

# Differenz- und Mischwerkzeuge für UML

Dirk Ohst

Praktische Informatik, Fachbereich Elektrotechnik und Informatik,  
Universität Siegen, D-57068 Siegen,  
ohst@informatik.uni-siegen.de

**Abstract:** Eine vielfach vernachlässigte Aufgabe in der Softwareentwicklung ist die Versionierung, Differenzberechnung und -anzeige sowie das Mischen von Versionen von UML-Diagrammen resp. Modellen. Diese Aufgabe erlangt durch große Entwicklergruppen, standardisierte Entwurfsprozesse und auch durch die Forderung nach hohen Qualitätsstandards eine immer größere Bedeutung. In der diesem Text zugrundeliegenden Dissertation wird eine Lösung für die genannten Fragestellungen beschrieben. Hier beschränken wir uns primär auf den Aspekt der Klassifizierung von Differenzen und deren Anzeige sowie das Mischen von Versionen.

## 1 Einleitung und Motivation

Unter Softwareartefakten versteht man alle Dokumente, die während der gesamten Softwareentwicklung entstanden sind, wie z.B. Analyse- oder Entwurfsdokumente. Der Werkzeugeinsatz in Entwicklergruppen, ohne den die moderne Softwareentwicklung nicht mehr denkbar ist, erleichtert einerseits die Erstellung der diversen Arten von Software-Dokumenten, andererseits führt er jedoch auch zu einem erhöhten Verwaltungsaufwand durch die steigende Anzahl an Dokumenten und Versionen dieser Dokumente. Die Verwaltung der so entstandenen Versionen, die Bestimmung und Anzeige von Differenzen sowie das Mischen von Versionen sind wichtige Aufgaben im Rahmen moderner Softwareentwicklungs-Prozesse. In den späten Phasen der Softwareentwicklung haben sich hierfür Versionsmanagement-Systeme (VM-Systeme) etabliert; die frühen Phasen insbesondere die Analyse und der Entwurf mithilfe der UML [OMG03] werden jedoch nur unzureichend unterstützt.

Der Grund für diese Situation ist, daß existierende VM-Systeme nur unzureichend an die Dokumenttypen der frühen Phasen angepaßt sind. Es gibt eine Vielzahl an existierenden VM-Systemen [CW98], die meisten von ihnen (z.B. CVS oder ClearCase) verwalten jedoch Textdateien und mit Einschränkung binäre Dateien. Im Gegensatz hierzu handelt es sich bei den Dokumenten der frühen Phasen um Diagramme und nicht um Text. Bei der UML sind es Modelle, welche durch einzelne Diagramme partiell visualisiert werden. Diese sind zwar häufig auch in Dateien gespeichert, jedoch ist die Bestimmung von Differenzen zwischen zwei Versionen durch Werkzeuge konventioneller VM-Systeme nicht praktikabel. Im Fall von binären Dateiformaten ist sie nur mit Hilfe externer Werkzeuge eingeschränkt möglich, oft sogar unmöglich.

Bei der textuellen Speicherung von UML-Modellen treten folgende Probleme auf [ZWR01, OWK03]: Ein Klasse eines Klassendiagramms kann z.B. in einer Textdatei (z.B. XMI) durch eine Folge von Zeilen repräsentiert sein. Die Reihenfolge dieser Textabschnitte in der Datei ist unerheblich, da die Position innerhalb eines Diagramms explizit durch Layoutdaten und die Klasse selbst durch eindeutige Identifizierer bestimmt wird. Daher kann sich die Reihenfolge der Textabschnitte zwischen den Dateiversionen unterscheiden, abhängig von der Realisierung des Werkzeugs. Die externe Repräsentation in einer Text- oder Binärdatei ist daher nicht für die Differenzberechnung und -Anzeige geeignet.

Die Visualisierung ist ein weiterer wichtiger Gesichtspunkt, der bei der Differenzbestimmung und beim Mischen von Modellen berücksichtigt werden muß. Textuelle Differenzen können leicht durch eine zweispaltige Anzeige mit farbig markierten Differenzen visualisiert werden. Das ist jedoch für Diagramme nicht möglich, da das Layout für die meisten Diagrammtypen keine Relevanz besitzt. Eine Ausnahme stellen die Sequenzdiagramme dar. Man muß daher zwischen Differenzen im Modell und im Layout unterscheiden.

Weiter muß auch Augenmerk auf die Anzahl der Differenzen gelegt werden. Üblicherweise gibt es Gruppen von Differenzen, die im Kontext derselben Änderungsaufgabe durchgeführt wurden. Diese Änderungen führen zu einer großen Anzahl an Differenzen zwischen Versionen, die sich über längere Zeit entwickelt haben. Oft gibt es daher das Problem, den Kontext einzelner Differenzen zu identifizieren.

Die diesem Text zugrundeliegende Dissertation [Ohs04] behandelt die Aufgabe der Versionierung, Differenzberechnung und -anzeige sowie das Mischen von UML-Modellen, resp. Diagrammen. In diesem Text skizzieren wir die entwickelten Konzepte, die in und auf der strukturell objektorientierten Datenbank (OMS) H-PCTE [Kel92] realisiert wurden. Den Aspekt der Differenzanzeige vertiefen wir.

Dieser Text ist wie folgt aufgebaut. In Abschnitt 2 diskutieren wir das verwendete Daten- und Versionsmodell, auf welchem die Differenz- und Mischwerkzeuge aufbauen, die wir in Abschnitt 3 beschreiben. Dieser Abschnitt beginnt mit einer Klassifizierung von Differenzen, beschreibt einen Vorschlag zu deren Anzeige, zum Mischen und zur Gruppierung von Differenzen/Konflikten. Anschließend skizzieren wir den Differenzalgorithmus (Abschn. 4) und geben einen Überblick über andere Arbeiten (Abschn. 5).

## **2 Datenmodell und Versionierung**

Für die folgende Diskussion unterscheiden wir zwischen Layout- und Modelldaten. Unter Layout verstehen wir die Positionsangaben und Größen der Diagrammknoten oder die Eckpunkte von Kanten, also alles was in einer textuellen Repräsentation als unwichtig angesehen würde. Die verbleibenden Daten sind die Modelldaten, die die Semantik des Dokuments ausdrücken. Die UML-Spezifikation [OMG03] unterscheidet ebenfalls zwischen Semantik (Kapitel 2) und Notation (Kapitel 3).

Kapitel 2 definiert die abstrakte Syntax von korrekten Diagrammen unter Verwendung von Klassendiagrammen und weiteren Mitteln. Die Modelldaten können als Instanzen der in diesen Diagrammen definierten Typen angesehen werden. Diese Art von Modellen

bezeichnen wir als feinkörnig. UML-Werkzeuge, insbesondere Diagrammeditoren, nutzen eine vergleichbare Struktur, um die Diagramme und die unterliegenden Modelle im Hauptspeicher oder einer objektorientierten Datenbank abzubilden. Die interne Struktur, wir bezeichnen es im folgenden als Editiermodell, ist stark von der verwendeten Programmiersprache oder dem Datenbankmodell abhängig.

Kapitel 3 von [OMG03] definiert viele Details wie UML-Diagramme auf dem Bildschirm dargestellt werden sollen. Es überläßt die Festlegung einiger Details jedoch dem Werkzeugentwickler. Daraus folgt, daß viele Details, die die Versionierung und die Differenzen zwischen Versionen betreffen, werkzeugspezifisch sind.

In dieser Arbeit legen wir ein feinkörniges Editiermodell zugrunde. Die Editiermodelle sind in der strukturell objektorientierten Datenbank (OMS) H-PCTE gespeichert und die UML-Editoren greifen im Rahmen von *Werkzeugtransaktionen* (WTA) [Pla99] direkt auf diese zu. Alle Änderungen an den in den UML-Editoren angezeigten Diagrammen werden direkt in das OMS propagiert [KM99].

Das feinkörnige Editiermodell von UML-Dokumenten erfordert eine andere Art der Versionierung als grobkörnig modellierte Dokumente wie z.B. Texte. Bei der Bearbeitung eines Diagramms ist es nicht sinnvoll, von allen Objekten des Editiermodells eine neue Version anzulegen. Es würden einerseits zu viele Objekte versioniert, andererseits würde auch von Objekten des Modells eine neue Version angelegt, die gar nicht im modifizierten Diagramm sichtbar sind. Daher ist es sinnvoll, jedes Objekt unabhängig zu versionieren.

Durch die feinkörnige Modellierung besteht ein Dokument aus einer Vielzahl an Objekten. Ein UML-Editor greift auf mehr als ein Objekt zu, so daß ggf. mehrere Objekte bei Änderungen unabhängig versioniert werden müssen. Das manuelle Anlegen von Versionen durch die Werkzeuganwender ist genausowenig praktikabel wie das Anlegen der Versionen durch die Werkzeuge, da es einen hohen Aufwand für die Anwender oder eine komplexe Werkzeugarchitektur bedeutet. In beiden Fällen müßte für jedes Objekt geprüft werden, ob eine Versionierung notwendig ist. Die Lösung besteht darin, das Anlegen von Objektversionen und den Zugriff darauf im OMS zu realisieren, da es die Objekte, deren Versionen und die Zugriffe auf diese verwaltet und kontrolliert. Durch die Kapselung aller Zugriffe in WTA ist es möglich, die Versionierungsfunktionalität in die WTA zu integrieren. Alle Schreibzugriffe führen somit automatisch zu einer Versionierung der Objekte, mit der Einschränkung, daß jedes Objekt nur einmal in einer WTA versioniert wird. Alle im Rahmen einer WTA veränderten Objekte faßt eine Konfiguration zusammen.

### **3 Differenz- und Mischwerkzeuge**

Bevor wir auf die Anzeige und das Mischen von Differenzen näher eingehen, ist es notwendig zu klären, welche Differenzen auftreten können. Daher betrachten wir die Differenzen in diesem Abschnitt eingehender. Der Begriff Differenz kann aus unterschiedlichen Perspektiven betrachtet werden (z.B. Differenzen im Editiermodell oder in der externen Darstellung). In dieser Arbeit nehmen wir die Sichtweise eines Entwicklers ein und bestimmen so die Differenzen, die zwischen 2 Versionen eines Diagramms auftreten können.

**Semantik des Layout.** Die einzelnen Diagrammtypen der UML können in zwei Kategorien unterteilt werden, Diagramme, deren Layout semantisch relevant ist und Diagramme deren Layout keine semantische Relevanz besitzt. Sequenzdiagramme gehören zur ersten Kategorie<sup>1</sup>. Alle anderen Diagrammtypen gehören zur zweiten Kategorie, auf die wir uns hier beschränken.

**Struktur der Diagramme.** Alle Diagramme mit Ausnahme der Sequenzdiagramme besitzen eine graphartige Struktur, genauer gesagt, besitzen sie Knoten und Kanten. Die Knotentypen sind vom Diagrammtyp abhängig, in Klassendiagrammen gibt es z.B. Klassen, Interfaces und Kommentare, in Aktivitätsdiagrammen gibt es verschiedene Arten von Zuständen.

Die Struktur der Knoten ist durch ihren Typ bestimmt. Die meisten Knotentypen besitzen nur eine geringe Anzahl an Attributen (oft nur den Bezeichner), während andere zusätzliche Attribute besitzen oder sogar eine komplexe innere Struktur. Beispiele für Knoten, die ausschließlich Bezeichner besitzen, sind einfache Zustände wie der initiale Zustand, Forks oder Joins. Knoten mit zusätzlichen Attributen sind z.B. Klassen oder komplexe Zustände und Beispiele für komplexe Knoten sind Pakete, die weitere Knoten gruppieren. Bezeichner gibt es auch an Beziehungen, z.B. als Rollenname.

**Arten von Knotenattributen.** Häufig handelt es sich bei den Knotenattributen um einfache Attribute. Beispiele sind der Bezeichner oder bestimmte Eigenschaften eines Knotens (`{abstract}`). Neben diesen einfachen Attributen gibt es Listen oder Mengen von Attributen, wie z.B. in einer Klasse die Attribut- und die Operationsliste oder die Menge von Stereotypen. In vielen Fällen besitzen diese Attribute eine interne Struktur; es kann aber sinnvoll sein, diese als Zeichenkette zu interpretieren.

Eine andere Art von Attributen kann (oder sollte) als Referenzen auf andere Diagrammelemente interpretiert werden. Beispiele sind Attributtypen in Klassen oder der Typ von Objekten in Klassendiagrammen. Diese Typen werden als Bezeichner dargestellt, sollten aber als dupliziert dargestellte Bezeichner der korrespondierenden Klasse interpretiert werden. Kollaborationsdiagramme beinhalten ausschließlich Referenzen (Objektypen und Methoden). Die folgenden Diagrammtypen können auch Referenzen auf andere Modellobjekte beinhalten: Zustands- und Aktivitätsdiagramme (die Events oder Operationen), Implementierungs- und Komponentendiagramme (Interfaces).

Noch ein anderer Attributtyp sind abgeleitete Attribute. Das einzig relevante Beispiel sind die Sequenznummern in Kollaborationsdiagrammen. Das Editiermodell ist hier abhängig von der Funktionsweise der Editoren. Im einfachsten Fall müssen die Sequenznummern manuell gesetzt werden, dann handelt es sich um einfache Attribute. Verwalten die Editoren jedoch die Operations-Sequenz explizit und bieten demzufolge Funktionen zum Einfügen oder Löschen einzelner Operationsaufrufe, so handelt es sich bei den Sequenznummern um durch die Editoren berechnete Attribute. In diesem Fall ist die Modifikation der Operations-Sequenz von mehr Relevanz als die geänderte Sequenznummer.

---

<sup>1</sup>Die Reihenfolge der Operationsaufrufe ist durch die graphische Anordnung der korrespondierenden Pfeile repräsentiert.

**Komplexe Knoten.** Einige Knoten besitzen eine komplexe innere Struktur. Diese Knoten gruppieren andere Knoten oder beinhalten ein vollständiges Sub-Diagramm, wie z.B. Pakete in einem Klassendiagramm, die wieder Pakete oder Klassen beinhalten können. Andere Beispiele sind Komponenten und Knoten in Komponenten- bzw. Implementierungsdiagrammen oder komplexe Zustände in Zustandsdiagrammen. Der Inhalt dieser komplexen Knoten wird oft als eigenständiges Diagramm dargestellt.

**Klassifizierung von Differenzen.** Die abstrakte Betrachtung der einzelnen Diagrammtypen [Ohs04] ermöglicht eine Gruppierung von Differenzen in Differenzklassen, die eine diagrammtypunabhängige Betrachtung hinsichtlich der Visualisierung ermöglichen. Die Klassen sind im einzelnen:

1. Struktur-Änderungen: die Graph-Struktur des Diagramms ist geändert worden: neue/gelöschte Knoten und Beziehungen
2. Intra-Knoten/Beziehungsänderungen: Änderungen der inneren Knotenstruktur:
  - (a) atomare Wertänderungen: z.B. Bezeichner, Rollen oder Kardinalitäten
  - (b) neue/gelöschte Elemente in Listen und Mengen
  - (c) Umordnen einer Liste
  - (d) Änderungen an komplexen Knoten: Diese Änderungen lassen sich auf die entsprechenden Änderungen am Sub-Diagramm zurückführen.
  - (e) Änderungen von intern referenzierten Modellobjekten: z.B. der Attributtyp, im Modell gibt es eine Referenz auf ein Objekt, welche eine Klasse repräsentiert, die den Attributtyp darstellt.
3. Inter-Knoten/Beziehungsänderungen: Verschieben von Komponenten oder Beziehungen zwischen Knoten

### 3.1 Anzeige von Differenzen

Die zentrale Problematik bei der Visualisierung der Differenzen liegt darin, unveränderte Diagrammelemente aus beiden zu vergleichenden Diagrammen (Basisdiagrammen) und Differenzen zwischen ihnen so darzustellen, daß die Entwickler diese leicht erkennen können. Der bei textuellen Dokumenten gebräuchliche Ansatz einer zweiseitigen Anzeige kann aufgrund der Dokument-Eigenschaften nicht verwendet werden. Aus diesem Grund werden beide Diagramme überlagert dargestellt, was man als ein *Vereinigungsdokument* interpretieren kann.

Das Vereinigungsdokument enthält alle Diagrammelemente<sup>2</sup> beider Basisdiagramme. Korrespondierende Diagrammelemente, also Diagrammelemente, die beide Basisdiagramme gemeinsam besitzen, werden nur einmal gezeichnet und die basisdiagrammspezifischen Diagrammelemente werden unterschiedlich eingefärbt. Die Wahl der Farbe richtet

<sup>2</sup>Unter Diagrammelement verstehen wir jedes Element eines Diagramms aus dem Editiermodell, welches in dessen graphischer Darstellung gezeichnet wird.

sich dabei nach dem ursprünglichen Basisdiagramm dieses Elementes. Abbildung 1 zeigt zwei Klassendiagramme und Abbildung 2 das dazugehörige Vereinigungsdiagramm.

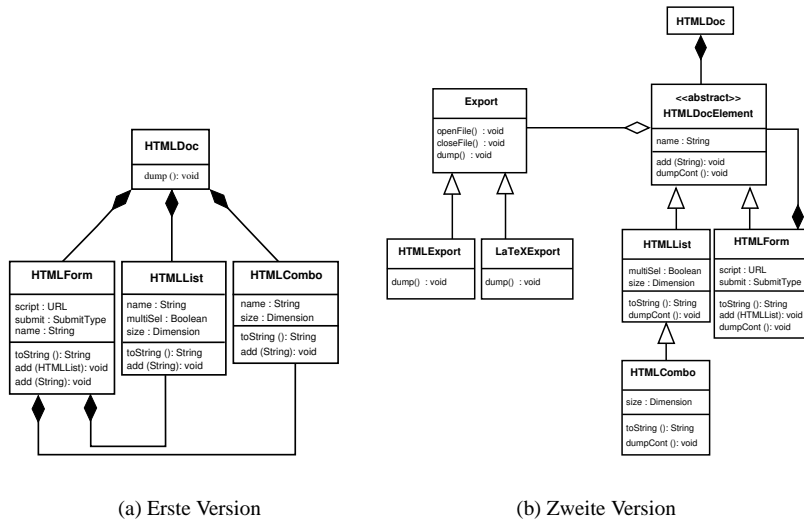


Abbildung 1: Entwicklung eines UML-Diagramms

Die Farben im Vereinigungsdiagramm kennzeichnen nicht, ob ein Diagrammelement erzeugt oder gelöscht wurde, sondern nur, zu welcher Diagrammversion es gehört. Die Aussage, ob ein Diagrammelement erzeugt oder gelöscht wurde, kann man nur dann treffen, wenn beide Basisdiagramme eine gemeinsame Vorgänger-Version besitzen und die bei der Differenz-Berechnung mit berücksichtigt wurde. Diese Unterscheidung würde jedoch auch die Anzahl der benötigten Farben im Vereinigungsdiagramm von 3 auf 5 erhöhen, was die Lesbarkeit des Diagramms verschlechtern würde [Yan94]. Dabei stellt sich die Frage, wie groß der Nutzen dieser zusätzlichen Informationen wäre.

**Layout.** Das Vereinigungsdiagramm enthält mehr Elemente als die beiden Basisdiagramme, daraus folgt, daß das Vereinigungsdiagramm ein anderes Layout besitzt als diese. I.d.R. sollte das jedoch kein Problem darstellen, da Entwickler meist an den Differenzen zwischen den Modellen und weniger an Differenzen im Layout interessiert sind. Sollten die Differenzen im Layout relevant sein, so kann die Technik des Vereinigungsdiagramms nicht verwendet werden. Das Layout des Vereinigungsdiagramms sollte ähnlich zu dem Layout (mindestens) eines Basisdiagramms sein, da andernfalls die Entwickler keines der Basisdiagramme mehr wiedererkennen können. Das würde die Interpretation des Vereinigungsdiagramms erheblich erschweren. Das Layout des Vereinigungsdiagramms kann man von dem Layout eines Basisdiagramms ableiten und um die zusätzlichen Elemente ergänzen. Algorithmen für automatisches Graphlayout (z.B. [See97]) sind ungeeignet, da sie das Layout vollständig neu berechnen.

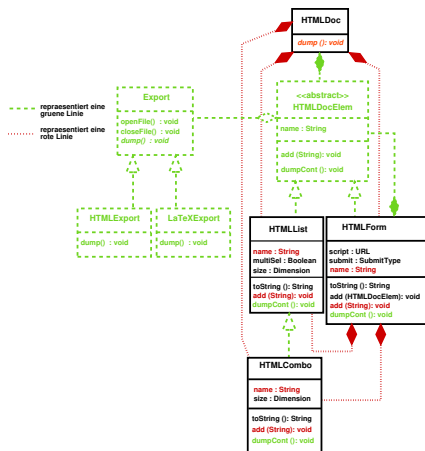
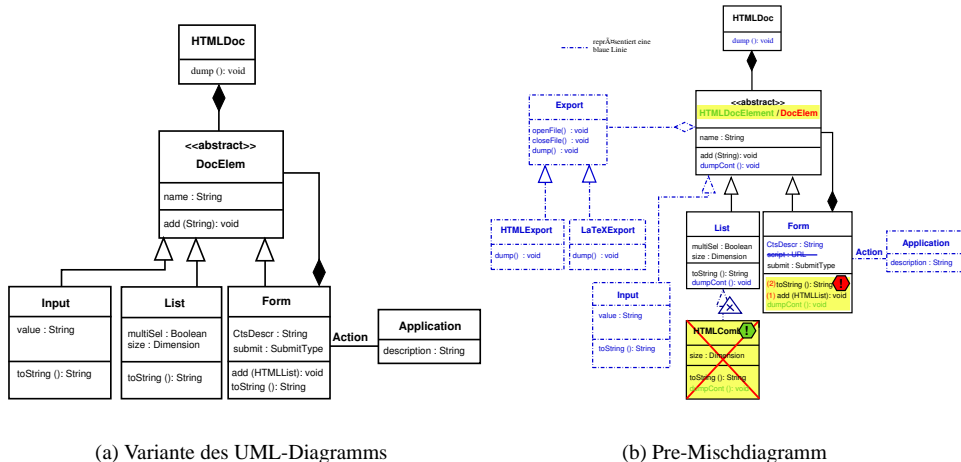


Abbildung 2: Vereinigungsdiagramm

### 3.2 Mischen von Differenzen

Im Gegensatz zur Differenzanzeige ist das Mischen von Versionen eine z.T. interaktive Tätigkeit, insbesondere beim Auflösen von aufgetretenen Mischkonflikten. Unsere Mischwerkzeuge arbeiten nach folgender Mischstrategie: (1) vorläufige Mischversion erstellen, (2) Konflikte manuell lösen und (3) endgültige Mischversion erstellen. Die vorläufige Mischversion eines Diagramms (*Pre-Mischversion*) ist vergleichbar mit dem Vereinigungsdiagramm. Sie unterscheidet sich jedoch in zwei wesentlichen Aspekten: 1. sie basiert auf den beiden zu mischenden Versionen einschließlich der gemeinsamen Vorgängerversion und 2. anhand eines 3-Wege-Mischalgorithmus wurden bereits viele Mischentscheidungen automatisch getroffen. Die Pre-Mischversion dient weiterhin als Ausgangspunkt für die Entwickler, die darauf aufbauend die Konflikte lösen und (automatisch oder manuell) getroffene Mischentscheidungen ändern können. Die Anzeige der Pre-Mischversion ist ähnlich der Anzeige des Vereinigungsdiagramms. Alle automatisch oder manuell getroffenen Mischentscheidungen sind darin blau gezeichnet, Konflikte sind ähnlich wie Differenzen im Vereinigungsdiagramm gezeichnet, s. Abb. 3. Alle getroffenen Entscheidungen zum Lösen von Konflikten werden nicht sofort im Editiermodell nachvollzogen, sondern erst im Werkzeug angezeigt und im Editiermodell als Anweisung notiert. Nachdem alle Konflikte gelöst wurden, wird die endgültige Mischversion erzeugt. Das bietet die Möglichkeit, einmal getroffene Entscheidungen einfach wieder rückgängig zu machen und den Konflikt durch Wahl der anderen Änderung zu lösen.



(a) Variante des UML-Diagramms (b) Pre-Mischdiagramm

Abbildung 3: Mischen von Versionen eines UML-Diagramms

### 3.3 Gruppierung von Differenzen

Der Nutzen des Vereinigungsdiagramms hängt entscheidend von dessen Übersichtlichkeit ab. Sind darin „zu viele“<sup>3</sup> Differenzen markiert, reduziert das die Übersichtlichkeit und somit den Nutzen deutlich. Aufbauend auf das Editiermodell und die Informationen, die die Versionsverwaltung des OMS liefert, ist es möglich, die angezeigten Differenzen anhand der Konfigurationen zu gruppieren. Eine Gruppierung auf Basis des Editiermodells faßt Differenzen, die einzelne Typen von Diagrammelementen betreffen, zusammen, wie z.B. Klassen, Methoden oder Beziehungen. Zur Verbesserung der Übersichtlichkeit kann die Markierung einzelner Differenzgruppen aufgehoben werden, indem sie nicht mehr farbig, sondern grau gezeichnet werden. Dadurch bleiben sie weiterhin erkennbar, lenken jedoch nicht mehr die Aufmerksamkeit der Entwickler auf sich.

## 4 Berechnung von Differenzen

Die Editiermodelle der zu vergleichenden Diagrammversionen sind die Grundlage der Differenz-Berechnung, die in drei Stufen verläuft: (1) Traversieren der Editiermodelle beider Diagramme, (2) Zuordnung korrespondierender Objektversionen, (3) Bestimmung der Unterschiede zwischen den Versionen. Zur Berechnung der Differenzen müssen die Editiermodelle abgeglichen und korrespondierende Objekte gefunden werden. Bei korrespondierenden Objekten handelt es sich um Versionen eines Objektes, die nicht not-

<sup>3</sup>Ab wann ein Vereinigungsdiagramm zu viele Differenzen anzeigt und damit unbrauchbar wird, ist abhängig vom einzelnen Entwickler, der ein Diagramm betrachtet und seinen Anforderungen an die Differenz-Anzeige.

wendigerweise unterschiedlich sein müssen. Hierzu traversiert man parallel beide Editiermodelle und vergleicht die Objekt-Identifizierer. Sind diese gleich, so hat man zwei korrespondierende Objektversionen gefunden. Die Differenzen lassen sich dann durch einen einfachen Vergleich der Attributwerte und der Beziehungen zwischen den Objekten bestimmen<sup>4</sup>. Im Modell verschobene Objekte lassen sich finden, indem man bei der Traversierung alle Objekte, für die kein korrespondierendes Objekt gefunden wurde, zwischenspeichert und nach der Traversierung untereinander noch einmal abgleicht. Wenn anschließend kein korrespondierendes Objekt gefunden wurde, so existiert dieses Objekt nur in einem Editiermodell.

## 5 Andere Arbeiten

Es wurden bereits viele VM-Systeme zur Versionsverwaltung von Dateien entwickelt oder neue Ansätze vorgeschlagen, einen Überblick gibt [CW98]. Diese Systeme eignen sich aus den in der Einleitung diskutierten Gründen nicht zur Versionierung von UML-Modellen. Hier sind Ansätze notwendig [OK02], die sich auf Objektgraphen konzentrieren. Die bisher vorgeschlagenen VM-Systeme versionieren jedoch alle Objekte, oder alle Objekte auf dem Pfad vom Wurzelobjekt bis zum modifizierten Objekt gemeinsam. Das führt zu einer Inflation von nicht notwendigen Versionen.

Desweiteren gibt eine Vielzahl an Algorithmen zur Differenzbestimmung zwischen Dokumenten, die jedoch vom Dokumenttyp abhängig sind. Algorithmen zur Differenzbestimmung zwischen Texten (z.B. [Mye86, Tic84]) sind nicht für Modelle der UML geeignet [ZWR01, RW98, OWK03]. Die Editiermodelle können oft als Bäume dargestellt werden. Differenzalgorithmen für Baumstrukturen (z.B. [CGM97, BCD95]) arbeiten auf geordneten oder ungeordneten Bäumen<sup>5</sup> und bestimmen ein Editskript, welches die eine Version in die andere Version überführt. Die Editskripte basieren auf einer Menge von Änderungsoperationen, wie einfügen, löschen, ändern oder verschieben von Knoten. Weitere Algorithmen bestimmen Differenzen zwischen XML-Dokumenten z.B. [CAM02, WDC03]. Sie sind erweiterte Versionen der Differenzalgorithmen für Bäume, die die speziellen Eigenschaften der XML-Dokumente berücksichtigen. Diese Algorithmen arbeiten auf unversionierten Dokumenten, so daß sie die Informationen eines VM-Systems nicht nutzen, insbesondere auch nicht die Knotenidentifizierer.

Differenzalgorithmen in VM-Systemen (z.B. [Wes91, Ask94]) berücksichtigen die Identifizierer bei der Suche nach korrespondierenden Knoten und besitzen daher eine lineare Laufzeit hinsichtlich der Anzahl der Knoten. Im Gegensatz hierzu ist die Differenzberechnung in ungetypten und ungeordneten Bäumen als NP-hart nachgewiesen [ZWS95].

Es gibt auch Vorschläge, die ein VM-System für Softwaredokumente wie Klassendiagrammen beschreiben (z.B. [RW98]), oder es gibt Forschungs-Prototypen, die eine Mischfunk-

<sup>4</sup>Ohne die Identifizierer müßten die Objektversionen anhand einer gemeinsamen Menge von Attribut-Werten, Komponenten und Beziehungen bestimmt werden, wie z.B. in [Weh04].

<sup>5</sup>Bei geordneten Bäumen existiert eine Ordnung zwischen Geschwisterknoten, bei ungeordneten Bäumen nicht.

tion oder Algorithmen um Differenzen zwischen Klassendiagrammen zu bestimmen anbieten, z.B. [ZWR01]. Sie basieren dabei auf einem feinkörnigen Editiermodell. ClearCase [Whi00], ein kommerzielles VM-System unterstützt auch die Versionierung von UML-Modellen. ClearCase nutzt dabei externe Werkzeuge, um Differenzen zwischen Dateiversionen zu berechnen, anzuzeigen oder um sie zu mischen. Um Versionen von UML-Modellen, die mit Rational Rose erstellt wurden, zu vergleichen wird der ModelIntegrator, der mit Rose ausgeliefert wird, verwendet. Der ModelIntegrator zeigt die Differenzen in einer Baumdarstellung an, die auch alle Metadaten enthält, wie z.B. die eindeutigen Identifizierer oder die Layoutdaten. Das hat den Nachteil, daß die Benutzer dieses Werkzeugs die ursprünglichen Diagramme nur mit einem erhöhtem Aufwand wiedererkennen können. Nach dem Lösen der im ModelIntegrator angezeigten Konflikte kann die neue Version in Rose angezeigt werden, jedoch ohne Differenzinformationen. Desweiteren berechnen Selonen [Sel03] und Alanen [AP03] Differenzen zwischen unversionierten UML-Modellen und mischen die Modelle. Das Mischen entspricht in diesen Fällen jedoch einer Vereinigung und berücksichtigt somit keine Löschungen.

Die meisten bekannten Konzepte und Werkzeuge sind generisch, so daß kein spezielles Metamodell vorausgesetzt wird (textuelle Differenzwerkzeuge sind prinzipiell auf alle Arten von Texten anwendbar, teilweise jedoch mit schlechten Ergebnissen). Der ModelIntegrator und unser vorgestelltes Konzept sind auf ein bestimmtes Metamodell ausgerichtet, so daß sie nicht mit beliebigen Dokumenttypen nutzbar sind.

Es gibt nur sehr wenige Werkzeuge, die Differenzen zwischen versionierten UML-Diagrammen anzeigen (ModelIntegrator und [Gir02]). Diese stellen das Modell entweder als Baum dar, oder verwenden eine große Anzahl an Farben, die die Übersicht der Differenzanzeige deutlich reduziert. Ansätze, die die Differenzen gruppieren und nur einzelne Differenzgruppen markieren sind uns nicht bekannt.

## 6 Zusammenfassung und Ausblick

In dieser Arbeit haben wir die Differenzen, die zwischen UML-Diagrammen auftreten können, klassifiziert und eine Methode zu deren Anzeige vorgestellt. Bei Differenzwerkzeugen für UML-Diagramme tritt, vergleichbar mit textuellen Differenzwerkzeugen, der Effekt auf, daß „zu viele“ Differenzen die Übersicht erheblich beeinträchtigen können. Das von uns vorgeschlagene Konzept bietet die Möglichkeit, Differenzen zu gruppieren und so die Anzeige auf für den Werkzeuganwender interessante Differenzen einzuschränken. Das ist möglich durch die Nutzung von Informationen, die durch das unterliegende VM-System geliefert werden.

Die diesem Text zugrundeliegende Dissertation betrachtet neben den hier skizzierten Versionsverwaltungskonzept und den Konzepten zur Differenzanzeige sowie des Mischens auch Möglichkeiten der synchronen Kooperation zwischen Entwicklern auf einer Version und die Verwaltung von Diagrammen anhand von Änderungsaufgaben.

Die beschriebenen Konzepte sind implementiert und funktionsfähig. Das Konzept zur Differenzanzeige läßt sich auch ohne das VM-System nutzen und wurde zwischenzeitlich als

ein Plugin für Fujaba realisiert, welches Differenzen zwischen unversionierten Modellen anzeigen kann [Weh04].

## Literatur

- [AP03] Marcus Alanen und Ivan Porres. Difference and Union of Models. Bericht 527, TUCS - Turku Centre for Computer Science, April 2003.
- [Ask94] Ulf Asklund. Identifying Conflicts During Structural Merge. In Boris Magnusson, Hrsg., *Proceedings of NWPER'94, Nordic Workshop on Programming Environment Research*, Juni 1994.
- [BCD95] David T. Barnard, Gwen Clarke und Nicolas Duncan. Tree-to-tree Correction for Document Trees. Bericht, Departement of Computing and Information Science Queen's University Kingston Ontario, Canada, Januar 1995.
- [CAM02] Grégory Cobéna, Serge Abiteboul und Amélie Marian. Detecting Changes in XML Documents. In *18. International Conference on Data Engineering (ICDE) San Jose, California, USA, February 26-March 1, 2002*, 2002.
- [CGM97] S. Chawathe und H. Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seiten 26–37, Mai 1997.
- [CW98] Reidar Conradi und Bernhard Westfechtel. Version models for software configuration management. *ACM Computing Surveys*, 30(2):232–282, Juni 1998.
- [Gir02] Martin Girschick. UMLDiff: Erkennung und Analyse von Unterschieden in Klassendiagrammen und Sequenzdiagrammen. Diplomarbeit, Technical University of Darmstadt, 2002.
- [Kel92] Udo Kelter. H-PCTE – a high-performance object management system for system development environments. In *Proceedings COMPSAC Illinois, September 23-25*, Seiten 45–50. IEEE Press, 1992.
- [KM99] Udo Kelter und Marc Monecke. Eine Architektur zur effizienten Konstruktion verteilter datenbankbasierter Anwendungen. In *Proceedings der 5. Fachkonferenz Smalltalk und Java in Industrie und Ausbildung (STJA)*, 28.-30. September 1999, Erfurt, 1999.
- [Mye86] E. W. Myers. An  $O(ND)$  difference algorithm and its variations. *Algorithmica*, 1:251–256, 1986.
- [Ohs04] Dirk Ohst. *Versionierungskonzepte mit Unterstützung für Differenz- und Mischwerkzeug*. Dissertation, Praktische Informatik, Universität Siegen, 2004. URN: urn:nbn:de:hbz:467-831.
- [OK02] Dirk Ohst und Udo Kelter. A Fine-grained Version and Configuration Model in Analysis and Design. In *Proc. of the IEEE International Conference on Software Maintenance 2002 (ICSM 2002)*, 3-6 October 2002, Montreal, Canada, 2002.
- [OMG03] OMG. *Unified Modeling Language Specification*. OMG, Marz 2003. Version 1.5 formal/03-03-01.

- [OWK03] Dirk Ohst, Michael Welle und Udo Kelter. Differences between Versions of UML Diagrams. In *Proc. of the fourth joint meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003), September 1-5, 2003, Helsinki, Finland*, Seiten 227–236. ACM Press, 2003.
- [Pla99] Dirk Platz. *Ein Werkzeugtransaktionskonzept für Objekt-Managementsysteme als Basis von Software-Entwicklungsumgebungen*. Dissertation, Praktische Informatik, Universität Siegen, Juni 1999.
- [RW98] Jungkyu Rho und Chisu Wu. 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, Dezember 1998.
- [See97] Jochen Seemann. Extending the Sugiyama Algorithm for Drawing UML Class Diagrams: Toward Automatic Layout of Object-Oriented Software Diagrams. In Giuseppe Di Battista, Hrsg., *Proc. 5th Int. Symp. Graph Drawing, GD*, number 1353 in Lecture Notes in Computer Science, LNCS, Seiten 415–424. Springer-Verlag, 18–20 September 1997.
- [Sel03] Petri Selonen. Set Operations for Unified Modeling Language. In *Proceedings of the Eight Symposium on Programming Languages and Tools, SPLST'2003, Kuopio, Finland, June*, Seiten 70–81. Kuopio, Finland: University of Kuopio, 2003.
- [Tic84] Walter F. Tichy. The String-to-String Correction Problem with Block Moves. *ACM Transactions on Computer Systems*, 2(4):309–321, November 1984.
- [WDC03] Yuan Wang, David J. DeWitt und Jin-Yi Cai. X-Diff: An Effective Change Detection Algorithm for XML Documents. In *19th International Conference on Data Engineering, March 5 - March 8, 2003 - Bangalore, India*, 2003.
- [Weh04] Jürgen Wehren. Ein XMI-basiertes Differenzwerkzeug für UML-Diagramme. Diplomarbeit, Universität Siegen, 2004. In Vorbereitung.
- [Wes91] Bernhard Westfechtel. *Revisions- und Konsistenzkontrolle in einer integrierten Softwareentwicklungsumgebung*, Jgg. 280 of *Informatik-Fachberichte*. Springer-Verlag, Berlin - Heidelberg - New York, 1991.
- [Whi00] Brian A. White. *Software Configuration Management Strategies and Rational ClearCase*. Addison-Wesley, 2000.
- [Yan94] Wu Yang. How to Merge Program Texts. *The Journal of Systems and Software*, 27(2):129–141, November 1994.
- [ZWR01] Albert Zuendorf, Joerg Wadsack und Ingo Rockel. Merging Graph-Like Object Structures. In Andre van der Hoek, Hrsg., *Tenth International Workshop on Software Configuration Management (SCM-10) New Practices, New Challenges, and New Boundaries May 14-15, 2001 Toronto, Canada (an workshop of 23th ICSE 2001)*. <http://www.ics.uci.edu/~andre/scm10/>, 2001.
- [ZWS95] Kaizhong Zhang, Jason Tsong-Li Wang und Dennis Shasha. On the Editing Distance between Undirected Acyclic Graphs and Related Problems. In Zvi Galil und Esko Ukkonen, Hrsg., *Combinatorial Pattern Matching, 6th Annual Symposium*, Jgg. 937 of *Lecture Notes in Computer Science*, Seiten 395–407, Espoo, Finland, 5-7 Juli 1995. Springer.