Hrsg.): *OMG Unified Modeling Language (OMG UML), Infrastructure, V. 2.1.2*. formal/2007-11-04. Needham, MA: Object Management Group, November 2007

[Ohs04] Ohst, Dirk: *Versionierungskonzepte mit Unterstützung für Differenz- und Mischwerkzeuge*. Siegen, Germany, Universität Siegen, Diss., 2004. `http://www.ub.uni-siegen.de/epub/diss/ohst.htm`

[OWK03] Ohst, Dirk ; Welle, Michael ; Kelter, Udo: Differences between versions of UML diagrams. In: *Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-11)*. New York, NY : ACM Press, 2003. – ISBN 1–58113–743–5, S. 227–236

[Pie08] Pietsch, Pit: *Optimierung des SiDiff-Algorithmus unter Ausnutzung modellspezifischer Eigenschaften und Strukturen*. Siegen, Germany, University of Siegen, Diplomarbeit, 2008

[Rau08] Rausch, Diana: *Ein Werkzeug zur Berechnung eines Matchings für UML-Klassendiagramme*. Bayreuth, Germany, University of Bayreuth, Zulassungsarbeit, 2008

[RFG+05] Reddy, Raghu ; France, Robert ; Ghosh, Sudipto ; Fleurey, Franck ; Baudry, Benoit: Model Composition - A Signature-Based Approach. In: *Aspect Oriented Modeling (AOM) Workshop, Montego*, 2005

[RNB07] Riesen, Kaspar ; Neuhaus, Michel ; Bunke, Horst: Bipartite graph matching for computing the edit distance of graphs. In: *Proceedings of the 6th IAPR-TC-15 International Conference on Graph-based Representations in Pattern Recognition*. Berlin, Heidelberg : Springer Verlag, 2007 (GbRPR'07). – ISBN 978–3–540–72902–0, S. 1–12

[RPB09] Rönnau, Sebastian ; Philipp, Geraint ; Borghoff, Uwe M.: Efficient change control of XML documents. In: *Proceedings of the 9th ACM Symposium on Document Engineering*. New York, NY, USA : ACM Press, 2009 (DocEng '09). – ISBN 978–1–60558–575–8, S. 3–12

[Rud05] Rudin, Walter: *Analysis*. 3. Auflage. München : Oldenbourg Wissenschaftsverlag GmbH, 2005

[RV08] Rivera, José E. ; Vallecillo, Antonio: Representing and Operating with Model Differences. In: *Proceedings of TOOLS Europe 2008, LNBIP 11*, Springer Verlag, 2008, S. 141–160

[SBPM09] Steinberg, Dave ; Budinsky, Frank ; Paternostro, Marcelo ; Merks, Ed: *EMF Eclipse Modeling Framework*. 2nd Edition. Upper Saddle River, NJ : Addison-Wesley, 2009 (The Eclipse Series)

[Sch03] Schneider, Christian: *CASE Tool Unterstützung für die Delta-basierte Replikation und Versionierung komplexer Objektstrukturen*, Technische Universität Braunschweig, Diplomarbeit, 2003

[SK83] Sankoff, David ; Kruskal, Joseph B.: *Time Warps, String Wdits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, MA : Addison-Wesley, 1983

[SZN04] Schneider, Christian ; Zündorf, Albert ; Niere, Jörg: CoObRA - a small step for development tools to collaborative environments. In: *Workshop on Directions in Software Engineering Environments; 26th international conference on software engineering (ICSE 2004)*. Scotland, 2004

[THRP05] Torsello, Andrea ; Hidovic-Rowe, Dzena ; Pelillo, Marcello: Polynomial-Time Metrics for Attributed Trees. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27 (2005), July, S. 1087–1099. – ISSN 0162–8828

[Tic84] Tichy, Walter F.: The String-to-String Correction Problem with Block Moves. In: *ACM Transactions on Computer Systems* 2 (1984), November, Nr. 4, S. 309–321

[Tre07] Treude, Christoph: *Einsatz multidimensionaler Suchstrukturen zur Optimierung der Bestimmung von Dokumentdifferenzen*. Siegen, Germany, University of Siegen, Diplomarbeit, 2007

[Tur04] Turau, Volker: *Algorithmische Graphentheorie*. 2. Auflage. München, Germany : Oldenbourg Verlag, 2004

[Uhr08] Uhrig, Sabrina: Matching Class Diagrams: With Estimated Costs Towards the Exact Solution? In: *Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models (CVSM 08)*. Leipzig, Germany : ACM Press, 2008, S. 7–12

[UW10] Uhrig, Sabrina ; Westfechtel, Bernhard: Strukturbezogener Vergleich von Modellversionen mit graphbasierten Optimierungsalgorithmen. In: Engels, Gregor (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Mayr, Heinrich C. (Hrsg.): *Modellierung 2010* Bd. 161. Klagenfurt, Österreich : GI, March 2010 (Lecture Notes in Informatics), S. 237–252

[WDC03] Wang, Yuan ; DeWitt, David J. ; Cai, Jin yi: X-Diff: An Effective Change Detection Algorithm for XML Documents. In: Dayal, Umeshwar (Hrsg.) ; Ramamritham, Krithi (Hrsg.) ; Vijayaraman, T. M. (Hrsg.): *Proceedings of the 19th International Conference on Data Engineering*, IEEE Computer Society Press, 2003. – ISBN 0–7803–7665–X, 519–530

[WHK] Wenzel, Sven ; Hutter, Hermann ; Kelter, Udo: Tracing Model Elements. In: *23rd IEEE International Conference on Software Maintenance (ICSM 2007), October 2-5, 2007, Paris, France*, IEEE Computer Society Press, S. 104–113

[WZJS94] Wang, J. T. L. ; Zhang, K. ; Jeong, K. ; Shasha, D.: A System for Approximate Tree Matching. In: *IEEE Transactions on Knowledge and Data Engineering* 6 (1994), August, S. 559–571. – ISSN 1041–4347

[XS05] Xing, Zhenchang ; Stroulia, Eleni: UMLDiff: an algorithm for object-oriented design differencing. In: Redmiles, David F. (Hrsg.) ; Ellman, Thomas (Hrsg.) ; Zisman, Andrea (Hrsg.): *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, ACM Press, 2005, S. 54–65

[XS06] Xing, Zhenchang ; Stroulia, Eleni: Refactoring Detection based on UMLDiff Change-Facts Queries. In: *13th Working Conference on Reverse Engineering (WCRE 2006), 23-27 October 2006, Benevento, Italy*, IEEE Computer Society Press, 2006, S. 263–274

[XS07] Xing, Zhenchang ; Stroulia, Eleni: Differencing logical UML models. In: *Automated Software Engineering* 14 (2007), June, S. 215–259. – ISSN 0928–8910

[YB07] Yujian, Li ; Bo, Liu: A Normalized Levenshtein Distance Metric. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29 (2007), S. 1091–1095. – ISSN 0162–8828

[Zha93] Zhang, Kaizhong: A New Editing based Distance between Unordered Labeled Trees. In: *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*. London, UK : Springer Verlag, 1993 (CPM '93). – ISBN 3–540–56764–X, S. 254–265

[ZS89] Zhang, Kaizhong ; Shasha, Dennis: Simple fast algorithms for the editing distance between trees and related problems. In: *SIAM Journal on Computing* 18 (1989), Nr. 6, S. 1245–1262

[ZSS92] Zhang, Kaizhong ; Statman, Rick ; Shasha, Dennis: On the editing distance between unordered labeled trees. In: *Information Processing Letters* 42 (1992), May, S. 133–139. – ISSN 0020–0190

[Zün01] Zündorf, Albert: Rigorous Object Oriented Software Development / University of Paderborn, Germany. 2001. – Forschungsbericht