Dokumentdifferenzen

Udo Kelter

22.03.2007

Zusammenfassung dieses Lehrmoduls

Berechnung und Anzeige von Differenzen zwischen Dokumenten sind grundlegende Funktionen von Versionsmanagementwerkzeugen, ferner das darauf aufbauende Mischen. Diese Grundfunktionen weisen für einzelne Dokumenttypen (textuelle bzw. graphische Dokumente, XML-Dateien usw.) erhebliche Unterschiede auf. Dieses Lehrmodul definiert einen Begriffsrahmen für den Problembereich; dabei wird versucht, die Unterschiede und Gemeinsamkeiten von Differenzbegriffen und -Werkzeugfunktionen bei den unterschiedlichen Dokumenttypen herauszuarbeiten. Ferner werden die wichtigsten Verfahren zur Bestimmung von Differenzen skizziert.

Vorausgesetzte Lehrmodule:

obligatorisch:
- Einführung in das Konfigurationsmanagement
- Einführung in CVS

Stoffumfang in Vorlesungsdoppelstunden: 6.0

Inhaltsverzeichnis

1 Motivation und Einordnung

Bei der Softwareentwicklung und in vielen anderen Zusammenhängen müssen Dateien – oder besser gesagt Dokumente – verglichen oder "abgeglichen" werden. In diesen Zusammenhängen kommen Begriffe wie "Differenz(en)", "Änderung", "Delta(s)", "Phantomdifferenzen", "inhaltliche Differenzen", "Differenzberechnung", "Korrespondenzen", "Konflikt", "gleiche Stelle", "Mischung" u.a. vor. Wenn man die Literatur incl. der Bedienungsanleitungen einiger wichtiger Konfigurationsmanagement-Werkzeuge durchsucht, findet man beliebig viele inkompatible oder unvollständige Definitionen der Grundbegriffe[1]. Teilweise sind diese Begriffsdifferenzen lediglich unterschiedliche Benennungen inhaltlich äquivalenter Begriffe – diese Differenzen lassen sich durch einheitliche Bezeichnungen leicht beheben –, teilweise liegen aber auch

- unterschiedliche Ziele bzw. Randbedingungen, warum Dokumente verglichen werden,
- dokumenttypspezifische Besonderheiten und
- unterschiedliche Betrachtungsebenen (abstrakt/konzeptionelle Ebene vs. äußere Darstellungen bis hin zu Bedienschnittstellen von Werkzeugen)

vor. Letztere machen eine saubere inhaltliche Trennung erforderlich.

Ziel dieses Lehrmoduls. Ein erstes Ziel besteht darin, eine im vorstehenden Sinne präzise Begriffswelt zu entwickeln. Diese Begriffswelt soll mehrere wichtige[2] Dokumenttypen abdecken:

[1] So reden viele Quellen von "den Unterschieden" bzw. "den Differenzen" (Plural!) zwischen zwei Dateien A und B, was 1 Differenz bzw. 1 Unterschied ist, bleibt undefiniert. Andere Quellen definieren "das Delta" zwischen zwei Dateien als "die Differenzen", es kommen aber auch "Deltas" vor.

[2] aber durchaus nicht alle denkbaren Dokumenttypen, z.B. LaTeX-Dokumente, HTML-Dokumente, Rechentabellen, graphische Textdokumente (aus WYSIWYG-Textprozessoren), Multimedia-Dokumente und viele weitere werden hier nicht betrachtet. Intendiert ist, alle für die Softwareentwicklung relevanten Dokumenttypen zu behandelt, aber diese Abgrenzung ist ziemlich unscharf.

- Textdokumente
- XML-Dokumente
- graphische Dokumente, namentlich UML-Diagramme und ähnliche bei der Softwareentwicklung eingesetzte Diagrammtypen
- Dateibäume

d.h. es sollen Gemeinsamkeiten und Besonderheiten der Differenzwerkzeuge für diese Dokumenttypen identifiziert werden.

Ein zweites, eher softwaretechnisches Ziel besteht darin, auf Basis dieser Begriffswelt

- Funktionen zu identifizieren, die in verschiedenen Kontexten als *wiederverwendbare Module* realisiert werden können,
- Differenzwerkzeuge für *unterschiedliche Dokumenttypen konsistent* zu konzipieren; dies beinhaltet auch Dokumenttypen, die *mehrere externe Darstellungen* (z.B. sowohl graphische als auch textuelle) haben;
- Anforderungen an bzw. Bewertungskriterien für Differenzwerkzeuge zu formulieren

1.1 Einsatzgebiete von Differenzen in der Praxis

In diesem Abschnitt skizzieren wir das Anwendungsspektrum von Differenzen. Warum sollte man sich überhaupt mit Dokumentdifferenzen befassen? Die Gründe dafür kann man in folgende Klassen einteilen:

1. *Speicherplatzersparnis*:

   Wenn mehrere ähnliche Dokumente zu speichern sind (insb. nur geringfügig veränderte Revisionen eines Dokuments), kann man durch eine sog. "Delta"-Speicherung Platz sparen.

   Die Delta-Speicherung ist eine interne Optimierungsmaßnahme innerhalb von Dokumentverwaltungssystemen bzw. Repositories, die beim Zugriff von Applikationen (also Werkzeugen) auf die Dokumente i.d.R. völlig verborgen bleibt.

2. *Vergleich von Dokumenten*:

Hier geht man davon aus, daß zwei oder mehr (ähnliche) Dokumente vorhanden sind, deren "Unterschiede" von Interesse sind. Gründe für dieses Interesse sind:

- bei zeitlich aufeinanderfolgenden Revisionen eines Dokuments: was ist in dem Dokument verändert worden?
- bei parallel existierenden Varianten eines Dokuments: was sind die Gemeinsamkeiten[3] und Unterschiede?

3. *Mischung von Dokumenten*:
   Auch hier geht man von zwei (ähnlichen) Dokumenten aus. Die Absicht ist hier jedoch, ein drittes Dokument zu erzeugen, das alle "Besonderheiten" der beiden Basisdokumente oder eine individuell bestimmte Teilmenge dieser Besonderheiten enthält. Basis ist auch hier eine Darstellung der Unterschiede.

Dokumente werden im Rahmen der Softwareentwicklung sehr häufig deshalb verglichen, weil sie gemischt werden sollen, also eine gemeinsame Nachfolgeversion gebildet werden soll. Die reine Anzeige von Dokumentunterschieden (ohne Mischabsicht) ist ebenfalls wichtig und muß unabhängig von Mischungen korrekt definiert werden. Eine häufige Situation, in der Mischen nicht geplant und in einem gewissen Sinn sogar nicht möglich ist, ist der Vergleich zweier Versionen eines Dokuments in einer Revisionskette. Die jüngere Revision ist definitionsgemäß Nachfolger der anderen, und ein Mischen der beiden Dokumente ist eher zu verstehen als ein partielles Aufheben bestimmter Änderungen, durch die die ältere Revision in die jüngere transformiert wurde (im Gegensatz dazu unterstellt der intuitive Begriff Mischen, daß beide Ausgangsdokumente in parallelen Entwicklungszweigen liegen und eigenständige Änderungen enthalten).

[3] Die Suche nach Gemeinsamkeiten entspricht im speziellen Fall von Quellprogrammen der Suche nach Clones, die im Kontext von Software-Refactoring auftritt mit dem Ziel, redundanten Code zu eliminieren.

1.2 Arten von Differenzwerkzeugen

Hier werden nur die gängigsten Werkzeugtypen beschrieben, nicht alle theoretisch denkbaren. Die beschriebenen Werkzeuge können sowohl autark aufrufbar sein als auch in eine komplexe Umgebung eingebunden sein.

1.2.1 Batch-Werkzeuge

Hierunter fallen diverse UNIX-Standardwerkzeuge und generell alle nicht-interaktiven Werkzeuge mit einer Kommandozeilen-Schnittstelle. Funktional können drei Gruppen unterschieden werden:

1. **Differenzberechnung**: Als Argumente werden 2 (manchmal mehr) Dokumente übergeben. Die Dokumente können direkt Dateien entsprechen oder aus Versionsarchiven oder anderen Datenverwaltungssystemen stammen. Weitere Argumente beeinflussen die Algorithmen, die die Differenz berechnen.

   Hauptergebnis ist eine textuelle Darstellung der Differenz (i.d.R. als Dateiinhalt), die oft als **Edit-Skript** bezeichnet wird. Ein Edit-Skript ist im einfachsten Fall eine textuell notierte Sequenz von Operationen, die das erste Dokument in das zweite umformen.

2. **Patch-Werkzeuge**: Als Argument werden ein Dokument und ein oder mehrere Edit-Skripte übergeben. Ergebnis ist ein geändertes Dokument, in dem die Änderungen gemäß den Edit-Skripten vorgenommen worden sind.

3. **nichtinteraktive Mischwerkzeuge**: Als Argumente werden 3 Dokumente übergeben, von denen eines die gemeinsame Vorgängerversion der beiden anderen ist. Ergebnis ist ein geändertes Dokument auf Basis der gemeinsamen Vorgängerversion, in dem alle Änderungen nachvollzogen worden sind, die in den beiden Nachfolgeversionen auftraten. Änderungen, die in Konflikt stehen, werden besonders behandelt. CVS beispielsweise beinhaltet eine derartige nichtinteraktive Mischfunktion.

1.2.2 Interaktive Anzeige- und Mischwerkzeuge

Diese Werkzeuge basieren intern stets auf einer nichtinteraktiven Differenzberechnung; die Argumente sind daher die gleichen wie bei Batch-Werkzeugen zur Differenzberechnung. Die Interaktivität betrifft vor allem die Steuerung der Anzeige, also den angezeigten Ausschnitt der beiden Dokumente oder die Darstellung der Unterschiede.

1. **Differenzanzeigewerkzeuge:** zeigen eine vergleichende textuelle oder graphische Darstellung zweier Dokumente D1 und D2 an. Die Dokumente selbst können i.d.R. nicht verändert werden.

2. **Interaktive Mischwerkzeuge:** Diese stellen erweiterte Versionen interaktiver Differenzanzeigewerkzeuge dar. Die Erweiterung besteht darin, daß ein drittes Dokument erzeugt werden kann, das alle gemeinsamen Teile beider Dokumente und individuell ausgewählte spezielle Teile der einzelnen Dokumente enthält.

   Manche Mischwerkzeuge haben ein weiteres Fenster, in dem das Mischergebnis direkt angezeigt wird, ggf. mit Markierungen, aus denen die Herkunft der Teile ersichtlich ist.

   Manche Mischwerkzeuge erlauben es darüber hinaus, das Mischergebnis sofort weiterzubearbeiten, d.h. sie bieten ähnliche Funktionen wie ein normaler Dokumenteditor an.

   Die meisten Mischwerkzeuge unterstützen dies aber nicht, hier kann das Mischergebnis nur als Datei (oder Dokument in einem anderen Datenverwaltungssystem) erzeugt werden, d.h. es muß bei Bedarf nach dem Mischen mit einem separaten Werkzeug angezeigt und weiterbearbeitet werden.

Differenz-Systemfunktionen. Potentiell mehrfach nutzbare interne Funktionen der vorstehenden Werkzeuge sind:

1. die Differenzberechnung: i.w. analog zu Batch-Werkzeugen zur Differenzberechnung, aber mit einer Programmschnittstelle (statt einer Kommandozeilen-Schnittstelle)
2. Anwenden eines Edit-Skripts: i.w. analog zu Patch-Werkzeugen, aber mit einer Programmschnittstelle

1.3 Formen der Anzeige von Differenzen

Differenzen können zu einem textuell in Form eines Edit-Skripts angezeigt werden. Ein Beispiel hierfür ist die Ausgabe des UNIX-Werkzeugs `diff`. Diese Form der Anzeige ist aber wenig anschaulich. Für die (halb-) graphische vergleichende Darstellung gibt es zwei Hauptformen, die integrierte Paralleldarstellung und die Darstellung in einem Vereinigungsdokument, sowie weitere Zwitterformen und Varianten.

1.3.1 Integrierte Paralleldarstellung

Bei allen Paralleldarstellungen werden beide Dokumente komplett in Fenstern oder Unterfenstern dargestellt und horizontal oder vertikal ausgerichtet einander gegenübergestellt.

In allen Fällen werden die "*gemeinsamen Teile*" bzw. der "Durchschnitt"[4] in einer unauffälligen Farbe dargestellt, während die "*speziellen Teile*" bunt dargestellt werden.

Unterschiede bestehen darin, wie *Korrespondenzen* zwischen gemeinsamen Teilen dargestellt werden. Bei der (integrierten) **Paralleldarstellung mit "Dehnstellen"** werden

- gemeinsame Komponenten nebeneinanderstehend angezeigt, Korrespondenzen also durch Layout sichtbar gemacht
- die nur in D1 bzw. D2 vorhandenen Komponenten in ihrer Spalte geeignet markiert angezeigt, in der Spalte gegenüber steht eine "Dehnstelle", d.h. dort wird in dem Dokument eigentlich nicht vorhandener Leerraum (Leerzeilen) angezeigt,
- veränderte Komponenten (bzw. an derselben Stelle vorhandene unterschiedliche Komponenten) nebeneinander angezeigt, wie gesagt optisch entsprechend markiert; die kürzere Komponente wird durch Leerraum verlängert, bis sie genauso lang wie die längere ist.

Weil Korrespondenzen durch räumliche Anordnung dargestellt werden, kann man in dieser Darstellungsform prinzipiell keine Verschiebungen darstellen.

[4] Diese Begriffe werden später genau untersucht.

Eine Variante der Paralleldarstellung, die z.B. in Eclipse für Textdokumente eingesetzt wird, besteht darin, Dateiabschnitte durch senkrechte Balken zu markieren und eine Korrespondenz zwischen zwei Abschnitten durch eine **Verbindungslinie** darzustellen. Dehnstellen werden so vermieden, d.h. die beiden Basisdokumente werden in dieser Hinsicht nicht verfälscht. Korrespondierende Komponenten liegen hier i.a. nicht mehr exakt gegenüber.

Verschiebungen können hier dargestellt werden; in diesem Fall überkreuzen sich die Verbindungslinien. Eine größere Anzahl von überlappenden Verschiebungen ist unübersichtlich, d.h. das Anzeigeverfahren stößt hier rasch an Grenzen[5]. Trotz dieser quantitativen Einschränkung ist das Anzeigeverfahren mit Verbindungslinien prinzipiell leistungsfähiger als das Anzeigeverfahren mit Dehnstellen, bei dem Verschiebungen prinzipiell nicht dargestellt werden können.

Das Hauptproblem aller integrierten Paralleldarstellungen liegt darin, daß man ein eigenes Anzeigesystem implementieren muß, das die Dokumente in ihrer normalen Form darstellen kann und zusätzlich den Leerraum, Verbindungslinien, ggf. eine Liste der lokalen Unterschiede, weitere Steuerungselemente usw. Dies ist bei vielen Dokumenttypen aus Aufwandsgründen unmöglich.

1.3.2 Interaktive Liste lokaler Unterschiede

Sofern für einen bestimmten Dokumenttyp keine integrierte Paralleldarstellung implementiert werden kann, aber ein von außen ansteuerbares Anzeigewerkzeug verfügbar ist, bietet sich folgende Lösung an, die man als Zwitter zwischen Edit-Skript und Paralleldarstellung ansehen kann.

In einem Steuerfenster wird eine **klickbare Liste lokaler Unterschiede** angezeigt. Die Liste sollte jeden lokalen Unterschied mit einem Kurztext beschreiben. Wenn man einen lokalen Unterschied anklickt, werden die beiden Dokumente in jeweils einem Editorfenster des vorhandenen Anzeigewerkzeugs angezeigt und möglichst auf die je-

[5] Dies ist aber normal, alle Anzeigeverfahren liefern bei sehr umfangreichen Unterschieden zwischen den Dokumenten keine übersichtlichen Darstellungen mehr.

weils involvierten Dokumentteile positioniert. Man sieht die Differenz hier nicht insgesamt, sondern nur stückweise. Die Differenzanzeige ist hier prinzipbedingt interaktiv.

Ein prinzipielles Problem dieser Anzeigeform liegt darin, daß ein Betrachter selbst durch Vergleich der beiden Fensterinhalte herausfinden muß, worin die Änderung in Detail besteht. Hierbei kann er auf mehrere Arten unterstützt werden:

- durch die textuelle Kurzbeschreibung in der Liste;
- durch Anzeige eines entsprechend kleinem Ausschnitts des Gesamtdokuments und Positionieren derart, daß in die lokale Differenz involvierten Dokumentelemente in der Mitte des Fensters stehen und möglichst wenige weitere Dokumentteile zu sehen sind;
- durch Einfärben oder anderes Hervorheben der betroffenen Dokumentelemente.

Die beiden letzten Punkte hängen stark davon ab, wie das Anzeigewerkzeug von außen steuerbar ist.

Ein Vorteil dieses Ansatzes liegt darin, daß die beiden Dokumentversionen sofort editiert werden können, wenn die Anzeigewerkzeuge zugleich Editoren sind.

1.3.3 Darstellung in einem Vereinigungsdokument

Bei der Anzeige in einem Vereinigungsdokument werden

- die gemeinsamen Teile nur einmal,
- veränderte Komponenten in beiden Varianten hinter- oder untereinander (oder nebeneinander bei graphischen Dokumenten) mit geeigneten Markierungen angezeigt;
- nur in D1 oder D2 vorhandene Komponenten mit geeigneten Markierungen oder farblichen Codierungen angezeigt

Hauptvorteil der Vereinigungsdarstellung ist der geringere Platzbedarf und das bessere Erkennen des gemeinsamen Kontextes eines lokalen Unterschieds. Auf Basis dieser Darstellung kann man auch bessere Mischwerkzeuge realisieren als auf Basis der Paralleldarstellung.

Das angezeigte Dokument ist ein völlig neues Dokument, es ist das Ergebnis einer (vollautomatischen) Mischung der beiden Ausgangsdokumente. Dies motiviert die Bezeichnung **Vereinigungsdokument**. Für diese Anzeigeform wird also ein vollautomatisches Mischverfahren benötigt. Neben dem Mischen des konzeptuellen Inhalts des Dokuments muß auch ein neues Layout gefunden werden; u.a. bei graphartigen Dokumenten ist es nicht einfach, ein gutes neues Layout zu finden.

Weiterhin hat das Vereinigungsdokument in vielen Fällen eine etwas andere Struktur als die Basisdokumente, oft gelten für das Vereinigungsdokument nur abgeschwächte Konsistenzkriterien. Infolgedessen können die Anzeige- und Editierwerkzeuge für den Basisdokumenttyp das Vereinigungsdokument nicht anzeigen. Man muß dann ein eigenes Anzeigewerkzeug realisieren; dies ist das Haupthindernis für diese Darstellungsform von Differenzen.

Verschiebungen sind im Vereinigungsdokument nicht darstellbar.

2 Grundbegriffe

2.1 Definitionen des Begriffs Differenz

In diesem Abschnitt motivieren und definieren wir den Begriff Differenz in mehreren Varianten und weitere damit zusammenhängende Grundbegriffe. Wir beginnen mit einer sehr allgemeinen, informellen Definition einer (Dokument-) Differenz:

allgemeine informelle Definition:

Eine **Differenz** ist *eine* Darstellung der Unterschiede zwischen zwei Dokumenten.

Falsche Begriffsübertragungen. Schon bei dieser informellen Definition besteht die große Gefahr einer falschen Begriffsübertragung. Der Begriff Differenz stammt aus der Mathematik und bezeichnet dort meistens das Ergebnis einer **Subtraktion**. Sehr wichtige Aspekte des mathematischen Begriffs Differenz sind *nicht* auf Dokumentdifferenzen übertragbar; dies führt regelmäßig zu Mißverständnissen:

1. *Das Begriffspaar Subtraktion-Differenz ist nicht auf Dokumente übertragbar.*

   Die Subtraktion ist eine Funktion im mathematischen Sinne, sie ordnet zwei Zahlen (oder ähnlichen Argumenten) einen eindeutigen Funktionswert zu. Im Gegensatz dazu ist die Differenz zwischen zwei gegebenen Dokumenten *nicht eindeutig definiert* (Details hierzu später). Daher redet man auch nicht von der Subtraktion eines Dokuments von einem anderen, sondern vom **Vergleich** von Dokumenten. *Eine* (nicht "die") Differenz zwischen zwei Dokumenten ist daher nur *eine von mehreren denkbaren Darstellungen* der Unterschiede zwischen den Dokumenten.

   In scheinbarem Gegensatz hierzu steht die praktische Erfahrung, daß diverse Differenzberechnungswerkzeuge immer genau eine Differenz als Ergebnis eines Vergleichs liefern. Dies beruht aber lediglich darauf, daß die Werkzeuge unter der Vielzahl möglicher Differenzen eine besonders schöne heraussuchen und dem Nutzer den Rest verschweigen. Bestenfalls gibt es Optimierungsvorgaben, so hat das

Unix-Werkzeug `diff` die Option `--minimal`: "Try hard to find a smaller set of changes"; auf Kosten höheren Rechenaufwands wird nach einer schöneren Differenz gesucht.

2. Differenzbegriffe aus der Mengenlehre (einfache Mengendifferenz, symmetrische Differenz) können nur sehr eingeschränkt übertragen werden; dies werden wir später ausführlich behandeln.
3. Differenzen entstehen meist, aber nicht immer als Ergebnis des Vergleichs von Dokumenten. Sie entstehen auch mittels anderer Verfahren, z.B. durch Beobachtung von Editiervorgängen oder Kombinieren vorhandener Differenzen. Der Begriff Differenz darf sich daher nicht auf ein Berechungsverfahren analog zur Subtraktion beziehen.
4. Dokumentdifferenzen werden auch zwischen 3 und mehr Dokumenten gebildet. Der Vergleich von 2 Dokumenten ist teilweise nur ein Sonderfall allgemeinerer Vergleiche.

Aspekte des Differenzbegriffs. Ziel dieses Abschnitts ist nur die genaue Analyse der Varianten des Begriffs Dokumentdifferenz. Wir konzentrieren uns in diesem Abschnitt auf

- den *konzeptuellen Inhalt einer Differenz*, also durch welche Angaben (letztlich durch welche Daten) die Unterschiede zwischen zwei Dokumenten dargestellt werden, und
- wie diese Angaben zu interpretieren sind, also *die Bedeutung einer gegebenen Differenz*

Der Begriff Differenz hat weitere Aspekte, die wir später separat betrachten werden und die nicht mit der Frage nach der Definition des konzeptuellen Inhalts vermischt werden dürfen:

1. die **"physische" Repräsentation** (Implementierung) einer Differenz, entweder persistent als Inhalt einer (XML-) Datei, Datenbank usw., oder transient in Form von Laufzeitobjekten.
2. die externe, oft **graphische Darstellung** einer Differenz; diese basiert meist auf der Standarddarstellung des Dokumenttyps und wurde schon oben behandelt

3. die **Berechnung einer Differenz**. Die Bedeutungserklärung von Differenzen darf sich nicht auf die Berechnung einer Differenz durch Vergleich zweier Dokumente beziehen. Differenzen können nicht nur durch Vergleich zweier Dokumente gewonnen werden, sondern auch durch Protokollierung von Editiervorgängen und durch Mischen, Bearbeiten usw. vorhandener Differenzen.

   Hinzu kommt, daß man leicht in die Versuchung kommt, Differenzen so zu definieren, daß sie bestimmte Qualitätsanforderungen erfüllen. Die Suche nach einer optimalen (oder guten) Differenz, die die Unterschiede zwischen zwei gegebenen Dokumenten darstellt, ist aber Sache des Berechnungsverfahrens, keine Frage der Definition einer Differenz.
4. das **Mischen** von Dokumenten (auch als Verschmelzen, Abgleichen, Vereinigen u.a. bezeichnet). Das Mischen basiert immer auf einer Differenz zwischen zwei oder mehr Dokumenten. Ziel des Mischens ist, ein neues Dokument zu erzeugen, das die "Besonderheiten" der zu mischenden Dokumente "vereinigt". Dies geht wesentlich über die Ziele der Differenzdarstellung hinaus.

   Beim Mischen können sogenannte Mischkonflikte auftreten. Der Begriff Mischkonflikt tritt nur im Kontext des Mischens auf, ist also nicht grundlegend für den Begriff Differenz.

In den beiden nächsten Abschnitten diskutieren wir zwei Hauptvarianten, wie die einleitende informelle Definition des Begriffs Differenz konkretisiert werden kann:

- asymmetrische Differenzen bzw. den operationalen Definitionsansatz
- symmetrische Differenzen bzw. den mengenorientierten Definitionsansatz

2.2 Asymmetrische Differenzen

Definition:

Eine **asymmetrische Differenz** *von* einem Dokument D1 *nach* einem Dokument D2 ist eine Transformation (bzw. Transformationsvorschrift), durch die D1 in D2 transformiert werden kann[6].

Eine **Transformation** besteht aus einer Sequenz von Operationsaufrufen, in denen aufzurufende Operationen gemäß einem Editierdatentyp (s.u.) und passende Parameter angegeben werden.

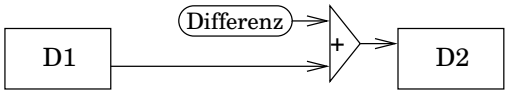

Abbildung 1: Differenz beim operationalen Ansatz

Ein konkretes Beispiel für eine Transformationsvorschrift ist die Ausgabe des UNIX-Werkzeugs `diff`. Die Ausgabe enthält in einer kryptischen Syntax Anweisungen mit Bedeutungen wie "lösche Zeilen 6 - 8" oder "füge hinter Zeile 10 folgenden Text ein".

Wegen der herausragende Rolle, die bei dieser Definition die Transformation des Dokuments spielt, kann man auch von einem **operationalen** oder **transformationellen** Definitionsansatz reden.

Man kann mit einer asymmetrischen Differenz von D1 nach D2 nicht wieder das Dokument D2 nach D1 zurücktransformieren, hierzu wird eine andere Transformation benötigt[7].

Asymmetrische Differenzen sind primär dazu gedacht, ein Dokument D1 mit Hilfe der Differenz in ein Dokument D2 zu transformieren

[6] Mit Dokumenten sind hier immer Dokumentzustände gemeint, keine Variablen. Ob die beiden Zustände nur nacheinander in einer Variablen existieren oder gleichzeitig - die Transformation erzeugt also einen komplett neuen Dokumentzustand -, ist hier unerheblich.

[7] Oft werden die asymmetrischen Differenzen für beide Richtungen gemeinsam dargestellt, diese gemeinsame Darstellung ist trotzdem keine symmetrische Differenz.

(s. Bild 1). Dies entspricht exakt der Denkweise bei Patch-Werkzeugen und der internen Delta-Speicherung in Dokument-Repositorys. Daher müßte man eigentlich eher von einem Summanden reden; der mathematischer Begriff Differenz unterstellt, daß das zweite Dokument vorher existiert bzw. daß die Differenz das Ergebnis einer Funktion und nicht ein Argument ist.

Editieroperationen. Asymmetrische Differenzen beschreiben Transformationen zwischen Dokumenten. Wir unterstellen generell, daß Dokumente typisiert sind, daß alle betrachteten Dokumente den gleichen Typ haben und daß Dokumente mittels bestimmter typspezifischer Operationen verändert ("editiert") werden können.

Die Editieroperationen fassen wir als einen abstrakten Datentyp[8] auf und bezeichnen diesen Datentyp als den **Editierdatentyp**. Die syntaktische Darstellung der Operationen ist hier irrelevant.

Editierdatentypen enthalten stets grundlegende Operationen zum Einfügen, Löschen, Ändern usw. von Dokumentelementen, die sich direkt aus der Grundstruktur der Dokumente ergeben. Hierdurch sind letztlich alle Änderungen, die mit den Editoren dieses Dokumenttyps vorgenommen werden, nachvollziehbar.

Zu den grundlegenden Editierdatentypen kann man optionale Operationen hinzunehmen, z.B. Verschiebungen von Dokumentteilen innerhalb eines Dokuments oder die Umbenennung einer Variablen in einem Quellprogramm. Derartige Operationen sind nicht unbedingt

[8]Der Vollständigkeit halber: Unter einem **abstrakten Datentypen** (ADT) verstehen wir eine Schnittstellenspezifikation, die die Namen, Signaturen und Wirkung von Operationen, die auf Instanzen des Datentyps arbeiten, definiert. Nicht gemeint ist eine konkrete Implementierung inkl. konkreter Datenstrukturen, die diese Schnittstelle implementiert. Wir abstrahieren auch von konkreten Programmiersprachen; in Sprachen wie C, Modula-2, Ada oder Java liegen jeweils eigene Sprachkonstrukte vor, mit denen man abstrakte ("opake") Datentypen realisieren kann.

Die Operationen des ADT treten in diversen Formaten und Codierungen in den Edit-Skripten auf, die Differenzwerkzeuge erzeugen oder benutzen. Diese Edit-Skripten bzw. die darin enthaltenen Editierkommandos werden von den Differenzwerkzeugen interpretiert. In welcher Sprache die Differenzwerkzeuge geschrieben sind, interessiert nicht.

notwendig, weil man den Effekt auch mit entsprechenden Löschungen und Einfügungen erzielen könnte. Sie sind aber kompaktere Darstellungen der Unterschiede zweier Dokumente und ermöglichen somit - je nach Qualitätsbegriff - qualitativ bessere Differenzen.

Zu einem Dokumenttyp kann es aufgrund der optionalen Operationen unterschiedliche Editierdatentypen geben: beispielsweise kann man Verschiebungen in den Editierdatentyp aufnehmen oder nicht. Dem Vorteil, qualitativ bessere Differenzen zu ermöglichen, stehen zwei Nachteile gegenüber: der Aufwand für die Berechnung von Differenzen steigt erheblich an, und komplexe Editieroperationen sind optisch schlecht darstellbar. Die Wahl des Editierdatentyps ist nicht trivial und wird in Abschnitt 3 noch im Detail diskutiert.

Bei asymmetrischen Differenzen darf der Editierdatentyp beliebig komplexe Operationen beinhalten.

Nichteindeutigkeit der Berechnung von asymmetrischen Differenzen. Für zwei gegebene Dokumente D1 und D2 kann es signifikant verschiedene Transformationsvorschriften geben, die D1 nach D2 transformieren. Die Berechnung "der" Differenz kann also zu unterschiedlichen korrekten Resultaten führen.

Ein erste Quelle für Unterschiede sind sinnlose Operationen in der Transformationsvorschrift (Beispiel: es wird etwas gelöscht und dann wieder eingefügt). Selbst wenn man dies ausschließt, ist die Berechnung nicht eindeutig. Als Beispiel betrachten wir in Bild 2 zwei Textdokumente D1 und D2. Wir unterstellen einen Editierdatentyp, bei dem man die i-te Zeile einfügen oder löschen kann. Mit diesen Operatoren kann man D1 in D2 umformen, indem man die 6. Zeile (oder alternativ die 7. Zeile) löscht, hinter der 4. (oder 5.) Zeile von D1 die Zeile "e" einfügt und dann hinter der 1. (oder 2.) Zeile von D1 die Zeile "b" einfügt. Insgesamt erhalten wir 8 Kombinationen bzw. Transformationsvorschriften, die alle nicht weiter vereinfacht werden können.

Ursache für die Nichteindeutigkeit der Berechnung waren die Dubletten in unserem Beispiel. Eine weitere potentielle Ursache sind komplexe Editierdatentypen. Dort können formell verschiedene, in-

lfd.Nr.	D1	D2
1	a	a
2	b	b
3	c	b
4	d	c
5	e	d
6	f	e
7	f	e
8		f

Abbildung 2: Beispiel zu vergleichender Textdokumente

haltlich aber äquivalente Transformationsvorschriften auftreten. Wenn wir beispielsweise bei einem Textdokument den Editierdatentyp um einen Operator erweitern, der in Zeile i an Position j das dort vorhandene Zeichen durch ein anderes ersetzt, könnten wir Zeilen umbauen anstatt sie komplett zu löschen und neu zu erzeugen.

2.3 Symmetrische Differenzen

Sehr häufig wird ein Dokument - ggf. vereinfachend - als *Menge* von Komponenten angesehen. Dieser mengenorientierte Ansatz liefert die konzeptuelle Basis für Mischwerkzeuge und die meisten Differenzanzeigewerkzeuge. Er wird sehr oft (und oft unbewußt) eingesetzt, vor allem bei einer informellen Betrachtung von Differenzen. Die Definitionen, die sich bei diesem Ansatz ergeben, variieren allerdings nicht unerheblich je nach dem Grad der Vereinfachung der Dokumentstruktur. Wir stellen daher i.f. mehrere Varianten vor und beginnen mit der stärksten Vereinfachung der Dokumentstruktur.

2.3.1 Dokumentstruktur: Menge

Wenn man ein Dokument als echte Menge von Komponenten auffaßt, kann man den aus der *Mengenlehre* gut bekannten Begriff der symmetrischen Differenz auf Dokumente übertragen.

Gegeben seien zwei Dokumente D1 und D2, die auch als **Basisdokumente** bezeichnet werden. Die Differenz zwischen D1 und D2 ist dann definiert als die Menge der Komponenten, die nicht im Durchschnitt der beiden Komponentenmengen enthalten sind.

Etwas anders betrachtet kann man eine Differenz zwischen D1 und D2 definieren als die zwei Mengen von Komponenten, die nur in D1 bzw. nur in D2 auftreten. Wir erhalten folgende

mengenbasierte Definition:

Eine **symmetrische Differenz** *zwischen* zwei Dokumenten D1 und D2, deren Dokumentstruktur eine Menge ist, besteht aus je einer Teilmenge

$$SK1 \subseteq D1 \text{ bzw. } SK2 \subseteq D2$$

die die speziellen Komponenten von D1 bzw. D2 angeben. SK1 und SK2, SK1 und D2 sowie SK2 und D1 müssen disjunkt sein.

Abbildung 3: Differenz beim mengenorientierten Ansatz

Die vorstehende Definition erlaubt keine ungünstige Differenzen, bei denen SK1 und SK2 nicht disjunkt und daher unnötig groß sind. Daher gibt es für zwei gegebene Dokumente D1 und D2 nur genau eine symmetrische Differenz, worin SK1 = D1-D2 und SK2 = D2-D1 ("-" steht hier für Mengensubtraktion). Diese Differenz ist zugleich optimal in dem Sinne, daß SK1 und SK2 möglichst klein sind.

Differenzen sind hier inhärent *symmetrisch*, weil keines der Dokumente eine besondere Rolle spielt und vom gemeinsamen Durchschnitt der beiden Basisdokumente ausgegangen wird.

Die aus der Mengenlehre übertragene Definition ist intuitiv naheliegend, sie eignet sich vor allem für eine sehr informelle Betrachtung von

Differenzen. Man kann sie allerdings fast nie wirklich anwenden, weil fast alle Dokumente Dubletten enthalten können, also Multimengen sind, und fast immer sogar eine nichttriviale Struktur haben – selbst simple Textdokumente müssen als Folge (also geordnete Multimenge) von Textzeilen aufgefaßt werden. Die Betrachtung solcher Dokumente als *Menge* von Komponenten vereinfacht zu stark. Bei strukturierten Dokumenten reicht es nicht aus, nur die speziellen Komponenten als solche zu kennen, man muß auch wissen, wo im Dokument sie liegen. Bei Dokumenten mit Dubletten ist nicht automatisch klar, welche Dubletten auf beiden Seiten korrespondieren sollen.

2.3.2 Dokumentstruktur: Multimenge

Die an die Mengenlehre angelehnte Definition kann man beim Vorhandensein von Dubletten retten, indem man sie um eine explizite Angabe erweitert, welche Dokumentkomponenten als korrespondierend angesehen werden sollen[9].

Eine Multimenge ist eine Menge, die das gleiche Element mehrfach enthalten kann. Üblicherweise gibt man bei einer Multimenge zu jedem enthaltenen Element nur die Anzahl seines Auftretens in der Menge an, die Dubletten sind völlig ununterscheidbar. Für unsere Zwecke ist dieses Vorgehen nicht brauchbar, weil die Komponenten von Dokumenten immer einzeln unterscheidbar sind und die multimengenbasierte Definition von Differenzen die Vorstufe zu weiteren Definitionen liefert, die Strukturen auf der Menge definieren.

Wir fassen daher den Begriff Multimenge geringfügig anders üblich auf und unterstellen irgendein "verstecktes" Merkmal, z.B. eine Numerierung, das bei Bedarf einzelne Elemente der Menge identifiziert, aber im Sinne der Gleichheit nicht zu ihrem Inhalt zählt. Die Identifikationsmerkmale spielen auch für die Gleichheit zweier Multimengen keine Rolle, d.h. zwei Multimengen, die sich nur bzgl. der Identifikationsmerkmale der Elemente unterscheiden, sind gleich.

[9] Allerdings leider nicht bei Textdateien, denn dies sind geordnete Multimengen. Unter Multimengen sind in diesem Abschnitt unsortierte, strukturlose Multimengen zu verstehen.

multimengenbasierte Definition:

Eine **Differenz** zwischen zwei Dokumenten D1 und D2, deren Dokumentstruktur eine Multimenge ist, besteht aus folgenden Angaben:

1. einer Menge von Korrespondenzen.
   Eine **Korrespondenz** ist ein Paar von je einer Komponente von D1 bzw. D2, die inhaltlich gleich sind und als einander entsprechend angesehen werden. Jede Komponente kann in höchstens einer Korrespondenz auftreten. Eine Komponente, die in einer Korrespondenz auftritt, wird auch als **gemeinsame** Komponente bezeichnet.

   Die Menge aller gemeinsamen Komponenten wird in Bild 4 als G(D1,D2) bezeichnet. Sie wird auch als **Durchschnitt** bezeichnet. Sie stellt selbst wieder eine Multimenge dar[10].

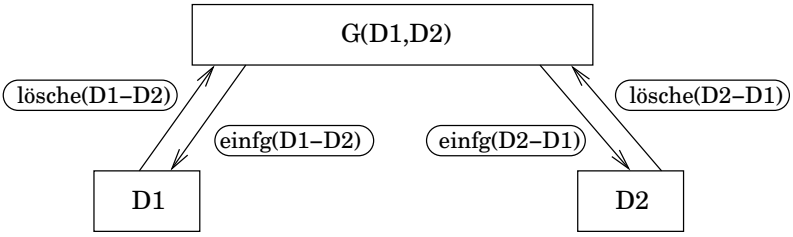

Abbildung 4: Symmetrische Differenz bei Multimengen

2. je einer **Einfüge-Transformation** pro Dokument, die ausgehend vom Durchschnitt das jeweilige Dokument D1 bzw. D2 rekonstruiert[11]. In Bild 4 werden diese Transformationen als einfg(D1-D2) bzw. einfg(D2-D1) bezeichnet. Jede Ein-

[10] In dieser Multimenge müssen die Dokumentkomponenten natürlich auch Identifikationsmerkmale haben. Hierzu können die Identifikationsmerkmale entweder von D1 oder von D2 übernommen werden. In beiden Fällen sind die beiden Multimengen in dem Sinne identisch, daß in beiden die gleichen Elemente mit der gleichen Häufigkeit enthalten sind.

[11] Wenn man will, kann man diese Einfüge-Transformationen auch als asymmetrische Differenzen vom Durchschnitt nach D1 bzw. D2 ansehen.

füge-Transformation enthält ausschließlich Einfügungen, keine Änderungen oder Löschungen. Der Editierdatentyp für Multimengen braucht hierzu nur eine einzige Operation "füge Element mit Wert X ein" zu enthalten[12].

Die durch diese Transformationsvorschriften eingefügten Komponenten werden als **spezielle Komponenten** von D1 bzw. D2 bezeichnet.

In Bild 4 sind zusätzlich inverse Transformationen lösche(D2-D1) bzw. lösche(D1-D2) eingetragen. Diese sind bei Multimengen leicht bestimmbar, sind aber nicht Teil der Differenz.

Die symmetrische Differenz zweier Multimengen ist eindeutig definiert, wenn man den Durchschnitt maximal wählt und Dubletten nicht unterscheidet, also deren Identifikationsmerkmale nicht mit betrachtet. Sobald man Dubletten unterscheidet, ist die Differenz nicht mehr eindeutig definiert, allerdings hat man dann der Multimenge eine zusätzliche Struktur gegeben; diesen Fall betrachten wir im nächsten Abschnitt.

Die vorstehende Definition für Multimengen behebt zwar das Problem der Zuordnung von Doubletten in Dokumentkomponenten, ist aber immer noch insofern (zu sehr) vereinfachend, als sie davon ausgeht, daß die Struktur dieser beiden Komponentenmengen keine Rolle spielt. Die Struktur der Komponentenmengen kann aber i.a. nicht vernachlässigt werden. Selbst ein so einfacher Dokumenttyp wie Text kann nicht als (unstrukturierte) Multimenge von Zeilen aufgefaßt werden.

Der Hauptnutzen dieser Definition liegt darin, daß sie beschreibt, was sehr viele Algorithmen zur Berechnung von Differenzen im Kern zunächst liefern, nämlich eine Tabelle von Korrespondenzen.

[12]Die ggf. für Dubletten erforderlichen Identifikationsmerkmale können dabei beliebig neu vergeben werden. Wie schon erwähnt sind die Identifikationsmerkmale unerheblich im Sinne der Gleichheit zweier Multimengen.

2.3.3 Strukturierte Dokumente

Bei vielen Dokumenttypen kann ein Dokument aufgefaßt werden als ein gerichteter Graph mit attributierten Knoten und Kanten oder noch etwas allgemeiner als eine Menge von Knoten und eine Struktur, die die Knotenmenge verwaltet.

Die Knoten und deren Attribute bilden dabei Dokumentkomponenten im Sinne der Multimengen-Differenzdefinition. Zwei Knoten sind Dubletten voneinander, wenn sie die gleichen Attribute haben und die Attribute paarweise gleiche Werte.

Die Kanten eines Graphen, ein Array oder weitere Typkonstruktoren bilden eine Struktur für die Multimenge der Knoten. Häufige Sonderfälle sind u.a. Sequenzen und Bäume.

Generell können für einen Dokumenttyp weitgehende Konsistenzkriterien definiert sein, die die erlaubten Strukturen einschränken. Diese Kriterien müssen natürlich von den Operationen des Editierdatentyps beachtet werden. Hieraus ergibt sich ferner, daß die Operationen des Editierdatentyps nur auf Dokumente anwendbar sind, die korrekt im Sinne dieser Konsistenzkriterien sind.

I.f. unterstellen wir, daß das grundlegende Transformationsschema aus Bild 4 beibehalten werden soll, weil dieses Voraussetzung für viele Differenzanzeigeformen und für Mischungen ist.

Die Konsistenzkriterien des Editierdatentyps stellen nun insofern ein Problem dar, als nicht mehr ohne weiteres beliebige Teilmengen eines Dokuments ein neues gültiges Dokument bilden. Wir müssen daher verlangen, daß nicht nur D1 und D2 den Konsistenzkriterien des Editierdatentyps entsprechen, sondern auch `G(D1,D2)`, d.h. `G(D1,D2)` muß ein Dokument von gleichen Typ wie D1 und D2 sein[13]. Dies führt zu der folgenden neuen Definition:

[13] Alternativ könnte man für G(D1,D2) schwächere Konsistenzkriterien, also letztlich einen anderen Editierdatentyp zulassen, wobei die resultierenden Fehler durch die beiden Einfüge-Transformationen behoben werden. Dies hätte aber diverse gravierende Nachteile, u.a. weil `G(D1,D2)` vielfach als normales Dokument angezeigt werden muß und weil diese Fehlerbehebung bei Mischungen nicht mehr gewährleistet werden könnte.

Definition für strukturierte Dokumente:

Eine **symmetrische Differenz** zwischen zwei strukturierten Dokumenten D1 und D2 besteht aus folgenden Angaben:

1. einer Menge von Korrespondenzen wie in der multimengenbasierten Definition
2. einem (gemeinsamen Zwischen-) Dokument G, das den gleichen Typ wie D1 und D2 hat und das genau die Komponenten enthält, die in den Korrespondenzen vorkommen
3. je einer Einfügetransformation pro Dokument, die ausgehend von G das Dokument D1 bzw. D2 rekonstruiert

Bei Definition 3 können die Transformationen `einfg(D1-D2)` und `einfg(D2-D1)` nur Einfügungen, aber *keine Änderungen* enthalten. Dies betrifft auch die Struktur der Komponenten von G, d.h. nicht nur die Komponenten von G müssen identisch in D1 und D2 auftreten, auch die Struktur gemäß G. Dies schränkt fallweise die Möglichkeiten, Differenzen zu bilden, erheblich ein und kann dazu führen, daß Komponenten aus D1 bzw. D2 nicht als korrespondierend definiert werden können, obwohl sie gleich sind. M.a.W. muß das Zwischendokument G kleiner gewählt werden, als es bei der multimengenbasierten Definition möglich wäre. Hierfür gibt es mehrere Ursachen, u.a.:

1. *komplexe Editierdatentypen*, bei denen z.B. die Operationen so gestaltet sind, daß man bestimmte Komponenten nur zusammen erzeugen oder verändern kann. Wenn von einem derartigen Paar zusammengehöriger Komponenten eine Komponente eine passenden Korrespondenzpartner im anderen Dokument hat und die andere Komponente nicht, dann kann gar keine Korrespondenz gebildet werden.

   Wenn die elementaren Operationen auf der Struktur der Komponentenmenge, z.B. elementare Graphoperationen bei einer Graphstruktur, zu inkonsistenten Zuständen führen können, dürfen sie offensichtlich nicht Teil des Editierdatentyps dieses Dokumenttyps sein. Der Kern des Problems liegt hier darin, daß die strukturierte Multimenge nur eine *Implementierung* des Dokuments auf einer

semantisch tieferen Schicht ist. Auf dieser Schicht dürfen nur solche Differenzen gebildet werden, die man "semantisch liften" kann, d.h. die beiden Einfügetransformationen müssen unter Verwendung zulässiger Operationen realisiert werden können.

2. *Strukturunterschiede* bei den gemeinsamen Komponenten. Dies sei am folgenden Beispiel erläutert:
   - D1 sei eine Textdatei mit den Zeilen "aa", "bb"
   - D2 sei eine Textdatei mit den Zeilen "bb", "aa"
   - Der Editierdatentyp soll eine Operation enthalten, die eine Zeile verschiebt.

   Intuitiv würde man hier den Unterschied zwischen beiden Dokumenten so beschreiben, daß die beiden Zeilen "aa" korrespondieren, die beiden Zeilen "bb" ebenfalls, und daß die Zeile "bb" ausgehend von D1 nach vorne verschoben wird.

   Die aktuell unterstellte Begriffsdefinition einer Differenz verbietet aber derartige Verschiebungen, man kann bestenfalls z.B. folgende Differenz, die qualitativ schlechter erscheint, bilden:

   - `G(D1,D2)` = der Text, der nur die Zeile "aa" enthält
   - `einfg(D1-D2)` = Einfügung von Zeile "bb" am Ende
   - `einfg(D2-D1)` = Einfügung von Zeile "bb" am Anfang

   Alternativ hätte man für `G(D1,D2)` die Zeile "bb" und die Einfügungen dazu passend wählen können.

Das zweite Beispiel zeigt auch, daß bei Differenz von strukturierten Dokumenten nicht eindeutig ist. Bei der multimengenbasierten Differenzdefinition ist der Durchschnitt der Dokumente eindeutig, sofern man die Identifikationsmerkmale von Dubletten außer Betracht läßt und maximal viele Korrespondenzen zwischen gleichen Komponenten bildet. Bei der Differenzdefinition für strukturierte Dokumente muß das Zwischendokument G i.a. echt kleiner gewählt werden. Zugleich wird hierdurch erstmals die Frage relevant, wie man die Qualität von Differenzen beurteilt.

2.3.4 Verschiebungen

Ein wesentlicher Nachteil der obigen Differenzdefinition für strukturierte Dokumente besteht darin, daß die Dokumentkomponenten, die "nur" verschoben worden sind, nicht als gemeinsame Komponenten betrachtet werden und nicht Teil des Zwischendokuments G sein können. Unter einer Verschiebung verstehen wir eine Dokumentänderung, bei der bestimmte Dokumentkomponenten ihre Position in der Dokumentstruktur ändern, der Wert der Dokumentkomponenten aber unverändert bleibt. Beispiele in Klassendiagrammen sind:

- die Verschiebung einer Operation von einer Klasse auf eine andere
- die Verschiebung eines Endes eines Beziehungstyps (also einer Rolle) von einer Klasse auf eine andere

Verschiebungen sind ändernde Operationen, dürfen somit in den Einfüge-Transformationen `einfg(D1-D2)` und `einfg(D2-D1)` nicht auftreten.

Die externe Darstellung von Verschiebungen ist schwierig, aber bei manchen Dokumenttypen und Darstellungsformen durchaus möglich. Dies legt es nahe, auch verschobene Komponenten als korrespondierend zuzulassen. Dann kann allerdings das Transformationsschema aus Bild 4 nicht mehr verwendet werden, stattdessen ist das in Bild 5 gezeigte erforderlich.

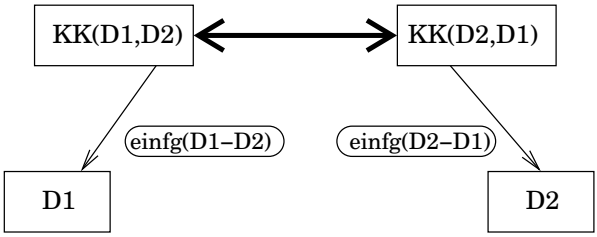

Abbildung 5: Transformationsschema bei verschobenen korrespondierenden Komponenten

Statt der Menge der gemeinsamen Komponenten `G(D1,D2)` sind hier zwei Mengen `KK(D1,D2)` und `KK(D2,D1)` aufgeführt. `KK(X,Y)`

ist die Menge der Komponenten in Dokument X, die eine korrespondierende Komponente in Dokument Y haben, zusammen mit der Struktur dieser Menge gemäß Dokument X. Der Doppelpfeil zwischen `KK(D1,D2)` und `KK(D2,D1)` soll darstellen, daß alle Komponenten in diesen beiden Teildokumenten eine korrespondierende Komponente im jeweils anderen Teildokument haben. Als Menge bzw. Multimenge betrachtet sind `KK(D1,D2)` und `KK(D2,D1)` gleich, ihre Struktur kann aber unterschiedlich sein. Wenn man diese Strukturunterschiede als unwesentlich betrachtet, kommt man zu der folgenden neuen Definition:

Definition für Dokumente mit Verschiebungen:

Eine **symmetrische Differenz** zwischen zwei strukturierten Dokumenten D1 und D2 besteht aus folgenden Angaben:

1. einer Menge von Korrespondenzen wie in der multimengenbasierten Definition
2. zwei Teildokumenten `KK(D1,D2)` und `KK(D2,D1)` von D1 bzw. D2, die jeweils genau die Komponenten enthalten, die in den Korrespondenzen vorkommen
3. einer Verschiebetransformation, die `KK(D1,D2)` in `KK(D2,D1)` umwandelt, und einer umgekehrten Verschiebetransformation
4. je einer Einfügetransformation pro Dokument, die ausgehend von `KK(D1,D2)` bzw. von `KK(D2,D1)` das Dokument D1 bzw. D2 rekonstruiert

Man muß allerdings vor der vorstehenden Definition insofern warnen, als sie in vielen Fällen nicht problemlos verwendbar ist. Die genaue Abgrenzung, welche Teile eines Dokument noch als Komponente betrachtet werden und welche als Struktur, kann von sehr detaillierten Annahmen, wie modelliert wird, abhängig sein, dies führt leicht zu Kommunikationsproblemen. Sobald man die Dokumentstruktur durch Komponenten modelliert, sind Strukturänderungen keine Verschiebungen mehr. Wenn unklar wird, was wirklich der gemeinsame Teil der Dokumente ist, geht man außerdem leicht unbewußt von einer

symmetrischen Differenz zu einem Paar gegenläufiger asymmetrischer Differenzen (i.w. die beiden Verschiebetransformationen) über.

Definitionsalternative mit Einfüge- und Verschiebetransformationen. Für das Problem der Verschiebungen gibt es folgende Lösungsalternative: man ändert die Definition für strukturierte Dokumente dahingehend ab, daß ausgehend von G das Dokument D1 bzw. D2 mit **Einfüge- und Verschiebetransformationen**[14] (anstatt mit reinen Einfügetransformationen) konstruiert wird. Ausgehend von G können also abwechselnd Einfüge- und Verschiebeoperationen benutzt werden, um D1 bzw. D2 zu erzeugen.

Eine Differenz gemäß der Definition für Dokumente mit Verschiebungen kann hierauf zurückgeführt werden: als Dokument G kann `KK(D1,D2)`, `KK(D2,D1)` und jedes beliebige Zwischenstadium bei der Umformung von `KK(D1,D2)` nach `KK(D2,D1)` verwendet werden. Von dort aus kann zuerst mit Verschiebungen `KK(D1,D2)` (oder `KK(D2,D1)`) und danach D1 (oder D2) erzeugt werden.

Die Umkehrung gilt nicht automatisch: eine Einfüge- und Verschiebetransformation kann Verschiebungen vorher eingefügter Komponenten enthalten, daher ist nicht garantiert, daß man die Operationssequenz so umsortieren kann, daß zuerst alle Verschiebungen kommen.

Der Definitionsansatz auf Basis von Einfüge- und Verschiebetransformationen ermöglicht theoretisch größere Mengen gemeinsamer Komponenten, also ein größeres Dokument G, es ist allerdings fraglich, ob G wesentlich größer wird und ob derartige Einfüge- und Verschiebetransformationen noch sinnvoll darstellbar sind. Dies kann nur individuell für einzelne Dokumenttypen diskutiert werden. Problematisch ist auch die ggf. stark steigende Zahl von Alternativen, Differenzen zwischen zwei Dokumenten zu bilden, je nachdem, auf welcher Seite man Verschiebungen anordnet.

[14] bzw. anders gesagt Operationen, die keine Dokumentkomponenten verändern oder löschen.

2.3.5 Ungleiche korrespondierende Komponenten

Alle bisherigen Definitionen erfordern, daß zwei korrespondierende Komponenten völlig identisch sind, da die Einfüge- und Verschiebetransformationen definitionsgemäß die im Durchschnitt bzw. Zwischendokument vorhandenen Komponenten nicht mehr ändern können. Diese Restriktion ist u.a. Voraussetzung für bestimmte Formen der Anzeige von Dokumenten und für die einfachen Mischverfahren.

Bei vielen Gelegenheiten möchte man auch nicht identische Komponenten als korrespondierend darstellen:

- Ein einfaches Beispiel ist der Wunsch, unterschiedliche Groß- und Kleinschreibung nicht als Unterschied aufzufassen. Man kann dies als Äquivalenz von bestimmten Datenwerten auffassen.
- Wenn sich z.B. zwei lange Textzeilen ungleich, aber sehr ähnlich sind, weil z.B. nur ein Wort eingefügt wurde, wird man es als unpassend empfinden, diese Zeilen nicht als korrespondierend dargestellt zu finden.

  Die hier vorliegende Ähnlichkeit von Komponenten kann man auch so verstehen, daß Zeilen ihrerseits aus Worten (oder Buchstaben) bestehen und man die Transformation zwischen den beiden Versionen nicht nur durch Austausch der ganzen Zeile darstellen kann, sondern auch "intelligenter": man unterstellt einen Editierdatentyp mit Operationen, die in einer Zeile ein Wort einfügen bzw. löschen. Hierdurch sind wir aber zu einem Komponentenmodell übergegangen, worin Komponenten in einer Baumstruktur überlappen können; diesen Fall diskutieren wir im folgenden Abschnitt.

Die Ursachen, warum ungleiche Komponenten korrespondieren können, sind strukturell verschieden und müssen im Einzelfall diskutiert werden. Was immer die Ursachen sind, sobald wir das grundlegende Paradigma aller bisherigen Definitionen, daß korrespondierende Komponenten identisch sind, aufgeben, können wir das darauf basierende Transformationsschema (s. Bild 4) nicht mehr aufrechterhalten.

Eine naheliegende Lösung ist vom Transformationsschema aus Bild 5 auszugehen und es dahingehend zu erweitern, daß zwischen den Ele-

menten in `KK(D1,D2)` und `KK(D2,D1)` nicht nur Verschiebungen, sondern auch bestimmte Wertänderungen erlaubt werden. Ob überhaupt und in welcher konkreten Ausgestaltung solche Differenzdefinitionen sinnvoll sind, kann nur individuell für konkrete Dokumenttypen entschieden werden.

2.3.6 Technische vs. mentale Komponenten

Bei allen bisherigen Definitionen hatten Dokumentkomponenten einige günstige, die Diskussion vereinfachende Eigenschaften:

- Komponenten waren gleichzeitig mentale Konzepte wie auch technische Speichereinheiten.

  Unter technischen Speichereinheiten verstehen wir hier das Abstraktionsniveau üblicher Analysemodelle, die zwar von vielen Details von konkreten Speicherformaten abstrahieren, aber trotzdem deren wesentliche Struktur repräsentieren. Dieses Abstraktionsniveau war auch beim Begriff Editierdatentyp unterstellt.

  Bei den bisher als Beispiel benutzten Texten sind die technischen Komponenten (Zeilen) konsistent mit dem üblichen mentalen Modell eines Textes.
- Komponenten waren inhaltlich "disjunkt" und selbständig erzeug- bzw. löschbar. Bei Texten ist jede Zeile inhaltlich völlig unabhängig von jeder anderen Zeile des Textes, und im Prinzip kann jede Zeile technisch selbständig gespeichert werden.
- Komponenten waren einzeln optisch darstellbar. Dies wurde implizit bei der Parallel- und der Vereinigungsdarstellung ausgenutzt, um Korrespondenzen oder spezielle Teile der einzelnen Dokumente darzustellen.

Diese günstigen Umstände haben es erlaubt, Differenzbegriffe auf einer technischen Ebene zu bilden und diese Begriffe bruchlos in mentale Modelle zu übertragen. Bei komplexeren Dokumenttypen, z.B. Dateibäumen, Klassendiagrammen, Paketdiagrammen und diversen weiteren graphstrukturierten Dokumenttypen, ist dies nicht so einfach:

- Die konzeptuelle (Daten-) Modellierung der mentalen Dokument-

strukturen ist nicht mehr trivial und in gewissem Rahmen Ermessenssache bzw. Interpretationsfrage. Beispiele sind diverse "inhaltliche Bestandteile" von UML-Diagrammen, die als schwierig zu modellieren gelten bzw. nach der Modellierung nicht immer eindeutig interpretiert werden: Parametertypen in Klassendiagrammen, Unterbrechungsbereiche in Zustandsautomaten, Reihenfolgen in Sequenzdiagrammen usw.

- Je nach der konzeptuellen Modellierung sind nicht alle technischen Entitäten / Attribute sinnvoll einzeln darstellbar und optisch gegenüberstellbar.
- Viele kleine mentale Einheiten, z.B. Namen von Klassen oder Attributen, können nicht sinnvoll selbständig erzeugt werden (selbst wenn das technisch ginge), sondern nur im Kontext größerer Einheiten, und werden deshalb nicht als echte mentale Dokumentkomponenten wahrgenommen.
- Viele mentale Dokumentkomponenten sind nicht disjunkt. In einem Klassendiagramm können z.B. sowohl Klassen als auch die darin enthaltenen Operationen mental als Dokumentkomponenten aufgefaßt werden und insb. Korrespondenzen bilden.

  Auf der technischen Ebene sind Entitäten dagegen im Prinzip immer selbständig.

Die Nichtdisjunktheit mentaler Dokumentkomponenten führt zu eigenen Interpretationsproblemen. Wir betrachten i.f. nur den Fall, daß eine Dokumentkomponente ganz in einer anderen enthalten sein kann[15]. Die Überlappungen bzw. Teil-von-Beziehungen von Dokumentkomponenten bilden unter dieser Annahme eine Baum- (oder Wald-) Struktur. Wir unterstellen hier weiter, daß diese Baumstrukturen i.w. auch bei den technischen Komponenten vorliegen, daß also insb. mentale Dokumentkomponenten, die Teil einer anderen Dokumentkomponente sind, technisch nur einmal repräsentiert werden. Demnach besteht die technische Repräsentation einer Komponente mit

[15]Eine teilweise Überlappung tritt nur bei sehr wenigen Dokumenttypen auf und kann hier vernachlässigt werden.

Teilkomponenten (also eines Nicht-Blattknotens) aus Referenzen auf die Repräsentationen der Teilkomponenten und den "direkt zugeordneten" Attributen, die die restlichen Eigenschaften repräsentieren, vergleichbar mit der Wurzel eines Baums. Zwei Dokumentkomponenten können unter diesen Annahmen nur dann gleich sein, wenn auch alle Teile gleich sind.

Der Begriff Komponente kann nunmehr bzgl. der Gleichheit auf zwei Arten interpretiert werden:

1. im Sinne der *Wurzel* eines komplexen Objekts ist.

   Bei dieser Interpretation sind nur diejenigen Merkmale einer Komponente für die Gleichheit zweier Komponenten relevant, die nicht einer Unterkomponente zugeordnet sind, die also auf der technischen Ebene durch direkt zugeordneten Attribute repräsentiert werden.

   Im Extremfall sind gar keine solchen Attribute bzw. Merkmale vorhanden; der "Wert" des Knotens im Sinne eines Gleichheitstests ist dann ein Nullwert, damit sind alle derartigen Knoten gleich und Dubletten, und es sind völlig unsinnige Korrespondenzen denkbar.
2. im Sinne eines ganzen *Teilbaums* (oder allgemeiner eines Teilgraphen).

   Beispielsweise sind zwei Klassen ungleich, sobald sie irgendeinen Detailunterschied aufweisen, z.B. ein anders benanntes Attribut, und können deshalb nach den bisherigen Definitionen auch nicht korrespondieren, obwohl die Wurzelknoten komplett identisch sind und beiden Klassen sehr ähnlich sind und intuitiv als korrespondierend empfunden werden.

Die Definition einer symmetrischen Differenz ist im Prinzip bei beiden Interpretationen anwendbar (auf der technischen Ebene), das Ergebnis ist aber in beiden Fällen, wenn man von der technischen auf die mentale Ebene zurückschließt, nicht zufriedenstellend: die Gleichheit zweier Wurzeln ist nicht immer hinreichend, um von einer sinnvollen Korrespondenz reden zu können, die Gleichheit kompletter Teilbäume ist als Kriterium zu streng.

Letztlich muß hier wenigstens eine der beiden folgenden grundsätzlichen Eigenschaften des mathematischen Begriffs Differenz zur Disposition gestellt werden:

1. Nur völlig gleiche Komponenten liegen im Durchschnitt bzw. korrespondieren. Anders gesagt gibt es keine irrelevanten Teile von Komponenten.

   Nur weil korrespondierende Komponenten völlig gleich sind, können sie eine einheitliche Basis für die Konstruktion der beiden speziellen Dokumente bilden und im Prinzip als irrelevant für die Darstellung der Unterschiede zwischen zwei Dokumenten behandelt werden.
2. Nur die in einer Komponente selbst vorhandenen Merkmale sind relevant dafür, ob zwei Komponenten als identisch betrachtet werden bzw. korrespondieren bzw. im Durchschnitt liegen können, keine außerhalb liegenden Merkmale und keine anderen Komponenten.

Anders formuliert wird bei den bisherigen Definitionen für *alle lokalen* und *keine nichtlokalen* Merkmale gefordert, daß sie bei korrespondierenden Komponenten gleich sind. Bei vielen graphartigen Dokumenttypen, insb. fast allen UML-Diagrammtypen, muß man hiervon abweichen.

Grundsätzlich kommen nur solche technische Komponenten – also Knoten des Dokumentbaum oder -Graphen – überhaupt für Korrespondenzen infrage, die mentale Komponenten repräsentieren, die selbständig erzeugt und gelöscht werden können und die aus Benutzersicht sinnvoll korrespondieren können, weil sie wahrnehmbar identisch oder ähnlich sind. Dies ist vor allem eine Frage der Modellierung. Beispielsweise sind einfache Namen meist keine sinnvollen Komponenten, auch wenn sie technisch als Knoten des Dokumentbaums repräsentiert werden.

Korrespondierende (technische) Komponenten sind nicht mehr unbedingt identisch, sondern nur ähnlich. Der Begriff ähnlich ist dokumenttypspezifisch zu definieren und muß im Extremfall sogar situationsspezifisch variierbar sein. Da die mentalen Komponenten durch mehrere technische Komponenten repräsentiert werden, können für

die Beurteilung der Ähnlichkeit von technischen Komponenten auch Merkmale relevant sein, die außerhalb dieser Komponenten liegen.

2.3.7 Zusammenfassung: asymmetrische vs. symmetrische Differenzen

Asymmetrische und symmetrische Differenzen weisen viele Gemeinsamkeiten auf. Symmetrische Differenzen bzw. der mengenorientierte Ansatz sind insofern einfacher und leichter verständlich, als eine sehr einfache und für alle Dokumenttypen einheitliche Struktur der Dokumente unterstellt wird. Selbst in der Version der Definition für Multimengen ist es nicht nötig, wie beim operationalen Ansatz den Editierdatentyp für den vorliegenden Dokumenttyp im Einzelfall komplett zu definieren, es werden nur die einfügenden Operationen benötigt, und diese sind meist trivial. Erhebliche Probleme treten auf, sobald komplexere Editierdatentypen und zugrundeliegende Konsistenzkriterien berücksichtigt werden müssen und die Komponentenmenge eine Baumstruktur oder eine noch komplexere Struktur hat, weil dann der Durchschnitt der Komponentenmengen i.a. gar kein Dokument mehr ist und die Auswahl eines Zwischendokuments sehr schwierig wird.

Die wichtigsten Unterschiede zwischen den beiden Ansätzen sind in der folgenden Tabelle schlagwortartig zusammengefaßt.

Merkmal	operationaler Ansatz	mengenorientierter Ansatz
intuitive Vorstellung von "Differenz"	eine Transformationsvorschrift	die speziellen Komponenten zweier Dokumente
unterstellte Struktur der Dokumente	beliebig komplexe Datentypen mit nichttrivialen Konsistenzkriterien	"flache" Menge, Multimenge, Sequenz, Verzeichnis o.ä.
Ausgangspunkt	1 Dokument, 1 Differenz	2 Dokumente + ggf. 1 Durchschnitt
typische Anwendung	Patch-Werkzeug	üblicher 2-spaltiger Dokumentvergleich
Symmetrie	asymmetrisch	symmetrisch

Beim Vergleich der grundlegenden Begriffswelten fällt auf, daß der Begriff der Korrespondenzen, der für symmetrische Differenzen ganz zentral ist, bei asymmetrischen Differenzen völlig fehlt bzw. nicht benötigt wird. Trivialerweise kann man Komponenten, die beim Anwenden einer asymmetrischen Differenz unverändert bleiben, als korrespondierend ansehen; hierzu muß man aber die Semantik der Editieroperationen kennen, d.h. es sind hierzu keine allgemeinen Aussagen möglich. Für gelöschte, geänderte oder eingefügte Komponenten liefert die operationale Begriffswelt keinen Denkansatz, wann sie als korrespondierend anzusehen sind[16].

2.4 Konversion von Differenzen

Konversion einer symmetrischen Differenz in zwei asymmetrische Differenzen. Aus einer gegebenen symmetrischen Differenz zwischen zwei Dokumenten D1 und D2 kann man bei allen obigen Definitionen zwei asymmetrische Differenzen von D1 nach D2 und von D2 nach D1 ableiten.

[16] Genaugenommen werden hier zeitlich nacheinander existierende Zustände *eines* Dokuments verglichen, nicht zwei parallel existierende Dokumente.

Hierbei unterstellen wir, daß man aus den Einfüge-Transformationen inverse Lösch-Transformationen ableiten kann. In Bild 4 sind derartige Lösch-Transformationen schon eingezeichnet (`lösche(D1-D2)` und `lösche(D2-D1)`). Bei Differenzen nach Definition 4 müssen auch die Verschiebungen invertiert werden können. Hierzu muß die Darstellung der Verschiebungen ausreichende Informationen enthalten.

Unter diesen Voraussetzungen kann man leicht aus einer symmetrischen Differenz zwischen D1 und D2 eine asymmetrische Differenz von D1 nach D2 ableiten: diese besteht gerade aus der Hintereinanderschaltung `lösche(D1-D2); einfg(D2-D1)`. Bei Differenzen nach Definition 4 müssen nach den Löschungen noch die Verschiebungen vorgenommen werden. Analog kann eine asymmetrischen Differenz von D2 nach D1 abgeleitet werden.

Bei ungleichen korrespondierenden Komponenten oder dokumenttypspezifischen Varianten des Begriffs symmetrische Differenz ist nicht offensichtlich, daß man immer inverse Transformationen ableiten kann; bei der Berechnung der Differenzen lassen sich dazu erforderliche Hilfsdaten aber immer gewinnen.

Anders gesagt kann man, sofern man überhaupt sinnvoll eine symmetrische Differenz bilden kann, diese auch in asymmetrische Differenzen konvertieren.

Die asymmetrischen Differenzen basieren auf dem gleichen Editierdatentyp wie die symmetrische Differenz und sind qualitativ insofern optimal, als keine überflüssigen Änderungen durchgeführt werden und im Rahmen des unterstellten Editierdatentyps auch keine "kleinere" Darstellung der Differenz gefunden werden kann.

Konversion asymmetrischer Differenzen in symmetrische Differenzen. Die Konversion einer asymmetrischen Differenz (oder sogar von zwei gegenläufigen asymmetrischen Differenzen) in eine symmetrischen Differenz ist nur in Ausnahmefällen möglich. Diese Konversionsrichtung ist aber wenig relevant, weil normalerweise zunächst symmetrische Differenzen gesucht werden.

Gegeben sei also eine asymmetrische Differenz von D1 nach D2, gesucht ist eine symmetrische Differenz.

Wenn der Editierdatentyp eine eindeutige Trennung von löschenden, verschiebenden und einfügenden Operationen erlaubt und wenn in der Sequenz der Einzeländerungen alle Löschungen vor der ersten Verschiebung liegen, dann kann der Anfangsteil bis zur letzten Löschung als Transformation `lösche(D1-D2)` (s. Definition 2) benutzt werden; der Rest der Sequenz entspricht der Transformation `einfg(D2-D1)`. Indem man `lösche(D1-D2)` auf D1 anwendet, kann man ein Zwischendokument G erzeugen. Für die symmetrische Differenz fehlt noch `einfg(D1-D2)`; alleine aus `lösche(D1-D2)` kann man diese Angaben nicht gewinnen, wenn aber D1 gegeben ist, können theoretisch die Angaben durch Beobachtung der Effekte von `lösche(D1-D2)` gewonnen werden.

Korrespondenzen können wie folgt definiert werden:

- Als korrespondierend können wir naheliegenderweise solche Komponenten definieren, die unverändert bleiben, die also nicht als Parameter von Löschungen oder Änderungen auftreten. Sofern wir die Wirkung der Editieroperationen nicht genau kennen, können wir aber die Änderungen, die die Differenz in D1 bewirkt, nicht einschätzen. Editieroperationen können globale Effekte haben, d.h. es können Komponenten betroffen sein, die nicht explizit als Parameter aufgeführt sind; daher können wir ohne Annahmen hinsichtlich der Wirkung der Editieroperationen nicht bestimmen, welche Komponenten unverändert bleiben.

  Immerhin trifft bei vielen Editierdatentypen die Annahme zu, daß nur solche Komponenten verändert werden, die auch explizit als Parameter in der asymmetrischen Differenz auftauchen.

- Korrespondenzen zwischen in D1 gelöschten und D2 eingefügten Komponenten können aus einer asymmetrischen Differenz nicht abgeleitet werden.

Wenn die Transformationsschritte nicht im obigen Sinne richtig sortiert sind, kann die Sequenz u.U. umsortiert werden, was aber bereits eine sehr genaue Kenntnis der Semantik des Editierdatentyps erfordert. Das Vorhaben scheitert, wenn die Operationen nicht kom-

mutieren. Ebenfalls nicht lösbar sind die Probleme bei komplexen Editierdatentypen.

2.5 Positionen und lokale Unterschiede

Differenzen bezogen sich bisher auf *ganze* Dokumente. Beim Vergleich von Dokumenten oder Diskussionen, woraus eine *Differenz im Detail besteht*, benutzt man häufig zusätzliche Begriffe wie "lokaler Unterschied", "gleiche Stelle in zwei Dokumenten" oder "Stelle (Position), an der sich die Dokumente unterscheiden". Wichtig ist vor allem der Begriff lokaler Unterschied, weil Darstellungen von Dokumentdifferenzen i.d.R. darin bestehen, einzelne lokale Unterschiede darzustellen, und bei Mischungen für jeden einzelnen lokalen Unterschied eine Mischentscheidung getroffen werden muß. Eine Differenz muß daher als Menge (in Ausnahmefällen als Folge) lokaler Unterschiede auffaßbar sein.

Wie wir i.f. zeigen werden, unterscheiden sich die o.g. Begriffe deutlich, wenn man den operationalen bzw. mengenorientierten Ansatz unterstellt.

Positionen. Informell definiert ist eine **Position** eine Angabe, mit der eine oder ggf. mehrere Komponenten eines Dokuments identifiziert werden können. Beispiele sind Zahlen, Intervalle, Namen, Identifizierer usw. U.U. ist an einer Position zur Zeit keine Komponente vorhanden, dort könnte aber eine Komponente eingefügt werden.

Positionen können sich auf eine vorhandene Komponente beziehen und heißen dann **relativ**; andernfalls heißen sie **absolut**. Was dies konkret bedeutet, hängt vom Typ des Dokuments ab. In Textdokumenten sind "die Zeile Nr. i" oder "das k. Zeichen in Zeile Nr. i" absolute Positionen, "der Raum hinter Zeile i und vor der Folgezeile" eine relative Position. Analog hierzu kann man Positionen bei allen Dokumenten definieren, deren Komponenten linear geordnet sind. In einem Telefonbuch ist der "Eintrag zu Telefonnr. X" eine absolute Position. Analog hierzu kann man Positionen bei allen Dokumenten definieren, deren Komponenten als Verzeichnis strukturiert sind. Re-

lative Positionen können hier nicht immer sinnvoll definiert werden.

Es sind auch Schachtelungen und Mischformen denkbar. In einem UML-Klassendiagramm, in dem keine Namensduplikate auftreten, sind die Klassen und Beziehungstypen in einem Diagramm als Verzeichnis anzusehen, die Menge der Operationen einer Klasse ebenfalls als Verzeichnis (mit Operationsname und Signatur als Identifizierer), eine Parameterliste als Sequenz.

2.5.1 Begriffe für asymmetrische Differenzen

Wir kommen noch einmal auf das in Abschnitt 2.2 und Bild 2 gezeigte Beispiel einer asymmetrischen Differenz zurück. Offensichtlich benötigt man ausschließlich Positionsangaben in Dokument D1, um die Differenz anzugeben. Positionen im zweiten Dokument werden für die Angabe einer Differenz nicht benötigt und interessieren beim operationalen Ansatz im Grunde überhaupt nicht.

Die Begriffswelt kommt auch völlig ohne den Begriff "gleiche (bzw. korrespondierende) Position in beiden Dokumenten" aus; derartige Paare von korrespondierenden Positionen brauchen nicht bekannt zu sein. Änderungsoperationen können nichtlokale Wirkungen haben (Beispiel: ändere überall den Namen einer Funktion). Die Änderungsoperation hat nur eine Komponente (Beispiel: das definierende Auftreten der Funktion) als Argument, nach der Änderungsoperation sind aber an vielen Positionen Unterschiede vorhanden. In solchen Fällen kann man überhaupt nicht mehr ohne weiteres von korrespondierenden Positionen reden.

Wie schon in Abschnitt 2.4 diskutiert kann man bei unveränderten Komponenten korrespondierende Positionen unterstellen, sofern die Editieroperationen keine nichtlokalen Effekte haben.

Der Begriff "lokaler Unterschied" ist beim operationalen Ansatz nur dann sinnvoll nutzbar, wenn die ändernden Operationen nur lokale Effekte erzeugen, also Effekte, die die als Argument benutzte Komponente ändern oder in unmittelbarer Nachbarschaft Komponenten erzeugen, löschen oder ändern. Diese Annahme ist z.B. in Syntaxeditoren nicht erfüllt, in denen eine Editieroperation die Anwendung

einer komplexe Grammatikregel realisiert und der Effekt weit auseinanderliegende Stellen im Zieldokument betreffen kann.

2.5.2 Begriffe für symmetrische Differenzen

Die weiteren Begriffe können bei einer symmetrischen Differenz vor allem auf die gegebene Menge von Korrespondenzen aufbauen.

Offensichtlich sollten Positionen, an denen korrespondierende Komponenten stehen, als **korrespondierende Positionen** gelten. Unklarer ist die Lage bei den speziellen Komponenten.

Wir betrachten erneut das in Bild 2 gezeigte Beispiel. Angenommen, wir unterstellen folgende Korrespondenzen - ausgedrückt als Paare von Zeilennummern: (1,1), (2,3), (5,6) und (7,8). Die Menge der speziellen Komponenten von D1 besteht dann aus der Zeile Nr. 6, die Menge der speziellen Komponenten von D2 aus den Zeilen Nr. 2 und 7. Die Tabelle in Bild 6 zeigt das Ergebnis.

D1		D2
a	=	a
	!	b
b	=	b
c	=	c
d	=	d
e	=	e
f	!	e
f	=	f

Abbildung 6: Beispiel korrespondierender Zeilen

In der mittleren Spalte zeigt ein Gleichheitszeichen eine Korrespondenz an. Ein Ausrufungszeichen in der mittleren Spalte zeigt einen "lokalen Unterschied" an. Dieser bezieht sich auf ein Paar korrespondierender ("gleicher") Positionen. Diese korrespondierenden Positionen konnten wir aber nur deshalb bestimmen, weil die Komponentenmenge linear geordnet ist und die speziellen Komponenten zwischen

Komponenten mit Korrespondenzen, also korrespondierenden Positionen liegen.

Ohne eine lineare Ordnung – z.B. in einem zweidimensionalen graphischen Dokument – hätten wir keine Intervalle von Positionen bilden können und daher für die speziellen Komponenten keine korrespondierenden Positionen bestimmen können. Als Konsequenz kann ein lokaler Unterschied immer nur aus einer Einfügung im Zwischendokument bestehen. Eine Gegenüberstellung von Alternativen wie in der vorletzten Zeile von Bild 6 ist nicht möglich.

2.6 Mehrwege-Dokumentvergleich

Bei den bisherigen Begriffen waren stets zwei zu vergleichende Dokumente unterstellt. Die Begriffe können teilweise auf den Vergleich von mehreren Dokumenten verallgemeinert werden.

Asymmetrische Differenzen. Bei asymmetrischen Differenzen ist eine echte Verallgemeinerung auf mehrere Dokumente nicht möglich. Man kann nur paarweise Differenzen bilden, z.B. indem man eines der Dokumente als Basis auswählt und alle anderen damit vergleicht oder indem man die Dokumente als Folge von Revisionen betrachtet und schrittweise vergleicht. Dies führt aber alles nicht zu Gesamtdarstellungen aller Unterschiede.

Auch aus Anwendungssicht besteht wenig Bedarf an asymmetrischen Differenzen zwischen n Dokumenten.

Symmetrische Differenzen. Bei symmetrischen Differenzen kann zumindest die mengen- und die multimengenbasierte Definition direkt auf n zu vergleichende Dokumente verallgemeinert werden, indem man den gemeinsamen Durchschnitt der Dokumente als die Menge bzw. Multimenge der Komponenten, die in allen Dokumenten auftreten, definiert. Die Einfüge-Transformationen sind auf dieser Basis wie vor zu definieren, Korrespondenzen sind hier n-Tupel mit je einer Komponente pro Dokument.

Die vorstehende Definition ist zwar mathematisch sauber, aber sie

ist nicht für alle Anwendungen sinnvoll. Wenn man 10 Dokumente zu vergleichen hat und eine bestimmte Komponente fehlt nur in einem einzigen Dokument, dann ist der oben definierte Durchschnitt keine gute Darstellung der Unterschiede und Gemeinsamkeiten der Dokumentmenge. Daß eine bestimmte Komponente in vielen Dokumenten auftritt, kann wichtig sein; Details hängen allerdings von speziellen Dokumenttyp ab.

Bei strukturierten Dokumenten stellt sich wie schon im 2-Wege-Fall das Problem, ob der Durchschnitt überhaupt ein korrektes Dokument ist bzw. daß ein korrektes Basisdokument, von dem aus die zu vergleichenden Dokumente erzeugt werden können, nicht eindeutig bestimmt und nur wegen Strukturdifferenzen sehr klein ist.

Der in Abschnitt 2.3.4 diskutierte Ansatz, durch eine spezielle Behandlung von Verschiebungen größere Basisdokumente möglich zu machen, ist nur mit Einschränkungen auf den Mehrwegefall übertragbar. Die Verschiebungen sind nämlich im Prinzip asymmetrische Differenzen, und alle Probleme, die oben schon für asymmetrische Differenzen zwischen n Dokumenten genannt wurden, treffen auch hier zu. So muß z.B. ein willkürlich gewähltes Dokument als Basis für die Verschiebungen gewählt werden, was dem Prinzip der Symmetrie widerspricht.

2.7 Dokumentvergleiche vs. -Mischungen

Mischungen gehen immer von einer symmetrischen Differenz der zu mischenden Dokumente aus. Dies ist insb. beim interaktiven Mischen offensichtlich, weil dort zunächst die Unterschiede zwischen den beiden zu mischenden Dokumenten gezeigt werden müssen. Mischungen basieren daher begrifflich in erster Linie auf symmetrischen Differenzen.

Im Vergleich zur reinen Differenzdarstellung kommt die Gewinnung eines dritten Dokuments D3 (s. Bild 7) als zentrales Thema hinzu. In diesem Zusammenhang entstehen neue Probleme und Begriffe, die wir anschließend diskutieren.

Die Transformation von einem Dokument D1 nach D2 (oder umgekehrt), die bei asymmetrischen Differenzen im Mittelpunkt des In-

teresses steht, interessiert bei Mischungen überhaupt nicht[17].

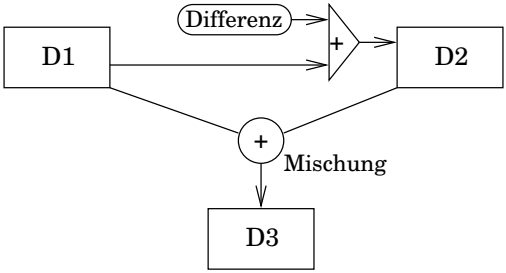

Abbildung 7: Differenzbetrachtung vs. Mischung

Ferner ergeben sich durch die Ziele und Randbedingungen des Mischens einige Besonderheiten bei der Bildung von symmetrischen Differenzen. Diese Besonderheiten führen auch zu eigenen Begriffen und werden in den folgenden Abschnitten ausführlicher behandelt.

2.7.1 2-Wege-Mischungen

Das letztliche Ziel des Mischens ist die Erzeugung eines neuen Dokuments, das aus den Komponenten der beiden Basisdokumente zusammengesetzt ist. Dies entspricht der grundlegenden Denkweise symmetrischer Differenzen bzw. des mengenorientierten Ansatzes, derzufolge Dokumente aus Komponenten zusammengesetzt sind; das Mischen geht insofern über den reinen Vergleich von Dokumenten hinaus, als aus Komponenten unterschiedlicher Herkunft *neue Dokumente konstruiert werden.* Für den Mischvorgang ist zentral, daß die vorhergehende Differenzbildung Informationen liefert über:

1. Korrespondenzen und

[17] Eine Transformation unterstellt auch, daß man D2 als Nachfolgeversion von D1 betrachtet, D1 also im Normalfall irrelevant ist. Eine Mischung beider Dokumente bedeutet dann, daß "alte Teile" des Dokuments in den aktuellen Stand zurückgeholt werden. Dieser Fall ist denkbar, ist aber aus Anwendersicht eher so zu verstehen, daß die Änderungen, die den alten Stand verändert haben, zurückgenommen werden sollen, also eine Teilmenge der Differenz gebildet wird, die auf D1 angewandt wird.

2. bei strukturierten Dokumenten über die Positionen der speziellen Komponenten

Die Korrespondenzen beeinflussen den Mischvorgang unmittelbar. Der Begriff Mischen stimmt bei Dokumenten nicht mit dem umgangssprachlichen Begriff Mischen überein. Wenn man in der Küche 500 Gramm Mehl und 300 Gramm Zucker mischt, erwartet man ein Ergebnis von 800 Gramm Gewicht[18]. Das Mischen von Dokumenten ist von Ausnahmen abgesehen weitaus weniger ertragreich; man geht stets von der Annahme aus, daß gemäß der Denkweise des mengenorientierten Ansatzes viele **gemeinsame Komponenten** vorhanden sind und daß diese unverändert und nur einmal (!) in das Mischergebnis übernommen werden.

Der intuitive Begriff "gemeinsame Komponenten" entspricht genau dem Begriff "korrespondierende Komponenten" bei symmetrischen Differenzen, allerdings nur solange korrespondierende Komponenten exakt identisch sind (s. Abschnitt 2.3.3). Bei korrespondierenden Komponenten, die nur ähnlich oder zueinander äquivalent sind, ist unklar, was in das Mischergebnis übernommen werden soll.

Wir machen hier keine Einschränkungen, woher die beiden zu mischenden Dokumente stammen; sie können unabhängig voneinander entstanden sein, also keine gemeinsame Ursprungsversion haben. Diesen Fall bezeichnet man auch als **2-Wege-Mischen**.

Wie schon erwähnt bilden die gemeinsamen Komponenten in beiden Dokumenten ein Dokument, das wir als **G(D1,D2)** bezeichnen (s. Bild 8). Die "Besonderheiten" der beiden verglichenen Dokumente stellen sich als Einfügungen der speziellen Komponenten dar (in Bild 8 als `einfg(D1-D2)` und `einfg(D2-D1)` bezeichnet), also als asymme-

[18] Außerdem wird umgangssprachlich mit dem Begriff Mischen oft gemeint, vorhandene Ordnungen und Strukturen aufzuheben ("Karten mischen"). Dies trifft auf Dokumentmischungen natürlich nicht zu. Daher werden statt der Bezeichnung Mischen öfter andere Bezeichnungen wie Verschmelzen, Vereinigen (Begriff aus der Mengenlehre!) und weitere verwendet. Sinnvoll wären auch Bezeichnungen wie fusionieren oder zusammenfassen, diese sind aber unüblich. Im Sprachgebrauch am häufigsten scheint die Bezeichnung Mischen zu sein.

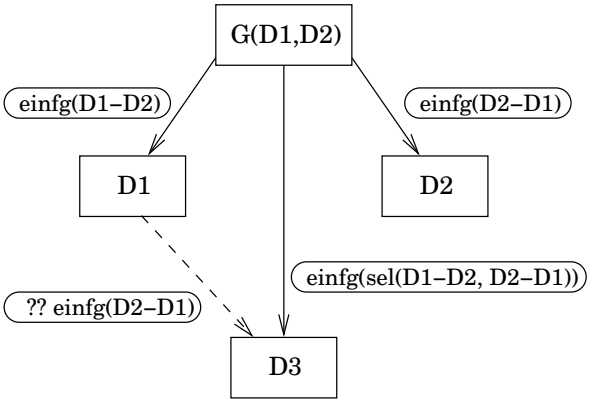

Abbildung 8: Rolle gemeinsamer Komponenten beim 2-Wege-Mischen

trische Differenzen, die ausschließlich aus Einfügungen bestehen, von G(D1,D2) nach D1 und D2.

Das Erzeugen des Mischergebnisses entspricht einem Nachvollziehen aller oder mancher der lokalen Einfügungen (s. mittleren Pfeil in Bild 8) in G(D1,D2). Die Notation `einfg(sel(D1-D2,D2-D1))` soll andeuten, daß im Prinzip eine Auswahl der speziellen Komponenten aus D1 und D2 in den gemeinsamen Durchschnitt eingefügt wird.

Daß nur eine Auswahl der speziellen Komponenten übernommen werden soll, kann eine willkürliche Entscheidung von Entwicklern sein. Die Auswahl muß dann natürlich manuell vorgegeben werden und begrifflich wie auch technisch ermöglicht werden. Der i.f. unterstellte Normalfall ist allerdings, daß möglichst automatisch alle Besonderheiten übernommen werden sollen.

Die Frage liegt auf der Hand, warum man überhaupt eine Auswahl braucht und ob man nicht viel einfacher das Mischergebnis D3 erzeugen kann, indem man, wie in Bild 8 angedeutet, die Einfügetransformation `einfg(D2-D1)` auf D1 anwendet (oder umgekehrt die Einfügetransformation `einfg(D1-D2)` auf D2). Die Antwort ist, daß dies nur bei trivialen Dokumentstrukturen (unsortierten Multimengen) funktioniert. Bei nichttrivialen Dokumentstrukturen treten erhebliche

Probleme auf:

1. Die Positionen, die in der Einfügetransformation `einfg(D2-D1)` benutzt werden, können sich verschoben haben. In einfachen Fällen kann für eine einzufügende spezielle Dokumentkomponente eine neue entsprechende Position ermittelt werden. Manchmal ist dies aber nicht eindeutig möglich; damit die Dokumentkomponente überhaupt irgendwo im Mischergebnis erscheint, kann notfalls eine beliebige Position gewählt werden.
2. Wenn der Editierdatentyp nichttriviale Konsistenzkriterien aufweist, z.B. eindeutige Bezeichner für bestimmte Kollektionen von Dokumentkomponenten, kann es sein, daß einzelne spezielle Dokumentkomponenten aus D2 überhaupt nicht in D1 eingefügt werden können. Die in der Einfügetransformation unterstellten Editieroperationen würden also nicht erfolgreich durchgeführt werden können.

Das Problem, daß spezielle Komponenten überhaupt nicht oder nicht an einer eindeutigen Position in das Mischergebnis übernommen werden können, führt zum Begriff **Mischkonflikt**. Diesen werden wir erst anschließend diskutieren, können aber hier schon festhalten, daß

- Mischkonflikte durch nichttriviale Dokumentstrukturen bzw. entsprechende Editierdatentypen verursacht werden und
- das Mischergebnis in Endeffekt nicht alle speziellen Dokumentkomponenten der beiden Ausgangsdokumente (unverändert) enthält.

Letzteres ist für die Vereinigungsdarstellung von Differenzen, die sich als Basis für interaktive Mischwerkzeuge besonders eignet, völlig unakzeptabel: alle speziellen Dokumentkomponenten müssen unbedingt sichtbar sein. Dieser Zwang führt fallweise dazu, daß man die Konsistenzkriterien in der Vereinigungsdarstellung aufgibt und damit *implizit zu einem anderen Editierdatentyp übergeht*, der die Dokumentstruktur in Richtung Multimenge vereinfacht. Das angezeigte Vereinigungsdokument ist dann kein Dokument im Sinne des ursprünglichen Editierdatentyps, erst nach manueller Bereinigung der Inkonsistenzen

kann es wieder als Dokument im Sinne des ursprünglichen Editierdatentyps verstanden werden (und z.B. wieder verglichen werden). Letztlich werden hier die Dokumente abwechselnd auf verschiedenen Konsistenzstufen bzw. unter Annahme wechselnder Editierdatentypen betrachtet, wobei (tückischerweise) die externe Darstellungen nahezu gleich bleiben. Dieser Wechsel zwischen Konsistenzstufen ist machbar, aber sehr irritierend, zumal die Übergänge zwischen den Konsistenzstufen nicht wirklich explizit und wegen der praktisch unveränderten externen Darstellung kaum erkennbar sind.

Beim Transformationsschema in Bild 8 wird jedenfalls ein Editierdatentyp unterstellt, der einheitlich für alle auftretenden Dokumente und Transformationen gilt.

2.7.2 3-Wege-Mischungen

Häufig sind die zu mischenden Dokumente Varianten voneinander, also Revisionen in verschiedenen Zweigen, die von einer gemeinsamen Ausgangsversion D0 ausgehen. Beim **3-Wege-Mischen** wird zusätzlich die Ausgangsversion herangezogen, allerdings primär für Mischentscheidungen (s.u.). Bei einigen Varianten wird die Ausgangsversion zusätzlich genutzt, um die gemeinsamen Komponenten zu bestimmen: hier werden nur solche Komponenten als "gemeinsam" angesehen, die in *allen drei* Dokumenten gleich vorhanden sind. In Bild 9 wird diese Menge als G(D0,D1,D2) bezeichnet.

Ein Unterschied zwischen G(D0,D1,D2) und G(D1,D2) tritt nur dann auf, wenn eine Komponente der Ausgangsversion in beiden Zweigen exakt gleich abgeändert worden sind. Bei solchen Komponenten scheint es auf den ersten Blick selbstverständlich, sie in der geänderten Form in das Mischergebnis zu übernehmen. Die Änderung an der vorliegenden Stelle kann indessen mit Änderungen an anderen Stellen des Dokuments zusammenhängen, die nicht in das Mischergebnis übernommen werden sollen. Daher ist es denkbar, daß im Mischergebnis der ursprüngliche Zustand wiederhergestellt wird, obwohl beide Nachfolgeversionen an dieser Stelle keinen Unterschied aufweisen. In solchen Fällen kann beim 3-Wege-Mischen das Mischergebnis nicht immer

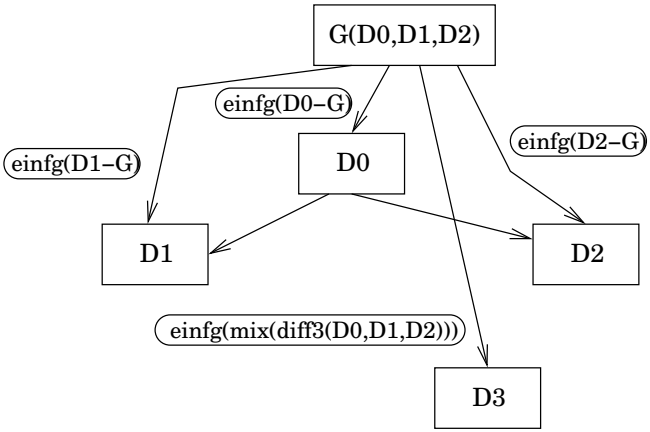

Abbildung 9: Rolle gemeinsamer Komponenten beim 3-Wege-Mischen

allein auf Basis von G(D1,D2) und `einfg(D1-D2)` und `einfg(D2-D1)` aus Bild 8 konstruiert werden[19].

Bzgl. der Differenzbildung liegt hier eine Variante der in Abschnitt 2.6 skizzierten Verallgemeinerung einer symmetrischen Differenz für Multimengen vor; zusätzlich werden hier im Rahmen des Vergleichs diejenigen Komponenten bestimmt, die in zwei Dokumenten (D0 und D1 bzw. D0 und D2) vorkommen. Diese spezielle Form der Differenzbildung ist in Bild 9 wie das UNIX-Standardwerkzeug `diff3` benannt worden.

Die beiden Differenzen von D0 nach D1 bzw. von D0 nach D2 in Bild 9 sind normale asymmetrische Differenzen und können insb. Löschungen beinhalten, während `einfg(D1-D2)` und `einfg(D2-D1)` nur Einfügungen beinhalten.

[19]Das UNIX-Standardwerkzeuge `diff3` zeigt hier z.B. einen Konflikt und die Version aus D0 an, obwohl die beiden verglichenen Dateien keinen lokalen Unterschied aufweisen.

2.7.3 Mischentscheidungen

Beim Mischen muß bei jedem lokalen Unterschied entschieden werden, welche der Alternativen in das Mischergebnis übernommen werden soll. Beim 2- und 3-Wege-Mischen sind die Alternativen nicht ganz identisch.

Beim 3-Wege-Mischen betrachtet man die unterschiedlichen lokalen Zustände in D1 bzw. D2 als Ergebnis je einer Änderung in D0 (also gemäß Bild 9 ein entsprechender Anteil der Gesamtdifferenz D1-D0 bzw. D2-D0). Hier sind zwei Fälle zu unterscheiden:

1. Einer der lokalen Zustände in D1 bzw. D2 ist unverändert gegenüber D0, es liegt also nur *eine Änderung* gegenüber D0 vor. Man hat dann die beiden Entscheidungsalternativen, diese eine Änderung zu übernehmen oder nicht.
2. Die lokalen Zustände in D1 und D2 sind beide verändert gegenüber D0, es liegen also *zwei Änderungen* vor. Folgende Entscheidungen sind hier denkbar:
   1. nur die Änderung gemäß D1 wird übernommen
   2. nur die Änderung gemäß D2 wird übernommen
   3. beide Änderungen werden übernommen; da der Gesamteffekt i.a. von der Reihenfolge, mit der die Änderungen vorgenommen werden, abhängt, ist zusätzlich über die Reihenfolge zu entscheiden
   4. keine der beiden Änderungen wird übernommen, d.h. der lokale Zustand gemäß D0 bleibt erhalten

   In den meisten Fällen ist nur die erste oder zweite Entscheidung sinnvoll; viele Mischwerkzeuge schränken die Auswahl deshalb auf diese beiden Entscheidungen ein, dies stellt aber einen Konstruktionsfehler dar und sollte nicht dazu verleiten, die Begriffswelt auf diese beiden Entscheidungen zu reduzieren. Die 3. Entscheidung, beide Änderungen zu übernehmen, ist z.B. sinnvoll, wenn an der gleichen Stelle zwei unabhängige Einfügungen vorgenommen wurden. Die 4. Entscheidung, keine Änderung zu übernehmen, ist nur sehr selten sinnvoll.

Beim 2-Wege-Mischen liegen zunächst nur unterschiedliche lokale Zustände vor. Diese sollte man allerdings gemäß Bild 8 als Einfügungen in G(D1,D2), also ebenfalls als *Änderungen eines Basisdokuments* auffassen. Die vorstehenden 4 Entscheidungsalternativen für den Fall von 2 Änderungen übertragen sich damit auch auf das 2-Wege-Mischen.

2.7.4 Automatisierung von Mischentscheidungen

Mischentscheidungen können immer manuell getroffen werden - hierzu müssen natürlich geeignete Bedienschnittstellen in Mischwerkzeugen vorhanden sein. Alle Entscheidungen manuell anzugeben ist aber oft zu zeitraubend oder nicht praxisgerecht. Das beste Beispiel ist das vollautomatische Mischen, das das Update-Kommando von CVS mit Dokumenten durchführt, die im Arbeitsbereich und im Repository verändert worden sind. Daher werden Verfahren benötigt, durch die die *Entscheidungen automatisiert getroffen werden.*

Beim 3-Wege-Mischen ist oft eine sehr einfache automatisierte Entscheidung möglich, und zwar dann, wenn nur *eine* Änderung gegenüber D0 (s.o.) vorliegt. Diese Änderung wird automatisch akzeptiert. Gewählt wird also der Zustand, der eine Änderung gegenüber der Ausgangsversion darstellt. Diese Entscheidung ist zwar nicht immer, aber fast immer die richtige, der Einsatz dieses Entscheidungsverfahrens also pragmatisch gerechtfertigt.

Wenn zwei Änderungen vorliegen, ist eine automatisierte Entscheidung nicht möglich (zumal bei 4 Alternativen). Der Sachverhalt, daß eine pragmatisch gerechtfertigte automatische Entscheidung fehlt, wird im Kontext des 3-Wege-Mischens als **Konflikt** bezeichnet. Der Begriff Konflikt wir aber auch allgemeiner benutzt (s.u.).

Beim 2-Wege-Mischen ist eine automatisierte Entscheidung *prinzipiell nicht möglich.* Daher ist auch der obige Begriff Konflikt hier prinzipiell nicht sinnvoll anwendbar, weil per definitionem bei allen lokalen Änderungen ein Konflikt vorliegt.

In manchen 2-Wege-Mischwerkzeugen kann man mit einem einzigen Kommando alle Entscheidungen zugunsten eines der Dokumente

voreinstellen. Danach kann man einzelne Entscheidungen manuell zugunsten des anderen Dokuments abändern. Es handelt sich bei solchen Voreinstellungen nicht um automatische Entscheidungen, sondern um explizite Entscheidungen, lediglich der Bedienungskomfort wird verbessert.

2.7.5 Konflikte

Wir haben den Begriff Konflikt im vorigen Abschnitt im speziellen Kontext des 3-Wege-Mischens definiert für den Fall, daß an *der gleichen Position* (im Durchschnitt) zwei Änderungen gegenüber der Ursprungsversion vorliegen; er drückt aus, daß keine automatische Entscheidung möglich ist, welche der Änderungen in das Mischergebnis übernommen werden sollen.

Bei der bisherigen Definition des Begriffs Konflikt blieb offen, warum man keine Entscheidung treffen kann. Naheliegend wäre eigentlich analog zum konfliktfreien Fall eine Übernahme aller Änderungen in das Mischergebnis. Es gibt zwei qualitativ verschiedene Gründe, warum dies nicht geht:

1. Je nach der Wahl der Reihenfolge der Änderungen kann sich ein signifikant anderes Endergebnis ergeben. Bei jeweils einer eingefügten Zeile ergeben sich zwei verschiedene Reihenfolgen der Zeilen in der Mischung. Bei einer Änderung einer Zeile in D1 und einer Löschung der Zeile in D2 ist nach der Löschung die Änderung nicht mehr möglich, d.h. diese Änderung geht verloren.
2. Die beiden Änderungen können inhaltlich unverträglich sein, d.h. das resultierende Dokument ist inkorrekt.

Das zweite Problem, also die Frage, ob das Mischergebnis korrekt ist[20], ist das wichtigere. Korrektheit kann sehr verschieden definiert sein, in einfachsten Fall geht es nur um elementare Syntax, im Ex-

[20]Dieses Problem tritt unabhängig davon auf, ob Mischentscheidungen manuell oder automatisiert getroffen werden. Bei manuellen Entscheidungen ist indes die Wahrscheinlichkeit höher, daß inkorrekte Mischergebnisse früh erkannt werden.

tremfall um die "Vereinigung" des Verhaltens oder der Bedeutung der beiden Dokumente.

Der ursprüngliche positionsbasierte Konfliktbegriff erkennt mögliche Fehler nur bei Paaren von Änderungen, die an korrespondierenden Positionen in den beiden Basisdokumenten liegen. Bei einigermaßen komplexen Dokumenten (insb. Programmquelltexten) können auch Änderungen an ganz unterschiedlichen Positionen zu Fehlern führen; Korrektheit ist oft eine globale Eigenschaft. Der positionsbasierte Konfliktbegriff erkennt solche semantischen Konflikte nicht.

Semantische Konflikte kann man informell wie folgt definieren: zwei Änderungen stehen in **semantischem Konflikt**, wenn das resultierende Mischergebnis Fehler enthält, die in den beiden Basisdokumenten nicht vorhanden waren. Fehler bezieht sich auf die Korrektheitskriterien des zugrundeliegenden Dokumenttyps[21]. Semantische Konflikte in vorstehenden Sinn müssen daher für jeden Dokumenttyp konkret definiert werden, ferner müssen spezielle Algorithmen, die diese Konflikte bestimmen, entwickelt werden. Die Effizienz dieser Algorithmen ist kritisch, zumal jede Änderung im ersten Dokument mit jeder Änderung im zweiten in Konflikt stehen kann, die Zahl der zu überprüfenden Paare von Änderungen also quadratisch steigt. Letztlich ist die exakte Suche nach allen semantischen Konflikten kaum praktikabel.

Ein Kompromiß besteht darin, die Menge der semantischen Konflikte nur approximativ zu bestimmen,

- indem man allgemein anwendbare Verfahren benutzt, die notwendigerweise auf einfachen Dokumentstrukturen und Kriterien basieren müssen, und
- indem man inkauf nimmt, daß einige vorhandene Konflikte nicht gefunden werden und einige Paare von Änderungen, die nicht in wirklichem Konflikt stehen, gefunden werden.

Oft wird der Begriff **Konflikt** im Kontext solcher Verfahren als Syn-

[21] also nicht nur die Korrektheitskriterien des Editierdatentyps, sondern weitergehende; per definitionem erfüllt das Mischergebnis immer die Korrektheitskriterien des Editierdatentyps.

onym für einen Treffer bei diesem Verfahren benutzt. Beispiele solcher Verfahren bzw. Heuristiken sind:

- Lokale Unterschiede, bei denen beide Varianten eine Änderung gegenüber der Ausgangsversion darstellen, gelten immer als Konflikt.
- Lokale Unterschiede an "nahe beieinanderliegenden Positionen" (z.B. weniger als 4 Zeilen Abstand) werden ggf. optional als Konflikt eingestuft.

2.7.6 Zusammenfassung

Die Unterschiede und Gemeinsamkeiten im Differenzbegriff bei reinen Dokumentvergleichen bzw. bei Mischungen können wie folgt zusammengefaßt werden:

- Mischungen unterstellen stets den mengenorientierten Denkansatz. 2-Wege-Mischungen basieren auf einer symmetrischen Differenz. 3-Wege-Mischungen basieren im Prinzip auf einer symmetrischen Differenz, die auf 3 Dokumente verallgemeinert ist.
- Die Problematik Konflikte ist nur für das 3-Wege-Mischen relevant, nicht hingegen für das 2-Wege-Mischen und die reine Differenzanzeige.
- Übliche Mischwerkzeuge kann man nur für solche Dokumenttypen und Editierdatentypen realisieren, bei denen der Durchschnitt identisch in die beiden Basisdokumente eingebettet werden kann (s. Abschnitt 2.3.3).
- Der Vergleich und die vergleichende externe Darstellung von Dokumenten ist beim Mischen stark geprägt durch das Problem, Mischkonflikte zu identifizieren und darzustellen; bei reinen Dokumentvergleichen tritt dieses Problem nicht auf.

3 Repräsentationen von Dokumenten

Die Diskussion von Dokumentdifferenzen wird dadurch erschwert, daß ein Dokument in verschiedenen Repräsentationen betrachtet werden kann bzw. muß. In diesem Abschnitt analysieren wir die möglichen Repräsentationen von Dokumenten und die Auswirkungen darauf, wie Differenzen berechnet und definiert werden können.

3.1 Vier Repräsentationen

Wir können vier Repräsentationen, die oft auch als Modelle des Dokuments bezeichnet werden, unterscheiden, und zwar die **physische**, **externe**, **semantische** und **Editier-Repräsentation** (Bild 10 gibt eine Übersicht).

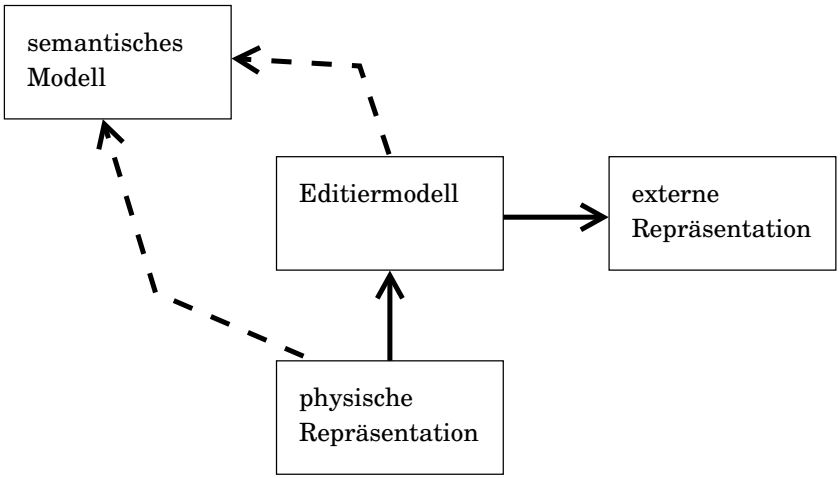

Abbildung 10: Repräsentationen bzw. Modelle von Dokumenten

1. die **physische Repräsentation** (bzw. das **Speichermodell**): hier betrachten wir die Dokumente auf der Ebene des Datenverwaltungssystems, also als Inhalt von Dateien, einer relationalen oder objekt-

orientierten Datenbank oder eines sonstigen Datenverwaltungssystems. Ein Dokument ist hier strukturiert in der Denkwelt (also im Datenbankmodell und den benutzten Schemata) dieses Datenverwaltungssystems.

Diese Repräsentation wird oft auch als **persistente Repräsentation** bezeichnet, weil alle anderen nicht persistent sind.

2. die **externe Repräsentation**: eine externe Repräsentation ist eine textuelle und/oder graphische Darstellung eines Dokuments auf einem Bildschirm oder auf Papier. Das gleiche Dokument kann mehrere verschiedene externe Repräsentationen haben, z.B. textuelle und graphische. Viele Details einer externen Repräsentation hängen von der verwendeten Software ab.

3. das **semantische Modell** eines Dokuments: dieses Modell stellt eine Abstraktion der physischen Repräsentation dar, in der sich ein Nutzer das Dokument strukturiert vorstellen und es inhaltlich verstehen kann. Details hängen von dem Typ des Modells ab. Bei XML-Dokumenten oder Programmtexten ist es der Syntaxbaum.

   In vielen anderen Fällen kann das semantische Modell nur gebildet werden, wenn die physische Repräsentation syntaktisch korrekt ist oder darüber hinaus bestimmte semantische Korrektheitskriterien erfüllt. In Bild 10 ist deswegen der Pfeil von der physischen Repräsentation zum semantischen Modell gestrichelt gezeichnet.

4. das **Editiermodell**: dieses Modell stellt eine Repräsentation dar, in der aus Nutzersicht Dokumenttransformationen (und damit auch Differenzen zwischen Dokumenten) leicht ausgedrückt werden können. Es liegt bzgl. seiner Abstraktion und seinen Konsistenzkriterien irgendwo zwischen der physischen Repräsentation und dem semantischen Modell.

Die verschiedenen Repräsentation unterscheiden sich bei textuellen Dokumenten nur wenig, bei graphischen Dokumenten oder XML-Dokumenten hingegen deutlich. Um vor allem die Unterschiede klarzumachen, betrachten wir später die einzelnen Repräsentationen für XML-Dokumente an einem Beispiel.

Anmerkungen zur externen Repräsentation. Die externe Repräsentation unterscheidet sich erheblich von den anderen Repräsentationen:

- Sie ist nicht eindeutig, es kann mehrere verschiedene externe Repräsentation für ein Dokument geben. Die Darstellung eines Dokuments kann ferner ggf. durch Werkzeugeinstellungen variiert werden.
- Während alle anderen Repräsentationen *ein* Dokument darstellen, müssen zur Darstellung von Differenzen i.d.R. *zwei* Dokumente dargestellt werden. Hierfür sind mehrere Varianten denkbar (Details werden in Abschnitt 1.3 diskutiert). Die Darstellungsformen für Dokumentpaare basieren fast alle auf einer Darstellungsform für ein einzelnes Dokument, beide Darstellungsformen sind also sehr ähnlich. Dennoch ist hier natürlich die Darstellungsform für Dokumentpaare entscheidend; diese entspricht inhaltlich nicht dem Editier-Modell *eines* Dokuments.

Obwohl also die externe Darstellung von Differenzen aus Benutzersicht im Vordergrund steht, ist sie im hier vorliegenden Kontext - Klärung der Rolle verschiedener Repräsentationen - nicht direkt relevant.

3.2 Editiermodell vs. Editierdatentyp

Das Editiermodell hängt sehr eng mit dem oben eingeführten Editierdatentyp zusammen. Wie schon in Abschnitt 2.2 betont ist der Editierdatentyp ein *abstrakter* Datentyp, er definiert nur (Editier-) *Operationen* auf Dokumenten. Die Implementierung der Dokumente bleibt völlig verborgen - zumindest in der Theorie.

In der Praxis läuft es meist umgekehrt: eine sehr beliebte Methode, abstrakte Datentypen zu spezifizieren, besteht darin, eine geeignete abstrakte Datenstruktur vorzugeben – z.B. eine Syntaxbaumstruktur – und die meisten Operationen des ADT einfach auf die generischen Operationen dieser Datenstruktur zurückzuführen. Lediglich komplexe, dokumenttypspezifische Operationen und zusätzliche dokumenttypspezifische Konsistenzbedingungen (z.B. kontextsensitive) müssen

zusätzlich beschrieben werden.

Das Editiermodell, genauer gesagt die hierbei benutzte abstrakte Datenstruktur, ist also *eine Repräsentation, mit deren Hilfe sich der Editierdatentyp besonders leicht spezifizieren läßt.* Es sei allerdings davor gewarnt, daß verschiedene Editierdatentypen, die leichte Unterschiede in der Menge der Operationen aufweisen, auf der gleichen abstrakten Datenstruktur arbeiten können. Die Datenstruktur alleine legt also den Editierdatentypen nicht fest! In Graphstrukturen kann man z.B. das Verschieben eines Knotens als Operation vorsehen oder nicht.

Die Gestaltung der abstrakten Datenstruktur des Editiermodells ist innerhalb gewisser Grenzen willkürlich, Beispiele von Editiermodellen für konkrete Dokumenttypen betrachten wir anschließend.

Das Editiermodell muß natürlich mit dem Speichermodell konsistent sein, auf dem auch die sonst für diesen Dokumenttyp benutzten Werkzeuge basieren, denn die Differenz- und Mischwerkzeuge müssen alle Dokumente, die von normale Editoren erzeugt werden, verarbeiten können. Umgekehrt müssen die von einem Mischwerkzeug produzierten Mischergebnisse von normalen Werkzeugen weiterverarbeitet werden können.

In vielen Fällen können Dokumente, die auf Basis des Editiermodells entstehen, in dem Sinne inkorrekt sein, daß *kein zugehöriges semantisches Modell* gebildet werden kann, in Bild 10 ist deswegen der Pfeil vom Editiermodell zum semantischen Modell gestrichelt gezeichnet.

3.3 Repräsentationen von XML-Dokumenten

Die Bemerkungen zu den Repräsentationen von XML-Dokumenten gelten größtenteils auch für HTML-Dokumente, LaTeX-Dokumente, Programmquelltexte und andere als Baum strukturierbare Dokumente.

1. physische Repräsentation: Wir unterstellen die Speicherung in einer Textdatei. Ein Vortragsliste mit Angabe von Termin, Vortragen-

dem und Titel könnte als folgender Auszug aus einem Dateiinhalt repräsentiert sein[22]:

```
<vortraege>
<vortrag>
   <termin>12.08.2004</termin> <name>Meier</name>
   <titel>Die unbekannte Differenz</titel>
</vortrag>
<vortrag>
   <termin>19.08.2004</termin> <name>Huber</name>
   <titel>Differenz vs. Distanz</titel>
</vortrag>
</vortraege>
```

2. Externe Repräsentation: In der Form, die z.B. von Mozilla und anderen WWW-Browsern erzeugt wird, wird eine XML-Datei in einer normierten Form angezeigt, d.h. Leerraum ist teilweise entfernt worden (innerhalb von Tags), teilweise zwecks Einrückungen hinzugefügt worden.

3. Semantisches Modell: Dies ist hier der entsprechende Syntaxbaum. Der Syntaxbaum kann gut durch einen Graphen dargestellt werden.

   Genaugenommen ist aber auch die strukturierte und kolorierte textuelle Darstellung von Mozilla ohne weiteres als Darstellung des Syntaxbaums interpretierbar.

   Sowohl die textuelle wie die graphische Darstellung drücken den entscheidenden Sachverhalt *nicht* aus, nämlich daß beim Syntaxbaum ein abstrakter Datentyp unterstellt wird, bei dem Bäume nur durch Anwendung bestimmter Grammatikregeln (die in Graphtransformationen resultieren) erzeugt werden; ein Syntaxbaum entsteht nicht, in dem man nacheinander Textzeilen aufschreibt oder

[22] Was man hier zeigen kann, ist eigentlich auch nur eine externe Repräsentation, den wirklichen Dateiinhalt kann man überhaupt nicht sehen, denn der ergibt sich nur als eine bestimmte Menge von Rückgabewerten, nachdem bestimmte Leseoperationen des APIs des Dateisystems aufgerufen worden sind. Diese externe Repräsentation stellt aber die Datei "1:1" dar, d.h. sie benutzt die gleiche Dokumentmodell, das auch dem API des Dateisystems zugrundeliegt.

Kästchen malt, die Kästchen mit Strichen verbindet und da und dort Texte in die Kästchen oder an die Striche schreibt.

4. Editiermodell: Für das Editiermodell und den damit spezifizierten Editierdatentyp sind mehrere Alternativen denkbar:

   Editierdatentyp A: wir wählen das Editiermodell i.w. identisch mit der physischen Repräsentation, d.h. die Dokumente werden verändert, indem Zeilen gelöscht oder neue Zeilen eingefügt werden.

   Wenn wir eine Zeile mit einem öffnenden Tag in eine Datei einfügen und die Datei dann speichern, ohne an der passenden Stelle eine Zeile mit dem schließenden Tag einzufügen, ist die Datei anschließend nicht mehr wohlgeformt, d.h. bei diesem Editierdatentyp können nichtwohlgeformte Dateien auftreten.

   Editierdatentyp B: wir wählen das Editiermodell i.w. identisch mit dem semantischen Modell, die zugehörige abstrakte Datenstruktur ist ein Syntaxbaum mit den zug. Graphtransformationen. Dokumente werden nun verändert, indem Teilbäume (also Komponenten) erzeugt oder gelöscht werden oder einzelne neue Knoten an einer bestimmten Stelle angehängt werden.

   Im Gegensatz zu Editierdatentyp A sind können hier nur wohlgeformte Dateien auftreten.

   Editierdatentyp C: wie Editierdatentyp B, aber mit einer zusätzlichen Operation, die einen Teilbaum des Syntaxbaums verlagert.

3.3.1 Beispiele für Differenzen

Die Wahl des Editiermodells und des zug. Editierdatentyps hat erhebliche Auswirkungen, wie Unterschiede dargestellt werden können. Hierzu betrachten wir folgendes zweite Dokument:

```
<vortraege>
```

```
<vortrag>
   <termin>12.08.2004</termin> <name>Meier</name>
   <titel>Differenz vs. Distanz</titel>
</vortrag>
</vortraege>
```

und fragen uns, mit welchen Operationen das vorige Dokument in das zweite transformiert werden kann, also wie eine asymmetrische Differenz vom ersten zum zweiten Dokument aussehen kann:

Editierdatentyp A: Differenz A1: Zeilen Nr. 4 - 7 löschen

Editierdatentyp B: Wir betrachten 2 Differenzen:

Differenz B1:

1. beide Elemente vom Typ `vortrag` löschen
2. das eine neue Element komplett neu erzeugen

Differenz B2:

1. im ersten Element vom Typ `vortrag` das `titel`-Element löschen,
2. dort ein neues passendes `titel`-Element erzeugen
3. das 2. Element vom Typ `vortrag` löschen.

Editierdatentyp C: zusätzlich zu den bei Editierdatentyp B genannten Differenzen B1 und B2 können wir auch nutzen:

Differenz C1:

1. im ersten Element vom Typ `vortrag` das `titel`-Element löschen,
2. an diese Stelle das im 2. Element vom Typ `vortrag` vorhandene `titel`-Element verschieben
3. das 2. Element vom Typ `vortrag` löschen.

Differenz C1 ist insofern besser als Differenz B2, als das Löschen und identische Neuerzeugen eines Elements vermieden wird.

Zu den vorstehenden asymmetrischen Differenzen passende symmetrische Differenzen und darauf aufbauende Visualisierungen der Unter-

schiede – wir unterstellen eine üblichen 2-spaltige Darstellung – sehen wie folgt aus:

- Differenz A: Zeilen 4 - 7 aus dem ersten Dokument werden als spezielle Zeilen angezeigt, jeweils die beiden ersten und die beiden letzten Zeilen werden als korrespondierend angezeigt
- Differenz B1: als korrespondierend werden nur die jeweils ersten Zeilen und die jeweils letzten angezeigt; diese Zeilen repräsentieren den Wurzelknoten unseres Beispiels. Alle restlichen Zeilen werden als spezielle Komponenten angezeigt (also ein wesentlich größerer Teil des Dokuments als bei Differenz A).
- Differenz B2: als korrespondierend werden die 1., 2., 3., 5. und letzte Zeile beider Dokumente angezeigt. Diese repräsentieren den Wurzelknoten, den ersten Vortrag-Knoten und darin den Termin- und den Namen-Knoten. Der Rest wird als spezielle Komponente angezeigt.
- Differenz C1: Im Vergleich zu B2 kommt eine weitere Korrespondenz hinzu, nämlich der Titel-Knoten im zweiten Vortrag-Knoten des ersten Dokuments korrespondiert zum Titel-Knoten im ersten Vortrag-Knoten des zweiten Dokuments.

  Mit der üblichen Paralleldarstellung kann man diese Menge von Korrespondenzen nicht graphisch darstellen (vgl. Abschnitt 1.3).

Bei der Differenz B2 fällt auf, daß die beiden jeweils ersten Vortrag-Knoten als korrespondierend definiert sind, obwohl die zugehörigen Teilbäume bzw. dadurch repräsentierten realen Vorträge nicht komplett identisch sind. Ursache dieser Auffälligkeit ist, daß man zwei verschiedene Definitionen für die Gleichheit von Komponenten des Syntaxbaums benutzt:

1. Komponenten sind die Knoten, die Kanten des Baums zählen als Struktur und nicht zum Inhalt der Komponenten. Für die Gleichheit von Komponenten ist nur der Knoten selbst relevant. Die inneren Knoten der hier vorliegenden Syntaxbäume dienen nur der Strukturierung und haben keinen Inhalt. Sie sind daher immer gleich.

2. Komponenten sind wiederum die Knoten. Für die Gleichheit von Komponenten ist der komplette davon ausgehende Teilbaum relevant. Dieser Fall kann dahingehend verallgemeinert werden, daß irgendwo positionierte andere Komponenten für die Gleichheit relevant sind.

Die zweite Definition von Gleichheit von Komponenten kann bei den vorhandenen Differenzbegriffen nicht ohne weiteres genutzt werden, weil implizit davon ausgegangen wird, daß die Komponenten eines Dokuments disjunkte Datenmengen sind.

3.3.2 Komplexität der Editierdatentypen

Aus dem vorigen Beispiel können wir ableiten, daß zu komplexe Editierdatentypen Probleme verursachen.

Bezüglich der externen Darstellungsformen kann man nur spekulieren, ob es vielleicht andere noch nicht erfundene externe Darstellungsformen gibt, die die genannten Probleme lösen; generell scheint es schwierig,

1. eine größere Zahl von Operatoren (also Operationen in der Typdefinition, nicht einzelne Operationsaufrufe) darzustellen, weil für jeden Operator ein eigenes graphisches Symbol benötigt wird; neben Einfügungen und Löschungen kommen als "generische" Operatoren das Verschieben und Kopieren in Betracht, ferner ggf. dokumenttypspezifische Operatoren;
2. Operationen (im Sinne von Operationsaufrufen) mit mehreren Parametern darzustellen
3. Parameter von Operationen, die nicht unverändert im Ergebnis erscheinen, darzustellen
4. die ggf. einzuhaltende Reihenfolge der Operationsaufrufen darzustellen (bei der Paralleldarstellung ist die Reihenfolge, in der die einzelnen lokalen Unterschiede behandelt werden, beliebig).

Ein weiteres Argument gegen komplexe Editierdatentypen ist, daß mit der Zahl der Editieroperationen der Aufwand zur Bestimmung

von Differenzen steigt, andererseits aber die Differenzberechnung in der Praxis ziemlich performant sein muß. Generell ergibt sich die Erkenntnis, daß

1. der Editierdatentyp nicht unabhängig von den Möglichkeiten zur (graphischen) Anzeige von Differenzen gestaltet werden kann und daß
2. im Editierdatentyp nur ein Bruchteil der Operationen verwendet werden kann, die die typischen Editoren für den jeweiligen Dokumenttyp anbieten; neben Löschungen und Einfügungen von Komponenten sind nur in seltenen Ausnahmefällen weitere Operationen möglich.

   Hieraus folgt, daß das Editiermodell erst recht nicht identisch mit oder auch nur ähnlich zu dem semantischen Modell sein kann, sofern letzteres auf vielen und/oder komplexen Operationen beruht; das Editiermodell muß also deutlich simpler sein.

Die Möglichkeiten zur (graphischen) Anzeige von lokalen Unterschieden in allen gängigen UML-Diagrammtypen wurden in [Oh04, OhWK03b] analysiert und wie folgt klassifiziert:

1. Struktur-Änderungen (neue/gelöschte Knoten und Beziehungen)
2. Intra-Knoten/Beziehungsänderungen (Änderungen der inneren Knotenstruktur):
   - atomare Wertänderungen, z.B. von Bezeichnern oder Kardinalitaten
   - neue oder gelöschte Elemente in Listen und Mengen
   - Verschiebungen in einer Liste (Umordnungen)
   - Änderungen an komplexen Knoten, die auf ein Unterdiagramm zurückzuführen sind
   - Änderungen an Referenzen auf andere Modellkomponenten
3. nichtlokales Verschieben von Komponenten oder Beziehungen zwischen Knoten

Diese Klassen entsprechen direkt bestimmten generischen Operationen, wenn man eine baumartige Implementierung der Dokumente unterstellt.

3.4 Repräsentationen von Textdokumenten

Unter einem Textdokument wird hier der Inhalt einer zeichenorientierten Datei, in der jedes Byte als Zeichen interpretiert wird, verstanden (Multi-Oktet-Zeichensätze verkomplizieren die Lage nur technisch, nicht inhaltlich). In solchen Textdokumenten werden klassischerweise Programmquelltexte, Latex-Quelltexte, einfache Briefe u.v.a. mehr gespeichert.

Die vier Repräsentationen eines Textdokuments unterscheiden sich fast nicht.

Bei Programmquelltexten und ähnlichen Inhalten betrachtet man teilweise Leerraum (Leerzeichen, Tabulatoren, Zeilenvorschübe) als vernachlässigbar. In solchen Fällen würde das semantische Modell einer Textdatei eine weitere Textdatei sein, in der z.B. vorne und am Satzende stehende Leerzeichen entfernt wären und im Satz Leerraum auf ein Leerzeichen reduziert wäre.

4 Berechnung von Differenzen

4.1 Übersicht

Behandelte Dokumentklassen. Wir behandeln in diesem Kapitel die Berechnung von Differenzen von Dokumenten, deren Komponentenstruktur

- eine Menge oder Multimenge ist oder
- eine Sequenz (Texte) oder
- ein Baum bzw. "Fast-Baum" oder
- ein allgemeiner attributierter gerichteter Graph

ist. Die vorderen Gruppen sind natürlich Sonderfälle der hinteren, haben aber eigene Berechnungsverfahren.

Es gibt diverse Dokumenttypen, die nicht in die vorstehenden Gruppen passen, z.B. Bitmap-Graphiken oder Rechentabellen, deren Struktur nicht sinnvoll als Graph aufgefaßt werden kann. Für Dokumente dieser Typen sind meist keine brauchbaren Differenzwerkzeuge vorhanden.

"Darstellungstiefe". Wie Differenzen berechnet werden, hängt im Detail stark von der Architektur der Werkzeuge, vom Typ des Dokuments und von Differenzbegriff ab. In diesem Abschnitt sollen nur die grundlegenden Verfahren und Algorithmen vorgestellt werden mit dem Ziel, einen Überblick über

- architektonische Voraussetzungen und technische Einsatzbedingungen,
- Komplexität und Größenordnung der Laufzeit der Verfahren,
- behandelbare Arten von Dokumenttypen

usw. zu gewinnen. Technische Detailprobleme (z.B. Datenaufbereitung bei der Eingabe), die in konkreten Werkzeugen zu lösen sind, diskutieren wir hier nicht, ebenso keine Algorithmen, die auf spezielle Merkmale eines einzelnen Dokumenttyps eingehen und nur auf Dokumente dieses Typs anwendbar sind.

Verfahren für asymmetrische Differenzen. Das einzige bekannte Verfahren, mit dem man direkt asymmetrische Differenzen gewinnen kann, basiert auf der Protokollierung von Editierkommandos in Editoren. Umgekehrt kann man mit diesem Verfahren auch nur asymmetrische Differenzen gewinnen. Dieses Verfahren ist nur in Ausnahmefällen praktikabel, weswegen wir es nur kurz in Abschnitt 4.2 diskutieren.

Verfahren für symmetrische Differenzen. Alle anderen wichtigen Verfahren basieren auf Dokumentzuständen und berechnen im Prinzip zunächst symmetrische Differenzen, die danach bei Bedarf zu asymmetrischen Differenzen konvertiert werden. Stark vereinfacht betrachtet besteht das Grobschema, nach dem die meisten dieser Verfahren arbeiten, aus folgenden Phasen bzw. Hauptfunktionen:

1. **Einlesen:** Einlesen der Dokumente und Strukturanpassungen; hierbei Konversion der Eingabedokumente in eine interne Graphdarstellung
2. **Kandidatensuche:** Suche nach ähnlichen oder identischen Komponenten, die Kandidaten für Korrespondenzbildungen werden
3. **Festlegung der Korrespondenzen,** also Auswahl aus den Kandidaten für Korrespondenzbildungen
4. **Ausgabe:** Ausgabe der Differenz und diverser Hilfsdaten, ggf. Ausgabe von Mischungen

Die vorstehenden Funktionen bauen zwar aufeinander auf, sind aber in konkreten Algorithmen nicht unbedingt als strikte Phasen organisiert. Außerdem besteht erheblicher Spielraum bei der Ausgestaltung der Funktionen.

Unter den zustandsbasierten Verfahren spielt das Verfahren auf Basis persistenter Identifizierer von Dokumentkomponenten, das in Abschnitt 4.3 vorgestellt wird, eine Sonderrolle: es funktioniert nur in geschlossenen Umgebungen, in denen alle Editoren persistente Identifizierer unterstützen, und selbst dann auch nicht ganz korrekt, ist aber sehr effizient und daher in der Praxis beliebt.

Es ist sinnvoll, die übrigen Verfahren danach zu klassifizieren, ob sie nur Korrespondenzen zwischen identischen Komponenten bilden oder auch zwischen ähnlichen; wir bezeichnen die Verfahren dementsprechend als **identitätsbasiert** bzw. **ähnlichkeitsbasiert**. Sowohl die Suche nach Kandidaten für Korrespondenzbildungen als auch die Festlegung der Korrespondenzen ist bei identitätsbasierten Verfahren wesentlich einfacher und daher effizienter durchführbar als bei ähnlichkeitsbasierten Verfahren. Daher besprechen wir in Abschnitt 4.4 zunächst die identitätsbasierten Verfahren.

Unter den identitätsbasierten Verfahren sind diejenigen für Mengen und unstrukturierte Multimengen am einfachsten, weil automatisch alle Differenzen optimal sind und man keinen Aufwand zur Optimierung der Differenz treiben muß. Bei strukturierten Multimengen ist trifft dies nicht mehr zu. Diese Dokumentklasse ist wesentlich schwieriger zu behandeln, weil

1. Varianten bei der Festlegung der Korrespondenzen möglich sind,
2. bei einer gegebenen Menge von Korrespondenzen zusätzlich Varianten für deren Struktur auftreten können und
3. für die Beurteilung der Qualität einer Differenz nicht nur die Zahl der Korrespondenzen herangezogen werden kann, sondern auch Merkmale der Struktur, die dokumenttypspezifisch sind.

Letztlich liegt hier ein Optimierungsproblem vor, dessen Lösung sehr rechenzeitaufwendig werden kann und für das dokumenttypspezifische Heuristiken eingesetzt werden müssen. Wir betrachten in den Abschnitten 4.4.2 und 4.4.3 sequentiell bzw. baumartig strukturierte Dokumente.

Spezielle Verfahren für einzelne Diagrammtypen. Alle vorstehend genannten Verfahren können im Prinzip abgesehen von technischen Anpassungen auf eine breite Klassen von Diagrammtypen angewandt werden. Darüber hinaus sind auch Verfahren vorhanden, die nur mit speziellen Diagrammtypen arbeiten.

In [XiS05] wird beispielsweise der Algorithmus UMLDiff vorgestellt, der ein spezieller Algorithmus ausschließlich für Klassendiagramme

ist. Im Gegensatz zu seiner Benennung ist er nicht auf andere UML-Diagrammtypen übertragbar, weil er auf Annahmen basiert, die nur in Klassendiagrammen erfüllt sind (selbst dort nicht immer), und auf spezifische Editieroperationen in Klassendiagrammen bezug nimmt. Weitere Beispiele derartiger spezieller Algorithmen sind [Gi02, RhW98]

4.2 Protokollbasierte Differenzbestimmung

Man kann Editoren für einen Dokumenttyp so konstruieren, daß sie Änderungskommandos an einem Dokument protokollieren und daß diese Protokolle für die Differenzbestimmung verfügbar gemacht werden. Änderungsprotokolle werden für die üblichen Undo- und Redo-Funktionen in Editoren ohnehin erstellt. Problematisch an der protokollbasierten Differenzbestimmung sind folgende Merkmale:

- Die Protokolle sind normalerweise auf den Zeitraum einer Sitzung beschränkt, was für viele Zwecke zu kurz ist. Will man längere Zeiträume abdecken, so muß das Protokoll zusammen mit dem eigentlichen Dokument persistent gespeichert werden. Alle im Laufe viele Sitzungen entstehenden Protokolle müssen geeignet integriert werden.
- Im Prinzip stellen die Protokolle asymmetrische Differenzen dar; diese können meist nur schwierig zu symmetrischen Differenzen konvertiert werden, die für die üblichen externen Darstellungen gebraucht werden.
- Editoren beinhalten oft komplexe Editieroperationen, für die externe Darstellung von Differenzen muß fast immer ein deutlich einfacheres Editiermodell benutzt werden. Das Änderungsprotokoll kann also nicht 1:1 dargestellt werden, sondern muß auf einfachere Operationen abgebildet werden. Hinzu kommen "Bereinigungen" dahingehend, daß Änderungen, die sich gegenseitig aufheben, gar nicht dargestellt werden sollten. Diese Vorverarbeitungen der Protokolle müssen dokumenttypspezifisch realisiert werden.
- Die Differenzwerkzeuge und *alle* Editoren für den jeweiligen Dokumenttyp müssen sich auf einen einheitlichen Satz von Operationen

in den Protokollen einigen und alle sonstigen Aspekte der Protokollierung einheitlich handhaben und verstehen. Dies widerspricht der generellen architektonischen Anforderung, daß eine Entwicklungsumgebung offen sein und die Integration neuer Werkzeuge nicht unnötig behindern sollte.

Eine Mischform zwischen zustandsbasierter und protokollbasierter Differenzbestimmung stellen Verfahren dar, die auf Sequenzen von Zuständen basieren. Zeitpunkte, an denen der Dokumentzustand protokolliert wird, können z.B. explizite check-ins in ein Repository sein, ferner jedes Speichern eines Dokuments. Die Protokolle können bzw. müssen hier durch das Repository-System oder eine Erweiterung des Dateisystems erstellt werden. Vorteilhaft ist, daß die Editoren nicht involviert sind und das Verfahren nicht von ihnen abhängt. Nachteilig ist, daß der hierzu nötige Eingriff in das unterliegende Dateisystem i.a. nicht möglich ist.

Wir werden i.f. protokollbasierte Verfahren nicht weiter betrachten.

4.3 Differenzbestimmung auf Basis von persistenten Identifizierern

Persistente Identifizierer bzw. Surrogate sind ein Konzept aus der Denkwelt objektorientierter Datenbanken und bedeuten, daß das DBMS jedem Objekt bei seiner Erzeugung einen Identifizierer zuweist, der während der gesamten Lebensdauer des Objekts unverändert bleibt und in der Objektbank nie wieder verwendet wird. Dieses Konzept kann auch bei Speicherung in Dateien imitiert werden, indem die Surrogate von Editoren vergeben werden und in versteckten Attributen oder in einer sonstigen Codierung in den Dateien gespeichert werden. Sofern nun

1. die Editoren die Surrogate bzw. persistenten Identifizierer erhalten (also nicht die Dokumente einlesen und später beim Schreiben neue Surrogate erzeugen), und
2. die zu vergleichenden Dokumente Revisionen voneinander sind,

können korrespondierende Dokumentteile einfach und äußerst effizient anhand gleicher Surrogate gefunden werden. Dieses Verfahren erfreut sich daher großer Beliebtheit und ist u.a. in [MaACM01, Oh04, OhK02] angewandt worden.

"Falsche" Korrespondenzen, also Korrespondenzen zwischen sehr unähnlichen Dokumentkomponenten, die man intuitiv nicht als korrespondierend empfinden würde, werden nur selten erzeugt, sind aber möglich. Wenn man Dokumente aus einer Revisionskette vergleicht, müßte eine Dokumentkomponente stark verändert worden sein, um diesen Fehler zu erzeugen. Wenn man zwei Dokumente aus verschiedenen Revisionssträngen vergleicht, sind falsche Korrespondenzen noch wahrscheinlicher, weil ein Dokumentelement aus der Ursprungsversion in beiden Strängen modifiziert werden kann.

Die Menge der sinnvollen Korrespondenzen wird nicht vollständig gefunden: Wenn ein Dokumentteil gelöscht und danach völlig identisch wieder manuell erzeugt wird, haben beide Teile verschiedene Surrogate, das surrogatbasierte Verfahren erkennt solche Korrespondenzen nicht. Wenn in zwei Revisionssträngen zufällig gleiche oder ähnliche Dokumentkomponenten erzeugt werden, bekommen diese Komponenten verschiedene Surrogate und bilden ebenfalls keine Korrespondenz. Wenn eine Komponente eines Dokuments kopiert wird, die Kopie i.w. unverändert bleibt und das Original stark verändert wird, die richtige Korrespondenz mit der Kopie nicht erkannt.

Letztlich sollte das surrogatbasierte Verfahren mit einem wertbasierten Verfahren kombiniert werden. Die Effizienzvorteile sind aber so hoch, daß Surrogate, sofern vorhanden, auf jeden Fall ausgenutzt werden sollten.

4.4 Identitätsbasierte Algorithmen

4.4.1 Mengen und unstrukturierte Multimengen

Grundalgorithmus zur Kandidatensuche: paarweiser Vergleich. Ein sehr simpler Algorithmus zur Suche nach Kandidaten für

Korrespondenzbildungen besteht darin, alle Paare von je einer Dokumentkomponente pro Dokument zu durchlaufen und deren Gleichheit zu prüfen oder Ähnlichkeit zu berechnen.

Hierzu durchläuft man alle Komponenten des ersten Dokuments und führt im Prinzip eine lineare Suche nach Kandidaten für Korrespondenzen, also gleichen bzw. ähnlichen Komponenten, im zweiten Dokument durch.

Hauptnachteil dieses Algorithmus ist, daß die Laufzeit quadratisch mit der Zahl der Dokumentkomponenten wächst.

Grundalgorithmus zur Kandidatensuche: Kandidatensuche mit Index. Analog zur Verbundberechnung mit Index in relationalen Datenbanken liegt es nahe, statt der linearen Suche eine Suchstruktur (z.B. Suchbäume oder binäre Suche in einem Array) einzusetzen, um gleiche Komponenten zu finden.

Da die Komponenten i.a. komplexe und umfangreiche Datenwerte sind, eignen sie sich nicht direkt dazu, als Schlüsselwert eingetragen zu werden. Die Lösung dieses Problems besteht darin, die Komponenten in **Hashwerte** umzurechnen und die Hashwerte in die Suchstruktur einzutragen. Bei einem ausreichend großen Zahlenraum (z.B. 64-Bit-Hashwerte) und einem guten Hash-Verfahren kann die Wahrscheinlichkeit von Kollisionen vernachlässigt werden.

Durch den Übergang zu Hashwerten werden auch die Datenvolumina in den vielen Fällen deutlich reduziert. Wir gehen jedenfalls davon aus, daß hauptspeicherbasierte Verfahren eingesetzt werden können und keine Zugriffe auf externe Speichermedien optimiert werden müssen. Insgesamt kann somit wie folgt verfahren werden:

1. für alle Komponenten beider Dokumente wird ein Hashwert bestimmt; Aufwand: O(n), n = Gesamtzahl der Komponenten
2. die Hashwerte des 2. Dokument werden in einen Suchbaum eingetragen oder in einem Array sortiert, der anschließend binär durchsucht werden kann; Aufwand: O(n*log(n))
3. alle Komponenten des 1. Dokument werden durchlaufen und es wird festgestellt, ob ein gleiches Element im 2. Dokument existiert; falls

ja, entsprechende Weiterverarbeitung, Aufwand: O(n*log(n))

Das Verfahren arbeitet insgesamt mit Aufwand O(n*log(n)) und ist damit bei großen Dokumenten wesentlich schneller als der paarweise Vergleich.

Eine Variante des Verfahrens besteht darin, die Hashwerte *beider* Dokumente in jeweils einem Array zu sortieren und dann beide Arrays parallel zu durchlaufen und nach gleichen Einträgen zu suchen. Dies entspricht genau der Kernidee des Misch-Verbunds (merge join) in relationalen Datenbanken. Die Größenordnung der Laufzeit ändert sich dabei nicht.

Das Index-Verfahren findet nur gleiche korrespondierende Komponenten, ist also primär ein identitätsbasiertes Verfahren.

Festlegung der Korrespondenzen. Nach der Kandidatensuche liegt in Prinzip eine Menge von Kandidatenpaaren vor; es muß nun entschieden werden, welche davon tatsächlich als Korrespondenzen bestimmt werden. Hierbei können je nach Dokumenttyp zwei Probleme auftreten:

- beim vorliegenden Dokumenttyp sind Dubletten möglich, Dokument 2 enthält die Dubletten y1 und y2, und beide sind identisch mit Komponente x aus Dokument 1. Es liegen also die Kandidatenpaare (x,y1) und (x,y2) vor, von denen nur eines gewählt werden kann.
- Die Komponentenmengen sind geordnet (allgemeiner: sind irgendwie strukturiert) und die Korrespondenzen "überkreuzen" einander (allgemeiner: sind nicht verträglich mit der Struktur)

Bei Dokumenten mit *Mengenstruktur* können die vorstehenden Probleme nicht auftreten, alle Kandidatenpaare können daher sofort als Korrespondenzen festgelegt werden.

Bei *Multimengen* kann nicht nur die Kandidatensuche mehrere Treffer liefern, es können auch im ersten Dokument Dubletten des gleichen Werts vorhanden sein. Daher sollten immer alle gleichen Werte in

beiden Dokumenten en bloc behandelt werden. Das oben erwähnte array-basierte Verfahren ist hier besonders vorteilhaft, denn die Dubletten stehen nach der Sortierung leicht erkennbar hintereinander in den beiden Arrays und können bei als Gruppe verarbeitet werden. Statt zweier Arrays kann man generell zwei Suchstrukturen verwenden, bei denen man die vorhandenen Einträge linear in aufsteigender Reihenfolge der Schlüsselwerte durchlaufen kann.

Bei einer unstrukturierten Multimenge werden die Komponenten, die in jedem Dokument genau einmal auftreten, als korrespondierend festgelegt (wie bei der Menge). Sofern Dubletten als potentielle Korrespondenzpartner vorliegen, kann eine beliebige ausgewählt werden. Auch hier ist also die Festlegung der Korrespondenzen ziemlich trivial.

Bei *strukturierten Multimengen* ist die Festlegung der Korrespondenzen viel stärker eingeschränkt, oben wurden schon sich "überkreuzende" Korrespondenzen als Beispiel genannt. Generell kann es sein, daß ein erheblicher Teil der Kandidaten für Korrespondenzen, die bei der Kandidatensuche gefunden werden, nicht akzeptiert werden kann. In solchen Fällen ist die obige "flächendeckende" Kandidatensuche nicht mehr immer sinnvoll. Es gibt in der Tat Differenzalgorithmen, die die Kandidaten auf völlig andere Art bestimmen (z.B. der nachfolgend beschriebene LCS-Algorithmus).

Es liegt nahe, die Struktur der Komponentenmenge auszunutzen, um die Zahl der zu analysierenden Komponentenpaare zu reduzieren. Hierzu müssen aber zumindest für Teile der Dokumente schon Korrespondenzen festgelegt worden sein, d.h. die Kandidatensuche und Festlegung der Korrespondenzen sind nicht mehr klar getrennt.

Die *Qualität der Differenzen* ist bei Mengen und Multimengen kein Thema, weil das einzig naheliegende Qualitätsmaß ein möglichst großer Durchschnitt ist und weil die obigen Verfahren nur in diesem Sinne optimale Differenzen liefern. Wie schon in Abschnitt 4.1 erwähnt ist bei nichttrivialen Dokumentstrukturen die Optimierung der Differenz ein wichtiges Thema.

4.4.2 Sequentiell strukturierte Dokumente

Das wichtigste Verfahren zur Bestimmung der Differenz zwischen zwei Texten ist der LCS- (Longest Common Subsequence) Algorithmus [My86]. Der Algorithmus bestimmt die längste gemeinsame Teilsequenz in zwei Zeichenketten. Die gemeinsame Teilsequenz kann aus beliebig vielen Abschnitten bestehen. Der Algorithmus berechnet zugleich die Positionen dieser Abschnitte in den beiden Texten. Die gemeinsame Teilsequenz stellt den Durchschnitt der beiden Texte dar, die restlichen Teile die speziellen Komponenten, es liegt also offensichtlich eine symmetrische Differenz vor. Aus den Daten, die bei dem Verfahren intern vorliegen, können aber auch direkt asymmetrische Differenzen gewonnen werden.

Der LCS-Algorithmus kann nicht nur auf Sequenzen von Zeichen, sondern allgemein auf Sequenzen von beliebigen Datenobjekten angewandt werden (z.B. ganze Textabschnitte oder Parameterlisten); Voraussetzung ist lediglich ein effizienter Test, ob zwei Datenobjekte gleich sind.

Verschiebungen werden nicht direkt erkannt. Anders gesagt enthält der unterstellte Editierdatentyp nur das Einfügen bzw. Löschen eines Textstücks, keine Verschiebungen.

Man kann trotzdem Verschiebungen mit wenig Mehraufwand erkennen: Wenn man die beiden Listen mit dem LCS-Algorithmus vergleicht, erhält man die beiden Mengen spezieller Komponenten. Diese sind nun paarweise auf Gleichheit zu prüfen: Paare gleicher Komponenten stellen Verschiebungen dar. Bei Dubletten sind die Zuordnungen nicht eindeutig.

Die Laufzeit des LCS-Algorithmus hat Größenordnung O(ND), worin N die Gesamtlänge der Sequenzen ist und D in etwa der Zahl der speziellen Komponenten entspricht. Der Algorithmus ist daher im besonders häufigen Fall, daß nur geringfügige Unterschiede zwischen den Dokumenten bestehen, sehr effizient.

4.4.3 Baumstrukturierte Dokumente

XML-Dateien, viele UML-Diagrammtypen, LaTeX-Texte und weitere Diagrammtypen können in erster Näherung als Bäume mit typisierten Knoten angesehen werden. Wir werden anschließend gewisse "Ausnahmen" von der Baumstruktur diskutieren. Die Baumstruktur entspricht der Teil-von-Struktur der Dokumente. Jeder Knoten bzw. der Unterbaum, dessen Wurzel der Knoten ist, repräsentiert einen Teil des Dokuments.

Die prinzipiellen Probleme, Differenzen von komplex strukturierten Dokumenten inhaltlich zu definieren, und die entstehenden Begriffsvarianten wurden schon in den Abschnitten 2.3.5 und 2.3.6 diskutiert. Diese Varianten übertragen sich natürlich auf die Berechnungsverfahren. In Abschnitt 2.3.6 wurden zwei Definitionen genannt, wann man zwei Dokumentkomponenten, die durch je einen Baumknoten repräsentiert werden, als identisch definieren kann:

- wenn alle lokalen Merkmale beider Knoten gleich sind bzw.
- wenn die kompletten Teilbäume, dessen Wurzel die Knoten sind, identisch sind.

Da wir hier nur identitätsbasierte Algorithmen betrachten, brauchen wir nur diese beiden "Extremfälle" zu betrachten, dazwischenliegende Fälle führen zu ähnlichkeitsbasierten Verfahren, die wir später diskutieren.

Kandidatensuche. In beiden Fällen können für die Kandidatensuche die Grundalgorithmen für Multimengen übernommen werden, wobei nur die Varianten sinnvoll sind, die zunächst Hashwerte der Dokumentkomponenten bilden. Bei der Berechnung dieses Hashwerts sind folgende Fälle zu unterscheiden:

- Wenn nur die lokalen Merkmale für die Identität relevant sind, dürfen natürlich nur diese in den Hashwert einfließen.
- Wenn komplette Teilbäume für die Identität relevant sind, müssen alle lokalen Merkmale und alle Merkmale aller Teilbäume in den

Hashwert einfließen. Sinnvollerweise berechnet man zuerst die Hashwerte alle Teilbäume und läßt diese Werte dann in den Gesamt-Hashwert einließen. Auf diese Weise können mit einem einzigen **bottom-up-Durchlauf** durch den Baum alle Hashwerte effizient berechnet werden: man berechnet erst die Hashwerte der Blätter, dann aufsteigend zur Wurzel die Hashwerte derjenigen inneren Knoten, bei denen schon für alle Teilbäume die Hashwerte vorliegen.

Bei *geordneten Bäumen* muß der Hashwert zusätzlich von der Reihenfolge der Kindknoten abhängen, z.B. indem ein Kriterium, das die Ordnung der Unterbäume darstellt, in die Berechnung der Hashwerte einfließt[23].

Bei *ungeordneten Bäumen* darf die Ordnung, nach der die Unterbäume zufällig gerade intern verwaltet werden, nicht in die Hashwerte einfließen[24].

Generell dürfen bei der Berechnung der Hashwerte zufällige Details der internen Darstellung der Dokumentkomponenten keine Rolle spielen, sonst bekämen konzeptuell gleiche Dokumentkomponenten unterschiedliche Hashwerte.

Festlegung von Korrespondenzen. Die Kandidatensuche liefert nunmehr Paare von Knoten in beiden Dokumentbäumen, die gemäß der gewünschten Identitätsvariante identisch sind. Diese Paare können aber noch nicht ohne weiteres als Korrespondenzen festgelegt werden, denn die beteiligten Knoten können an ganz verschiedenen Stellen der Bäume liegen. Wenn ein Teilbaum innerhalb des Dokuments verschoben wird und der alte und neue Zustand verglichen werden, dann ist der Teilbaum jeweils identisch vorhanden, aber der Elternknoten ist jeweils ein anderer; wenn umgekehrt beim Vergleich zweier Dokumente zwei identische Teilbäume gefunden werden, die nicht korrespondierende Elternknoten haben, kann man das so interpretieren, daß dieser Teilbaum verschoben worden ist.

[23] Der LaDiff-Algorithmus [ChRGW96] ist ein Beispiel für einen Algorithmus, der für geordnete Bäume gedacht ist.

[24] Algorithmen für ungeordnete Bäume sind u.a. MHDiff [ChG97], XDiff [WaDC03] und XyDiff [CoAM02].

Verschiebungen können bei vielen Dokumenttypen nicht sinnvoll extern dargestellt werden und sind dann nicht zulässig. Stattdessen wird man oft fordern, daß nur solche Komponenten korrespondieren, deren Elternkomponenten ebenfalls korrespondieren. Anders gesehen erzwingt die Korrespondenz der Elternkomponenten die Korrespondenzen der Kinder.

Setzt man diese Forderung rekursiv im Baum bis zur Wurzel fort, ergibt sich als Forderung, daß *die Wurzeln der beiden Bäume korrespondieren* müssen! Dies ist ohnehin das einzig naheliegende, denn der Sachverhalt, daß man zwei Dokumente vergleicht, unterstellt bereits, daß diese beiden Dokumente einander entsprechen. Wenn aber per Definitionem die Wurzelknoten der beiden Dokumente zueinander korrespondieren, dann kann zumindest dieses Korrespondenz nicht bedeuten, daß die kompletten Teilbäume - in diesem Fall die ganzen Dokumente - gleich sind. Im Endeffekt ergibt sich, daß man die *Wurzelknoten und i.d.R. auch die oberen Ebenen der Baumstruktur als Ausnahmen von der Regel behandeln* muß und speziell bei diesen Knoten eine Korrespondenz nur ausdrücken kann, daß die lokalen Merkmale gleich sind oder daß typspezifische Ähnlichkeitskriterien (die wir erst später behandeln werden) erfüllt sind.

Die Erkenntnis, daß man in baumstrukturierten Dokumenten rein identitätsbasierte Verfahren nicht durchgängig ohne Ausnahmen nutzen kann, sollte nicht zu dem Fehlschluß verleiten, sie seien überflüssig und man bräuchte nur ähnlichkeitbasierte Verfahren zu betrachten. Letztere sind wesentlich ineffizienter und meist nur dokumenttypspezifisch definierbar. Die sinnvolle Kooperation beider Verfahrenstypen liegt auf der Hand: zunächst sollten Korrespondenzen zwischen identischen Komponenten mit einem effizienten identitätsbasierten Verfahren gesucht werden, die dann verbliebenen Komponenten können mit einen ähnlichkeitsbasierten Verfahren behandelt werden.

4.4.3.1 Typisierung

In den meisten Fällen sind die Dokumentbäume typisiert, d.h. die Knoten und ggf. Kanten haben Typen. Generell können nur Knoten

gleichen Typs miteinander korrespondieren.

Die beiden Grundalgorithmen zur Kandidatensuche, die alle Knoten miteinander vergleichen, können leicht an diese Situation angepaßt werden: für jeden Knotentyp werden die beiden Mengen der Knoten dieses Typs miteinander verglichen. Statt zweier großer Mengen werden also mehrere Paare kleinerer Mengen verglichen. Dies ist im Prinzip sogar günstig für die Laufzeiten bei großen Modellen, insb. bei dem ineffizienten paarweisen Vergleich. Beim Einsatz von Indexen bzw. sortieren Arrays ist dagegen nicht mit nennenswerten Vorteilen zu rechnen.

Allgemeine Graphalgorithmen, die in zwei untypisierten Graphen nach möglichst großen isomorphen Teilbäumen suchen, sind zur Bestimmung von Differenzen zwischen typisierten Bäumen (oder Graphen) nicht geeignet, weil sie unzulässige Korrespondenzen liefern können und weil ihre Laufzeit wesentlich ungünstiger ist.

4.4.3.2 Lokale Verschiebungen in geordneten Bäumen

In Softwaredokumenten treten Dokumentteile auf, deren direkt untergeordnete Komponenten die Struktur einer Liste haben, also geordnet sind, z.B. die Parameter einer Parameterliste. I.f. bezeichnen wir solche Mengen von Kindkomponenten als **geordnete Kinder**.

Ob die Kinder einer Komponenten geordnet sind oder nicht, hängt meist vom Typ einer Komponente ab; jedenfalls können im gleichen Dokument sowohl Komponenten mit geordneten wie ungeordneten Kindern vorkommen. Sofern die Kinder bestimmter Komponenten geordnet sind, ist dies eine Eigenschaft, die über die reine Baumstruktur hinausgeht.

Bei geordneten Kindern ist eine Änderung der Ordnung eine Änderung, die wir als **lokale Verschiebung** bezeichnen.

Man kann Verschiebungen in Sequenzen relativ leicht mit dem erweiterten LCS-Algorithmus erkennen (s. Abschnitt 4.4.2).

Oft ist eine Differenz mit einer Verschiebung qualitativ besser als eine Differenz mit einer Löschung und Einfügung der verschobenen Komponente, z.B. wenn man sie z.B. extern sinnvoll darstellen kann oder wenn eine asymmetrische Differenz abgeleitet werden soll. Dies

hängt vom Dokumenttyp und Komponententyp ab.

4.4.3.3 Nichtlokale Verschiebungen

Beispiele für nichtlokale Verschiebungen sind das Verschieben einer Operation oder eines Attributs von einer Klasse in eine andere oder das Verschieben einer Klasse von einem Paket in ein anderes.

Nichtlokale Verschiebungen können auch in *ungeordneten Bäumen* auftreten.

Wie schon bei lokalen Verschiebungen ist es oft sinnvoll, aber nicht immer, nach nichtlokalen Verschiebungen zu suchen.

Speziell bei kleinen Dokumentteilen, ist es zwar technisch möglich, aber nicht sinnvoll, eine Verschiebung anzunehmen, weil die Darstellung der Differenz zu unübersichtlich wird und weil es nicht ganz unwahrscheinlich ist, daß der verschobene Teil tatsächlich gelöscht und neu eingefügt worden ist.

Wenn nach Verschiebungen gesucht wird, erhöht sich i.a. der Rechenaufwand signifikant, denn es ergeben sich deutlich mehr Alternativen, aus den Kandidatenpaaren Korrespondenzen bilden. Für die Auswahl benötigt man Qualitätskriterien. Letztlich gewinnt die Frage, wie die Qualität von Differenzen definiert und praktisch überprüft wird, erheblich an Bedeutung, wenn auch Verschiebungen erkannt werden sollen.

Verschobene Dokumentteile können natürlich auch "leicht modifiziert" sein, oder Teile können weiterverschoben worden sein. Diesen Fall diskutieren wir unter den ähnlichkeitsbasierten Verfahren.

4.4.3.4 Kopien

Analog zu Verschiebungen kann man auch fordern, daß der Effekt von *Kopieroperationen* erkannt werden soll. Dieser stellt sich so dar, daß z.B. im zweiten Dokument mehrere Komponenten identisch mit einer einzigen Komponente des ersten Dokuments sind und sonst keinen passenden Korrespondenzpartner haben. Höchstens einer dieser Kopien kann einen Elternknoten haben, der mit dem Elternknoten des Originals korrespondiert, alle anderen haben nicht korrespondierende Elternknoten und ähneln insofern Verschiebungen.

Bei sehr kleinen Dokumentteilen lohnt sich die Suche nach Kopien, ähnlich wie bei Verschiebungen, kaum, es ist auch unklar, ob die Kopie durch eine Kopieroperation erzeugt wurde oder von Hand neu angelegt wurde; bei größeren Dokumentteilen werden identische Kopien kaum auftreten[25].

Insgesamt ist es fraglich, ob das Erkennen von Kopien deutliche praktische Vorteile bringt. Auf jeden Fall nachteilig ist die kompliziertere externe Darstellung der Differenzen.

4.4.3.5 Dubletten

Bei reinen Multimengen ist die Behandlung von Dubletten trivial, weil ohne Qualitätsverluste beliebige Korrespondenzen eingerichtet werden können. In baumartig strukturierten Dokumenten werfen Dubletten die gleichen Probleme wie nichtlokale Verschiebungen und Kopien auf.

Das Dublettenproblem tritt unabhängig davon auf, ob für die Identität von Knoten nur die lokalen Merkmale oder der ganze Teilbaum relevant ist.

Bei Klassendiagrammen und diversen anderen Dokumenttypen sind Dubletten ein erhebliches praktisches Problem, weil sie massenhaft auftreten. Das Musterbeispiel hierfür sind in Klassendiagrammen die Datentypen, die als Parameter von Operationen einer Klasse oder als Typ von Attributen auftreten. Diese Komponenten sind zwar je nach Modellierung i.d.R. nur Blätter der Baumstruktur, also sehr kleine Teilbäume, das ändert aber nichts an der prinzipiellen Problematik.

Dubletten sind äußerst lästig für die Differenzberechnung, weil man identische Teilbäume nicht sofort einander zuordnen kann: theoretisch kann jede Dublette des ersten Dokuments mit jeder gleichen Dublette des zweiten Dokuments korrespondieren.

Algorithmen, die Dubletten nicht sinnvoll behandeln, sind daher ungeeignet für den Vergleich von Softwaredokumenten. Ein Beispiel hierfür ist LaDiff [ChRGW96]; dieser Algorithmus ist für LaTeX-Quelltexte konzipiert. In normalen LaTeX-Quelltexten sind komplett

[25] Beim modifizierten Kopien können nur ähnlichkeitsbasierte Verfahren benutzt werden, die wie später diskutieren.

identische Textabschnitte äußerst unwahrscheinlich.

4.4.4 Fast baumartig strukturierte Dokumente und Referenzen

Bei genauem Hinsehen stellt man fest, daß Klassendiagramme, Zustandsautomaten und viele andere Diagrammtypen eine Struktur haben, die punktuell die Baumstruktur durchbricht. Beispiele hierfür sind Attribute oder Parameter, deren Typ eine Klasse ist. Derartige Attribute sind eigentlich im Sinne eines Fremdschlüssels als **Referenz** auf eine andere Klasse bzw. i.a. auf andere Diagrammelemente zu interpretieren. Solche Referenzen können auf zwei verschiedene Arten interpretiert werden:

- Die Referenz, z.B. der Typ eines Parameters, wird als Text bzw. autarker *Datenwert* verstanden. Entscheidend ist nur dieser Wert, nicht der Zustand der referenzierten Dokumentkomponente, die sich beliebig stark verändern darf. Für die schon erwähnten Parametertypen ist diese Interpretation meist passend.
- Die Referenz wird als "*Stellvertreter*" der referenzierten Dokumentkomponente verstanden. Der konkrete Wert der Referenz spielt keine Rolle, entscheidend ist nur die Gleichheit der Werte in der Referenz und in einem entsprechenden Attribut der Zielkomponente. Zwei Referenzen sind hier identisch, wenn die beiden Zielkomponenten korrespondieren.

  Diese Interpretation ist nahezu zwingend, wenn die Referenz graphisch und nicht textuell (also über einen Wert) ausgedrückt sind. Beispiel ist das Sachverhalt, daß eine Klasse eine Rolle in einem Beziehungstyp spielt; in Klassendiagrammen wird dies durch eine Linie ausgedrückt,,

Die Referenzen auf Dokumentkomponenten werden bei einer sinnvollen Modellierung mehr oder minder direkt durch technische Referenzen realisiert werden, so daß sich die beiden Interpretationsmöglichkeiten auch auf die technische Ebene übertragen. Es ist aber zu betonen, daß diese Ambivalenz der Interpretation unabhängig von der

Speicherungsform besteht, sie besteht auch dann, wenn die Diagramme nur auf dem Papier stehen.

Derartige Referenzen haben *keine "Teil-von"-Semantik im Editierdatentyp*, d.h. beim Löschen einer Operation, die einen Parameter hat, dessen Typ eine andere Klasse ist, wird die andere Klasse natürlich nicht gelöscht. Im Sinne der "Teil-von"-Semantik des Editierdatentyps durchbrechen die Referenzen daher die Baumstruktur nicht.

Bei der Interpretation der Referenzen als Werte können diese wie normale Attribute behandelt werden und machen keine Probleme.

Bei der Interpretation als Stellvertreter treten dagegen erhebliche Probleme auf:

1. Die referenzierten Dokumentteile - in unserem Beispiel Klassen - liegen in der Teil-von-Hierarchie höher als die referenzierende Komponente; somit kann die grundlegende Vorgehensweise, von den Blättern der Teil-von-Baumstruktur ausgehend Hashwerte zu bestimmen, hier nicht aufrechterhalten bleiben.

   Sofern die Teil-von-Baumstruktur zusammen mit den Referenzen wenigstens noch eine Halbordnung ist, können immer noch mit einer entsprechend angepaßten Steuerung alle Hashwerte in einem Durchlauf berechnet werden.

2. Es sind *Zyklen* möglich; z.B. kann die referenzierte Klasse eine Operation mit einem Parameter enthalten, dessen Typ die Klasse ist, in der die aktuell betrachtete Referenz liegt.

   Diese Zyklen brauchen bei der Berechnung der Hashwert nicht durchlaufen zu werden, d.h. sobald ein Zyklus geschlossen würde, kann die entsprechende Dokumentkomponente einfach ignoriert und der Zyklus insofern an dieser Stelle aufgebrochen werden. Die aufgebrochenen Zyklen können aber auf komplizierte Weise überlappen, und die an den Komponenten befindlichen Hashwerte hängen davon ab, wo ein Zyklus aufgebrochen wurde, d.h. der Hashwert für eine Komponente muß *mehrfach* berechnet werden. Ferner entstehen erhebliche Aufwände für die Zyklustests. Direkt für den Gleichheitstest verwendbar sind nur die Hashwerte, die sich bei der Berechnung ergeben, die im jeweiligen Knoten startete. Insgesamt

ist mit einem vielfach höheren Berechnungsaufwand im Vergleich zu halbgeordneten Strukturen zu rechnen.

"Entfernte Ähnlichkeitsmerkmale". Das oben diskutierte Problem der Behandlung von Referenzattributen kann man auch anders auffassen: wenn man z.B. je ein Attribut zweier Klassen vergleicht, dann ist der Typ eines Attributs für die Ähnlichkeitsbestimmung relevant, aber nicht in dem Teilbaum enthalten, der das Attribut repräsentiert; dieses Merkmal liegt "entfernt" und ist nur über die Referenz erreichbar.

Dieses Problem tritt in Diagrammtypen wie Petri-Netzen, Anwendungsfalldiagrammen u.a. verschärft auf: wenn man die durch die UML-Metamodelle vorgegebene Modellierung unterstellt, haben viele Diagrammelemente in diesem Modelltypen nur wenige lokale Ähnlichkeitsmerkmale und stattdessen viele entfernte, teilweise sehr kompliziert zu erreichende Ähnlichkeitsmerkmale.

4.5 Ähnlichkeitsbasierte Algorithmen

Grundannahme aller Algorithmen in Abschnitt 4.4 war, daß nur identische Komponenten korrespondieren können. Diese Annahme ist zwar restriktiv, aber für viele Anzeigeverfahren notwendig. Unter anderem bei der Diskussion von baumartig strukturierten Dokumenten wurde andererseits klar, daß trotz mehrerer denkbarer Identitätsdefinitionen keine völlig zufriedenstellend ist und man letztlich auf ähnlichkeitsbasierte Definitionen übergehen muß.

4.5.1 Ähnlichkeitsmaße

Bei ähnlichkeitsbasierten Algorithmen können Komponenten korrespondieren, wenn Sie eine ausreichende Ähnlichkeit aufweisen. Diese informelle Definition führt zu den Folgefragen, wie die Ähnlichkeit definiert und wann die Ähnlichkeit ausreichend hoch ist.

Während die Gleichheit von Komponenten sozusagen ein "generischer" Begriff ist, weil er für alle Dokument- und Komponententy-

pen immer implizit als definiert gilt, müssen Ähnlichkeitsdefinitionen – oder anders gesagt **Ähnlichkeitsmaße** – für *jeden Dokument- und Komponententyp individuell definiert* werden. Ein Ähnlichkeitsmaß ordnet zwei Dokumentkomponenten eine Zahl zwischen 0 und 1 zu. Identische Komponenten müssen Ähnlichkeitsmaß 1 erhalten.

In einfachsten Fall werden zwei Komponenten als Kandidaten für eine Korrespondenz eingestuft, wenn ihre Ähnlichkeit einen Wert überschreitet, der global für den Komponententyp festgelegt ist. Es gibt auch flexiblere und kompliziertere Definitionen, wann zwei Komponenten als Kandidaten für eine Korrespondenz eingestuft werden.

Auf jeden Fall müssen die Ähnlichkeitsmaße und die Verfahren zur Bildung von Kandidaten zusammenhängend betrachtet werden, letztlich kommt es im Endeffekt nur darauf an, welche Korrespondenzen gebildet worden sind und ob diese aus Benutzersicht sinnvoll sind (s. Abschnitt 4.6).

4.5.2 Mengen und unstrukturierte Multimengen

Von den beiden identitätsbasierten Grundalgorithmen zur Kandidatensuche ist (leider) nur der ineffiziente paarweise Vergleich auf den ähnlichkeitsbasierten Fall übertragbar: statt die Gleichheit zweier Komponenten zu prüfen, wird deren Ähnlichkeit berechnet. Im einfachsten Fall wird nur eine ggf. komponententypspezifische untere Schranke angesetzt, ab der alle Komponentenpaare als Kandidaten gelten.

Die effizientere Suche nach Kandidaten mittels Index oder sortiertem Array scheitert daran, daß sich die Ähnlichkeit von Komponenten nicht ohne weiteres auf eine lineare Skala abbilden läßt.

Festlegung der Korrespondenzen und Qualitätsmaße für Differenzen. Während bei identitätsbasierten Grundalgorithmen nur in Ausnahmefällen (bei Dubletten) eine Wahlmöglichkeit bzw. Entscheidungsbedarf bestand, Korrespondenzen festzulegen, können derartige Entscheidungssituationen viel leichter auftreten, sofern es reicht, wenn Korrespondenzparter ähnlich sind.

Verschärft wird das Problem dadurch, daß die bildbaren Mengen von Korrespondenzen unterschiedlich gut sein können, weil die Ähnlichkeitsmaße der Kandidatenpaare unterschiedlich sind[26]. Generell ist eine Differenz umso besser, je ähnlicher die korrespondierenden Komponenten sind. Aus Sicht jeder einzelnen Komponente liegt nach der Ähnlichkeitsberechnung eine (Präferenz-) Liste ähnlicher Komponenten im anderen Dokument vor. Die Aufgabe ist nun ähnlich wie bei dem aus der Graphentheorie bekannten Heiratsproblem, eine global optimierte Paarbildung vorzunehmen. Im Gegensatz zum Heiratsproblem ist es hier aber durchaus vertretbar, einige Komponenten "ledig" zu lassen. Außerdem können die Qualitätsmaße für Differenzen andere Aspekte als nur die Durchschnittsähnlichkeit der korrespondierenden Paare heranziehen.

Allgemeiner gesagt liegt hier ein Optimierungsproblem vor, aus der i.a. sehr großen Menge von bildbaren Differenzen auf Basis eines Qualitätsmaßes für Differenzen eine möglichst gute Differenz zu wählen.

4.5.3 Baumstrukturierte und graphartige Dokumente

Für baumstrukturierte Dokumente bzw. Dokumente mit allgemeiner Graphstruktur bietet sich für die Kandidatensuche wieder die schon bekannte paarweise Ähnlichkeitberechnung an, ggf. zusätzlich nach Komponententypen gruppiert.

Ein wesentlicher Unterschied zu Multimengen sind die Ähnlichkeitsmaße, bei deren Definition die Baumstruktur ausgenutzt werden kann. Während bei identitätsbasierten Verfahren i.w. nur zwei Interpretationsmöglichkeiten für Korrespondenzen vorlagen (identische Wurzelknoten und identische Teilbäume, s. Abschnitt 4.4.3), können hier Ähnlichkeitsbegriffe benutzt werden, die sich auf beliebige über die Baum- oder Graphstruktur erreichbare Knotenmengen beziehen.

Metamodellbasierte Ähnlichkeitsmaße. In vielen Fällen definiert man die Ähnlichkeit von Komponenten dadurch, daß man

[26] Bei identitätsbasierten Verfahren tritt das Problem nicht auf, alle bildbaren Differenzen sind gleich gut.

- bei elementaren Attributen (also Blättern des Baums) typspezifische Ähnlichkeitsmaße benutzt,
- für innere Knoten des Baums die Kindknoten gewichtet und den gewichteten Durchschnitt der Einzelähnlichkeiten bildet.

Im einfachsten Fall gewichtet man alle Kinder der inneren Knoten gleich und muß dann nur noch die Menge der Kindknoten pro Dokumentelementtyp kennen. Diese wiederum kann man in einfachen Fällen aus einem Schema (oder Metamodell) der Dokumente entnehmen, d.h. man erhält im Idealfall ein Ähnlichkeitsmaß, ohne weitere Details von Hand angeben zu müssen. Speziell bei der UML bietet sich dieses Vorgehen wegen der vielen Diagrammtypen an und wird vielfach angewandt [KeWN05, OhK02, OhWK03a, OhWK03b, Oh04, Se03, XiS05].

Derartige Maße sind aber nicht besonders gut, weil die Attribute in der Realität eben nicht gleichwichtig sind. Beispielsweise ist für die Ähnlichkeit zweier Klassen deren Name äußerst relevant, die Eigenschaft, abstrakt zu sein, dagegen nur wenig relevant. Im Endeffekt führt eine Gleichgewichtung nicht zu sinnvollen Ergebnissen, und man muß zumindest eine individuelle Gewichtung einstellen können (wie z.B. im SiDiff-System [KeWN05, Si07]). Dies ist indessen mit erheblicher Handarbeit verbunden, d.h. der ursprüngliche Reiz metamodellbasierter Ähnlichkeitsmaße, "vollautomatisch" verfügbar zu sein, geht schon hier weitgehend verloren.

Noch problematischer wird die metamodellbasierte Definition (und Implementierung) von Ähnlichkeitsmaßen, wenn entfernte Attribute relevant sind. Unter **entfernten Attributen** seien hier Daten verstanden, die zwar für die Ähnlichkeit eines Komponententyps relevant sind, die aber nicht innerhalb des Teilbaums liegen, der eine Komponente repräsentiert. Musterbeispiel hierfür sind Zustände in Zustandsübergangsdiagrammen; für deren Ähnlichkeit ist vor allem der Kontext relevant, also die ausgehenden Zustandsübergänge und die dadurch erreichbaren Zustände.

Bei der Ableitung von Ähnlichkeitsmaßen aus Metamodellen kann man Teil-von-Beziehungstypen ausnutzen, um die Baumstruktur zu

erkennen. Die Beziehungstypen, die zu entfernten Attributen führen, können keine Teil-von-Semantik haben, weil sie i.d.R. zu Zyklen führen können.

Ähnlichkeitsberechnung in zyklischen Graphen. In den meisten Fällen entfernter Ähnlichkeitsattribute können auch Zyklen auftreten. Dies bedeutet bei den o.g. Algorithmen, daß in die Berechnung der Ähnlichkeit zweier Komponenten indirekt deren Ähnlichkeit wieder einfließt. In solchen Fällen müssen iterative Verfahren eingesetzt werden, die Ähnlichkeiten in mehreren Iterationsrunden im Graphen verbreiten.

Ein derartiges Verfahren wird in [Me02] unter der Bezeichnung der Similarity-Flooding-Algorithmus vorgestellt. Der Algorithmus ist primär für den Vergleich von Datenbankschemata konzipiert worden. Er ermittelt zuerst Ähnlichkeiten von Knotenpaaren ausgehend von den Blättern und propagiert diese dann iterativ in den Kontext der Knoten. Überraschenderweise werden auch Ähnlichkeiten von größeren Dokumentteilen zu deren Bestandteilen propagiert. Zyklen in der Graphstruktur sind erlaubt, allerdings konvergiert der Algorithmus dann nicht immer. Die Ergebnisse hängen ferner von einer eher zufällig bestimmten initiale Menge von Korrespondenzen ab.

Die erwähnten Probleme deuten an, daß eine effiziente Ähnlichkeitsberechnung in zyklischen Graphen außerordentlich anspruchsvoll ist.

4.6 Qualität von Differenzen

Wie schon früher mehrfach erwähnt ist die Differenz zwischen zwei Dokumenten (in Sinne eines Edit-Skripts, analog bei bei allen Varianten dieses Begriffs) nicht eindeutig bestimmt. Wenn man "dumme" Differenzen ("lösche das 1. Dokument; erzeuge das 2. Dokument komplett neu") hinzunimmt, kommt sogar eine enorm große Anzahl an Differenzen zustande. Ernsthaft betrachten wird man aber nur Differenzen, die bestimmte **Qualitätskriterien** erfüllen. Die beinhalten immer bestimmte *quantitative Minimalitätsanforderungen*, z.B.

- eine möglichst kleine Zahl von Einzeloperationen im Edit-Skript, die sich gegenseitig kompensierende Kommandos ausschließt, und
- möglichst kleine Parameter, z.B. bei Textdokumenten gemessen in der Länge der einzufügenden oder zu löschenden Texte

Es liegt nahe, zur Bestimmung der Qualität einer Differenz alle Angaben, aus denen die Differenz (im Sinne der Definitionen von Kapitel 2) besteht, heranzuziehen und auf ein Kostenmaß abzubilden. Diese eindimensionale Betrachtung der Qualität von Differenzen ist aber nicht immer angemessen:

- Die Kosten unterschiedlicher Editieroperationen und die Merkmale der Korrespondenzenmenge sind nicht immer direkt vergleichbar (was ist besser: eine Korrespondenz mehr oder 2 Verschiebeoperationen weniger?) Letztlich liegt hier ein multidimensionales Optimierungsproblem vor, es gibt nicht nur eine einzige optimale Lösung, sondern i.a. eine Gruppe Pareto-optimaler Lösungen.
- Die bisherigen numerischen Qualitätsgrößen haben den Vorteil, automatisch berechenbar zu sein, und suggerieren Objektivität. In Wirklichkeit ist das einzig wirkliche Maß für die Qualität einer Differenz, wie gut sie einem Betrachter die Unterschiede zwischen zwei Dokumenten vermittelt und wie sehe sie damit zusammenhängende Arbeitsschritte erleichtert. Damit zusammen hängt auch die graphische Darstellbarkeit, weswegen man i.d.R. Verschiebungen, die zu kleineren Differenzen führen, mit einem erheblichen Malus versehen muß.

  Die meisten *qualitativen Kriterien* hängen stark vom Dokumenttyp ab, müssen also individuell für den jeweiligen Dokumenttyp formuliert werden.

Die Berechnungsverfahren beinhalten i.d.R. nur eine implizite "Optimierung" der Qualität der Differenzen, d.h. die Berechnungsalgorithmen beinhalten Heuristiken, die fast alle schlechten oder sehr schlechten Differenzen von vorneherein ausschließen. Nicht garantiert wird damit, daß die gefundene Differenz unter den denkbaren eine besonders gute oder sogar die beste ist. Nur in Ausnahmefällen kann man

die Verfahren bzw. Werkzeuge dahingehend steuern, auf Kosten von erhöhter Rechenzeit nach qualitativ besseren Differenzen zu suchen. In vielen Fällen lohnt eine Optimierung nicht, weil die Qualität der gefundenen Differenzen in der Praxis ausreicht. Ein zentrales Problem bleibt ferner die fehlende oder unscharfe Definition, wann eine Differenz besser als eine andere oder sogar optimal ist.

4.7 Interaktive Korrektur von Differenzen

Die durch einen Algorithmus gefundene Differenz ist nicht immer zufriedenstellend (oder sogar optimal). Als Beispiel betrachten wir ein kurzes Shell-Skript, das die Definition einer Funktion `t1` enthält. Diese Funktion ist kopiert und in `t2` umbenannt worden, die bisherige Funktionsdefinition ist leicht modifiziert worden. Die beiden Dateiinhalte sind in Teil A von Bild 11 in zwei Spalten gegenübergestellt

Teil B von Bild 11 zeigt das Vergleichsergebnis, das das UNIX-Standardwerkzeug `sdiff` liefert. Die gefundene Differenz besagt, daß im ersten Dokument hinter der ersten Zeile die 4 mit einem `>` markierten Zeilen einzufügen sind. Diese Differenz ist dahingehend optimal, daß alle Gleichheiten von Zeilen ausgenutzt werden. Sie ist aber nicht optimal dahingehend, die strukturellen Veränderungen in den Programmen zu veranschaulichen. In dieser Hinsicht wäre es besser, eine der beiden Funktionsdefinitionen in der zweiten Datei komplett als Einfügung zu betrachten[27]; welche Alternative besser ist, kann nur der Benutzer entscheiden, aus dem Programmtext selbst ergeben sich keine eindeutigen Anhaltspunkte für die Entscheidung. Eine der Alternativen sieht wie in Teil C von Bild 11 gezeigt aus.

Es lassen sich leicht weitere, komplexere Beispiele finden, in denen ein Benutzer bessere Korrespondenzen erkennt als ein Algorithmus.

Man kann zwei Hauptmotive unterscheiden, warum ein Benutzer die Differenzbestimmung beeinflussen bzw. korrigieren möchte:

[27] Obwohl dann in den übrigen 3 Zeilen ein Unterschied auftritt und nur noch 2 Zeilen als korrespondierend erkannt werden, d.h. beide Lösungen maximieren nicht die Zahl der als korrespondierend erkannten Zeilen.

A) Die zu vergleichenden Textdateien:

```
t1 () {                          t1 () {
  date '+%Y-%m-%d'                 date '+%A, %d. %B %Y'
}                                }

                                 t2 () {
                                   date '+%Y-%m-%d'
                                 }
```

B) Ausgabe von `sdiff`:

```
t1 () {                          t1 () {
                               >   date '+%A, %d. %B %Y'
                               > }
                               >
                               > t2 () {
  date '+%Y-%m-%d'                 date '+%Y-%m-%d'
}                                }
```

C) Bessere Differenz:

```
t1 () {                          t1 () {
  date '+%Y-%m-%d'             |   date '+%A, %d. %B %Y'
}                                }
                               >
                               > t2 () {
                               >   date '+%Y-%m-%d'
                               > }
```

Abbildung 11: Vergleich zweier Dateien

1. bessere Veranschaulichung der Unterschiede beider Dokumente; je nach Dokumenttyp kann hierunter auch die Korrektur des Layouts der Parallel- oder Vereinigungsdarstellung fallen
2. Verbesserung der Mischmöglichkeiten in einer nachfolgenden Phase,

in der ein Mischdokument bestimmt wird.

Speziell bei Textdokumenten fällt hierunter der häufig auftretende Fall, daß mehrere inhaltlich unabhängige Änderungen an der gleichen Textstelle bei der Differenzbildung zu einem einzigen Cluster "verklumpt" werden und nicht mehr isoliert zur Übernahme in das Mischergebnis angewählt werden können.

Korrekturkommandos. Welche Möglichkeiten zur Einflußnahme auf die Differenzbestimmung ein Benutzer haben kann und wie geeignete Kommandos und Bedienschnittstellen zu gestalten sind, hängt stark von jeweiligen Dokumenttyp ab[28]. Weitgehend unabhängig vom Dokumenttyp sind folgende Korrekturfunktionen sinnvoll:

- Vorgabe von korrespondierenden Komponenten[29]
- Aufteilen von Clustern

Als Basis für eine Bedienschnittstelle, in die die Korrekturkommandos integriert werden können, eignet sich vor allem die Paralleldarstellung, weil hier z.B. korrespondierende Komponenten gut erkennbar markiert werden können. Die Darstellung mit einem Vereinigungsdokument ist hier weniger günstig, da die unveränderten Komponenten u.U. doppelt markiert werden müssen; anders gesagt basiert das Vereinigungsdokument punktuell gerade auf falsch erkannten Korrespondenzen.

Bei Textdokumenten (allgemeiner bei linear geordneten Dokumenten) können beide Korrekturfunktionen durch eine einzige Eingabeform abgedeckt werden, die Angabe je eines Textabschnitts in beiden

[28] Ferner wird man fast immer zugleich Kommandos für Mischprozesse berücksichtigen müssen, d.h. die Kommandos zur Korrektur von Differenzen sind in diesem Gesamtkontext zu gestalten.

[29] Ob es auch sinnvoll ist, *nicht korrespondierende* Paare von Komponenten vorzugeben, ist weniger klar. Einsetzbar wäre eine solche Vorgabe, wenn bei der Differenzbestimmung eine falsche Korrespondenz berechnet wurde. Als falsch wird eine Korrespondenz aber i.d.R. nur dann erkennbar sein, wenn man für wenigstens eine der involvierten Komponenten eine bessere Korrespondenz angeben kann. Wird diese (positive) Korrespondenz vorgegeben, wird die falsche Korrespondenz implizit aufgelöst, ein explizite Auflösung ist nicht erforderlich.

Dokumenten. Einer der Abschnitte kann leer sein. Sind beide Abschnitte gleich, handelt es sich um gemeinsame Komponenten, sind sie verschieden, handelt es sich um gelöschte, eingefügte oder veränderte Komponenten.

Neuberechnung der Differenz nach einer manuellen Korrektur. Die Neuberechnung der Differenz nach einer expliziten Teilung eines Paar korrespondierender Cluster ist trivial: das bisher für diese Cluster vorhandene Delta ist zu ersetzen durch zwei neue Deltas für die jeweiligen Teile der Cluster.

Die Neuberechnung der Differenz nach Vorgabe einer neuen Korrespondenz ist wesentlich komplizierter: oft muß die Differenz komplett neu berechnet werden. Die beiden neu korrespondierenden Komponenten (oder Cluster) können nämlich bisher Teilnehmer anderer Korrespondenzen gewesen sein, die aufgelöst werden müssen (s. obere und untere Korrespondenz in Bild 12); die beiden hierbei freiwerdenden Komponenten stehen nun erneut zur Bildung von Korrespondenzen zur Verfügung. In einer Art Kettenreaktion kann sich das Auflösen und Neubilden von Korrespondenzen beliebig weit fortpflanzen.

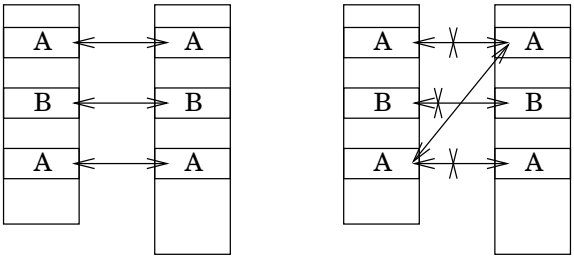

Abbildung 12: Korrektur von Korrespondenzen in Textdokumenten (Korrespondenzen werden durch Doppelpfeile angezeigt, aufgelöste Korrespondenzen sind durch ein Kreuz markiert)

Besonders kraß können die Auswirkungen bei Textdokumenten sein. Wegen der linearen Ordnung des Dokuments müssen bei einer neu vorgegebenen Korrespondenz nicht nur die beiden direkt betroffenen

bisherigen Korrespondenzen aufgelöst werden, sondern auch alle dazwischenliegenden (s. mittlere Korrespondenz in Bild 12), denn die Korrespondenzen dürfen sich bildlich gesprochen nicht überkreuzen[30].

Da die Algorithmen zur Berechnung von Differenzen zunächst die Korrespondenzen der beiden zu vergleichenden Dokumente bestimmen, kann man sie leicht dahingehend erweitern, daß bestimmte Korrespondenzen vorab als sicher erkannt vorgegeben werden. Die Korrektur einer Differenz nach expliziter Vorgabe einer neuen Korrespondenz kann daher am einfachsten dadurch implementiert werden, die Differenz mit dieser Vorgabe (und ggf. schon früher festgelegten Vorgaben) komplett neu zu berechnen.

Bei linear geordneten Dokumenten (insb. Textdokumenten) werden die beiden Dokumente durch n vorgegebenen Korrespondenzen in bis zu n+1 paarweise korrespondierende Abschnitte geteilt, die jeweils einzeln verglichen werden können. Die Gesamtdifferenz ist dann einfach die "Vereinigung" der lokalen Differenzen.

Es sind auch inkrementelle Verfahren denkbar, die nach Festlegung einer weiteren vorgegebenen Korrespondenz die Differenz nicht komplett neu berechnen. Ob sich hierdurch viel Performance gewinnen läßt, hängt sehr von der Größe und Komplexität der Dokumente ab.

4.8 Äquivalente Komponenten und Filterungen

Unter zwei äquivalenten Komponenten verstehen wir zwei Komponenten aus den beiden Basisdokumenten, die als "gleich" angesehen werden, obwohl sie nicht exakt gleich sind. Stattdessen ist eine vereinfachte, "gefilterte" Version der Komponenten, die man oft auch als normierte Darstellung auffassen kann, identisch.

[30]Hierbei ist unterstellt, daß der Editierdatentyp keine Verschiebeoperation enthält; wäre eine solche vorhanden, könnte die neue Korrespondenz auch als Verschiebung eingestuft werden. Ob die so entstandene Differenz optimal oder auch nur gut ist, ist aber fraglich. Daher ist durch Verschiebeoperationen letztlich nichts gewonnen.

4.8.1 Layout-Daten

Beispiele für Layout-Daten sind:

- in textuellen Dateien der Leerraum am Zeilenanfang, durch den Einrückungen realisiert werden, die die Blockstruktur eines Programms besser erkennbar machen;
- in UML-Diagrammen die Positionen von Knoten in Graphen, Stützpunkte von Verbindungskanten, die Größe von Einrahmungen u.a. Details.

Layout-Daten sind typischerweise irrelevant für die "inhaltliche" Konsistenz des Dokuments, aber wichtig für die Lesbar- bzw. Überschaubarkeit.

Es gibt auch Fälle, wo die textuelle oder graphische Anordnung eine inhaltliche Bedeutung hat. Beispiele sind: die Reihenfolge der Anweisungen eines Programms, die Reihenfolge der Parameter in einer Parameterliste und die vertikale Anordnung der Operationsaufrufe in einem Sequenzdiagramm, die die Reihenfolge der Aufrufe ausdrückt[31].

Relevanz der Layout-Daten für Differenzen. Nach unseren bisherigen Begriffsbildungen sind Layout-Daten Teil des Editiermodells des Dokuments, denn sie können ja durch die üblichen Werkzeuge bearbeitet werden. Somit sind sie prinzipiell als relevant für die Differenz zwischen Dokumenten ansehbar.

Dennoch wird man sie vielfach nicht als relevant ansehen. Ob die Layout-Daten relevant für die Differenzbestimmung sind oder nicht, *muß vom Benutzer bei jeder Differenzbestimmung wählbar sein*, es sind sogar abgestufte Formen von Nichtberücksichtigung denkbar.

Die Nichtberücksichtigung der Layout-Daten kann so aufgefaßt werden, daß sie beim Einlesen der Dokumente weggefiltert werden. Bei diesem Denkmodell sind sie nicht mehr im Editiermodell enthalten. Dies hat allerdings mehrere Nachteile:

- Es widerspricht der grundsätzlichen Definition des Editiermodells.

[31] Der Abstand der Aufruflinien ist hingegen semantisch irrelevant, ebenso die horizontale Anordnung der Objekte bzw. Lebenslinien.

- Aus dem Editiermodell kann man nicht mehr eine externe Darstellung der Dokumente ableiten, die das exakte ursprüngliche Layout hat (bei der Paralleldarstellung) bzw. beim Vereinigungsdokument ein Layout, das an das Layout eines der Basisdokumente angelehnt ist.

Daher sollten Layout-Daten als Teil des Editiermodells angesehen werden.

Die Nichtbeachtung der Layout-Daten kann alternativ als Option des Differenzbestimmungsverfahrens aufgefaßt werden.

4.8.2 Äquivalenz von Komponenten

Abstrakt betrachtet liegt hier ein Äquivalenzbegriff für Komponenten des jeweiligen Dokumenttyps vor. Gilt z.B. vornestehender Leerraum in einer Textzeile als irrelevant, so sind zwei Zeilen äquivalent, wenn sie nach Entfernen von vornestehendem Leerraum gleich sind.

Das Entfernen von vornestehendem Leerraum oder die Reduktion von Leerraum auf ein einziges Leerzeichen kann auch als Transformation in eine **normierte Darstellung** der Komponente verstanden werden. Zwei Komponenten gelten dann als **äquivalent**, wenn ihre normierte Darstellung identisch ist. Dieser Begriff kann auf *Cluster*, also zusammenhängende Folgen eingefügter Zeilen, ausgedehnt werden, wenn z.B. Leerzeilen irrelevant sein sollen[32].

Manche Äquivalenzen (z.B. Tabulatoren vs. Leerzeichen) haben die Eigenschaft, daß äquivalente Komponenten in der externen Darstellung gleich aussehen.

Die Äquivalenzen haben den Effekt, daß korrespondierende Komponenten nicht mehr exakt identisch sind. Bei der Paralleldarstellung kann man sie trotzdem gegenüberstellen. Bei der Darstellung als Vereinigungsdokument kann nur eine der beiden Varianten übernommen

[32] Eine Leerzeile (bzw. eine Zeile, deren normierte Darstellung eine Leerzeile ist) hat ggf. keine korrespondierende Zeile im anderen Dokument. Daher kann die Option, daß Leerzeilen (allgemeiner: "leere" Komponenten) nicht beachtet werden sollen, nicht auf die Äquivalenz von Komponenten zurückgeführt werden, nur auf die Äquivalenz von Clustern.

werden; dies ist solange problemlos, wie hieraus das andere Dokument noch erkennbar ist. Speziell bei UML-Diagrammen, bei denen das Layout als irrelevant betrachtet wird, muß i.a. ein komplett neues Layout berechnet werden; günstig ist es, wenn dieses dem Layout eines der beiden Basisdokumente ähnlich ist.

4.8.3 Irrelevante Änderungen

Bei der Speicherung von strukturierten Dokumenten in XML-Dateien, relationalen Datenbanken oder ähnlich strukturierten physischen Repräsentationen werden oft eindeutige Nummern oder Bezeichner für die Dokumentkomponenten (und ggf. weitere Zwecke) vergeben. Diese Bezeichner werden oft in Fremdschlüsselattributen, IDREF-Attributen oder ähnlichen Datenwerten eingesetzt, die Beziehungen zwischen Dokumentteilen (i.d.R. sogar Dokumentkomponenten) darstellen. Analog zu Variablennamen in Programmen kann man von einem definierenden und ggf. mehreren nutzenden Auftreten eines Bezeichners sprechen.

Bzgl. der Differenzberechnung sind diese Bezeichner oft sehr störend, weil der konkrete Wert keine Rolle spielt, es kommt nur auf die Gleichheit des Werts beim definierenden und nutzenden Auftreten an. Der konkrete Wert ist austauschbar (solche Bezeichner werden fast nie manuell vergeben, sondern praktisch immer automatisch). Manche Werkzeuge vergeben daher beim Laden oder Speichern eines Dokuments komplett neue Bezeichner, z.B. eine neue durchlaufende Numerierung.

Eine marginale Änderung des Dokuments und nachfolgende Speicherung kann zu einer erheblichen Differenz von der vorherigen zur folgenden physischen Repräsentation führen, weil viele Fremdschlüsselwerte neu vergeben werden. Dies bestätigt die schon früher gewonnene Erkenntnis, daß man auf Basis der physischen Repräsentation keine Differenzen berechnen kann.

Sofern die Unterschiede beim Übergang zum Editiermodell komplett verschwinden, ist das Problem gelöst. Dies ist aber nicht der Fall, wenn interne Bezeichner "wörtlich" in der externen Darstellung angezeigt werden. Die Bezeichner sind dann auch Teil des Editiermo-

dells; dies sollte möglichst vermieden werden.

Ein ähnlich strukturierter, aber weitaus kritischerer Fall sind Fälle, von Sequenzen im semantischen Dokumentmodell durch Nummern im Editiermodell realisiert werden. Ein Beispiel hierfür sind Sequenznummern in Kollaborationsdiagrammen. Diese drücken eine Reihenfolge aus, in der Operationen aufgerufen werden[33]. Wenn beim Editieren mitten in eine derartige Sequenz ein weiterer Operationsaufruf eingefügt wird, müssen alle nachfolgenden Sequenznummern erhöht werden. Diese Änderungen der Nummern sind eigentlich irrelevant, denn letztlich ist nur die dadurch ausgedrückte Reihenfolge relevant.

Sofern man die (in der externen Darstellung sichtbaren!) Sequenznummern als Teil des Editiermodells auffaßt[34], sind alle Änderungen irrelevant bis auf die erste hinter der Einfügestelle.

Beiden Beispielen ist gemein, daß man die irrelevanten Änderungen im Editiermodell vermeiden kann, indem man darin direkt mit applikationsnäheren Datenstrukturen bzw. Datenmodellierungskonzepten (wie Listen, "abstrakten" Referenzen) arbeitet. Letztlich nähert man so das Editiermodell an die semantische Repräsentation des Dokuments an. Dementsprechend komplex wird der Editierdatentyp und entsprechend erhöht wird der Mindestkonsistenzgrad, den alle Editoren für diese Dokumentklasse garantieren müssen; beides ist, wie schon früher erwähnt wurde, sehr problematisch.

Algorithmen, die Editiermodelle unterstellen, bei denen die Fremdschlüsselwerte sichtbar sind und die bei der Differenzberechnung den Austausch eines Fremdschlüsselwerts als irrelevante Änderung erkennen, können wie folgt vorgehen:

- Das Schlüsselattribut, auf das sich die Fremdschlüssel beziehen, muß inhaltlich insofern als irrelevant angesehen werden, als mit die-

[33] Wie die Sequenznummern genau zu vergeben und zu interpretieren sind, soll hier nicht diskutiert werden.

[34] Man muß dies nicht unbedingt tun, sondern kann den Editierdatentyp auch so gestalten, daß darin direkt mit Sequenzen und entsprechenden Operationen darauf gearbeitet wird; die in der externen Darstellung sichtbaren Sequenznummern müssen dann jedesmal generiert werden.

sem Attribut keine Korrespondenzen gebildet werden können. Die Komponenten, bei denen dieses Attribut auftritt, müssen daher ein zweites Schlüsselattribut haben (z.B. den Namen der Klasse); nur über dieses "relevante" Schlüsselattribut dürfen Korrespondenzen gebildet werden.

- Werte in Fremdschlüsselattributen werden als unverändert angesehen, wenn die beiden verglichenen Werte zu korrespondierenden Objekten führen.

Literatur

[AlP03] Alanen, M.; Porres, I.: Difference and Union of Models; p.2-17 in: UML 2003 - The Unified Modeling Language: Modeling Languages and Applications, LNCS 2863, Springer; 2003

[BaJ05] Basit, H.A.; Jarzabek, S.: Detecting higher-level similarity patterns in programs; SIGSOFT Softw. Eng. Notes, 30(5):156–165; 2005

[ChG97] Chawathe, S.; Garcia-Molina, H.: Meaningful Change Detection in Structured Data; p.26-37 in: Proc. ACM SIGMOD International Conference on Management of Data; 1997

[ChRGW96] Chawathe, S.S.; Rajaraman, A.; Garcia-Molina, H.; Widom, J.: Change detection in hierarchically structured information; p.493-504 in: Proc. ACM SIGMOD Intl. Conference on Management of Data; 1996

[ChTZZ01] Chien, S.-Y.; Tsotras, V. J.; Zaniolo, C.; Zhang, D.: Storing and Querying Multiversion XML Documents using Durable Node Numbers; in: Proc. 2nd International Conf. on Web Information Systems Engineering (WISE), Kyoto, Japan, Dec. 2001; 2001

[CoAM02] Cobéna, G.; Abiteboul, S.; Marian, A.: Detecting Changes in XML Documents; Proc. 18. International Conference on Data Engineering (ICDE) San Jose; 2002

[CoW98] Conradi, Reidar; Westfechtel, Bernhard: Version models for software configuration management; ACM Computing Surveys, 30(2):232282; 1998

[Gi02] Girschick, Martin: UMLDiff: Erkennung und Analyse von Unterschieden in Klassendiagrammen und Sequenzdiagrammen. Diplomarbeit, Technical University of Darmstadt; 2002.

[1] Fujaba-Projekt; `www.fujaba.de`

[KeWN05] Kelter, U.; Wehren, J.; Niere, J.: A Generic Difference Algorithm for UML Models; Proc. GI-Fachtagung Software Engineering 2005, Essen, LNI; 2005

[Li01] Lindholm, T.: A 3-way Merging Algorithm for Synchronizing Ordered Trees - the 3DM merging and differencing tool for XML; Thesis, Helsinki Univ. Technology, Dep. Computer Science; 2001

[Lu06] Lück, Stephan: Differenzberechnung von hierarchischen Diagrammen; Diplomarbeit, Fachgruppe Praktische Informatik, Universität Siegen; 2006

[MaACM01] Marian, A.; Abiteboul, S.; Cobena, G.; Mignet, L.: Change-Centric Management of Versions in an XML Warehouse. p.581590 in: Proc. 27th Intl. Conf. Very Large Data Bases: Roma, Italy, Sept. 2001; Morgan Kaufmann Publishers; 2001

[Me02] Mens, T.: A State-of-the-Art Survey on Software Merging; IEEE Transactions on Software Engineering 28:5, p. 449-462; 2002

[My86] Myers, Eugene W.: An O(ND) difference algorithm and its variations; Algorithmica 1(2):251-266; 1986

[Oh04] Ohst, D.: Versionierungskonzepte mit Unterstützung für Differenz- und Mischwerkzeuge; Dissertation, FB 12, Univ. Siegen, URN: urn:nbn:de:hbz:467-831; 2004

[OhK02] Ohst, D.; Kelter, U.: A Fine-grained Version and Configuration Model in Analysis and Design; in Proc. IEEE International Conference on Software Maintenance 2002 (ICSM 2002), 3-6 October 2002, Montreal, Canada; 2002

[OhWK03a] Ohst, D.; Welle, M.; Kelter, U.: Difference Tools for Analysis and Design Documents; in: Proceedings IEEE International Conference on Software Maintenance 2003 (ICSM2003), Amsterdam, pages 1322, September 2003.

[OhWK03b] Ohst, D.; Welle, M.; Kelter, U.: Differences between Versions of UML Diagrams; in: Proceedings ESEC/FSE'03, September 1-5, 2003, Helsinki, Finland; 2003.

[RhW98] Rho, Jungkyu; Wu, Chisu: An Efficient Version Model of Software Diagrams; in: Proc. 5th Asia-Pacific Software Engineering Conf., 2-4 December 1998 in Taipei, Taiwan, ROC, IEEE Computer Society; 1998/12

[Se03] Selonen, Petri: Set Operations for Unified Modeling Language. p. 7081 in: Proceedings Eight Symposium on Programming Languages and Tools, SPLST'2003, Kuopio, Finland, University of Kuopio; 2003

[Si07] SiDiff Differenzwerkzeuge; `http://www.sidiff.org`

[St06] Störrle, Harald: Management großer Modelle (Mini-Tutorium); GI-Fachtagung Software Engineering 2006, Universität Leipzig; 2006; `http://ebus.informatik.uni-leipzig.de/se2006/programm/mini-tutorien/stoerrle.html`

[WaDC03] Wang, Yuan; DeWitt, David J.; Cai, Jin-Yi: X-Diff: An effective change detection algorithm for XML documents; in: 19th International Conference on Data Engineering, March 5 - 8, 2003, Bangalore, India; 2003

[We04] Wehren, J.: Ein XMI-basiertes Differenzwerkzeug für UML-Diagramme; Diplomarbeit, FG PI, FB12, Univ. Siegen; 2004

[XiS05] Xing, Z.; Stroulia, E.: UMLDiff: An Algorithm for Object Oriented Design Differencing; in Proc. 20th IEEE Conf. Automated Software Engineering; 2005

[ZuWR01] Zündorf, A.; Wadsack, J.; Rockel, I.: Merging Graph-Like Object Structures: in: Andre van der Hoek (ed.): Tenth International Workshop on Software Configuration Management (SCM-10) New Practices, New Challenges, and New Boundaries May 14-15, 2001 Toronto, Canada, http://www.ics.uci.edu/andre/scm10/; 2001

[KM1] Kelter, U.: Lehrmodul "Einführung in das Konfigurationsmanagement"; 2001

Index

