Transaktionsorientiertes Mischen von Modellen

Maik Schmidt, Sven Wenzel, Udo Kelter
Praktische Informatik, Universität Siegen
{mschmidt,wenzel,kelter}@informatik.uni-siegen.de

Abstract: Modellbasierte Softwareentwicklung wird zunehmend gängige Praxis. Da immer umfangreichere Modelle arbeitsteilig in Teams entwickelt werden, sollte das parallele Bearbeiten von Modellen durch ähnliche Mischfunktionen unterstützt werden, wie man es für textuelle Dokumente gewohnt ist. Wir analysieren zunächst die technischen Randbedingungen bei der modellbasierten Softwareentwicklung; demzufolge müssen Mischprozesse bei Modellen anders gestaltet werden als bei Texten. Auf dieser Basis stellt das Papier ein Design für Mischwerkzeuge vor, das einen hohen Grad an Konsistenz der Mischergebnisse erzielt und gleichzeitig Verluste an Arbeit infolge von Konfliktauflösungen minimiert.

1 Motivation

Modellbasierte Softwareentwicklung wird zunehmend gängige Praxis, sowohl unter Nutzung allgemeiner Modellierungssprachen wie der UML oder von Modelltypen wie Matlab/Simulink oder ASCET, die für bestimmte Domänen spezialisiert sind. Zudem werden inzwischen größere Modelle arbeitsteilig entwickelt, was üblicherweise durch Repositories unterstützt wird [EKSW07]. Ein klassisches Problem besteht in der gleichzeitigen Bearbeitung desselben Dokumentes durch verschiedene Teammitglieder, da die parallelen Änderungen wieder zusammengeführt werden müssen. Für die modellbasierte Softwareentwicklung braucht man daher Mischwerkzeuge, die analoge Leistungen wie Mischwerkzeuge für textuelle Dokumente anbieten. Das Mischen von Modellen unterscheidet sich in einigen Randbedingungen erheblich vom Mischen textueller Dokumente. In Abschnitt 4 stellen wir ein Mischverfahren für Modelle vor, das einen hohen Konsistenzgrad der Mischergebnisse gewährleistet und zugleich die Vollständigkeit der Mischung maximiert.

2 Begriffsdefinitionen

3-Wege-Mischen. Beim 3-Wege-Mischen sind zwei Dokumente D1 und D2 gegeben, die beide Revisionen einer Basisversion D0 sein müssen. Wir unterstellen immer, dass alle Dokumente den gleichen Typ haben. Die Mischung ist ein neues Dokument, das möglichst alle Änderungen in D1 bzw. D2 gegenüber D0, wozu auch Löschungen zählen, enthalten soll. Die Änderungen in D1 bzw. D2 gegenüber D0 können im Prinzip auf beliebige Weise bestimmt werden, z.B. durch Protokollierung in Editoren [SZN04]; üblicher ist die

Berechnung einer asymmetrischen Differenz [KSW08] von D0 nach D1 bzw. D2 oder einer speziellen 3-Wege-Differenz.

Editierdatentyp. Wir unterstellen, dass für jeden Modelltyp elementare Editieroperationen definiert sind, u.a. Operationen zum Einfügen, Löschen und Ändern von Modellelementen. Konzeptuell liegen diese Operationen den Kommandos in Editoren zugrunde. Wir unterstellen ferner, dass die Änderungen in D1 bzw. D2 gegenüber D0 mittels dieser Operationen ausgedrückt werden und dass diese Operationen durch einen abstrakten Datentyp definiert sind. Wir bezeichnen diesen Datentyp als den Editierdatentyp. Editierdatentypen unterstellen immer eine bestimmte Repräsentation der Modelle, i.d.R. als abstrakter Syntaxgraph mit mehr oder minder umfangreichen kontextsensitiven Konsistenzbedingungen.

Modelleditoren können nur solche Modelle darstellen und verarbeiten, die gewisse Konsistenzbedingungen einhalten. Mischverfahren arbeiten dagegen oft auf einer "internen" Repräsentation der Modelle, die diese Konsistenzbedingungen nicht einhält. Die Mischergebnisse sind dann nicht konsistent genug für graphische Editoren und müssen z.B. als XML-Datei mit entsprechenden Editoren korrigiert werden.

Konflikte und Mischentscheidungen. Ein Paar von je einer Änderung aus D1 bzw. D2 gegenüber D0 kann unverträglich insofern sein, dass beide Änderungen zusammen technisch nicht möglich sind[1] oder einen Modellzustand erzeugen, der bestimmte, vom Modelltyp abhängende Konsistenzbedingungen verletzt. Ein solches Paar von Änderungen bildet einen **Konflikt**. Änderungen, die nicht in Konflikt stehen, können gemeinsam in das Mischergebnis übernommen werden. Bei einem Konflikt muss hingegen eine der beiden Änderungen weggelassen werden. Dies bezeichnen wir als eine **Mischentscheidung**. Hierfür sind zwei Vorgehensweisen in der Praxis bedeutend:

1. **nichtinteraktives Mischen:** Es wird zunächst ein Mischergebnis erzeugt, in dem *beide Änderungen* dargestellt werden. Hierzu müssen ggf. die veränderten Stellen dupliziert oder zusätzliche *Hilfsdaten* angelegt werden, die weder zu D1 noch D2 gehören[2] Verschiebungen und andere komplexe Operationen können so i.d.R. nicht dargestellt werden. Das Mischergebnis ist durch die Hilfsdaten oder die unverträglichen Änderungen inkonsistent und muss manuell nachbearbeitet werden. Erst hierbei wird die eigentliche Mischentscheidung getroffen. Ein Vorteil dieser Vorgehensweise ist, dass eine beliebige weitergehende Editierung zur Konfliktauflösung möglich ist. Die Variante, die Opfer der Konfliktauflösung ist, kann nach Modifikationen trotzdem in das Mischergebnis übernommen werden und muss nicht komplett neu eingegeben werden.
2. **interaktives Mischen:** Ein Entwickler wird während des Mischens um die Mischentscheidung gebeten, der Mischvorgang wird also interaktiv. Denkbare Entscheidungen sind: Übernahme der Änderung gemäss D1 oder D2, Übernahme keiner Änderung, Übernahme von modifizierten Änderungen. Bei diesem Ansatz wird garantiert, dass das Mischergebnis wieder ein konsistentes Modell ist. Hauptnachteil dieses Ansatzes ist, dass für jeden Modelltyp ein eigenes interaktives Mischwerkzeug realisiert werden muss. 3-Wege-Mischwerkzeuge sind deutlich komplexer als reine Differenzanzeigewerkzeuge [KSW08] bzw. davon relativ leicht ableitbare 2-Wege-Mischwerkzeuge.

[1] Technisch gesehen würde wenigstens eine Ausführungsreihenfolge einen Fehler erzeugen und scheitern.
[2] Die automatische Mischfunktion in RCS (bzw. CVS oder SVN) arbeitet nach diesem Prinzip.

3 Vorhandene Lösungsansätze

Das Mischen von Modellen ähnelt dem Mischen von Programmen, wenn man diese als abstrakte Syntaxgraphen betrachtet. Die Algorithmen hierfür [Mens02] orientieren sich allerdings bzgl. der Charakteristika der Graphen und beim Konfliktbegriff stark an den Strukturen in Quellprogrammen und sind auf fast alle Modelltypen nicht übertragbar. Das "Mischen" von Modellen wird weiterhin adressiert durch einige Papiere, die Repository-Systeme für Modelle vorstellen, z.B. [MCGW08]. Tatsächlich sind die dort als "Mischen" bezeichneten Funktionen eher patch-Funktionen.

Nur sehr wenige praktisch nutzbare Werkzeuge zum Mischen von Modellen sind verfügbar. In der Praxis meidet man daher notgedrungen Situationen, in denen Mischen notwendig wird [BE08], d.h. verschiedene Entwickler dürfen nur strikt nacheinander oder an disjunkt partitionierten Modellen arbeiten.

Das Graphical Merge Tool von Matlab/Simulink[3] ist ein Differenzanzeigewerkzeug für Matlab/Simulink-Modelle, in das rudimentäre Mischfunktionen integriert sind. Es zeigt zwei zu vergleichende Modelle in ihrer grafischen Repräsentation und als Objektbaum an. Im Objektbaum sind die lokalen Differenzen markiert. Man kann für einzelne lokale Differenzen die Zustände vom einen Modell in das andere übertragen. Es handelt sich um ein reines 2-Wege-Mischen, für jede einzelne lokale Differenz ist daher eine manuelle Mischentscheidung nötig; dies ist ein langwieriger und fehlerträchtiger Prozess.

Die Vergleichskomponente des Rational Software Architect[4] unterstützt neben der reinen Differenzdarstellung sowohl das 2-Wege- als auch das 3-Wege-Mischen. In der grafischen Darstellung werden das Basisdokument, die zu mischenden Varianten sowie das vorläufige Mischergebnis in der gewohnten grafischen Notation angezeigt. Zudem wird eine Liste der Mischkonflikte angezeigt. Zusammenhängende Änderungen, die sich z.B. auf das gleiche Modellelement beziehen, werden gruppiert. Bei jedem Konflikt ist eine der möglichen Änderungen interaktiv zu wählen. Für Gruppen von Konflikten können alle Änderungen des einen oder des anderen Dokuments ausgewählt werden, wodurch die Zahl der Einzelentscheidungen reduziert wird. Zwei in Konflikt stehende Änderungen können mit dem Rational Software Architect nicht in das Mischergebnis übernommen werden.

4 Transaktionsorientiertes Mischen von Modellen

In diesem Abschnitt stellen wir ein Verfahren zum 3-Wege-Mischen von Modellen vor, das sich weniger an den aktuellen Zuständen der zu mischenden Modelle orientiert, sondern vielmehr an den elementaren Editierungen, die zu diesen Zuständen geführt haben.

[3] http://www.mathworks.de/products/simulink, 2007
[4] http://www.ibm.com/software/awdtools/architect/swarchitect, 2008

4.1 Transaktionen

Editiervorgänge innerhalb eines Versionszweigs lassen sich in Gruppen einteilen, die jeweils einer bestimmten Arbeitsaufgabe entsprechen, z.B. dem Beheben eines Fehlers oder dem Einfügen einer neuen Funktion, und die das Modell von einem konsistenten Zustand in einen anderen überführen. Eine solche Gruppe von Editieroperationen bezeichnen wir als *Transaktion*. Eine Grundidee des transaktionsorientierten Mischens ist, die Transaktionen, also Sequenzen von Änderungen, als "Bürger erster Ordnung" aufzufassen, die als Datenobjekte editiert und ggf. sogar dauerhaft gespeichert[5] werden.

Werden Modelle mit einem Versionsmanagementsystem verwaltet, lassen sich die Änderungen zwischen zwei Revisionen in erster Näherung als eine Transaktion ansehen. Bei diszipliniertem Gebrauch dokumentiert die *commit message* sogar den Zweck der Transaktion. Das Mischwerkzeug sollte Funktionen anbieten, mit denen man solche "Roh-Transaktionen" zusammenlegen oder aufteilen kann, wenn z.B. mehrere fachliche Änderungen innerhalb einer Revision umgesetzt wurden. Die Editieroperationen in einer Transaktion können im Prinzip durch Vergleich der beiden Revisionen (vgl. [KWN05]) bestimmt werden. Um allerdings Modellelemente über die ganze Versionshistorie hinweg korrekt zu identifizieren, muss die komplette Versionshistorie bis zum gemeinsamen Basisdokument analysiert werden. Eine Technik hierzu ist in [WHK07] beschrieben. Alternativ kann man Transaktionen auch durch Protokollierung in Editoren und anschließende Bereinigung der Protokolle gewinnen.

4.2 Abhängigkeiten und Konflikte

Editieroperationen werden immer auf einem Kontext, der aus referierten Modellelementen besteht, ausgeführt und erzeugen, verändern oder entfernen Modellelemente. Zwischen zwei *Editieroperationen im gleichen Zweig* liegt eine **Abhängigkeit** vor, wenn die spätere Operation auf Ergebnissen der früheren Operation aufbaut. Zwischen zwei *Editieroperationen in verschiedenen Zweigen* liegt ein **Konflikt** vor, wenn sie im einfachsten Fall dasselbe Element verändern oder den benötigten Kontext der anderen Operation verändern. Allgemeiner liegt ein Konflikt vor, wenn beim gegebenen Editierdatentyp die beiden Operationen bei Hintereinanderausführung andere Ergebnisse erzeugen, wenn man die Ausführungsreihenfolge vertauscht. Je nach Editierdatentyp sind "nichtlokale" Konflikte möglich, z.B. durch Kardinalitäten im Metamodell: zwei Editieroperationen, die jeweils ein Element anlegen, können zusammen eine Kardinalitätsgrenze überschreiten, d.h. die zweite Operation würde bei Hintereinanderausführung scheitern.

Zwei *Transaktionen* sind abhängig voneinander bzw. stehen in Konflikt, wenn sie wenigstens ein Paar von Editieroperationen enthalten, die abhängig voneinander sind bzw. in Konflikt stehen.

[5] Bei scharfen Produkthaftungsbedingungen, die z.B. für eingebettete Software in Fahrzeugen typisch sind, müssen ohnehin alle Entwicklungsschritte genauestens dokumentiert werden.

4.3 Transaktionsorientiertes Mischen

Die Gruppierung einzelner Editieroperationen zu Transaktionen kann den Mischprozess in mehrfacher Hinsicht vereinfachen und verbessern: Die Transaktionen gruppieren die aus Anwendersicht i.d.R. zu kleinteiligen Einzeloperationen und bilden in sich geschlossene Änderungen, die ggf. einzelnen Änderungsanforderungen zugeordnet werden können. Mischentscheidungen werden so verständlicher und ggf. ohne Mehraufwand dokumentiert. Die Verwendung von Transaktionen als Granularität der Mischentscheidungen verringert die Anzahl der manuell zu entscheidenden Konflikte und stellt zudem sicher, dass Mischergebnisse im Sinne der intendierten Änderungen semantisch vollständig sind. Der Mischvorgang auf Basis von Transaktionen beinhaltet folgende Hauptschritte:

1. Automatische Analyse der Historien einzelner Modellelemente gemäß [WHK07].
2. Definition der Transaktionen: in einfachsten Fall anhand von check-ins in einem Repository, ggf. manuelle Nachbearbeitung (s.o.)
3. Abhängigkeiten und Konflikten zwischen Operationen und Transaktionen bestimmen.
4. Berechnung des "Transaktionswertes" und anderer Hilfsdaten; der **Wert einer Transaktion** ist typischerweise definiert als die Zahl der enthaltenen Änderungen, die je nach involvierten Elementtypen gewichtet werden. Allgemeiner können Transaktionen mittels Differenzmetriken [Wen08] bewertet werden. Die Bewertung unterstützt später die Auswahl der in das Mischergebnis übernommenen Änderungen.
5. Berechnung eines initialen Vorschlags für Mischentscheidungen auf Transaktionsebene: Oft ist es sinnvoll, einem der beiden Zweige höhere Priorität zuzuordnen und bei allen Konflikten zunächst zugunsten dieses Zweiges zu entscheiden. Alternativ kann es sinnvoll sein, den Gesamtwert der gewählten Transaktionen anhand einfacher Heuristiken zu optimieren, z.B. die jeweils wertvolleren Transaktionen (unter Berücksichtigung von abhängigen Transaktionen) zu wählen. Weitergehende Optimierungen können darin bestehen, z.B. die Zahl bestimmter Modellelemente im Ergebnis zu maximieren.
6. Iterative manuelle Korrektur der Mischentscheidungen auf Transaktionsebene, jeweils Neuberechnung des Gesamtwerts der gewählten Transaktionen
7. Manuelle Behandlung der "geopferten" Transaktionen: i.w. wird hier die Operationsliste einer solchen Transaktion editiert und die modifizierte Transaktion am Ende freigegeben. Hierzu sollten für eine betroffene Transaktion die darin enthaltenen Editieroperationen angezeigt werden. Ein Entwickler mit entsprechendem Domänenwissen kann dann die Operationsliste verändern, indem einzelne Operationen hinzugewählt oder deselektiert werden oder indem Parameter modifiziert werden. Während die vorigen Schritte nur komplette Transaktionen in das Mischergebnis übernehmen und damit annahmegemäß automatisch die Konsistenz der Modelle erhalten, liegt die Verantwortung für die Konsistenzerhaltung hier beim Entwickler.
8. Generierung eines Mischergebnisses auf Basis der Mischentscheidungen

Die ausgewählten Transaktionen bzw. Einzeloperationen, die in das Mischergebnis aufgenommen werden sollen, lassen sich einfach speichern. Dies bietet zwei Vorteile: erstens kann der Mischvorgang jederzeit unterbrochen und später fortgesetzt werden. Zweitens

kann ein Test des Mischergebnisses zeigen, dass bei Editieren der geopferten Transaktionen Fehler gemacht wurden; diese Fehler können behoben werden und das Mischergebnis kann leicht neu generiert werden.

5 Fazit

Das Mischen von Modellen wird bislang nur unzureichend von Werkzeugen unterstützt. In diesem Papier haben wir die wesentlichen Probleme beim Mischen von Modellen analysiert; viele Prinzipien und Details von Mischvorgängen, die sich bei textuellen Dokumenten bewährt haben, können nicht für Modelle übernommen werden. Ferner haben wir ein Mischverfahren vorgestellt, das die Änderungshistorie der Modelle ausnutzt und eine Mischung auf Basis von Transaktionen erlaubt. Hierdurch wird die Zahl der Mischentscheidungen reduziert. Während konventionelle Mischverfahren sich vor allem an den Zuständen der zu mischenden Dokumente orientieren, ist dieses Verfahren änderungs- und historienorientiert und bezieht sich explizit auf einen konsistenzerhaltenden Editierdatentyp für den jeweiligen Modelltyp. Hierdurch und weil Transaktionen konsistente Zustände erhalten, wird die Überwachung der syntaktischen und semantischen Korrektheit der Mischung unterstützt. Das Verfahren erlaubt Mischenentscheidungen sowohl auf der Ebene von der Transaktionen als auch auf der Ebene einzelner Editieroperationen, und reduziert hierdurch den Verlust an geleisteter Entwicklungsarbeit.

Literatur

[BE08] Bendix, L.; Emanuelsson, P.: Diff And Merge Support For Feature Oriented Development; In: Proc. ICSE Workshop CVSM, 2008; S.31-34

[EKSW07] Ebert, J.; Kelter, U.; Schürr, A.; Westfechtel, B.: Workshopbericht Vergleich und Versionierung von UML-Modellen, Hamburg; Softwaretechnik-Trends 27:2; 2007

[KSW08] Kelter, U.; Schmidt, M.; Wenzel, S.: Architekturen von Differenzwerkzeugen für Modelle; S.155-168 in: Software Engineering 2008, München; 2008

[KWN05] Kelter, U.; Wehren, J.; Niere, J.: A Generic Difference Algorithm for UML Models; In: Proc. GI-Fachtagung Software Engineering, Essen, 2005

[Mens02] Mens, T.: A State-of-the-Art Survey on Software Merging; In: IEEE Transactions on Software Engineering 28:5; 2002

[MCGW08] Murta, L. et al: Towards Odyssey-VCS 2: Improvements Over A UML-based Version Control System; In: Proc. ICSE Workshop CVSM, 2008; S.25-30

[SZN04] Schneider, C.; Zündorf, A. ; Niere, J.: CoObRA - a small step for development tools to collaborative environments; In: Proc. WoDiSEE, 2004

[Wen08] Wenzel, S.: Scalable Visualization of Model Differences In: Proc. ICSE Workshop on Comparison and Versioning of Software Models, Leipzig, 2008; S.41-46

[WHK07] Wenzel, S.; Hutter, H.; Kelter, U.: Tracing model elements; In: Proc. of the 23rd International Conference on Software Maintenance, Paris, 2007