Berechnung und Propagation von Modelländerungen auf der Basis von Editieroperationen[1]

Timo Kehrer[2]

Abstract: Modellbasierte Softwareentwicklung hat sich in verschiedensten Applikationsdomänen fest etabliert. Größtenteils visuelle Modelle wie bspw. diverse Varianten verschiedener Modelltypen der UML werden hierbei zum integralen Bestandteil aller Phasen modellbasierter Entwicklungsprozesse. Modelle unterliegen somit kontinuierlichen Änderungen und existieren im Laufe ihrer Evolution in zahlreichen Versionen und Varianten. In der Praxis zeigt sich sehr deutlich, dass die gleichen grundlegenden Werkzeugfunktionen für das Versionsmanagement von Modellen benötigt werden, die man von klassischen Repository-Systemen für textuelle Dokumente gewohnt ist. Qualitativ hochwertige und für ein breites Spektrum unterschiedlicher Modelltypen einsetzbare Differenz-, Patch- und Mischwerkzeuge sind jedoch nicht verfügbar. Einen umfassenden und generischen Lösungsansatz beschreibt die hier zusammengefasste Dissertation. Diverse Zusammenarbeiten mit anderen Wissenschaftlern und der Industrie zeigen das große Interesse an der Arbeit und belegen ferner die breite Anwendbarkeit der entwickelten Konzepte und Techniken.

Keywords: Modellbasierte Softwareentwicklung, Softwareevolution, Konfigurationsmanagement, Versionsverwaltung, Differenzberechnung, Änderungspropagation

1 Einführung

Modellbildung ist ein Mittel zur Abstraktion auf nahezu allen Gebieten der Informatik. In der Softwaretechnik spielen Modelle eine zunehmend zentrale Rolle, um die stetig wachsende Komplexität softwareintensiver Systeme zu beherrschen. Modellbasierte Softwareentwicklung (MBSE) [BCW12] hat sich in einigen Applikationsdomänen fest etabliert, zu nennen sind hier insbesondere der Automotive-, der Luft- und Raumfahrt- sowie der Telekommunikationssektor. Die Einsatzbereiche für Modelle sowie die eingesetzten Modelltypen sind vielfältig. Größtenteils visuelle Modelle, so z.B. diverse Varianten verschiedener Modelltypen der UML, werden hierbei zum integralen Bestandteil aller Phasen modellbasierter Entwicklungsprozesse. Modelle sind somit primäre Artefakte, unterliegen ständigen Änderungen und existieren im Laufe ihrer Evolution in zahlreichen Versionen und Varianten. Letztere müssen oftmals parallel weiter entwickelt und gewartet werden. Ferner muss die kollaborative Entwicklung von Modellen koordiniert werden. Die Versionierung von Software wird klassischerweise durch Repository-Systeme wie CVS, SVN oder Git unterstützt. Essentielle Repository-Dienste sind, neben Basisfunktionen zur Verwaltung von Versionsgraphen und administrativer Daten, der Dokumentenvergleich sowie die

[1] Englischer Titel der Dissertation: “Calculation and Propagation of Model Changes Based on User-level Edit Operations: A Foundation for Version and Variant Management in Model-driven Engineering” (URN: urn:nbn:de:hbz:467-9633) [Ke15a]. Die Dissertation entstand während der Arbeit des Autors als Wissenschaftlicher Mitarbeiter an der Universität Siegen unter der Betreuung von Prof. Dr. Udo Kelter (Universität Siegen) und Prof. Dr. Gabriele Taentzer (Philipps-Universität Marburg). Der Autor wurde teilfinanziert durch das DFG-Schwerpunktprogramm 1593 “Design for Future – Managed Software Evolution”.

[2] Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Italy.

Propagation von Dokumentänderungen, einerseits zwischen unterschiedlichen Entwicklungszweigen (“Cherry Picking” [CSFP04]), andererseits vom zentralen Repository auf lokale Arbeitsbereiche (“Workspace Update” [CSFP04]). Klassische Differenz-, Patch- und Mischwerkzeuge wurden jedoch für Texte ohne spezielle Struktur konzipiert bzw. reduzieren die konzeptuelle Struktur der verwalteten Dokumente auf eine einfache Sequenz von Textzeilen. Für Source Code und andere textuelle Dokumente führen zeilenbasierte Verfahren zu akzeptablen Ergebnissen, nicht aber für Modelle, insbesondere nicht für visuelle Modelle. Die Softwaretechnik-Forschung hat sich mit dem Aufkommen des MBSE-Paradigmas daher zunehmend der Versionierung von Modellen gewidmet. Die Online Bibliographie[3] zum Versions- und Variantenmanagement von Modellen enthält über 400 Einträge, die meisten davon datieren aus dem Jahr 2003 oder später. Allgemeiner Konsens besteht darin, dass hier Verfahren benötigt werden, welche sich an der für Modelle typischen Graphstruktur orientieren. Dennoch beklagt die industrielle Praxis immer noch einen Mangel an qualitativ hochwertigen und für ein breites Spektrum unterschiedlicher Modelltypen einsetzbaren Differenzwerkzeugen [BE09, Em12]. Auch in der Forschung wird die Modellversionierung noch einige Jahre nach dem Erscheinen der ersten Publikationen auf diesem Gebiet als eine der zentralen Herausforderungen zur Etablierung der modellbasierten Entwicklung identifiziert [Al09].

Abschnitt 2 skizziert die gravierenden Probleme derzeitig verfügbarer Werkzeuge für das Versions- und Variantenmangement von Modellen. Abschnitt 3 leitet daraus die Zielsetzung der hier zusammengefassten Dissertation ab und gibt einen Überblick über die Kernideen des entwickelten Ansatzes. Abschnitt 4 fasst die wesentlichen Ergebnisse und wissenschaftlichen Beiträge zusammen, abschließend resümieren wir in Abschnitt 5.

2 Problemmotivation

Werkzeugfunktionen zum Vergleichen, Patchen und Mischen von Dokumenten basieren im Kern auf einer Repräsentation der Unterschiede zwischen Dokumenten. Konzeptuelle Frameworks [CW98] definieren eine (gerichtete) Differenz $\Delta(v_1, v_2)$ als Sequenz von Änderungsschritten $s_1, ..., s_n$, die, bei Anwendung auf Dokumentversion v_1, Version v_2 erzeugt. Ein Änderungsschritt repräsentiert den Aufruf einer Änderungsoperation mit aktuellen Parametern. Die konzeptuelle Repräsentationsform sowie die für einen Dokumenttyp zur Verfügung stehenden Änderungsoperationen werden von diesen grundlegenden Definitionen offen gelassen. Klassische Verfahren basieren auf Texten und unterstellen sehr einfache Änderungsoperationen, i.W. die Einfügung und Löschung einzelner Textzeilen; ein Ansatz der, wie bereits erwähnt, nicht sinnvoll auf Modelle übertragen werden kann. Im weiteren Verlauf dieses Abschnitts betrachten wir die für Modelle üblichen Interpretationen der in obigen Definitionen genannten Variationspunkte. Daraus leiten wir die Problemmotivation für die hier zusammengefasste Dissertation ab[4].

Elementweise Betrachtung von Modelländerungen. Differenzwerkzeuge für Modelle sollen sich an der für Modelle typischen Graphstruktur orientieren. Eine hierfür geeignete,

[3] `http://pi.informatik.uni-siegen.de/CVSM`

[4] Wir berufen uns hierbei auf den Stand der Technik zu Beginn des Promotionsvorhabens Anfang des Jahres 2011. In der Zwischenzeit publizierte, verwandte Arbeiten werden in der Dissertation ausführlich beleuchtet und von dem hier skizzierten Ansatz abgegrenzt.

konzeptuelle Repräsentationsform ist der abstrakte Syntaxgraph (ASG), dessen Knoten- und Kantentypen sowie ggf. weitere Wohlgeformtheitskriterien üblicherweise durch ein Metamodell definiert werden. Die für Texte übliche, zeilenbasierte Betrachtung von Änderungen wird jedoch nahezu analog auf die ASG-Struktur übertragen: Modelländerungen werden elementweise beschrieben, als verfügbare Änderungsoperationen werden primitive Graphoperationen zum Erzeugen und Löschen von Knoten und Kanten des ASG unterstellt [Al09, KKT11, SC13]. Gewöhnliche Benutzer sind jedoch mit den Details von Metamodellen und der ASG-basierten Repräsentation von Modellen nicht vertraut. Die Beschreibung von Modelländerungen auf Basis primitiver Graphoperationen führt daher zu "low-level" Differenzen, welche schwer zu verstehen sind und hochgradig konfus wirken [Al09, KKT11]. Bei der Propagation von Modelländerungen führen primitive Graphoperationen zu weiteren Problemen, da sie elementare Konsistenzkriterien verletzen können. Im schlimmsten Fall kann ein Modell so inkorrekt werden, dass es mit Standard-Modelleditoren nicht mehr verarbeitet werden kann. Dies gilt insbesondere für visuelle Modelle, welche in diesem Fall nur noch mit einfachen Texteditoren auf Basis der Repräsentation des serialisierten Datenformats (z.B. XML) korrigiert werden können.

Nicht-Verallgemeinerbarkeit existierender Lösungsansätze. Lösungsansätze existieren lediglich für einzelne Modelltypen, so z.B. Prozessmodelle [Kü08], oder unterstellen die Verfügbarkeit umgebungsspezifischer Zusatzinformationen, bspw. die Existenz von Editierprotokollen [HK10]. Dies führt zu erheblichen Einschränkungen. Die vorgeschlagenen Verfahren sind somit nicht in der Breite einsetzbar, wie dies für klassische Repository-Systeme der Fall ist. Die Erfahrung auf dem Gebiet der Versionierung von Source Code hat jedoch gezeigt, dass eine breite Einsetzbarkeit ein wesentlicher Erfolgsfaktor ist [Es05]. Dies wird auch für Modelle erwartet, weshalb wir nicht verallgemeinerbare Ansätze für das Versionsmanagement von Modellen hier nicht näher betrachten. Für einen Überblick sei auf die Beschreibung des Stands der Technik in der Dissertation verwiesen.

3 Zielsetzung und Überblick

Werkzeuge für das Versions- und Variantenmanagement von Modellen sollen nicht auf elementweisen, kleinteiligen Änderungsbeschreibungen basieren, sondern auf Änderungsoperationen, welche für den Benutzer verständlich sind und ferner elementare Konsistenzkriterien zur externen Darstellbarkeit der Modelle erhalten. Beispiele für solche konzeptuellen Beschreibungen von Änderungen sind *Editieroperationen*, wie sie bspw. von Modelleditoren oder modernen Refactoring-Werkzeugen angeboten werden. Benutzer sind mit der Bedienung dieser Werkzeuge und den zur Verfügung stehenden Editierkommandos vertraut. Weitere komplexe Editieroperationen lassen sich aus wiederkehrenden, schematischen Editiertätigkeiten ableiten. Editieroperationen nehmen im Kontext der Modellversionierung zwei zentrale Rollen ein: *(i)* Eine *deskriptive* Rolle zur Beschreibung der zwischen zwei Modellversionen beobachtbaren Änderungen, insbesondere zwischen zeitlich aufeinander folgenden Versionen einer Modellhistorie; *(ii)* Eine *präskriptive* Rolle zur Spezifikation von Änderungen, welche auf einer existierenden Version eines Modells auszuführen sind, insbesondere im Rahmen der Propagation von Modelländerungen.

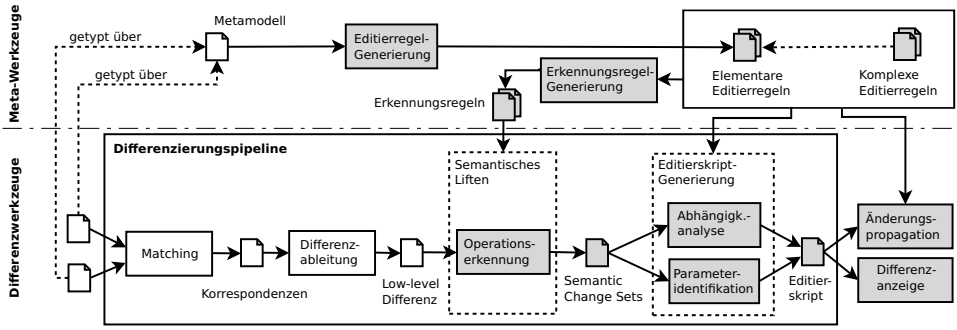

Abb. 1: Übersicht über den entwickelten Ansatz

Ziel der hier zusammengefassten Dissertation ist die systematische Anhebung von Konzepten, Algorithmen, Techniken und Werkzeugen zur Modellversionierung auf ein höheres Abstraktionsniveau basierend auf Editieroperationen. Kernidee des Ansatzes ist es, die für einen Modelltyp verfügbaren Editieroperationen in Form von In-Place Transformationen zu spezifizieren. Hierzu wird die Modelltransformationssprache Henshin [Ar10] genutzt, welche auf Graphtransformationskonzepten [Eh06] basiert. Formale Spezifikationen von Editieroperationen, i.F. als *Editierregeln* bezeichnet, dienen als Konfigurationsparameter für diverse generische Werkzeugkomponenten (s. Abb. 1). Zunächst werden existierende Differenzwerkzeuge für Modelle um die Erkennung von Editieroperationen erweitert (s. Abschnitt 4.1). Die so erhaltenen "semantisch gelifteten Modelldifferenzen" können bei Bedarf in konsistenzerhaltende Editierskripte konvertiert werden (s. Abschnitt 4.2). Diese bilden die Grundlage für die kontrollierte Propagation von Modelländerungen. Wir beschreiben hierzu einen flexiblen Ansatz zum Patchen von Modellen, welcher gängige Propagationsszenarien, insbesondere das "Cherrypicking" und die Aktualisierung lokaler Arbeitsbereiche ("Workspace Update"), unterstützt (s. Abschnitt 4.3). Die Erzeugung konsistenzerhaltender Editierregeln erfolgt größtenteils automatisiert auf Basis des Metamodells für einen gegebenen Modelltyp (s. Abschnitt 4.4). Die erarbeiteten Techniken und Methoden werden in einem universellen Framework zusammengefasst. Eine Referenzimplementierung auf Basis des weit verbreiteten Eclipse Modeling Framework (EMF) zeigt die technische Umsetzbarkeit und ist unter dem Namen *SiLift* [Ke12a] bekannt und frei verfügbar. Die Konzeption des Frameworks basiert auf etablierten Methoden zur Entwicklung von Softwareproduktlinien und ermöglicht es, spezialisierte Werkzeuge und Werkzeugfunktionen für das Versionsmanagement von Modellen zu konfigurieren (s. Abschnitt 4.5).

4 Resultate und wissenschaftliche Beiträge

4.1 Semantisches Liften von Modelldifferenzen

Ziel. Gängige Verfahren des Modellvergleichs liefern große und unstrukturierte Mengen von kleinteiligen Änderungen auf dem ASG. Diese sollen so partitioniert werden, dass jede Teilmenge den Effekt einer Editieroperation repräsentiert. Eine solche Teilmenge bezeichnen wir als *Semantic Change Set*, das Verfahren zur Transformation von low-level

Differenzen hin zu konzeptuellen Beschreibungen von Änderungen auf Basis von Editieroperationen wird als *semantisches Liften von Modelldifferenzen* bezeichnet.

Ansatz. Als Ausgangspunkt für das semantische Liften von Modelldifferenzen unterstellen wir die übliche Struktur zustandsbasierter Vergleichsverfahren, d.h. die beiden ersten Verarbeitungsschritte der in Abb. 1 dargestellten Differenzierungspipeline. Ein graphbasierter Matching-Algorithmus bestimmt im ersten Schritt die korrespondierenden ASG-Elemente der beiden zu vergleichenden Eingabemodelle. Die Ableitung einer low-level Differenz erfolgt in einem zweiten Schritt; nicht korrespondierende ASG-Elemente werden als eingefügt bzw. gelöscht erachtet. Eine zentrale Beobachtung ist, dass Editieroperationen zu charakteristischen Änderungsmustern in der strukturellen Repräsentation von low-level Differenzen führen. Die Erkennung von Semantic Change Sets resultiert somit i.W. in einem Mustererkennungsproblem. Die zu erkennenden Änderungsmuster weisen eine hochgradig komplexe Struktur auf. Eine manuelle Musterspezifikation wird daher bereits für kleine Mengen an Editieroperationen extrem aufwendig und fehleranfällig. Zur Lösung wird ein regelbasierter Ansatz vorgestellt [KKT11]. Die benötigten Suchmuster werden in Form von *Erkennungsregeln* in Henshin formuliert, welche automatisch aus den zugehörigen Editierregeln generiert werden. Erkennungsregeln sind ferner so konstruiert, dass sie parallel auf eine gegebene low-level Differenz angewendet werden können. Falsch-Positive, d.h. überlappende Semantic Change Sets, werden in einem nachgelagerten heuristischen Verfahren behandelt, um eine möglichst optimale Partitionierung der Gesamtmenge an low-level Änderungen zu bestimmen.

Experimentelle Ergebnisse. Experimente mit verschiedenen Testmodellen zeigen, dass sich die Anzahl der in einer Differenz enthaltenen Editierschritte durch semantisches Liften drastisch komprimieren lässt. Die Kompressionsraten variieren je nach Modelltyp und Testreihe; für die UML wurden Kompressionsfaktoren von bis zu 18,0 gemessen [KKT11]. Dies kann als Indikator für die Verbesserung der Verständlichkeit von Differenzen betrachtet werden. Untersuchungen zur Laufzeit des semantischen Liftens zeigen ferner, dass das Verfahren auch für reale Modelle skaliert.

4.2 Generierung ausführbarer Editierskripte

Grundlegende Definitionen und Ziele. Semantisches Liften von Modelldifferenzen verbessert deren Verständlichkeit, die gewonnenen Informationen reichen jedoch nicht aus, um Differenzen in Szenarien der Änderungspropagation auch ausführen zu können. Hierzu definieren wir eine erweiterte Form der gerichteten Differenz, die wir als *Editierskript* bezeichnen. Ein Editierskript enthält zusätzlich zu den auszuführenden Editieroperationen auch aktuelle Parameter und Informationen zu sequentiellen Abhängigkeiten von Editierschritten. Gegeben eine geliftete Differenz sowie die Menge der für einen Modelltyp zur Verfügung stehenden Editierregeln sollen Editierskripte automatisch generiert werden.

Generierung von Editierskripten. Ausgehend von einer semantisch gelifteten Differenz erfolgt die Generierung eines Editierskripts in zwei Schritten: *Parameteridentifikation* und *Abhängigkeitsanalyse* (s. Abb. 1). Aktuelle Regelparameter können anhand der Matches der Erkennungsregeln bestimmt werden. Um eine effiziente Abhängigkeitsanaly-

se zur Laufzeit zu gewährleisten, werden alle kombinatorisch möglichen Paare von Editieroperationen bereits statisch auf potentielle Abhängigkeiten untersucht. Die geschieht auf Basis der Technik der Kritischen-Paar-Analyse [Eh06]. Die so gewonnenen kritischen Paare repräsentieren potentielle Abhängigkeitsbeziehungen in einem minimalen Kontext. Zur Laufzeit prüfen wir für jedes Paar von Editierschritten, deren Editieroperationen potentiell in einer sequentiellen Abhängigkeitsbeziehung stehen, ob diese in einer Differenz tatsächlich auftritt. Details des Verfahrens wurden in [KKT13] vorab veröffentlicht.

4.3 Kontrollierte Propagation von Modelländerungen

Ziele. Die Propagation von Modelländerungen soll insofern kontrolliert ablaufen, als dass die synthetisierten Ergebnisse in Standard-Editoren extern repräsentiert werden können. Diverse Fehlerfälle und Konflikte sollen erkannt und, sofern eine automatisierte Fehlerbehebung bzw. Konfliktlösung nicht plausibel erscheint, dem Benutzer in geeigneter Weise präsentiert werden. In diesem Fall muss eine manuelle Intervention ermöglicht werden.

Ansatz. Zur kontrollierten Propagation von Modelländerungen wird ein flexibles Patch-Verfahren für Modelle vorgeschlagen, welches konsistenzerhaltende Editierskripte als Patches verwendet. Unter einem *konsistenzerhaltenden Editierskript* verstehen wir ein spezielles Editierskript, welches ausschließlich konsistenzerhaltende Editieroperationen verwendet. Der einzuhaltende Konsistenzgrad orientiert sich an Standard-Editoren für Modelle. Dies ermöglicht eine schrittweise und interaktive Anwendung von Patches. Da jeder Editierschritt in einem graphisch darstellbaren Folgezustand resultiert, können die Auswirkungen jedes einzelnen Editierschritts in der externen Darstellung des Zielmodells sichtbar gemacht werden. Darüber hinaus werden diverse Fehlerfälle behandelt und Möglichkeiten für manuelle Eingriffe angeboten [KKK13]. Basierend auf dem Prinzip des Patchens wird ferner ein neuartiger Ansatz zur Aktualisierung von Workspaces vorgestellt, welcher sich technisch und methodisch vom klassischen 3-Wege Mischen unterscheidet [KKR14]. Statt einer aufwendigen Konflikterkennung und -behandlung wird, zusätzlich zu den Fehlererkennungsroutinen des Patch-Verfahrens, eine Warnungsroutine eingeführt, welche das blinde Überschreiben lokaler Änderungen in Arbeitsbereichen verhindert.

Evaluation. Aufgrund des substantiellen Mangels an Patch-Werkzeugen für Modelle konzentriert sich die Evaluation auf das neuartige Verfahren zur Aktualisierung von Arbeitsbereichen. Wir vergleichen hierzu den Ansatz mit dem traditionellen 3-Wege Mischen. Einige Vorzüge sind offensichtlich. So ist z.B. die grafische Benutzeroberfläche des interaktiven Werkzeugs deutlich weniger komplex als jene interaktiver Mischwerkzeuge. Erkauft wird dies durch eine gewisse Unschärfe in der Konflikterkennung; die vorgestellten Varianten zur Realisierung einer Warnungsroutine basieren auf Heuristiken. Benchmarking-Ergebnisse zeigen jedoch, dass Konflikte, deren Erkennung von 3-Wege Mischwerkzeugen erwartet wird, auch von unserem Ansatz detektiert werden.

4.4 Generierung konsistenzerhaltender Editierregeln.

Ziel. Konsistenzerhaltende Editierskripte sind essentielle Grundlage für die kontrollierte Propagation von Modelländerungen. Die Anpassung an einen gegebenen Modelltyp sowie

eine hierfür unterstellte Editierumgebung erfolgt durch die Spezifikation konsistenzerhaltender Editieroperationen. Hierzu wird eine vollständige Menge an konsistenzerhaltenden Editieroperationen benötigt, welche jede mögliche Änderung an beliebigen konsistenten Modellen eines gegebenen Typs abdeckt. Die Spezifikation soll daher durch geeignete Meta-Werkzeuge unterstützt werden.

Ansatz. Zur Unterstützung der Erzeugung konsistenzerhaltender Editierregeln beschreiben wir einen Algorithmus welcher, gegeben ein Metamodell mit Multiplizitäten, eine vollständige Menge an Editierregeln erzeugt. Der Erhalt elementarer Konsistenzkriterien und Multiplizitäten wird formal nachgewiesen. Der vorgestellte Ansatz geht davon aus, dass Standard-Metamodelle zunächst zu effektiven Metamodellen für eine gegebene Editierumgebung reduziert werden. Grundlegende Ideen wurden in [Ke13b, RKK14, Ke16] publiziert.

Evaluation. Die durchgeführten Untersuchungen zeigen, dass nicht-lokal wirksame Wohlgeformtheitskriterien für effektive Metamodelle eine sekundäre Rolle spielen. Lediglich ausgereifte, meist kommerzielle UML-Editoren fordern die Einhaltung weniger solcher Kriterien. Der Aufwand zur Nachbearbeitung generierter Editierregeln ist daher gering. Meist sind nur wenige Regeln um zusätzliche Anwendungsbedingungen zu ergänzen.

4.5 Methodik zur Entwicklung von Werkzeugen für die Modellversionierung

Ziel. Für ein qualitativ hochwertiges Versionsmanagement von Modellen werden für jeden Modelltyp eigene, dedizierte Werkzeuge benötigt, welche zusätzlich an Benutzerpräferenzen und technische Randbedingungen angepasst werden müssen. Hierfür wird ein methodischer Ansatz benötigt, um dies mit vertretbarem Aufwand leisten zu können.

Ansatz. Zur Umsetzung der oben skizzierten Anforderungen wird ein umfassendes Framework vorgestellt. Die Konzeption des Frameworks basiert auf Methoden zur Entwicklung von Softwareproduktlinien. In einer Feature-orientierten Domänenanalyse werden zunächst die variablen und gemeinsamen Eigenschaften der benötigten Werkzeuge identifiziert. Ein wichtiger Variationspunkt, die Menge der verfügbaren Editieroperationen, wurde bereits diskutiert. Weitere Variationspunkte sind bspw. die Anzeigeform für Differenzen, die Fehlerbehandlung bei der Änderungspropagation oder die Integration mit anderen Werkzeugen einer MBSE-Umgebung. Zur technischen Umsetzung der Variabilität wird ein kompositionaler Ansatz verfolgt. Werkzeug-Eigenschaften (Features) werden auf wiederverwendbare Implementierungskomponenten abgebildet, welche flexibel kombiniert und zu fertigen Werkzeugen gebunden werden können. Grundlegende Betrachtungen zur Adaptierbarkeit von Differenzwerkzeugen wurden in [Ke12b] vorab veröffentlicht.

Evaluation. Die *Flexibilität* des Frameworks wird durch die Entwicklung diverser Werkzeuge für unterschiedlichste Anwendungskontexte und Problemstellungen unter Beweis gestellt. Das Anwendungsspektrum umfasst zum einen Differenz-, Patch- und Mischwerkzeuge für Modelle, einige davon wurden in referierten Werkzeugdemonstrationen auf internationalen Konferenzen [KKR14, Ke15b] und in wissenschaftlichen Fachzeitschriften [KKT14] präsentiert. Zum anderen wurden Werkzeugfunktionen zur Lösung konkreter

Probleme in anderen Forschungsprojekten realisiert, so z.B. für statistische Analysen zur Evolution von Modellen [GRK14, Ya13, Ya14, Ya15], zur semantischen Klassifikation von Modelländerungen [Bü15] oder im Kontext der delta-orientierten Entwicklung modellbasierter Softwareproduktlinien [Pi15, Vo15]. Die *Adaptierbarkeit* der entwickelten Komponenten wird durch die Unterstützung einer Vielzahl verschiedenartiger Standard- [KKR14, KKT14, Pi15] und domänenspezifischer [Bü15, Ke13a, Ke15b, Vo15] Modelltypen unterstrichen.

5 Resümee

Hauptproblem der modellbasierten Softwareentwicklung ist, die Entwicklungsprozesse durch hochwertige Werkzeuge zu unterstützen. Dies zeigt sich insbesondere im Kontext des Versions- und Variantenmangements. Klassische Verfahren für Texte sind auf Modelle nicht anwendbar. Verfügbare Werkzeuge für Modelle verfolgen zwar einen strukturellen Ansatz, beschreiben Änderungen an Modellen jedoch mittels primitiver Graphoperationen. Solch kleinteilige Beschreibungen sind für Benutzer unverständlich und bergen im Kontext der Änderungspropagation die Gefahr der Synthetisierung inkonsistenter Modelle. Zur Lösung dieser Probleme wird in der hier zusammengefassten Dissertation ein umfassender Ansatz vorgestellt. Existierende Konzepte und Techniken zur Berechnung und Propagation von Modelländerungen werden auf einen höheren Abstraktionsgrad basierend auf Editieroperationen angehoben. Die konzipierten Verfahren sind generisch und hochgradig anpassbar. Hierzu wird ein methodischer Ansatz zur Implementierung einer Familie grundlegender und für ein breites Spektrum unterschiedlicher Modelltypen und Szenarien einsetzbarer Werkzeugfunkionen vorgestellt. Der Ansatz ist durch die formale Spezifikation von Editieroperationen theoretisch fundiert, gleichzeitig wird der praktische Nutzen durch zahlreiche Experimente und Fallstudien belegt. Diverse Zusammenarbeiten mit anderen Wissenschaftlern und der Industrie zeigen das große Interesse an der Arbeit und belegen ferner die breite Anwendbarkeit der entwickelten Konzepte und Techniken. Die Arbeit ermöglicht die Modellversionierung auf einer Abstraktionsebene, welche für die Programmversionierung so bisher nicht erreicht wurde. Ein interessanter Ausgangspunkt für weitere Forschungsprojekte ist die Frage, wie weit die Ideen zur Programmversionierung eingesetzt werden können. Die Arbeit bildet ferner die Basis für ein kürzlich bewilligtes DFG-Projekt, in dem die hier vorgestellten Konzepte auf die Versionierung koevolvierender Modelle erweitert werden sollen.

Literaturverzeichnis

[Al09] Altmanninger, Kerstin; Brosch, Petra; Kappel, Gerti; Langer, Philip; Seidl, Martina; Wieland, Konrad; Wimmer, Manuel: Why model versioning research is needed!? An experience report. In: Intl. MoDSE-MCCM Workshop. 2009.

[Ar10] Arendt, Thorsten; Biermann, Enrico; Jurack, Stefan; Krause, Christian; Taentzer, Gabriele: Henshin: advanced concepts and tools for in-place EMF model transformations. In: Intl. Conf. on Model Driven Engineering Languages and Systems, S. 121–135. Springer, 2010.

[BCW12] Brambilla, Marco; Cabot, Jordi; Wimmer, Manuel: Model-driven software engineering in practice. Synthesis Lectures on Software Engineering, 1(1):1–182, 2012.

[BE09] Bendix, Lars; Emanuelsson, Pär: Collaborative work with software models–industrial experience and requirements. In: Intl. Conf. on Model-Based Systems Engineering. IEEE, S. 60–68, 2009.

[Bü15] Bürdek, Johannes; Kehrer, Timo; Lochau, Malte; Reuling, Dennis; Kelter, Udo; Schürr, Andy: Reasoning about product-line evolution using complex feature model differences. Automated Software Engineering, S. 1–47, 2015.

[CSFP04] Collins-Sussman, Ben; Fitzpatrick, Brian; Pilato, Michael: Version control with subversion. O'Reilly Media, Inc., 2004.

[CW98] Conradi, Reidar; Westfechtel, Bernhard: Version models for software configuration management. ACM Computing Surveys, 30(2):232–282, 1998.

[Eh06] Ehrig, Hartmut; Ehrig, Karsten; Prange, Ulrike; Taentzer, Gabriele: Fundamentals of Algebraic Graph Transformation. Springer, 2006.

[Em12] Emanuelsson, Pär: There is a strong need for diff/merge tools on models. Softwaretechnik-Trends, 32(4):30–31, 2012.

[Es05] Estublier, Jacky; Leblang, David; Hoek, André van der; Conradi, Reidar; Clemm, Geoffrey; Tichy, Walter; Wiborg-Weber, Darcy: Impact of software engineering research on the practice of software configuration management. ACM Transactions on Software Engineering and Methodology, 14(4):383–430, 2005.

[GRK14] Getir, Sinem; Rindt, Michaela; Kehrer, Timo: A Generic Framework for Analyzing Model Co-Evolution. In: Intl. Workshop on Models and Evolution. S. 12–21, 2014.

[HK10] Herrmannsörfer, Markus; Koegel, Maximilian: Towards a generic operation recorder for model evolution. In: Intl. Workshop on Model Comparison in Practice. S. 76–81, 2010.

[Ke12a] Kehrer, Timo; Kelter, Udo; Ohrndorf, Manuel; Sollbach, Tim: Understanding model evolution through semantically lifting model differences with SiLift. In: Intl. Conf. on Software Maintenance. IEEE CS, S. 638–641, 2012.

[Ke12b] Kehrer, Timo; Kelter, Udo; Pietsch, Pit; Schmidt, Maik: Adaptability of model comparison tools. In: Intl. Conf. on Automated Software Engineering. ACM, S. 306–309, 2012.

[Ke13a] Kehrer, Timo; Pietsch, Pit; Yazdi, Hamed Shariat; Kelter, Udo: Detection of High-Level Changes in Evolving Java Software. Softwaretechnik-Trends, 33(2), 2013.

[Ke13b] Kehrer, Timo; Rindt, Michaela; Pietsch, Pit; Kelter, Udo: Generating Edit Operations for Profiled UML Models. In: Intl. Workshop on Models and Evolution. S. 30–39, 2013.

[Ke15a] Kehrer, Timo: Calculation and Propagation of Model Changes Based on User-level Edit Operations. Dissertation, University of Siegen, 2015.

[Ke15b] Kehrer, Timo; Pietsch, Christopher; Kelter, Udo; Strüber, Daniel; Vaupel, Steffen: An Adaptable Tool Environment for High-level Differencing of Textual Models. In: Intl. Workshop on OCL and Textual Modeling. S. 62–72, 2015.

[Ke16] Kehrer, Timo; Taentzer, Gabriele; Rindt, Michaela; Kelter, Udo: Automatically Deriving the Specification of Model Editing Operations from Meta-Models. In: Intl. Conf. on Model Transformations. Jgg. 9765 in LNCS. Springer, S. 173–188, 2016.

[KKK13] Kelter, Udo; Kehrer, Timo; Koch, Dennis: Patchen von Modellen. In: Software Engineering 2013: Fachtagung des GI-Fachbereichs Softwaretechnik. Jgg. 213 in LNI. GI, S. 171–184, 2013.

[KKR14] Kehrer, Timo; Kelter, Udo; Reuling, Dennis: Workspace updates of visual models. In: Intl. Conf. on Automated Software Engineering. ACM, S. 827–830, 2014.

[KKT11] Kehrer, Timo; Kelter, Udo; Taentzer, Gabriele: A rule-based approach to the semantic lifting of model differences in the context of model versioning. In: Intl. Conf. on Automated Software Engineering. IEEE, S. 163–172, 2011.

[KKT13] Kehrer, Timo; Kelter, Udo; Taentzer, Gabriele: Consistency-preserving edit scripts in model versioning. In: Intl. Conf. on Automated Software Engineering. IEEE, S. 191–201, 2013.

[KKT14] Kehrer, Timo; Kelter, Udo; Taentzer, Gabriele: Propagation of Software Model Changes in the Context of Industrial Plant Automation. Automatisierungstechnik, 62(11):803–814, 2014.

[Kü08] Küster, Jochen M; Gerth, Christian; Förster, Alexander; Engels, Gregor: Detecting and resolving process model differences in the absence of a change log. In: Business Process Management, S. 244–260. Springer, 2008.

[Pi15] Pietsch, Christopher; Kehrer, Timo; Kelter, Udo; Reuling, Dennis; Ohrndorf, Manuel: SiPL - A Delta-based Modeling Framework For Software Product Line Engineering. In: Intl. Conf. on Automated Software Engineering. ACM, 2015.

[RKK14] Rindt, Michaela; Kehrer, Timo; Kelter, Udo: Automatic Generation of Consistency-Preserving Edit Operations for MDE Tools. In: Intl. Conf. on Model Driven Engineering Languages and Systems. 2014.

[SC13] Stephan, Matthew; Cordy, James R: A Survey of Model Comparison Approaches and Applications. In: Modelsward. S. 265–277, 2013.

[Vo15] Vogel-Heuser, Birgit; Folmer, Jens; Kowal, Matthias; Schaefer, Ina; Lity, Sascha; Fay, Alexander; Lamersdorf, Winfried; Kehrer, Timo; Tichy, Matthias; Beckert, Bernhard: Selected Challenges of Software Evolution for Automated Production Systems. In: Intl. Conf. on Industrial Informatics. IEEE, S. 314–321, 2015.

[Ya13] Yazdi, Hamed Shariat; Pietsch, Pit; Kehrer, Timo; Kelter, Udo: Statistical Analysis of Changes for Synthesizing Realistic Test Models. In: Software Engineering 2013: Fachtagung des GI-Fachbereichs Softwaretechnik. Jgg. 213 in LNI. GI, S. 225–238, 2013.

[Ya14] Yazdi, Hamed Shariat; Mirbolouki, Mahnaz; Pietsch, Pit; Kehrer, Timo; Kelter, Udo: Analysis and Prediction of Design Model Evolution Using Time Series. In: Advanced Information Systems Engineering Workshops. Jgg. 178 in LNBIP. Springer, S. 1–15, 2014.

[Ya15] Yazdi, Hamed Shariat; Pietsch, Pit; Kehrer, Timo; Kelter, Udo: Synthesizing realistic test models. Computer Science-Research and Development, 30(3-4):231–253, 2015.

Timo Kehrer wurde geboren am 3. Dezember 1981 in Reutlingen. Er machte sein Abitur am Wildermuth-Gymnasium Tübingen und studierte Medieninformatik und Angewandte Informatik in Stuttgart und Siegen. Sein Informatik-Diplom erlangte er Anfang des Jahres 2011. Anschließend arbeitete Kehrer als Wissenschaftlicher Mitarbeiter in der Fachgruppe Praktische Informatik der Naturwissenschaftlich-Technischen Fakultät der Universität Siegen. Seine Promotion schloss er 2015 mit der hier zusammengefassten Dissertation ab. Zum Oktober 2015 erhielt Kehrer ein Forschungsstipendium an der Politecnico di Milano und arbeitet dort seither als Postdoktorand. Zu seinen Arbeitsgebieten zählt u.a. die modellbasierte Softwareentwicklung mit einem Schwerpunkt auf der Unterstützung und Untersuchung der Evolution von Modellen.