Karlsruher Institut für Technologie

Rückkopplung von Softwaremodelländerungen in textuelle Spezifikationen

Diplomarbeit
von

Bugra Derre

An der Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Erstgutachter:	Prof. Walter F. Tichy
Betreuender Mitarbeiter:	Dipl.-Inform. Sven J. Körner

Bearbeitungszeit: 01. Dezember 2009 – 31. Mai 2010

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Karlsruhe, 31. 05. 2009

Danksagung

Zuerst möchte ich mich bei den wichtigsten Personen meines Lebens, meinen Eltern, bedanken. Sie ermöglichten mir das Studium in allerlei Hinsicht. Angefangen bei der moralischen Unterstützung, über die Aufheiterung an schlechten Tagen hin zur finanziellen Unterstützung im Studium. Ich möchte mich auch für die anstrengenden Tage mit mir entschuldigen und dankbar für euch sein. Danke, dass es euch gibt.

Außerdem möchte ich unserem Betreuer Sven J. Körner für die tolle Unterstützung, die immer wieder aufheiternden Geschichten in unseren wöchentlichen Treffen unseres Teams RECAA danken. Meinen Teamkollegen des Projekts RECAA's gilt ebenfalls besonderer Dank. Für jedes unangenehme Problem hatten sie immer ein offenes Ohr und viele interessante Lösungsansätze zu bieten.

Dr. Tom Gelhausen gilt besonderer Dank. Er hat mich auf das IPD am Lehrstuhl von Prof. Tichy aufmerksam gemacht und mich zum dortigen Werdegang animiert.

Dem Team von GrGen.NET gilt ebenfalls besondere Aufmerksamkeit für die lobenswerte Hilfestellung und ihre technische Unterstützung bei Problemen und Fragen bei Problemen mit Graphersetzungen.

Diese Arbeit möchte ich meinem Großvater widmen, dem ich leider nicht mehr erzählen kann, dass ich mein Studium abgeschloßen habe. Er brachte mich immer zum Lachen und verlor nie seinen Humor, auch nicht in schlechten Lebenslagen.

Danke.

Kurzfassung

Die Anforderungsanalyse ist ein fehleranfälliger Prozess. Wünsche und Anforderungen der Kunden an das Produkt werden häufig fehlinterpretiert und führen zu einer inkorrekten oder unvollständig dokumentierten Spezifikation. Der Softwareanalyst hat die Aufgabe die textuell definierten Anforderungen der Spezifikation in ein Softwaremodell zu überführen. Die Fehlinterpretationen der Spezifikation werden durch den Softwareanalysten und Domänenexperten nachträglich im Softwaremodell geändert.

Dieser Prozess führt zu Modelländerungen die die ursprüngliche Spezifikation nicht enthält. Der Kunde kann die Änderungen am Modell nicht nachvollziehen und diese weder bestätigen noch korrigieren, da die Änderungen sich nicht in der Spezifikation widerspiegeln. Somit ist nicht gesichert, dass Modelländerungen ohne Auswirkungen auf den eigentlichen Kundenwunsch sind. Eine Rückkopplungskomponente soll die Modelländerungen in die vorhandene Spezifikation einfügen, um Änderungen für den Kunden zu dokumentieren.

Bei existierenden Ansätzen wird das Softwaremodell dabei nicht mit der ursprünglichen Spezifikation verknüpft, sondern natürlichsprachlicher Text aus dem Modell erzeugt. Änderungen sind nicht verfolgbar und erlauben dem Kunden keinen direkten Vergleich zur ursprünglichen Version der Spezifikation. Die neu erzeugte Spezifikation beschreibt die Modellelemente, nicht aber deren Prozesse.

Das im Rahmen dieser Arbeit erstellte Werkzeug ist Teil des Gesamtprojekts Requirements Engineering Complete Automation Approach (RECAA) [KDGL], das sich mit der Automatisierung des Anforderungsanalyseprozesses beschäftigt.

Unser vorgestellter Ansatz bietet eine Modellrückkopplungskomponente die Modelländerungen in die Spezifikation einpflegt und den iterativen Prozess der Anforderungsanalyse unterstützt. Änderungen gegenüber der ursprünglichen Spezifikation werden dem Kunden nachvollziehbar dargestellt. Unser Werkzeug verknüpft hierbei das Softwaremodell mit der textuellen Spezifikation um Änderungen detailliert darzustellen und die einzelnen Entitäten zu synchronisieren.

Inhaltsverzeichnis

1. Einleitung

Die Anforderungsanalyse ist eine umfangreiche und komplexe Disziplin der Softwaretechnik [SS97]. Geschulte Softwareanalysten und Domänenexperten sammeln Anforderungen der Kunden in einem hauptsächlich manuellen Prozess. Die gesammelten Anforderungen werden analysiert und in Spezifikationen niedergeschrieben. Die anschließende Validierung prüft, ob alle Anforderungen stimmig sind und sich insbesondere nicht wiedersprechen. Aufgabe des Anforderungsanalysten ist es, diesen Prozess zu steuern und verwalten. Weitere Aufgaben des Analysten sind die Verbesserung und Dokumentation der verfeinerten Anforderungen. Um die Anforderungen der Spezifikation visuell darzustellen und sie (formal) zu verifizieren, wird sie in ein Modell überführt. Modelle verschaffen dem Analysten einen Überblick über das Systemverhalten und seine Struktur. Diese Modelle können auch für die automatische Codegenerierung nach dem Prinzip der Modellgetriebenen Architektur (MDA[1]) genutzt werden. Eine frühere Arbeit von Gelhausen und Tichy bietet mit SAL$_e$ MX eine Lösung an, um Modelle aus natürlicher Sprache aus Spezifikationen automatisch zu extrahieren [GT07]. Wir wollen unsere Arbeit mit SAL$_e$ MX evaluieren.

Die Gründe für die Komplexität der Anforderungserhebung sind vor allem der Unterschied zwischen der textuell definierten Spezifikation und dem zu erstellenden System. Anforderungen werden in der Problemdomäne festgehalten; sie beschreiben die Probleme und Anforderungen die das System erfüllen muss. Die Lösungsdomäne hingegen beschreibt die technischen Lösungsmöglichkeiten der gestellten Anforderungen. Die hierbei stattfindende Transformation ist verlustbehaftet.

Eine weitere Schwierigkeit für den technischen Lösungsraum ergibt sich aus der informalen Sprache der Spezifikation. Die natürlichsprachlich erstellten Anforderungen sind kaum an Beschränkungen oder Bedingungen gebunden. Der Lösungsraum ist dabei geprägt von mathematisch formalen Modellen, die sich nicht direkt auf informelle Anforderungen abbilden lassen. Weitere im Anforderungsanalyseprozess anzutreffende Komplexitäten und Schwierigkeiten werden in [CA07, Rup07, BW84] beschrieben.

[1] Model Driven Architecture ist eine Strategie der Object Management Group

1.1. Der Prozessablauf

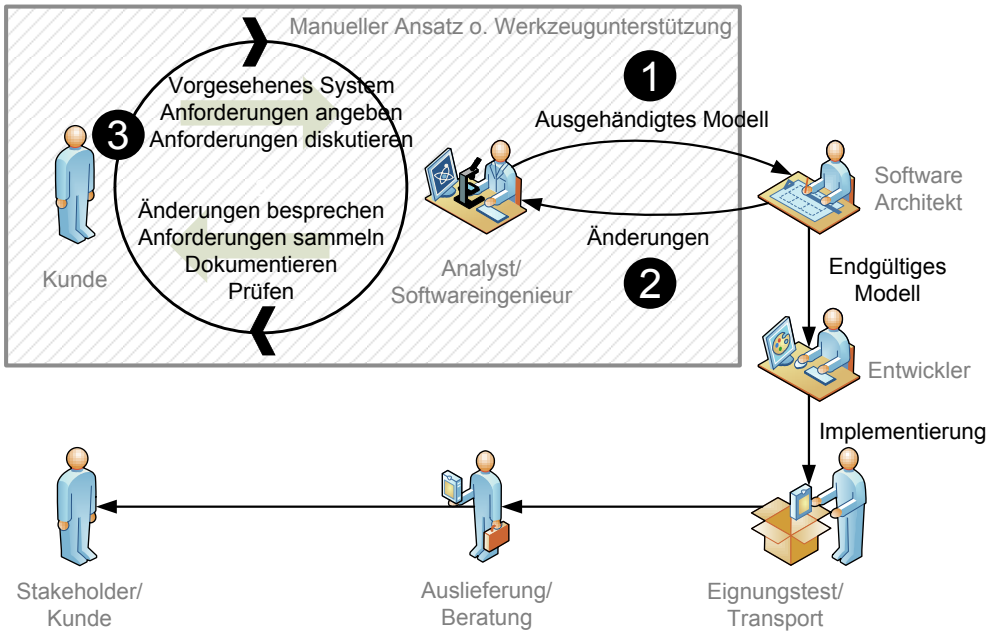

Abbildung 1: Der Anforderungsanalyseprozess

Der typische Prozessablauf der Anforderungsanalyse beginnt beim Kunden. Seine Vorstellungen und Wünsche, werden von einem geschulten Analysten gesammelt und in textuellen Anforderungen festgehalten. Die Liste der Anforderungen wird dann in ein Domänenmodell überführt und dem Softwarearchitekten vorgelegt (1), in Abbildung 1.

Der Architekt ändert das Modell (2) und nimmt Entwurfsänderungen vor, während er ein Softwaremodell aus der Spezifikation modelliert. Eine Rückkopplungskomponente soll die Änderungen automatisch in die Spezifikation einfügen. Die gemachten Änderungen beeinflussen die ursprüngliche Anforderungsspezifikation. Die Kunden können darüber nicht informiert werden. Die gemachten Änderungen des Softwaremodells werden nicht in die natürlichsprachliche Spezifikation rückgekoppelt. Die Spezifikation ist aber das einzige Dokument, das für den Kunden verständlich und relevant für den Auftrag ist. Zusätzlich dient die Spezifikation als vertraglich bindendes Dokument zwischen beiden Parteien: dem Kunden und dem Softwareentwickler. Der Architekt selbst ist kein Domänenexperte; von ihm gemachte Modelländerungen können den Anforderungen des Kunden widersprechen oder sie blockieren. Umso wichtiger ist es, diese Änderungen an den Kunden gut dokumentiert zurück zu liefern.

Nach den Modelländerungen müssen die Auswirkungen auf die Spezifikation untersucht und die Kunden informiert werden (3). Die betroffenen Anforderungen werden

zusammen mit dem Analysten diskutiert und als geänderte Anforderung in die Spezifikation aufgenommen. Manche Anpassungen werden vom Kunden negiert, so dass eine erneute Anpassung des Softwaremodells notwendig ist.

Unser vorgestellter Ansatz erweitert SAL$_e$ MX um eine Modellrückkopplungskomponente. Der Architekt führt Änderungen am Modell durch, die automatisch in die ursprüngliche Version der Spezifikation eingefügt werden. Die geränderte Spezifikation kann vom Kunden zusammen mit dem Analysten auf Korrektheit untersucht werden. Änderungen gegenüber der ursprünglichen Version der Spezifikation werden durch Textvergleich angezeigt.

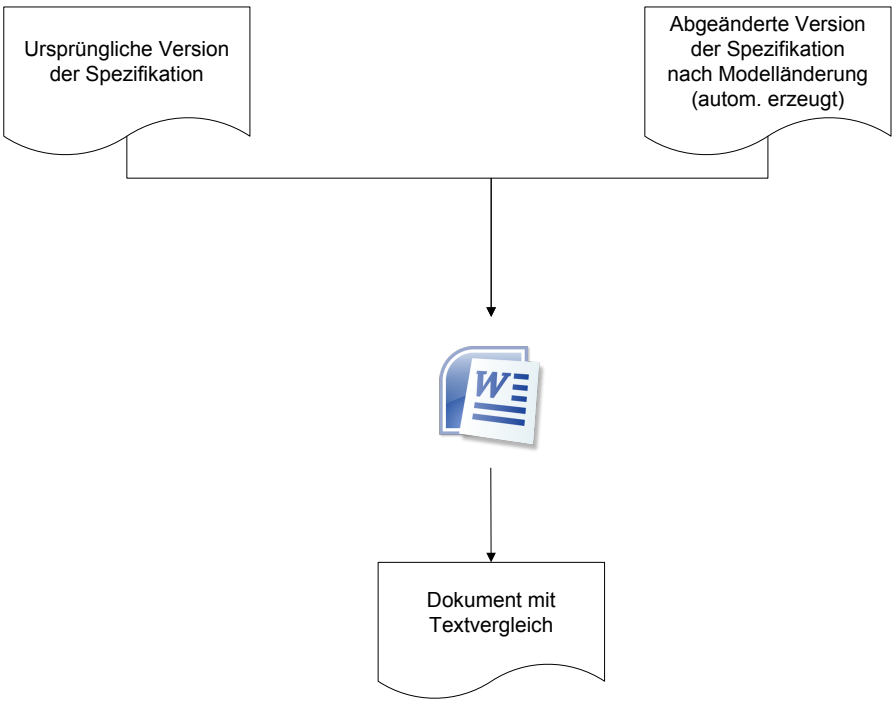

Abbildung 2: Textvergleich mit einem Textbearbeitungsprogramm

1.2. Systemüberblick

Um den Anforderungen einer automatischen Rückkopplung für Spezifikationen nachzukommen, ist eine Modellrückkopplungskomponente unverzichtbar. Die Rückkopplungskomponente wird am Beispiel des Modellextraktors SAL$_e$ MX verifiziert [GT07].

Im ersten Schritt werden die Anforderungen durch den Analysten ermittelt und in einer Textdatei als Anforderungsdokument niedergeschrieben. Die Textdatei ist für die Modellextraktion nicht ausreichend. Um das Softwaremodell aus dem Dokument zu extrahieren ist eine semantische Annotierung notwendig, siehe [Gel10]. Die semantische Annotierung geschieht durch die Auszeichnung thematischer Rollen an die einzelnen Konstituenten des Dokuments (Punkt (1) in Abbildung 3).

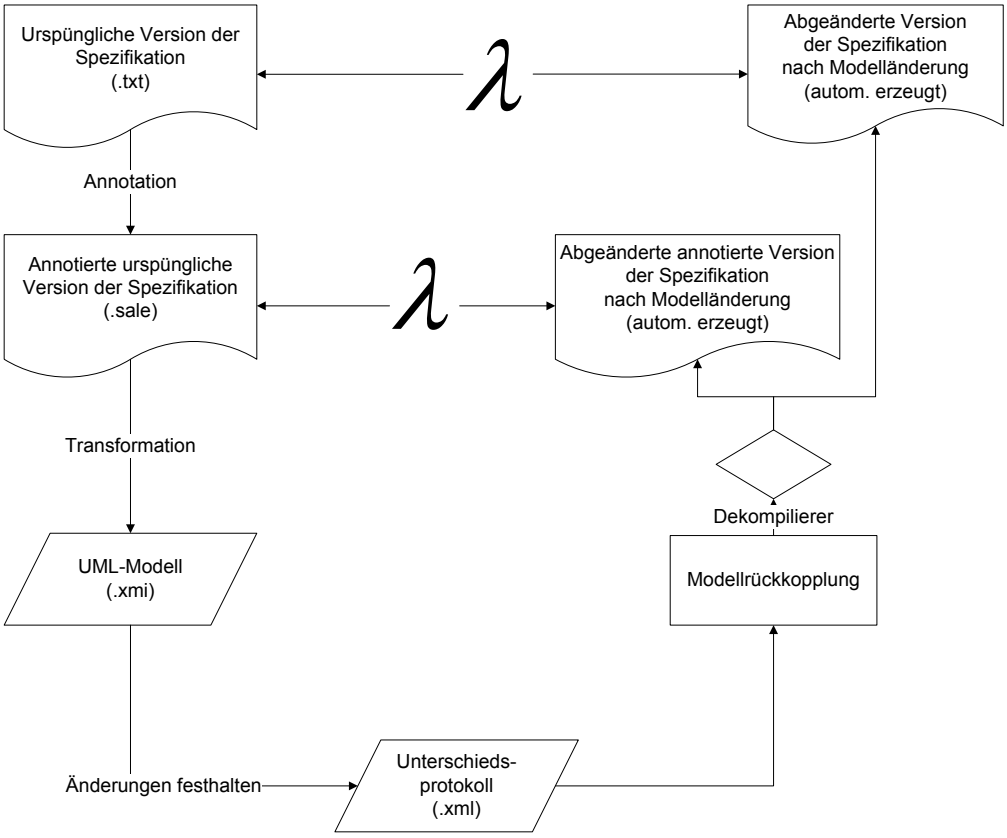

Abbildung 3: Übersicht über den Prozessverlauf mit REFS

Die Modellextraktion (2) liefert ein UML-Softwaremodell, das in dem XMI-Austauschformat gespeichert wird. Diese XMI-Datei [Gro07b] wird im Anschluss in ein CASE-Werkzeug importiert, um das Modell manuell zu überarbeiten oder zu verfeinern.

Das überarbeitete bzw. verfeinerte Modell enthält Änderungen, die in einem Unterschiedsprotokoll festgehalten werden (3). Das Protokoll beschreibt die Änderungen des Modells durch den Analysten.

Das Unterschiedsprotokoll (4) dient der Modellrückkopplungskomponente als Eingabedatei. Durch sie ist die Rückkopplungskomponente in der Lage, die individuellen Änderungen zu erkennen und in die ursprüngliche Spezifikation einzufügen. Die Änderungen unterscheiden sich folgendermaßen in ihrem Typ;

- das Hinzufügen neuer Modellelemente führt zu einer Texterzeugung in der Spezifikation,
- das Löschen vorhandener Modellelemente führt zu einer Textauslassung der betroffenen Anforderungskomponente in der textuellen Spezifikation,
- Aktualisierungen am Modell beschreiben Wertänderungen, wie z. B. Namensänderungen.

Nachdem die Modelländerungen in die Spezifikation eingefügt wurden, gibt der SAL_{E}-Dekompilierer ein neues Spezifikationsdokument aus, dass alle Modelländerungen in der Spezifikation enthält, siehe Punkt (5). Mit einem beliebigen Textbearbeitungsprogramm[2] können Unterschiede zwischen der originalen und der geänderten Spezifikation analysiert und überprüft werden. Zusammen mit den Kunden kann der Analyst die Änderungen aufklären und besprechen.

1.3. Zielsetzung der Arbeit

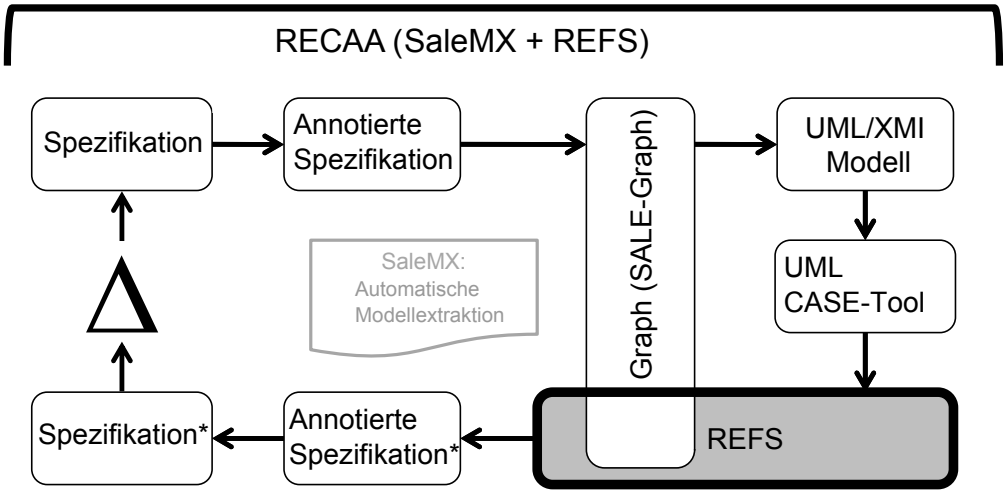

Abbildung 4: Die Integration von REFS in SAL_e MX

Die automatisierte Modellrückkopplungskomponente soll Analysten und Kunden einen Überblick über die geänderten Anforderungen der ursprünglichen Spezifikation verschaffen um Änderungen besser nachzuvollziehen. REFS wird in den bestehenden Modellextraktor SAL_e MX integriert um das Ergebnis der Arbeit zu evaluieren. Die Modellrückkopplung ist aktuell beschränkt auf UML-Klassendiagramme.

Die Änderungen werden von REFS evaluiert und in die ursprüngliche annotierte Version der Spezifikation rückgekoppelt, siehe Abbildung 4. Anschließend wird eine abgeglichene Version der Spezifikation ausgegeben, die alle Modelländerungen beinhaltet. Ein einfacher Textvergleich, wie in Microsoft Word, versetzt den Kunden in die Lage, Änderungen an der Spezifikation schnell und transparent einzusehen.

Weiterhin wird mit REFS die Idee von RECAA [KDGL], einem Werkzeug zur Unterstützung des gesamten Anforderungsprozesses, unterstützt.

[2]Das Textbearbeitungsprogramm muss einen Ähnlichkeitsvergleich zweier Dokumente anbieten. Microsoft Office Word oder Open Office Writer bieten einen solchen Ähnlichkeitsvergleich an.

1.4. Gliederung der Arbeit

Aus der Notwendigkeit von Iterationen im Anforderungsanalyseprozess folgen spezielle Anforderungen an ein Modellrückkopplungssystem. Welche Anforderungen aus dieser Problemstellung resultieren wird in Abschnitt 2 am Beispiel für SAL$_e$ MX und REFS detailliert beschrieben. Existierende Lösungsansätze vergleichbarer Werkzeuge werden anschließend vorgestellt.

Den Entwurf und das Konzept für die technische Lösung der gestellten Anforderungen der Arbeit stellen wir in Abschnitt 3 vor. Abschnitt 4 vertieft das Konzept der Modell-Rückkopplung mit Auszügen des Quellcodes.

Die Evaluation, siehe Abschnitt 5, demonstriert das Ergebnis der vorgestellten Modellrückkopplungskomponente. An Fallbeispielen zeigen wir, wie nützlich ein Rückkopplungssystem für den Analysten bzw. Kunden ist.

2. Analyse

Dieses Kapitel gibt gewonnenene Kenntnisse über die Rückkopplung von Modelländerungen in die zugehörige Spezifikation wieder.

In Unterabschnitt 2.1 wird gezeigt, wie die Modellrückkopplung erfolgen kann. Hierzu werden grundlegende Konzepte vorgestellt, die unabhängig von einer konkreten Implementierung sind.

Die automatisierte Modellierung natürlichsprachlicher Anforderungen in der Anforderungsanalyse ist ein Forschungsbereich, der bis in die frühen 1980er reicht [Che83]. Allerdings gibt es kaum Konzepte und Methoden für unseren vorgestellten Ansatz der automatisierten Modellrückkopplung. In Unterabschnitt 2.2 werden die seither gewonnenen Erkenntnisse zusammengefasst und die Ursprünge der Modellrückkopplung aufgezeigt. Des Weiteren werden verwandte Arbeiten wie LIDA und TESSI vorgestellt und mit unserem in SAL_e MX integrierten Ansatz verglichen.

2.1. Anforderungen

SAL$_e$ MX überführt Wörter und Satzelemente, z. B. Satzzeichen, in ein SAL$_E$-Graphmodell [KDGL]. Im SAL$_E$-Graphmodell werden die so überführten Wörter und Satzelemente als Konstituenten bezeichnet. Alle SAL$_E$-Graphknoten sind vom Typ *Constituent*. Sie bildet den Basistyp für die SAL$_E$-Graphknoten, die sich in ihren erweiterten Typen unterscheiden. Konstituenten haben zusätzliche Eigenschaften, wie z. B. Positionsanzeiger. Durch sie wird die Reihenfolge der einzelnen Konstituenten in einer Phrase bestimmt.

SAL$_e$ MX wendet Graphtransformationen an, die aus einem SAL$_E$-Graphen, bzw. einem Teil davon, ein UML-Modellgraphen erzeugen. Die Transformation wird im Folgenden als mathematische Abbildung von der SAL$_E$-Quellmenge in die UML-Zielmenge beschrieben. SAL$_e$ MX bildet eine Konstituente der Quellmenge (SAL$_E$) auf mindestens ein Modellelement der Bildmenge (UML) ab.

Um Änderungen einfügen zu können, müssen die modellierten Anforderungen rückverfolgbar sein. Das bisherige Graphmodell erzeugt zwei voneinander unabhängige Graphen. Die Unabhängigkeit der Graphen gibt keinen Aufschluß über die Abbildungsvorschrift eines Konstituenten der Quellmenge in ein Modellelement der Bildmenge. Um festzustellen welche Konstituente von einer Änderung des Modellelements betroffen ist benötigen wir eine Verknüpfung der zwei Graphen miteinander. Eine Kante verbindet die Konstituente mit ihrem abgebildeten Modellelement und macht so die modellierte Anforderung rückverfolgbar.

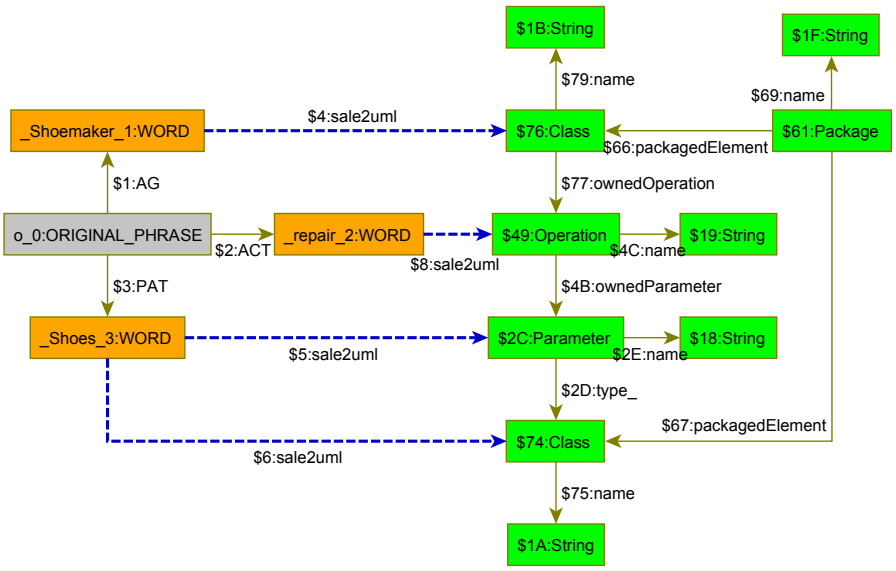

Abbildung 5: Grafische Darstellung des SAL$_E$-Graphen mit rückverfolgbaren Abbildungskanten zum UML-Graphen.

Abbildung 5 illustriert die Rückverfolgbarkeit der Abbildungskanten. Die ursprüng-

liche SAL$_E$-Phrase besteht aus den Konstituenten *Shoemaker repair Shoes*. Zuerst wird durch den SAL$_E$-Übersetzer die Phrase in seine einzelnen Konstituenten aufgeteilt und als Graph geladen.

Anschließend wird aus dem SAL$_E$-Graphen das UML-Graphmodell mit den rückverfolgbaren Kantenverbindungen erzeugt. Die Konstituente *Shoemaker* wird zu einer Klasse abgebildet; die Rückverfolgbarkeit auf seine Konstituente ist durch eine Verbindung gewährleistet. Die Konstituente *repair* wird auf eine Operation abgebildet, dessen Besitzer die Klasse *Shoemaker* ist. Auch hier haben wir eine Abbildungskante, die die erzeugte Operation rückverfolgbar macht. Die nächste Konstituente *Shoes* erhält zwei Abbildungskanten. Zum einen wird er als Klasse abgebildet, zum anderen wird ein Parameter in der Operation *repair* erzeugt.

Um mehr über das Prinzip von SAL$_E$ zu erfahren, sei auf [KDGL] verwiesen.

2.2. Existierende Lösungsansätze

Ein wichtiges Thema in der Softwaretechnik ist die Wartung von Software. Obwohl Softwaresysteme scheinbar nicht „altern“ unterliegen sie ständigen Anpassungen. Dabei unterteilen Lientz und Swanson [LS81] die Gründe für Anpassungen in verschiedene Wartungstypen;

- korrektive Wartung; die Beseitigung von Fehlern
- perfektionierende Wartung; Verbesserung von Attributen wie etwa der Performanz oder der Wartbarkeit. Darunter fällt insbesondere die Bereinigung des Entwurfs oder der Implementierung durch Reengineering (Software), Refactoring usw.
- adaptive Wartung; Anpassung der Software an veränderte technische Bedingungen der Umgebung

Wird ein Softwaresystem gewartet, so erosiert das System. Modellierte Anforderungen werden geändert aber nicht mehr in der Spezifikation reflektiert.

Briand et. al beschreiben in [BLO03] die Art der Änderungen die an einem Modell auftreten können. Eine Analyse von UML-Modelldiagrammen zeigt drei mögliche Änderungen die ein Modell erfahren kann. Im Modell werden

1. neue Elemente hinzugefügt (`Creation`),
2. Werte vorhandener Elemente geändert (`Update`) oder
3. vorhandene Elemente gelöscht (`Deletion`).

In [RCW07, Pie78] wird die Problematik der Modellerosion aufgegriffen. Sie wenden die Rückverfolgbarkeit modellierter Anforderungen an, um sie über den gesamten Produktlebenszyklus verfolgen zu können. Tritt eine Änderung an der modellierten Anforderung auf, so können die ursprünglich definierten Anforderungen ebenfalls geändert werden.

Die Konsistenz zwischen mehreren Diagrammen wird, von der in dieser Arbeit besprochenen Modellrückkopplungskomponente, nicht garantiert. Wird ein Modellelement in einem Diagramm d_1 geändert, so wirkt sich die Änderung nicht auf das Modellelement in einem anderen Diagramm d_2, wobei $d_1 \neq d_2$ aus. Eine Klasse, die in einem Klassendiagramm und als Objektinstanz in einem anderen Sequenzdiagramm verwendet wird, wird nicht unbedingt konsistent gehalten. Wird die Klasse im Klassendiagramm gelöscht, kann das Objekt im Sequenzdiagramm immer noch vorhanden sein. Für die Konsistenz zwischen den Modelldiagrammen gibt es bereits Verfahren und Werkzeuge, wie den ViewIntegra-Ansatz [Egy01].

Existierende Lösungen für ein Gesamtsystem zur automatischen Modellierung von Anforderungen und einer integrierten Rückkopplungskomponente für geänderte Modellelemente gibt es nicht. Die Literaturrecherche ergab zwei Werkzeuge mit denen wir unser System vergleichen konnten. Im Folgenden untersuchen wir TESSI [Kro00, KGR06] und LIDA [OLR01], die zwar den geforderten Iteratationsprozeß unterstützen, aber einen anderen Ansatz bei der Modellrückkopplung in die Spezifikation verfolgen.

2.2.1. TESSI - Textual Assistent

Kroha stellt mit dem „TExtual aSSIstent“ (TESSI) eine Methode und ein Werkzeug zur Unterstützung des Domänenexperten vor [Kro00, KGR06]. TESSI unterstützt die Erfassung, das Verfeinern und Modellieren von Anforderungen sowie die automatische Texterzeugung zur Beschreibung des erzeugten Modells. Die erzeugte Beschreibung kann vom Analysten herangezogen werden, um sie mit der ursprünglichen Spezifikation zu vergleichen und auf Änderungen zu untersuchen. Das Prinzip von TESSI basiert auf Verarbeitung natürlicher Sprache mit einem integriertem UML-Entscheider.

TESSI verlangt eine Anforderungsspezifikation vom Analysten als Eingabedokument. Im ersten Prozessschritt unterstützt TESSI den Analysten bei der halbautomatischen Modellgenerierung. Anschließend fordert es den Analysten durch komplementäre Fragen zum analysierten Textteil auf, das System genauer zu beschreiben um so ein detailliertes Modell erzeugen zu können. Unvollständigkeiten der Spezifikation werden im Modell vervollständigt und Fehler minimiert.

Iterationen in der Anforderungsanalyse werden durch eine Texterzeugungskomponente unterstützt. Eine automatische erzeugte Textbeschreibung des fertigen Modells schreibt die modellierten Anforderungen in natürlichsprachlichen Text, um sie dann auf Korrektheit zu überprüfen. Die Texterzeugung wird nach einem Fließbandprinzip mit drei texterzeugenden Komponenten realisiert [RD00].

Der „Dokumentplaner“ analysiert die Semantik des Modells mit einer Ontologie. In TESSI ist die Ontologiedatenbank das Modell selbst, d. h. es wird keine aussenstehende Ontologie abgefragt. Für die Ausgabe des Dokumentplaners nutzt TESSI ein hybrides System. Es werden domänenspezifische Schablonen 1) für einzelne Textpassagen zur Texterzeugung genutzt und 2) miteinander in Beziehung stehende *atomare* Textpassagen in Hierarchieordnungen ausgegeben. Der „Mikroplaner“ legt den Inhalt einzelner Sätze mit der ermittelten Semantik fest. Die „Oberflächenrealisierung“ für die Sätze formatiert die Ausgabe benutzerfreundlich um. Wortendungen werden angehängt und ausgegebene

Wörter werden in die richtige Reihenfolge gebracht. Die so erzeugte textuelle Beschreibung hilft dem Analysten bestimmte Anforderungen zu überprüfen und sie dem Kunden zu zeigen. In gemeinsamer Absprache können die gemachten Änderungen rückgängig gemacht oder in die Anforderungsspezifikation übernommen werden.

Anders als in unserem Ansatz wird die ursprüngliche Spezifikation nicht mit dem Modell verknüpft. Ein direkter Vergleich der ursprünglichen Version der Spezifikation und der abgeglichenen Version ist nicht automatsiert möglich. Die erzeugte Modellbeschreibung muss händisch verglichen werden.

2.2.2. LIDA - Linguistic Assistant for Domain Analysis

Mit dem „Linguistic assistant for Domain Analysis“ (LIDA) stellt Overmyer ein Werkzeug zur Unterstützung des Analysten vor [OLR01]. LIDA bereitet eine domänenunabhängige linguistische Textanalyse vor. Sogenannte Part-Of-Speech Tagger bestimmen in der Spezifikation vorkommende Nomen. Der Analyst entscheidet, ob die ausgewählten Nomen in Klassen oder als Klassenattribute modelliert werden sollen.

Im nächsten Prozessschritt werden Adjektive auf Zugehörigkeit von Klassen untersucht. Danach wird der Text auf Zugehörigkeit von Operationen, Assoziation und Generalisierungsbeziehungen hin untersucht. Neben einem Modell für ein Klassendiagramm wird ein Aktivitätsdiagramm erstellt, um das Laufzeitverhalten des Systems zu beschreiben.

Das erstellte Modell ist allerdings nicht portabel. Mit einem eigenen Austauschformat bietet LIDA nur eine beschränkte Weitergabe und Verarbeitung mit anderen CASE-Werkzeugen an.

Die für die Iteration zuständige Komponente ist ähnlich zu TESSI. Ein Texterzeuger erstellt eine Beschreibung des generierten Modells. Der Text wird von „Model Explainer“ (MODEX) [LRR97] generiert, wirkt aber statisch beim Lesen. Eine Verknüpfung zwischen der Spezifikation und dem Modell gibt es auch hier nicht. Der Analyst muss die erzeugte Modellbeschreibung mit der ursprünglichen Spezifikation manuell vergleichen.

2.2.3. Weitere Werkzeuge zur Modellextraktion

Weitere Systeme zur automatischen Modellierung textueller Anforderungen ohne eine Modellrückkopplungskomponente beschreiben wir im Folgenden nur kurz. „Static UML Model Generator from Analysis of Requirements“ (SUGAR) extrahiert Modelle aus natürlich-sprachlichen Spezifikationstexten. SUGAR [DS08] erwartet die Spezifikationen in einer bestimmten, vordefinierten grammatikalischen *Subjekt-Verb-Objekt* Struktur. Ist die grammatikalische Struktur den Erwartungen von SUGAR unzutreffend, fordert eine Dialogeingabe den Benutzer zur grammatikalischen Umformung des Satzes auf. Liegen alle Sätze in der korrekten grammatikalischen Form vor, extrahiert SUGAR das Modell daraus. Das Modell wird in einem SUGAR-eigenen Format gespeichert und kann nicht zum Austausch mit anderen Werkzeugen genutzt werden.

„UML Model Generator from Analysis of Requirements“ (UMGAR) [DB09] ist eine Weiterentwicklung von SUGAR. Zum einen werden die UML Modelle in einem XMI-

Dokument gespeichert, zum anderen bietet UMGAR Funktionen wie „Keyword in Context“ (KWIC). Durch die Auswahl eines Modellelements wird der dazugehörige Kontext in der Spezifikation angezeigt. Ansätze zur Rückverfolgbarkeit der modellierten Anforderungen sind vorhanden. In beiden Systemen, SUGAR und UMGAR, wird aber keine Iteration im Anforderungsanalyseprozess unterstützt. Es gibt keinen Texterzeuger zur automatischen Beschreibung des Modells, im Gegensatz zu TESSI oder LIDA.

2.2.4. Texterzeuger

Um im Modell neu hinzugefügte UML-Modellelemente in die bestehende Spezifikation zu integrieren sind Texterzeuger zur Beschreibung von Modellen oder Teilen des Modells notwendig. Sie erzeugen natürlichsprachliche Dokumentationstexte zur Beschreibung des Modells oder eines Modellteils. In unserer Arbeit verwenden wir einen eigenen, einfach gehaltenen Texterzeuger, da ein aufwändiger Texterzeuger den Rahmen der Diplomarbeit sprengen würde. Es gibt einen eigenen Forschungsbereich und viele Werkzeuge die bereits gute Arbeit in der Modellbeschreibung leisten. Wir wollen einige der Werkzeuge vorstellen, die man bei einer erweiterten Version von REFS benutzen kann um bessere Resultate bei der Texterzeugung zu erzielen.

Meziane veröffentlichte mit GeNLangUML [MAA08] einen Texterzeuger zur Beschreibung von Klassendiagrammen. Die natürlichsprachige Spezifikation wird ausschließlich aus dem Klassendiagramm des UML-Modells erzeugt. Um Mehrdeutigkeiten bei der semantischen Bedeutung von benutzten Wörtern in Klassendiagrammen auszuschließen bedient sich GeNLangUML der linguistischen Ontologiedatenbank WordNet [Fel98].

Meziane stützt die Erzeugung der natürlichsprachlichen Spezfikation auf einer umfangreichen Studie. Er und sein Team untersuchten 45 Klassendiagramme aus unterschiedlichen Fachbüchern und Lieraturquellen um Annahmen über typische Namenskonventionen von Anwendern in Klassendiagrammen zu treffen. Er gibt zu bedenken, dass die gemachten Annahmen unter Umständen nicht auf Namenskonventionen der Industrie übertragbar seien. Ein interessantes Beispiel sind Abkürzungen die in der Industrie häufig genutzt werden um bekannte Verhältnisse schnell darzustellen.

In der Evaluation von GeNLangUML nennt Meziane Schwächen seines Systems. Die einseitige Nutzung von strukturellen Klassendiagrammen kann zu falschen oder fehlenden Informationen in der erzeugten Sepzifikation führen. Die zusätzliche Ausnutzung von Verhaltensdiagrammen führt zu einer informativeren Spezifikation, die Aussagen über zeitliche Abhängigkeiten der in dem Klassendiagramm vorkommenden Strukturelemente treffen kann. Beispielsweise verlangt die Spezifikation für ein Kuchenrezept zuerst den Teig fertig zu rühren und ihn danach im Backofen für 45 Minuten zu backen. Die zeitliche Reihenfolge wird nur durch die zusätzliche Ausnutzung eines Verhaltensdiagramms beachtet. Nur mit einem Klassendiagramm würde diese Information verloren gehen.

Eine weitere Schwäche stellt die von Meziane automatisiert erzeugte Spezifikation selbst dar. Sie wirkt statisch und objektorientiert auf den unerfahrenen Leser. Die erzeugten Sätze sind in einem akademischen Umfeld und für erfahrene Modellierer lesbar. Kunden dürften hier Verständnisprobleme haben.

MODEX [LRR97] ist ein von Lavoie geleitetes Projekt für die Erzeugung natürlicher

Sprache aus objektorientierten Modellen. Die Modelle werden durch die ODL beschrieben, einer Spezifikation zur Beschreibung von Objekttypen in der ODMG [CBB+00]. MODEX ist ein allgemeines Spracherzeugungssystem und somit nicht an die Gegebenheiten einer spezifischen Domäne gebunden.

Bei der Benutzung kann der Anwender Modelldokumentation hinzufügen, die bei der späteren Textgenerierung berücksichtigt und ausgegeben werden. Solche Dokumentationen ergänzen im Modell fehlende Angaben und sollen das Verständnis für den späteren Leser erleichtern. Ob die Dokumentationen sinnvoll beschrieben und auch vollständig im gesamten Modell weitergeführt werden, hängt stark vom Analysten ab. Zu den anwendbaren UML-Diagrammtypen werden keine Angaben gemacht, dem Anschein nach (S.255 r.u.) handelt es sich ähnlich wie bei GeNLangUML ausschließlich um Klassendiagramme.

2.3. Zusammenfassung

Wie in den bisherigen Kapiteln gezeigt wurde, wird der Prozess der Iteration im teilweise automatisierten Analyseprozess nur von wenigen Werkzeugen unterstützt. Außerdem bieten die existenten Lösungsansätze keine Verbindung zwischen der textuellen Spezifikation und den daraus modellierten Anforderungen. Unser Ansatz nutzt die Präsentation der Spezifikation als Graph um die Modellelemente mit den Anforderungen zu verbinden und die Rückverfolgbarkeit zu garantieren. Eine Verknüpfung zwischen beiden Entitäten entsteht, und die ursprüngliche Struktur der Spezifikation wird bei einer Abgleichung beibehalten.

3. Entwurf

In diesem Kapitel erfolgt die ausführliche Beschreibung des vorgestellten Lösungsansatzes für die Modellrückkopplungskomponente.

Unterabschnitt 3.1 beschreibt den Prozessablauf, angefangen bei der Spezifikation über das erzeugte Modell und der abschließenden Rückkopplung von Modelländerungen in die Spezifikation. Außerdem wird die Verknüpfung zwischen der Spezifikation und dem daraus erzeugten Modell behandelt.

In Unterabschnitt 3.2 beschreiben wir, welche Merkmale das System haben wird und welche Implementierungsentscheidungen wir treffen.

In einer abschließenden Zusammenfassung in Unterabschnitt 3.3, gehen wir auf die wichtigsten Merkmale des vorgestellten Konzepts ein und fassen diese zusammen.

3.1. Grundlagen

Das vorgestellte System besteht aus mehreren aneinandergereihten Prozessen die durchlaufen werden. Abbildung 6 zeigt eine Übersicht der Prozessschritte.

Zu Anfang muss der Benutzer die vorliegende Domänenbeschreibung (normaler Text) annotieren. SAL$_E$-Quelldateien sind einfache Textdateien, so dass dieser Schritt mit jedem beliebigen Texteditor durchgeführt werden kann. Ein Editor mit Syntaxhervorhebung und einer Funktion zur automatischen Vervollständigung erleichtert die Arbeit; entsprechende Plug-Ins für die Entwicklungsumgebung „Eclipse“ sind auf der RECAA-Seite abrufbar [KDGL]. Der Annotationsschritt [Gel10] ist in Abbildung 6 mit (1) markiert.

Der nächste Schritt ist die Übersetzung des semantisch annotierten Textes mit dem SAL$_E$-Übersetzer (2). Der SAL$_E$-Übersetzer ist ein einfacher, ANTLR-basierter [Par09] Übersetzer, der eine Textdatei mit der Endung „grs“ erzeugt (Punkt (3)). Die Endung „grs“ steht für „Graph Rewrite Script“ und deutet darauf hin, dass die Datei eine lineare Folge von Instruktionen enthält, die in dem Graphersetzungssystem GrGen.NET [GBG$^+$06] einen Graphen erzeugen. Der so erzeugte Graph repräsentiert das übersetzte SAL$_E$-Dokument.

GrGen.NET ist eine virtuelle Maschine die Graphersetzungsbefehle ausführt. Das Programm muss mit zwei Arten von Dateien parametrisiert werden: mit Modelldateien und mit Regeldefinitionen („grg“) siehe Punkt (4), (5) oder (10). Die Modelldateien definieren die Knoten- und Kantentypen sowie deren Attribute und Typbeziehungen, die zur Laufzeit in der virtuellen Maschine zur Verfügung stehen. In einer früheren Arbeit wurde GrGen.NET um UML-Modelldateien erweitert [GDG08]. Damit ist GrGen.NET in der Lage UML-Graphmodelle zu erzeugen. Die Regeldefinitionen enthalten Regeln der Form „linke Seite $\mapsto$ rechte Seite“. Die linke Seite ist ein Graphmuster, das im SAL$_E$graphen gesucht wird. Wird es gefunden, *kann* der Graph an dieser Stelle entsprechend der rechten Seite abgeändert werden. Im Gegensatz zu herkömmlichen (Term-) Ersetzungssystemen wie die funktionale Programmiersprache Haskell gibt es in GrGen.NET keine durch die Deklarationsreihenfolge festgelegte Anwendungssemantik. Stattdessen steuert ein weiteres „Graph Rewrite Script“ die Auswahl.

In einem ersten Transformationsschritt erzeugt GrGen.NET aus dem geladenen SAL$_E$-Graphen ein UML-Softwaremodell in Form eines Graphen (4). Der SAL$_E$-Graph und sein zugehöriger UML-Graph sind voneinander unabhängig und zusammenhangslos. Eine Erweiterung der Transformationsregeln verbindet beide Graphen. SAL$_E$-zu-UML-Abbildungskanten machen einzelne Modellelemente des UML-Modells verfolgbar. Die einzelnen Konstituenten der SAL$_E$-Quellmenge werden lokalisiert. Die abgebildeten Modellelemente können auf die einzelnen Anforderungen der Spezifikation zurückverfolgt werden. Das Konzept der Abbildungskanten führt das Prinzip der Rückverfolgbarkeit von Anforderungen ein, wie Gotel sie bespricht [Got94].

Sind beide Graphen miteinander verknüpft, wird das UML-Modell in einem weiteren Transformationsschritt in einer XMI-Datei ausgegeben, siehe (5) und (6). Diese XMI-Datei kann der Benutzer im Anschluss in sein CASE-Werkzeug [Vol09] importieren, um

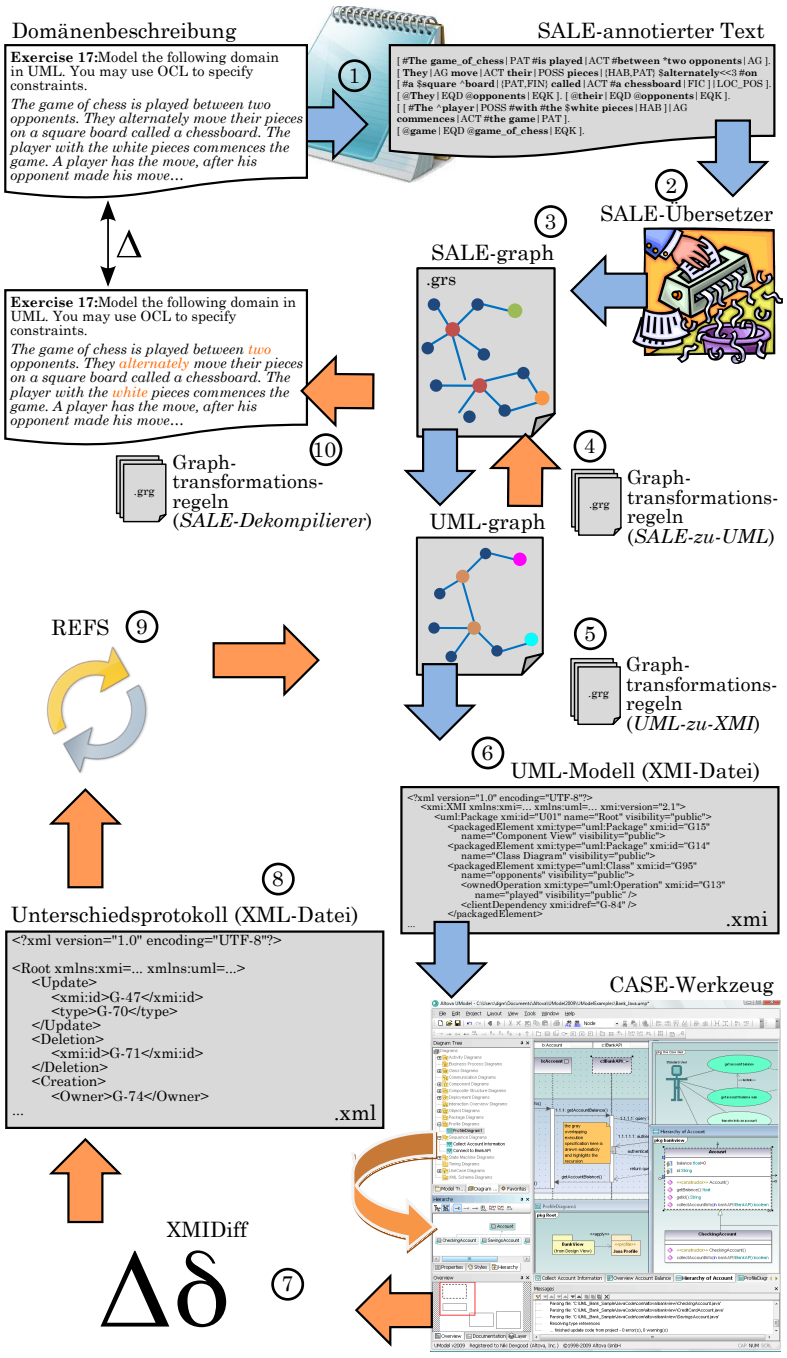

Abbildung 6: Detailübersicht: Modellerzeugung mit SALe MX und Modellrückkopplung mit REFS

das Modell zu überarbeiten oder zu verfeinern. Beim Austausch über XMI werden nur die Modelldaten übertragen. Darstellungsinformationen, wie sie beispielsweise für ein konkretes Klassendiagramm benötigt werden (Positionen, Linienzüge für Assoziationen, etc.), sind in einer XMI-Datei nicht enthalten. Dies vereinfacht die Implementierung, erfordert aber beim Importieren jeweils das Anlegen und Anordnen der gewünschten Diagramme. Je nach Leistungsfähigkeit des verwendeten CASE-Werkzeugs werden beim Importieren der Modellelemente automatisch mehr oder weniger brauchbare Diagramme angelegt und grob angeordnet. Die Diagramme in dieser Arbeit sind jedoch jeweils von Hand angeordnet, um sie für den Druck zu optimieren.

Das importierte Diagramm wird nun im CASE-Werkzeug geändert. Anpassungen führen zu Änderungen im Modell die üblicherweise nicht von CASE-Werkzeugen protokolliert werden. Mit ModelDiff [Ok10] steht uns ein eigenes Werkzeug zur Protokollierung von Modelländerungen zweier verschiedener Modellversionen, zur Verfügung (7). ModelDiff listet alle Modelländerungen auf. Dieses Unterschiedsprotokoll dient im nächsten Schritt der Modellrückkopplungskomponente als Eingabedatei um die protokollierten Modelländerungen zuerst im ursprünglichen UML-Graph abzugleichen (8).

Erst danach kann die ursprüngliche Spezifikation und deren SAL_E-Graphen durch REFS abgeglichen werden. Die Abgleichung der Spezifikation erfolgt je nach Rückkopplungsart:

- `Creation`s fügen neue Textphrasen an das Ende des Dokuments ein. Dabei werden vorgegebene thematische Rollen den neu erzeugten Konstituenten in den Phrasen zugeordnet.
- `Update`s aktualisieren den textuellen Wert der betroffenen Konstituenten im SAL_E-Graphen,
- `Deletion`s lassen den betroffenen Konstituenten im betroffenen Kontext der Spezifikation aus oder führen zur Löschung einer bestimmten thematischen Rolle.

Nach dem Spezifikationsabgleich durch REFS wird der abgeglichene SAL_E-Graph mit dem SAL_E-Dekompilierer dekompiliert und in einem Dokument gespeichert. Der SAL_E-Dekompilierer kann das Dokument wahlweise als SAL_E-annotierte Spezifikation oder Spezifikation ohne SAL_E-Annotation ausgegeben, siehe Punkt (10).

Der Vorteil gegenüber anderen Ansätzen liegt in der Erhaltung der ursprünglichen grammatikalischen Struktur der Spezifikation. Es wird ein direkter Vergleich der ursprünglichen Version der Spezifikation und seiner abgeglichenen Version ermöglicht. Die Dokumente sind für den automatischen Vergleich in einem beliebigen Textbearbeitungsprogramm bereit. Die Kunden und Analysten müssen nicht nach den geänderten Stellen in der Spezifikation suchen.

3.2. Vorgehen

Die Vorgehensweise bei aktualisierten und gelöschten Elementen ist klar und transparent. Die entsprechende Elemente werden aktualisiert `Update` oder gelöscht (`Deletion`). Bei hinzugefügten Elementen (`Creation`) müssen neue Textphrasen hinzugefügt werden.

Neue Elemente definieren neue Anforderungen für die textuelle Spezifikation, die nach dem Textmuster in Tabelle 1 in die Spezifikation rückgekoppelt werden.

Tabelle 1: Textuelle Abbildung neuer Modellelemente.

Modellelement	**Abbildung in Spezifikation**
Klasse	There is/are <*NameDerKlasse*>.
Attribut	<*NameDesAttributs*> <*NameDerKlasse*>.
Operation	<*NameDerKlasse*> <*AttributnameDerOperation*> <*NameDerOperation*>.
Parameter	<*KlassennameDerOperation*> <*NameDerOperation*> <*NameDesParameters*>.
Assoziation	The <*NavigierenderAttributName*> <*NavigierenderAttributtyp*> <*RollennameDerAssoziation*> the <*NavigiertesAttributName*> <*NavigierterAttributTyp*>.
Generalisierung	<*NameDerErweiterndenSubklasse*> is/are <*NameDerErweitertenSuperklasse*>.

3.3. Zusammenfassung

Die Erweiterung der SAL$_E$-zu-UML-Graphtransformationsregeln um die Abbildungskanten bildet den Grundstein für die Rückverfolgbarkeit der modellierten Anforderungen. Mit den rückverfolgbaren Kanten ist REFS in der Lage Modelländerungen wieder zurück in die Spezifikation einzupflegen. Eine reale Verknüpfung zwischen der Spezifikation und den modellierten Anforderungen entsteht. Der Vorteil liegt in der abgeglichenen Spezifikation mit dem Erhalt der ursprünglichen grammatikalischen Struktur. Die Struktur der Spezifikation bleibt nach der Abgleichung weitgehend erhalten und hat einen hohen Wiedererkennungswert für den Kunden und Analysten. Ein direkter Vergleich zwischen den Spezifikationen ist damit möglich.

4. Implementierung

Dieses Kapitel beschreibt die Implementierung des in Abschnitt 3 vorgestellten Entwurfs. Unterabschnitt 4.1 beschreibt die Erweiterung der SAL$_E$-zu-UML-Graphtransformationsregeln um die rückverfolgbaren Abbildungskanten. Die Rückverfolgbarkeit durch die Abbildungskanten garantiert die Verknüpfung zwischen den textuellen Anforderungen der Spezifikation und dem Softwaremodell. Die in einer Graphrepräsentation vorliegende initiale SAL$_E$-Spezifikation und das zugehörige erzeugte Modell werden gespeichert. Sie enthalten die ursprünglichen Informationen der Spezifikation und des Modells. Diese werden dann durch die vorgestellte Modellrückkopplungskomponente REFS abgeglichen.

In Unterabschnitt 4.2 wird das Vergleichsprotokoll vorgestellt. Das Vergleichsprotokoll listet alle Änderungen, die am ursprünglichen Modell gemacht wurden. Das Vergleichsprotokoll dient als Instruktionsfolge für die Modellabgleichungs- und die Modellrückkopplungskomponente.

Mit der Modellabgleichungskomponente wird das initiale Softwaremodell mit den Modelländerungen abgeglichen, siehe Unterabschnitt 4.3.

Wurde das ursprüngliche Modell synchronisiert, beginnt die Arbeit der Rückkopplung in die ursprüngliche Spezifikation. In Unterabschnitt 4.4 stellen wir REFS mit allen Einzelheiten der Rückkopplung vor.

In Unterabschnitt 4.5 transformiert der SAL$_E$-Dekompilierer den geänderten SAL$_E$-Graphen in ein Dokument, wahlweise mit oder ohne SAL$_E$-Annotierung. Dieses Dokument wird mit der ursprünglichen Version verglichen.

4.1. Die rückverfolgbaren Abbildungskanten

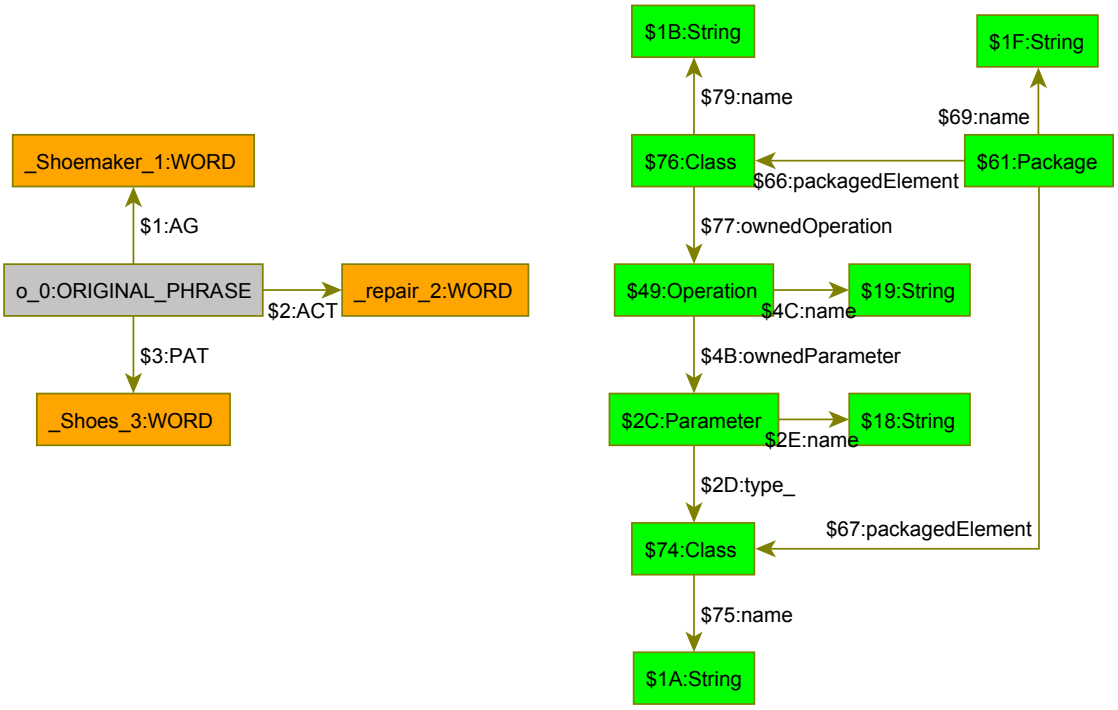

Abbildung 7: Grafische Darstellung des geladenen SAL$_E$-Graphen und daraus erzeugten UML-Graphmodells.

Die bestehenden SAL$_E$-zu-UML Graphtransformationsregeln sind für die Erzeugung des UML-Softwaremodells verantwortlich. Aus den einzelnen textuellen, semantisch annotierten Anforderungen wird das UML-Softwaremodell in Form eines serialisierten Graphen erzeugt. Der geladene SAL$_E$-Graph und erzeugte UML-Graph sind voneinander unabhängig, d. h. zusammenhangslose Graphen, siehe Abbildung 7. Um eine Verbindung zwischen dem SAL$_E$-Graphen und dem daraus erzeugten UML-Modell herzustellen, erweitern wir die Modelltransformationsregeln. Abbildungskanten sollen Konstituenten mit den daraus erzeugten Modellelementen verbinden und die Rückverfolgbarkeit von den modellierten Anforderungen auf die textuellen Anforderungen der Spezifikation garantieren.

Bisher wurde aus einer Konstituente das entsprechende Modellelement abgebildet und zum UML-Graphen hinzugefügt ohne sie miteinander zu verbinden. Die eingeführte Graphtransformationsregel, siehe Auszug 1, führt die rückverfolgbare Abbildungskante ein. Sie wird bei jeder Modellabbildungsregel aufgerufen. Die Regel verbindet die Konstituente mit seinem abgebildeten Modellelement. Eine Verbindung zwischen der Quell- und Bildmenge entsteht.

```
/* Trace the passed UML element. */
rule TraceSALE2UML( c :CONSTITUENT, cmofNode :CMOF_NODE )
{
   negative
   { c -:sale2uml-> cmofNode; }
   modify
   { c -:sale2uml-> cmofNode; }
}
```

Auszug 1: Graphtransformationsregel verbindet eine Konstituente mit einem UML-Modellelement.

Die Regel erwartet jeweils eine SAL_E-Konstituente und ein UML-Modellelement als Eingabeargumente um sie mit der SAL_E-zu-UML-Kante zu verbinden. Die *negative* Regelanwendung verhindert doppelte Kanten zwischen den Verbindungspunkten um redundante Abbildungskanten zu vermeiden.

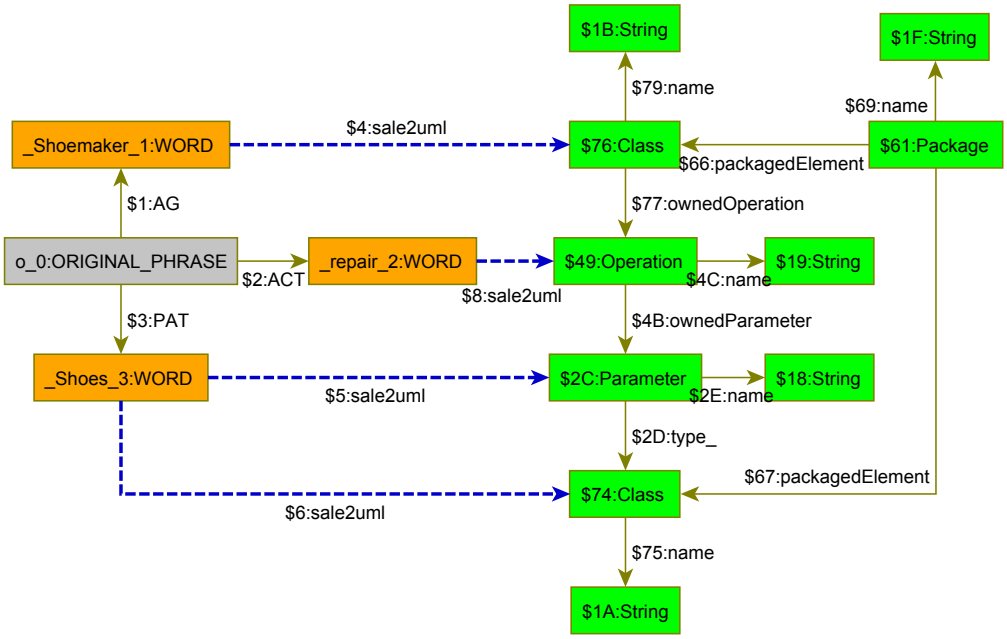

Abbildung 8: Grafische Darstellung der, durch *sale2uml*-Rückverfolgungskanten, verbundenen Graphen.

Abbildung 8 illustriert die Rückverfolgbarkeit der Abbildungskanten. Die SAL_E-Phrase besteht aus den Konstituenten *Shoemaker repair Shoes*. Die Konstituente *Shoemaker* wird zu einer Klasse abgebildet. Die Klasse ist rückverfolgbar auf seine Konstituente. Die Konstituente *repair* wird auf eine Operation abgebildet, dessen Besitzer die Klasse *Shoemaker* ist. Auch hier haben wir eine Abbildungskante, die die Operation rückverfolgbar

macht. Die nächste Konstituente *Shoes* erhält zwei Abbildungskanten. Zum einen wird er als Klasse abgebildet, zum anderen bildet er einen Parameter in der Operation *repair*. Abbildungen müssen nicht zwingend bijektiv sein, sondern können surjektiv abgebildet werden. Jedes Element der Bildmenge (UML) nimmt mindestens einen Funktionswert an, hat also mindestens eine Konstituente der SAL_{E}-Quellmenge.

4.2. Das Unterschiedsprotokoll

Um Änderungen der ursprünglichen Version und der geänderten Version des Modells zu synchronisieren, müssen die einzelnen Unterschiede festgehalten werden. Eine Lösung bietet die Protokollierung der Modellunterschiede in einem strukturierten und maschinen- sowie menschenlesbaren Dokument. Gut geeignet ist hierfür das strukturierte XML-Format. Die Beschreibung von Modelländerungen im XML-Dokument ist transparent. Ein eigens definiertes XML Schema legt die Protokollierung der Änderungen fest. Die Protokollierung von Modelländerungen wird in den folgenden Kapiteln ausführlich erklärt. Alle Modelländerungen sollen im selben XML-Dokument gelistet werden. Die Reihenfolge der einzelnen Protokollknoten ist beliebig. Der eindeutige Kennzeichner eines mit SAL$_e$ MX erzeugten Modellelements beginnt mit einem *G*-; ein abschließender mehrstelliger, ganzzahliger Wert dient zur eindeutigen Kennzeichnung.

Der Creation-Protokollknoten. `Creation`-Protokollknoten definieren neu hinzugefügte Modellelemente.

```
<!-- Füge neues Modellelement hinzu -->
<Creation>
   <Owner>G-xxx</Owner>
   <owned...  xmi:type="uml:SomeType" xmi:id="G-xxxxx"
         attr1=".." attr2=".." ...>
      <anotherOwned... xmi:type="uml:SomeType" xmi:id="G-xxxxx"
            attr1=".." attr2=".." ...>
      </anotherOwned...>
   </owned...>
</Creation>
```

Auszug 2: `Creation`-Protokollknoten im Unterschiedsprotokoll.

Ein `Creation`-Protokollknoten referenziert im Owner-Kindknoten den Besitzer des neu hinzuzufügenden Modellelements. Ist der Besitzer festgelegt, kann das neue Modellelement dem Besitzer untergeordnet und dem UML Modell hinzugefügt werden. Das Modellelement kann seine Kindelemente im selben `Creation`-Protokollknoten definieren ohne einen neuen `Creation`-Knoten in der Datei zu definieren. Der Prozess kann rekursiv für alle weiteren Kindelemente angewendet werden.

Der Update-Protokollknoten. Aktualisierte Werte eines Modellelements werden in einem `Update`-Protokollknoten beschrieben.

```
<!-- Aktualisiere Modellelement-->
<Update>
   <xmi:id>G-xxx</xmi:id>
   <type_1>value</type_1>
   <type_2>value</type_2>
   ...
   <type_n>value</type_n>
</Update>
```

Auszug 3: `Update`-Protokollknoten im Unterschiedsprotokoll.

Der erste `xmi:id` Kindknoten des `Update`-Knotens referenziert das bestehende Modellelement dessen Werte aktualisiert werden sollen. Alle danach definierten Kindknoten des `Update`-Knotens beschreiben die Werte, die aktualisiert werden. Es kann mehr als eine Aktualisierung beschrieben werden, wenn es sich um eine Aktualisierung des selben Modellelements handelt.

Der Deletion-Protokollknoten. Gelöschte Modellelemente werden in einem `Deletion`-Protokollknoten definiert.

```
<!-- Lösche Modellelement -->
<Deletion>
   <xmi:id>G-xxx</xmi:id>
</Deletion>
```

Auszug 4: `Deletion`-Protokollknoten im Unterschiedsprotokoll.

Der `xmi:id`-Kindknoten referenziert das zu löschende Modellelement. Kindelemente des gelöschten Modellelements werden ebenfalls gelöscht. Die Löschung wird rekursiv bis zum letzten Kindelement wiederholt. Die Löschung der Kindelemente müssen daher nicht erneut beschrieben werden. Die Angabe des obersten Kindelements reicht damit zur Beschreibung der Löschung aus.

4.3. Modellabgleich mit SUDiff

Um die Rückkopplung zu realisieren benötigen wir einen Abgleich des Modells. Die Modelländerungen aus dem Unterschiedsprotokoll müssen zuerst mit dem ursprünglichen Modell abgeglichen werden bevor die Rückkopplung in die Spezifikation durchgeführt werden kann. „Synchronize Uml Differences“ (SUDiff) fügt die Modelländerungen in das ursprüngliche Modell ein. Als Eingabedatei für SUDiff dient das Unterschiedsprotokoll das die Modelländerungen beschreibt.

Bei Modellelementen unterscheiden wir zwischen Beziehungen und Objekten. Objekte werden durch Beziehungen verbunden. Um Konflikte mit dem ursprünglichen Modell zu vermeiden ist eine differenzierte Behandlung in der Reihenfolge des Modellabgleichs nötig. SUDiff synchronisiert das Modell in der Create/Update/Read (`C/U/D`)-Reihenfolge.

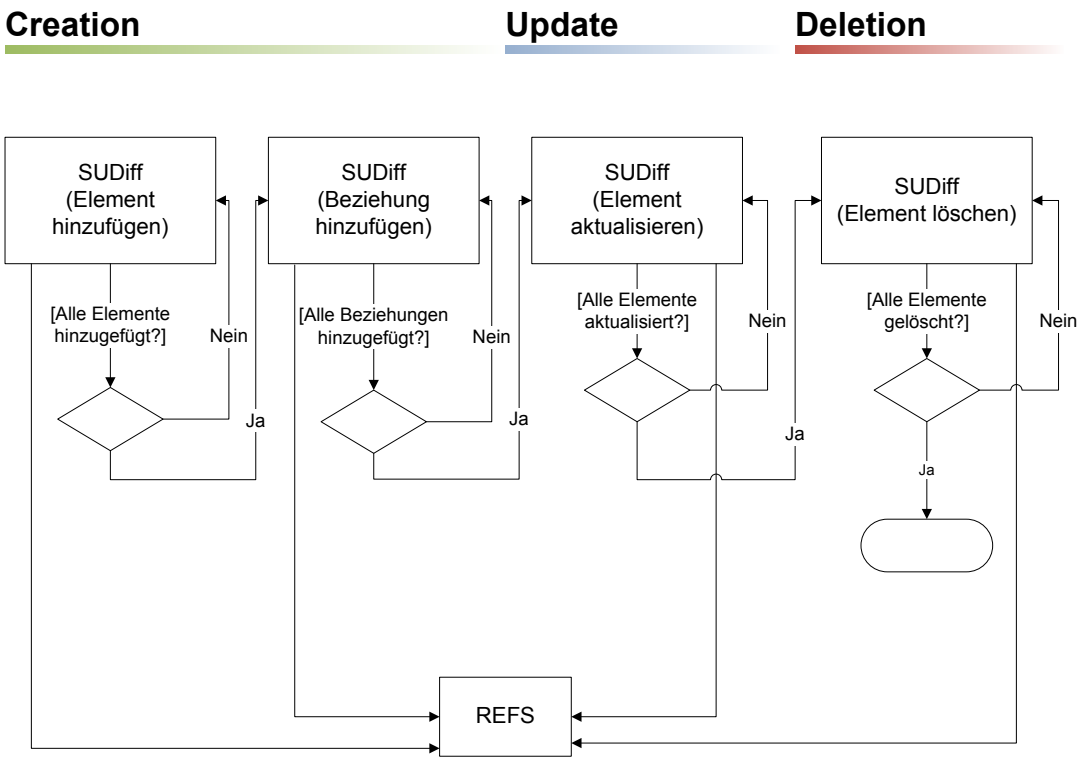

Abbildung 9: Prozessablauf von SUDiff.

SUDiff instruiert zuerst das Hinzufügen neuer Elemente. So ist die korrekte Ausführung aktualisierter Beziehungen gewährleistet. Beziehungen können neue Modellelemente referenzieren, die im Modell vorhanden sein müssen. Nach dem das neue Modellelement im ursprünglichen Modell abgeglichen wurde ruft die SUDiff-Applikation REFS auf um das neue Modellelement in die Spezifikation einzufügen. Der Prozess von REFS wird in Unterabschnitt 4.4 genauer beschrieben.

Das Aktualisieren bzw. Löschen eines Modellelements benötigt keine besondere Ablauffolge für den Modellabgleich. Die jeweiligen Instruktionen beinhalten bei jedem Durch-

lauf eines solchen Protokollknotens den Aufruf von REFS um die Rückkopplung der Änderung in die Spezifikation vorzunehmen.

SUDIFF prüft das Unterschiedsprotokoll nicht auf semantische Korrektheit und garantiert damit nicht die Korrektheit des Modells.

4.4. Spezifikationsabgleich mit REFS

Das „Requirements Engineering Feedback System“ (REFS) pflegt Modelländerungen in die ursprüngliche Spezifikation ein. Die von Modelländerungen betroffenen Konstituenten können durch die in Unterabschnitt 4.1 vorgestellten, rückverfolgbaren Abbildungsskanten lokalisiert werden. So kann zu jeder modellierten Anforderung des Softwaremodells die zugehörige Konstituente im SAL$_E$-Graphen gefunden werden.

Die Bedeutung der in den folgenden Beispielen vorkommenden thematischen Rollen kann Anhang A entnommen werden. Für eine vollständige Liste aller in SAL$_e$ MX verfügbaren thematischen Rollen sei auf die projekteigene Internetseite [KDGL] bzw. die Dissertation von Gelhausen [Gel10] verwiesen.

4.4.1. Neue Elemente hinzufügen

Bei der Modellprüfung können neue Elemente hinzugefügt werden um fehlende oder mangelhaft dokumentierte Teile der Spezifikation zu ergänzen. Werden neue Elemente hinzugefügt dann müssen auch neue Anforderungen bzw. Anforderungsteile in der Spezifikation definiert werden. Um diese neuen Elemente in der Spezifikation zu reflektieren benötigen wir einen Texterzeuger. Er erzeugt die textuellen Anforderungen zu den neu hinzugefügten Elementen des Modells. Der Texterzeuger kann, je nach Wahl des Benutzers, eine SAL$_E$-annotierte Version oder eine rein natürlichsprachliche (NL) Version des abgeglichenen Dokuments ausgeben lassen. Die natürlichsprachliche Version verzichtet auf die Ausgabe der annotierten thematischen Rollen. Jede erzeugte Textphrase wird an das Ende des bestehenden Dokuments eingefügt. In den folgenden Abschnitten wollen wir erläutern wie der generierte Text für die einzelnen Elemente des Klassendiagramms wiedergegeben wird. Die Rückkopplung wird an der folgenden beispielhaften Spezifikation veranschaulicht, siehe Auszug 5.

```
1 [ {Boots Sneakers Sandals AND Mocassins}|FIN #are Shoes|FIC ].
2 [ Shoemaker|AG repair|ACT $black @Shoes|PAT ].
3 [ Shoemaker|AG produce|ACT $leathern @Shoes|OPUSP ].
4 [ @Shoemaker|POSS #have *20 Tools|HAB ].
```

Auszug 5: Ursprüngliche SAL$_E$-annotierte Spezifikation

Ausgangssituation. *Boots*, *Sneakers*, *Sandals* und *Mocassins* sind *Shoes*. Die Auszeichnung des thematischen Rollenpaars `FIN` und `FIC` drückt die Ist-Ein-Beziehung zwischen den Konstituenten aus. Die Beziehung wird bei der SAL$_E$-zu-UML-Transformation im UML-Klassendiagramm [Gro07a] zu einer Generalisierungsbeziehung zwischen den erzeugten Klassenelementen modelliert, siehe Abbildung 10.

In der nächsten Phrase übt der Schuhmacher (`AG`) eine Tätigkeit aus. Er repariert (`ACT`) Schuhe (`PAT`). Dieses wird in einem Klassendiagramm als Klasse mit einer eingebundenen Operation modelliert. Die Operation besitzt einen Parameter des Typs *Shoes*, der beim Aufruf übergeben wird.

In Zeile 3 produziert (`ACT`) der *Shoemaker* (`AG`) *Shoes* (`OPUSP`). Der agierende Schuhmacher übt eine Tätigkeit aus, ähnlich wie in Zeile 2 der Spezifikation. Hier erzeugt der

Schuhmacher ein Produkt Schuhe, das dann zurückgegeben wird. Die Schuhe werden mit der Rolle `OPUSP` ausgezeichnet um sie in der *erschaffenden* Operation *produce* als Rückgabeparameter zu modellieren. Eine zusätzliche Assoziation zwischen den beiden Typen Schuhmacher und Schuhen stellt die Navigierbarkeit und des Eigentums der Schuhe dar.

Das SAL$_{\text{E}}$-Attribut *leathern* ordnet der Klasse Schuhe ein Attribut mit einen booleschen Wert zu. Dieses Attribut kann dann in der Operation *produce* genutzt werden um den Schuhmacher lederne Schuhe produzieren zu lassen.

In der letzten Phrase wird eine Verbindung zwischen dem Schuhmacher und seinen Werkzeugen beschrieben. Der *Shoemaker* (`POSS`) ist der Inhaber von *Tools* (`HAB`), wobei er 20 Stück davon besitzt. Im erzeugten Klassendiagramm beschreibt eine besitzanzeigende, navigierbare Assoziation zwischen dem Schuhmacher und seinen Werkzeugen das Besitzverhältnis. Die Spezifikation wird mit SAL$_e$ MX in folgendes Softwaremodell transformiert.

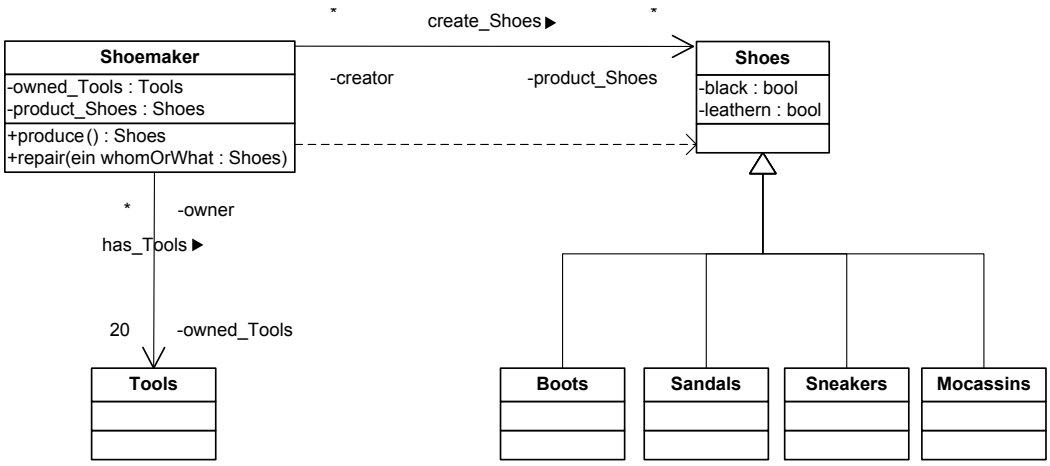

Abbildung 10: Mit SAL$_e$ MX erzeugtes, initiales Klassendiagramm.

Hinzufügen neuer Klassen. Für eine neu eingefügte Klasse wird in der textuellen Spezifikation eine neue Phrase in das bestehende Dokument eingefügt. Eine neue *Phrasen*-Konstituente mit einer angehängten *Word*-Konstituente repräsentiert die neu hinzugefügte Klasse. Die *Word*-Konstituente wird anschließend mit der Klasse verbunden, um die Rückverfolgbarkeit der modellierten Klasse zu gewährleisten. Die textuelle Ausgabe der erzeugten Phrase in der abgeglichenen Spezifikation sieht wie folgt aus:

- SAL$_E$: [#{*Thereis/are*} <*NameDerKlasse*>|OBJECTROLE].
- NL: There is/are <*NameDerKlasse*>.

Das Einfügen einer Klasse wollen wir an Hand eines kleinen Beispiels fortführen um die Grundlagen der Implementierung zu festigen. Wir ergänzen das ursprüngliche Modell um zwei neue Klassen. Die neuen Klassen `Person` und `Flip-Flop`, siehe Abbildung 11, sollen das extrahierte Modell der ursprünglichen Spezifikation ergänzen.

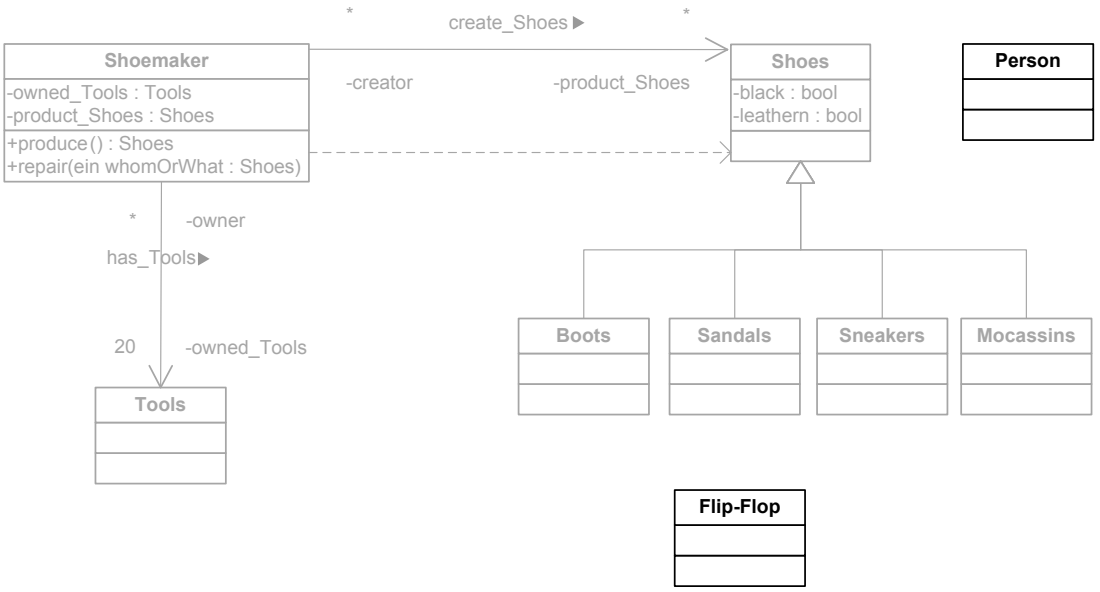

Abbildung 11: Zwei Klassen ergänzen das Klassendiagramm.

Das Unterschiedsprotokoll, siehe Auszug 6, beschreibt die neu eingefügten Klassen als Unterschiede zum ursprünglichen Modell. Der Besitzer der Klassen gibt das Paket an, in dem die Klassen als Kindelemente angehängt werden. Das oberste Paket ist in diesem Falle das oberste UML-Wurzelpaket.

Der nächste XML-Kindknoten definiert die Klasse, mit all ihren Attributen und Kindelementen. REFS verarbeitet die Protokollknoten und erzeugt Textphrasen. Die erzeugten Textphrasen der neuen Klassen sind in Auszug 12, Zeile 5 und 6, dokumentiert.

```
<!-- Füge Klasse 'Person' hinzu -->
<Creation>
   <Owner>G-110</Owner>
   <packagedElement xmi:type="uml:Class" name="Person"
      xmi:id="G-299">
   </packagedElement>
</Creation>

<!-- Füge Klasse 'Flip-Flop' hinzu -->
<Creation>
   <Owner>G-110</Owner>
   <packagedElement xmi:type="uml:Class" name="Flip-Flop"
      xmi:id="G-309">
   </packagedElement>
</Creation>
```

Auszug 6: Auszug aus dem Unterschiedsprotokoll: Zwei neue Klassen *Person* und *Flip-Flop* werden gelistet.

Hinzufügen neuer Attribute. REFS führt neue Attribute in die ursprüngliche textuelle Spezifikation zurück. Die neu hinzugefügten Attribute müssen vom Typ *Boolean* sein, da anders typisierte Attribute nicht von SAL$_e$ MX unterstützt werden und nicht in der Spezifikation reflektiert werden können. Dazu wird eine neue Phrase mit einem neuen SAL$_E$-Attribut zusammen mit der Referenz auf seinen Besitzer in den Graphen eingefügt. REFS erzeugt eine neue Textphrase, die wie folgt textuell ausgegeben wird:

- SAL$_E$: [$<*NameDesAttributs*> @<*NameDerKlasse*>|OBJECTROLE].
- NL: <*NameDesAttributs*> <*NameDerKlasse*>.

Das Hinzufügen von Attributen wollen wir an folgendem Beispiel illustrieren: das Modell wird um ein neues Attribut erweitert. In das Modell fügen wir das neue Attribut *iron_made* in die Klasse *Tools* ein, siehe Abbildung 12.

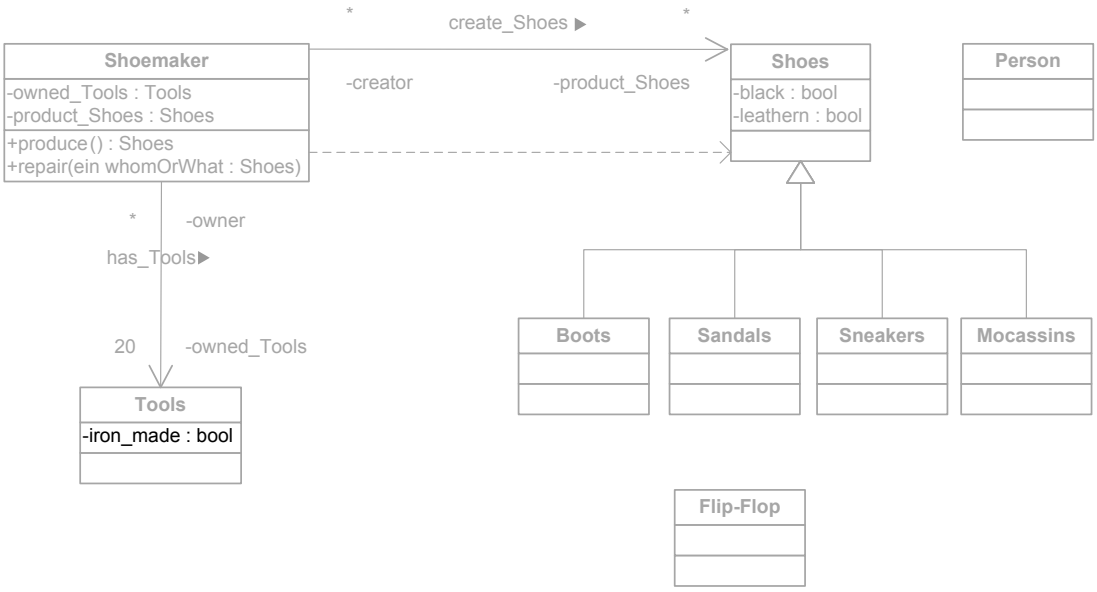

Abbildung 12: Klassendiagramm ergänzt um ein neues Attribut.

Im Unterschiedsprotokoll, siehe Auszug 7 wird das Attribut in einem `Creation`-Protokollknoten beschrieben. Das Ergebnis der Textausgabe ist in Zeile 8, siehe Auszug 12, niedergeschrieben.

```
<!-- Füge Attribut 'iron_made' in Klasse 'Tools' hinzu -->
<Creation>
   <Owner>G-113</Owner>
   <ownedAttribute name="iron_made" xmi:type="uml:Property"
      xmi:id="G-349" ... >
      <type xmi:idref="G-77" />
   </ownedAttribute>
</Creation>
```

Auszug 7: Auszug aus dem Unterschiedsprotokoll: Einfügen eines neuen Attributs in der Klasse *Tools*.

Hinzufügen neuer Operationen. Operationen werden in der textuellen Spezifikation in einer neue Phrase zusammen mit der Klasse, die sie beinhaltet, in das bestehende Dokument eingefügt. Der generierte Text wird nach folgendem Muster erzeugt:

- SAL$_E$: [<*NameDerKlasse*>|OBJECTROLE [$<*AttributnameDerOperation*>]? <*NameDerOperation*>|METHODROLE].
- NL: <*NameDerKlasse*> <*AttributnameDerOperation*> <*NameDerOperation*>.

Wir erweitern unser Beispielmodell um eine Operation `worn_by` in der Klasse `Shoes`, siehe Abbildung 13.

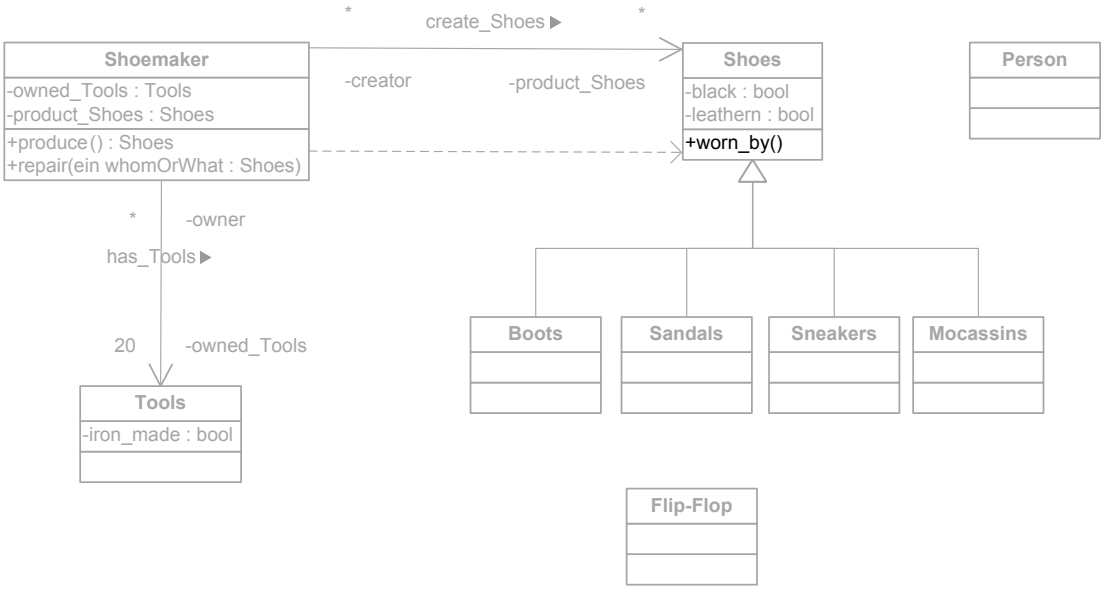

Abbildung 13: Klassendiagramm ergänzt um eine neue Operation.

Der Analyst fügt eine neue Operation `worn_by` in die Klasse `Shoes` ein um die Aktivität zu kennzeichnen, dass Schuhe getragen werden können.

Das Unterschiedsprotokoll listet die neue Operation auf, siehe Auszug 8. Das Ergebnis der Verarbeitung steht in Auszug 12, Zeile 7.

```
<!-- Füge Operation 'worn_by' in Klasse 'Shoes' ein -->
<Creation>
   <Owner>G-118</Owner>
   <ownedOperation name="worn_by" xmi:type="uml:Operation"
      xmi:id="G-319" ... >
      <ownedParameter name="whomOrWhat" xmi:type="uml:Parameter"
         xmi:id="G-320" ... >
         <type xmi:idref="G-299" />
      </ownedParameter>
   </ownedOperation>
</Creation>
```

Auszug 8: Auszug aus dem Unterschiedsprotokoll: Einfügen einer neuen Operation in der Klasse *Shoes*.

Hinzufügen neuer Parameter. Wird ein neuer Parameter in eine Operation eingefügt, wirkt sich das auf die Spezifikation aus. An das Ende des Spezifikationstextes wird eine neue Phrase mit folgendem Inhalt eingefügt.

- SAL_{E}: [<*KlassennameDerOperation*>|OBJECTROLE <*NameDerOperation*>|METHODROLE <*NameDesParameters*>|ADJUNCTROLE].
- NL: <*KlassennameDerOperation*> <*NameDerOperation*> <*NameDesParameters*>.

Einfügen eines neuen Parameters in eine vorhandene Operation.

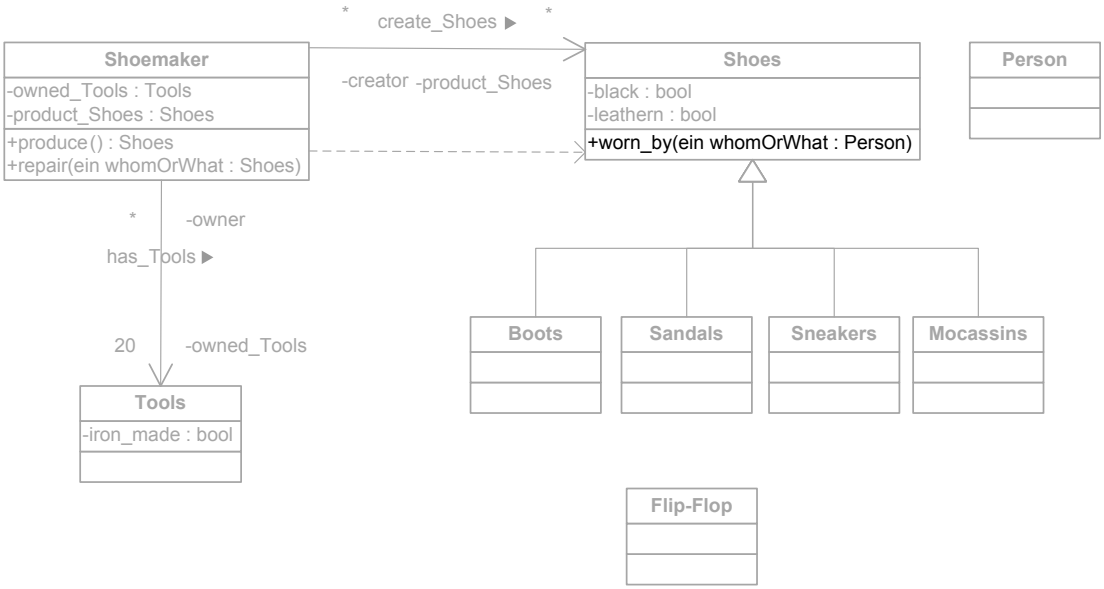

Abbildung 14: Klassendiagramm ergänzt um eine neuen Parameter in der Operation.

Die Operation `worn_by` der Klasse `Shoes` wird mit dem neuen Parameter `whomOrWhat` des Typs `Person` versehen.

Der Protokollknoten, siehe Auszug 9 vermerkt den Unterschied zum ursprünglichen Modell, so dass REFS diese Änderung in das bestehende Modell einarbeiten und die Spezifikation mit der neuen Anforderung aktualisieren kann. Zeile 8 in Auszug 12 demonstriert den Einbau des neuen Parameters und die Auswirkung auf die Spezifikation.

```
<!-- Füge Operation 'worn_by' in Klasse 'Shoes' ein -->
<Creation>
   <Owner>G-118</Owner>
   <ownedOperation name="worn_by" xmi:type="uml:Operation"
      xmi:id="G-319" ... >
      <ownedParameter name="whomOrWhat" xmi:type="uml:Parameter"
         xmi:id="G-320" ... >
         <type xmi:idref="G-299" />
      </ownedParameter>
   </ownedOperation>
</Creation>
```

Auszug 9: Auszug aus dem Unterschiedsprotokoll: Einfügen eines neuen Parameters in die Operation *worn_by*.

Hinzufügen neuer Assoziationen. Eine neue Assoziation verbindet zwei Klassen miteinander. Oft wird auch ein Rollenname für die Assoziation und deren Enden angegeben. In der derzeitigen Implementierung werden nur binäre Assoziationen unterstützt, da bei den bisher untersuchten CASE-Werkzeugen kein Werkzeug mit Unterstützung für n-äre Assoziationen gefunden wurde.

Die Textausgabe für eine eingefügte Assoziation lautet:

- SAL$_E$: [#The #<*NavigierenderAttributName*> <*NavigierenderAttributtyp*>|POSS #<*RollennameDerAssoziation*> #the #<*NavigiertesAttributName*> <*NavigierterAttributTyp*>|HAB].
- NL: The <*NavigierenderAttributName*> <*NavigierenderAttributtyp*> <*RollennameDerAssoziation*> the <*NavigiertesAttributName*> <*NavigierterAttributTyp*>.

Nachfolgend wollen wir eine beispielhafte Ergänzung einer Assoziation in das bestehende Modell betrachten.

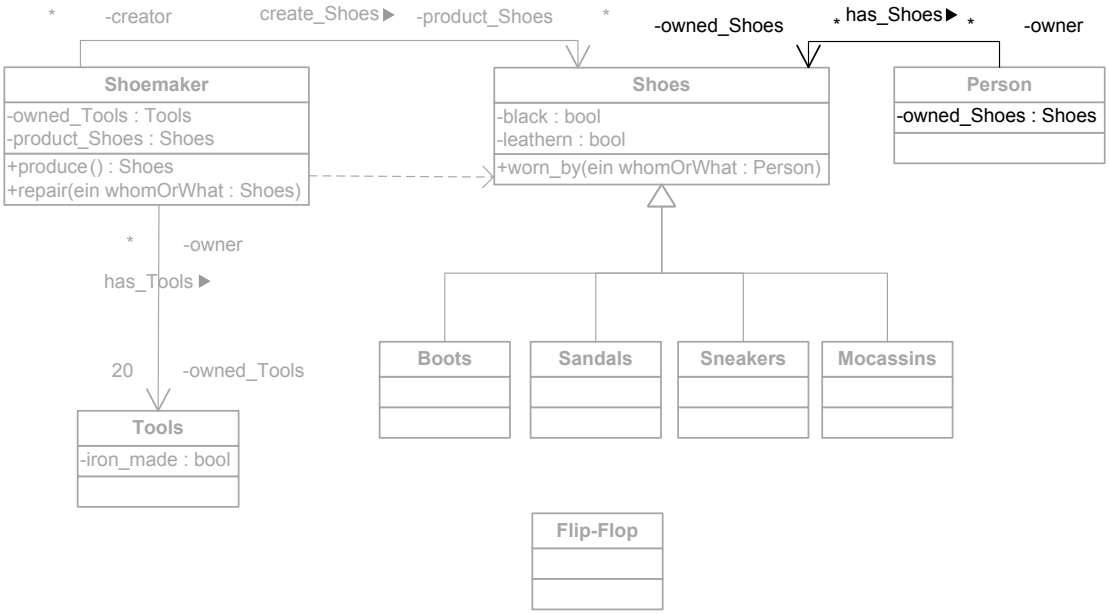

Abbildung 15: Klassendiagramm ergänzt um eine neue Assoziation zwischen *Person* und *Shoes*.

Eine neue Assoziation zwischen der Klasse `Person` und `Shoes` soll den Besitz von Schuhen einer Person ausdrücken. Die neue Assoziation ist mit dem entsprechenden Rollennamen und den Namen der Assoziationsenden versehen, siehe Abbildung 15.

Das Unterschiedsprotokoll versieht die Änderung in einem `Creation`-Protokollknoten, siehe Auszug 10. Die Auswirkung der eingefügten Assoziation auf die Spezifikation wird in der letzten Zeile im Auszug 12 demonstriert.

```
<!-- Füge Assoziation 'has_Shoes'
      zwischen 'Person' and 'Shoes' ein -->
<Creation>
   <Owner>G-110</Owner>
   <packagedElement name="has_Shoes" xmi:type="uml:Association"
      xmi:id="G-329" ... >
      <ownedEnd name="owner" xmi:type="uml:Property"
         xmi:id="G-330" ... >
         <type xmi:idref="G-299" />
         <association xmi:idref="G-329" />
      </ownedEnd>
      <memberEnd xmi:idref="G-330" />
      <memberEnd xmi:idref="G-339" />
   </packagedElement>
</Creation>
```

Auszug 10: Auszug aus dem Unterschiedsprotokoll: Einfügen einer neuen Assoziation zwischen der Klasse *Person* und *Shoes*.

Hinzufügen neuer Vererbungsbeziehungen. Eine neue Vererbungsbeziehung deutet auf eine *is-a*-Beziehung zwischen den Elementen hin. Das bedeutet, dass das erbende Element ein spezialisierter Typ seiner Superklasse ist. Dieser Sachverhalt spiegelt sich ebenfalls in der Spezifikation wider. Der erzeugte Text folgt dem Muster:

- SAL_E: [<*NameDerErweiterndenSubklasse*>|FIN #is/are <*NameDerErweitertenSuperklasse*>|FIC].
- NL: <*NameDerErweiterndenSubklasse*> is/are <*NameDerErweiterten-Superklasse*>.

Das folgende Beispiel zeigt eine neu eingeführte Vererbung.

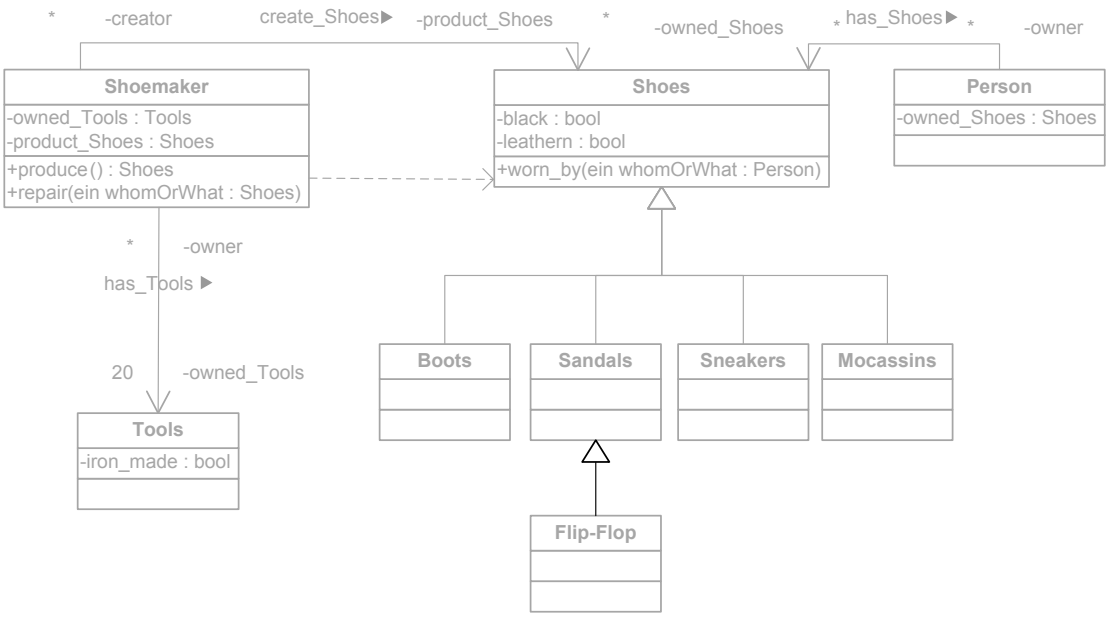

Abbildung 16: Klassendiagramm ergänzt um eine neue Vererbungsbeziehung.

Die Klasse `Flip-Flop` soll die Klasse `Sandals` erweitern, siehe Abbildung 16.

```
<!-- Füge Generalisierung
      zwischen 'Flip-Flop' und 'Sandals' hinzu -->
<Creation>
   <Owner>G-309</Owner>
   <generalization xmi:type="uml:Generalization" xmi:id="G-310">
      <general xmi:idref="G-116" />
   </generalization>
</Creation>
```

Auszug 11: Auszug aus dem Unterschiedsprotokoll: Einfügen einer neuen Vererbungsbeziehung zwischen den Klassen *Sandals* und *Flip-Flop*.

Das Unterschiedsprotokoll beschreibt die neue Vererbungsbeziehung, siehe Auszug 11. Das Ergebnis wird in der vorletzten Zeile in Auszug 12 demonstriert.

Die neu erzeugte Spezifikation mit ihrer Annotation. Die Abgleichung des SAL_{E}-Graphen resultiert in dem folgenden, automatisch ausgegebenen Spezifikationsdokument. Neue Modellelemente werden in neuen Phrasen am Ende des Dokuments gespeichert.

```
1  [ {Boots Sneakers Sandals AND Mocassins}|FIN #are Shoes|FIC ].
2  [ Shoemaker|AG repair|ACT $black @Shoes|PAT ].
3  [ Shoemaker|AG produce|ACT $leathern @Shoes|OPUSP ].
4  [ @Shoemaker|POSS #has *20 Tools|HAB ].
5  [ #{There is/are} Person|OBJECTROLE ].
6  [ #{There is/are} Flip-Flop|OBJECTROLE ].
7  [ @Shoes|OBJECTROLE worn_by|METHODROLE ].
8  [ @Shoes|OBJECTROLE worn_by|METHODROLE @Person|ADJUNCTROLE ].
9  [ $iron_made @Tools|OBJECTROLE ].
10 [ @Flip-Flop|FIN #{is/are} @Sandals|FIC ].
11 [ #{The owner} @Person|POSS #has_Shoes #{the owned_Shoes} @Shoes|HAB].
```

Auszug 12: Spezifikation nach Rückkopplung hinzugefügter Modellelemente

4.4.2. Elemente aktualisieren

Wird ein Modell geändert, können auch Aktualisierungen auftreten. Eine Aktualisierung ändert einen bestimmten Wert eines vorhandenen Modellelements. Dabei wird das existierende Modellelement beibehalten.

Die Aktualisierungen, die REFS, vornimmt wollen wir mit einem Beispiel vertiefen. Die folgende kurze SAL_E-annotierte Spezifikation, siehe Auszug 13, bildet die Grundlage für die im weiteren Text vorkommenden Beispiele.

```
1 [ {Boots Sneakers AND Mocassins}|FIN #are Shoes|FIC ].
2 [ Shoemaker|AG repair|ACT @Shoes|PAT ].
3 [ @Shoemaker|POSS #has *20 Tools|HAB ].
```

Auszug 13: Ursprüngliche SAL_E-annotierte Spezifikation

Die annotierte Spezifikation wird mit SAL_e MX automatisch in ein Modell überführt, siehe Abbildung 17.

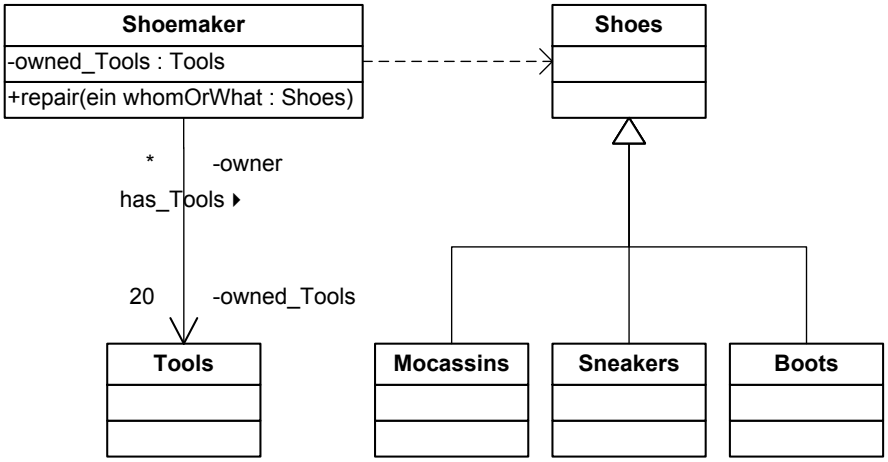

Abbildung 17: Das initiale von SAL_e MX erzeugte Modell.

Ausgangssituation. Die Mengenelemente *Boots*, *Sneakers* und *Mocassins* erweitern die Klasse *Shoes*. In SAL_E werden die Konstituenten durch eine `FIC` und `FIN` Beziehung ausgezeichnet, siehe Anhang A. Die Mengenelemente stellen eine Rolle dar. Sie können die Rolle der Klasse *Shoes* annehmen. Die semantische Beziehung wird in der Modellierung als eine „ist-ein“ Beziehung modelliert. Eine Vererbungsbeziehung in der die Mengenelemente die *Shoes* erweitern ist die korrekte Modellierungsart.

Außer den Schuhen gibt es einen Shoemaker `AG` der eine Tätigkeit `ACT` an Dingen PAT ausübt. *Shoemaker* reparieren *Shoes*. Die korrekte Modellierung stellt eine Operation in der Klasse *Shoemaker* dar. Die Operation erhält einen Parameter um auszudrücken, dass der *Shoemaker Shoes* repariert.

In der letzten Phrase bekommt der *Shoemaker* noch Werkzeuge (*Tools*) mit denen er arbeiten kann, um die *Shoes* zu reparieren. Insgesamt stehen ihm 20 *Tools* zur Auswahl um *Shoes* zu reparieren. Diese Beziehung wird in einem Klassendiagramm als Assoziation zwischen dem *Shoemaker* und den *Tools* modelliert.

Namenswerte aktualisieren. In vielen Fällen ist der Name eines Modellelements unpassend gewählt. Gründe dafür können Missverständnisse in der Spezifikation, unverständliche Abkürzungen des Domänenbereichs oder auch Tippfehler sein. Der Domänenexperte bzw. Systemanalyst passt den Namen des Elements an, um den Fehler zu korrigieren.

In der Implementierung bedeutet dies, dass alle Konstituenten der Quellmenge, die auf das korrigierte Modellelement zeigen, angepasst werden. Die SAL_E-Graphelemente, die Konstituenten repräsentieren, sind mit einem Graphknoten des Typs *String* verknüpft, der den Namen der Konstituente speichert. Der Wert des Strings wird überschrieben. Der neue Wert wird beim Dekompilieren des SAL_E-Graphes ausgegeben.

Eine Konstituente kann dabei auf mehr als ein Zielelement verweisen. Diese Konstituente wird auch dann aktualisiert, wenn nur ein Zielelement aktualisiert wurde. Dies führt zu einer impliziten Aktualisierung auf den neuen Namen aller Zielelemente bei der nächsten automatischen Modellgenerierung durch SAL_e MX.

Wir wollen eine Namensaktualisierung im folgenden Beispiel betrachten. Dabei wird der Name einer Operation geändert.

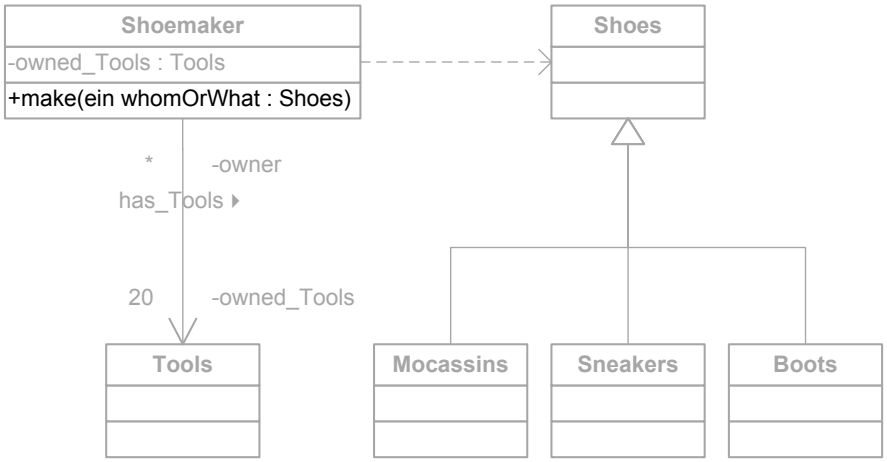

Abbildung 18: Aktualisiere den Namen der Operation *repair* auf *make* in der Klasse *Shoemaker*.

In Abbildung 18 demonstrieren wir die Aktualisierung der *repair* Operation in der Klasse *Shoemaker*. Wir unterstellen ein Mißverständnis in der Spezifikation. Wir ändern den Namen der Operation von *repair* auf *make*. Das ist das Einzige, was tatsächlich geändert werden muß. Die Parameter bleiben erhalten.

Der Namensunterschied spiegelt sich im Unterschiedsprotokoll wieder. Das zu aktualisierende Element wird mit der `xmi:id` referenziert um dann den Wert des Namens ändern zu können.

```
<!-- Update operation name from 'repair' to 'make' -->
<Update>
   <xmi:id>G-58</xmi:id>
   <name>make</name>
</Update>
```

Auszug 14: Auszug aus dem Unterschiedsprotokoll: Aktualisiere den Namen der Operation *repair* zu *make*.

Parametertypen aktualisieren. REFSunterstützt ebenfalls die Rückkopplung von Typänderungen an vorhandenen Parametern an. Typisierte Parameter wie sie in Operationen vorkommen können Typänderungen auf andere Typen erfahren. Dabei wirkt sich die Typänderung auch auf die Spezifikation.

Das Verhalten von Typaktualisierungen bei Parametern in REFS wollen wir im folgenden Abschnitt genauer betrachten. Jeder Parameter wird in einer Operation geführt. Bestimmt man die Kontextumgebung für die Operation im SAL_E-Graphen liefert das auch die den exakten SAL_E-Konstituenten des ungeänderten UML-Parameterelements. Die thematischen Rollen der alten Konstituente die für die Parametererzeugung verantwortlich sind werden in einer Liste gesammelt. Die gesammelten Rollen werden dann von dem alten Konstituenten auf die Konstituente des neuen, geänderten Parameterelements umgeleitet. So werden die selben thematischen Rollen für die neuen Konstituente beibehalten und der semantische Kontext für die Erzeugung der Operation inklusive seiner Parameter bewahrt.

Wie sich die Typänderung auf die textuellen Anforderungen genau auswirkt, kann in dem fortführenden Beispiel, siehe Abbildung 19 und deren finale dekompilierte Ausgabe exemplarisch betrachtet werden.

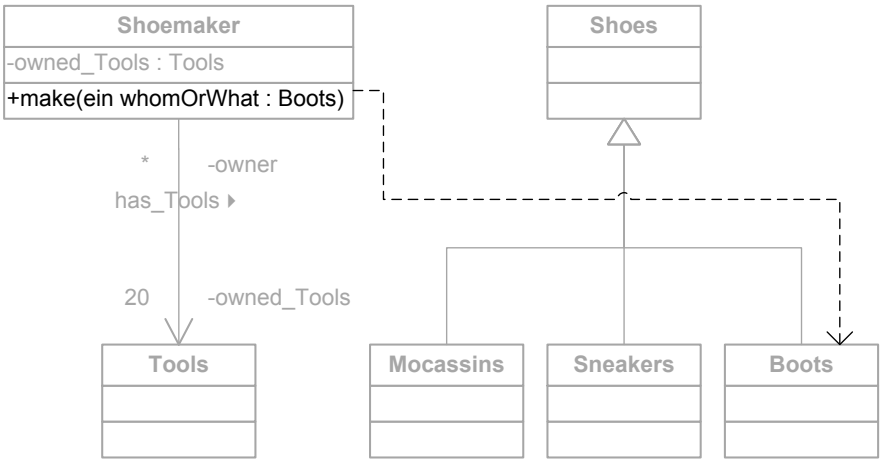

Abbildung 19: Aktualisiere den Typ des Parameter *whomOrWhat* in der Operation *make*.

Der Parameter *whomOrWhat* in der Operation *make* der Klasse *Shoemaker* besitzt den Typ *Shoes*. Der Typ des Parameters soll so geändert werden, dass der Schuhmacher nicht mehr alle Arten von Schuhen anfertigt. Der Schuhmacher soll nur noch Stiefel (Klasse *Boots*) anfertigen, siehe Abbildung 19

Das Ergebnis der Aktualisierung kann in Zeile 3 von Auszug 17 betrachtet werden. Vergleicht man das Dokument mit dem initialen Spezifikationstext fällt der Unterschied sofort auf. Im alten Text werden noch Schuhe repariert, wohingegen im neuen Dokument Stiefel gefertigt werden.

```
<!-- Update Parameter type from 'Shoes' to 'Boots' -->
<Update>
   <xmi:id>G-47</xmi:id>
   <type>G-70</type>
</Update>
```

Auszug 15: Auszug aus dem Unterschiedsprotokoll: Aktualisiere den Parametertyp von *Shoes* zu *Boots*.

Andere Werte aktualisieren. Außer Umbenennungen und Typänderungen gibt es noch andere Arten um ein Modell zu ändern. Kardinalitäten drücken eine Mengenbeziehung aus, die beschreibt wie viele Objekte in Relation zu anderen Objekten einer Assoziation stehen.

In der Abbildungsvorschrift vom SAL_{E}-Graphmodell hin zum UML-Graphmodell ist die Abbildung eines Mengenkonstituenten (*Multiplicity*) bijektiv. Eine Konstituente wird zu genau einer Multiplizitätsangabe abgebildet. Die Implementierung nutzt diese Gegebenheit. Die Ausnutzung der Bijektivität vereinfacht eine Rückkopplung in die Spezifikation.

Eine einzige Ausnahme bilden Multiplizitätsangaben für Mengen, sogenannte `Sets`. Hier wird eine Multiplizität auf mehrere Kardinalitäten für mehrere Assoziationen abgebildet. Um diesem Problem zu begegnen, wird bei einer Kardinalitätsänderung die zugehörige Konstituente aus der Menge extrahiert. Die Konstituente wird wieder hinter die Menge mit einem neuen Mengenkonstituenten eingefügt.

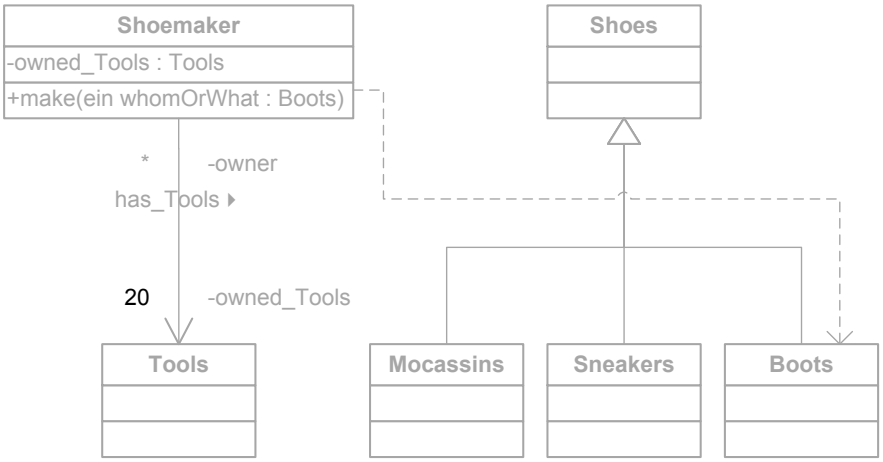

Abbildung 20: Aktualisiere die Menge der inhabenden Werkzeuge.

In Abbildung 20 wird die Mengenbeziehung zwischen dem Schuhmacher und seinen Werkzeugen geändert. Der Wert wird von 20 verfügbaren Werkzeugen auf 99 angehoben.

```
<!-- Update multiplicity value from '20' to '99' -->
<Update>
   <xmi:id>G-43</xmi:id>
   <value>99</value>
</Update>
```

Auszug 16: Auszug aus dem Unterschiedsprotokoll: Aktualisiere die Kardinalität der Mengenbeziehung zu Tools.

Das Unterschiedsprotokoll, siehe Auszug 16, definiert in einem `Update`-Protokollknoten die Anhebung der Werkzeuganzahl.

Das Ergebnis kann mit der Ausgabe der synchronisierten Spezifikation, siehe Auszug 17 verglichen werden. In Zeile 3 werden nun 99 Werkzeuge für den Schuhmacher beschrieben, statt wie vorher definiert wurde, nur 20.

Die abgeglichene Spezifikation mit ihrer Annotation. Die Aktualisierung des SAL_{E}-Graphen und dessen Ausgabe resultiert in folgendem Dokument.

```
1 [ {Boots Sneakers AND Mocassins}|FIN #are Shoes|FIC ].
2 [ Shoemaker|AG make|ACT @Boots|PAT ].
3 [ @Shoemaker|POSS #has *99 Tools|HAB ].
```

Auszug 17: Spezifikation nach Rückkopplung aktualisierter Modellelemente

4.4.3. Elemente löschen

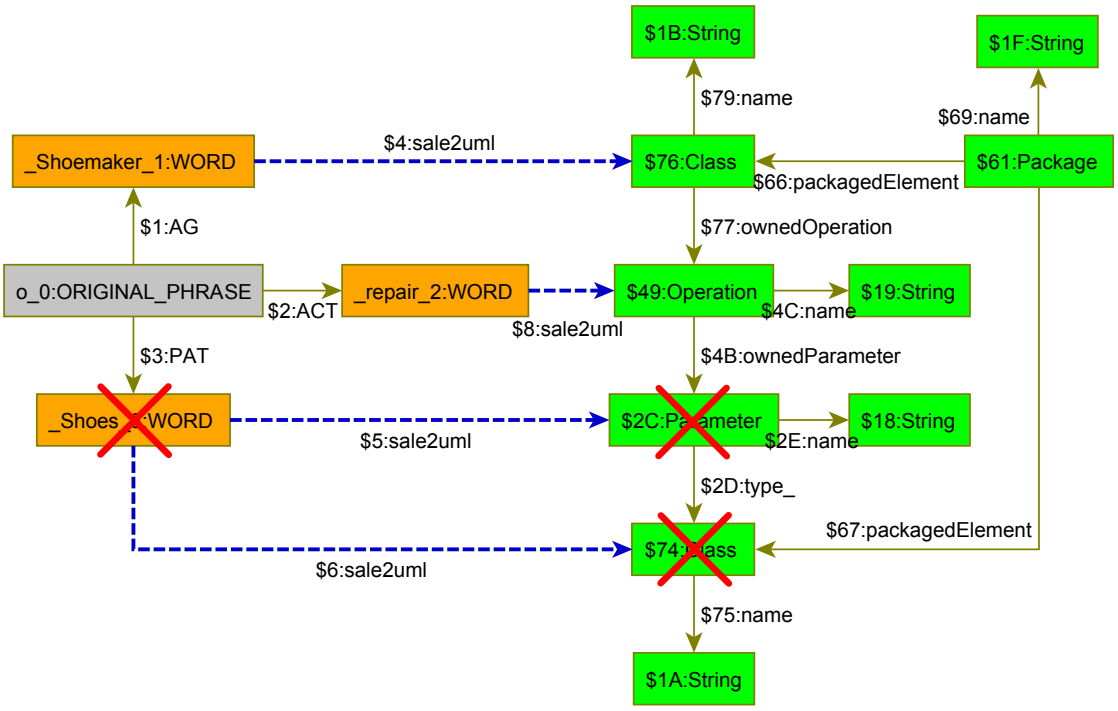

Abbildung 21: `Deletion` eines Klassen- und Parameterelements, und deren Auswirkung auf das SAL$_E$-Graphmodell.

Bei Modelländerungen kann es zur Löschung vorhandener Modellelemente kommen. Wird ein Modellelement gelöscht, kann dies zur Löschung seiner Konstituenten aus der Quellmenge führen bzw. zur Löschung bestimmter thematischer Rollen der Konstituente, siehe Abbildung 21. Die Klasse und der Parameter des Typs *Shoes* werden aus dem Modell gelöscht. Die Rückverfolgung über die *sale2uml*-Kanten führen zur ursprünglichen Konstituente *Shoes* im SAL$_E$-Graphmodell. Die Löschung im Modell führt zur Löschung der Konstituente *Shoes*.

Die Löschung einer einzelnen Konstituente aus einer Phrase führt zur Auslassung der Konstituente in der Phrase. Bei der Auslassung in einer Menge (`SET`) müssen Besonderheiten für die Menge beachtet werden. Es können drei verschiedene Fälle auftreten:

1. Enthält ein `SET` ursprünglich zwei Konstituenten und soll eine Konstituente gelöscht werden, so wird der `SET` aufgelöst. Die übriggebliebene Konstituente wird mit seinen Attributen in die Vaterphrase bzw. -`SET` eingefügt.

2. Enthält ein `SET` mehr als zwei Konstituenten und ist die zu löschende Konstituente die zuletzt aufgelistete Konstituente im `SET` so wird der Mengenbezeichner um eine Konstituente im Index verschoben. Die vorletzte Konstituente nimmt den Platz der letzten Konstituente in der Reihenfolge im `SET` ein.

3. Enthält das SET mehr als zwei Konstituenten und ist die zu löschende Konstituente nicht das letzte Element im SET dann kann die Konstituente ohne Einschränkung im SET ausgelassen werden.

In einigen Fällen genügt es die thematischen Rollen der Konstituente zu löschen, die zur Erzeugung eines bestimmten Modellelements führen. Bei der Löschung in einer Phrase werden die thematischen Rollen für die jeweilige Konstituente ausgelassen. Thematische Rollen für eine Konstituente zu löschen, der in einem SET vorkommt erfordert ein differenziertes Verhalten. Es können zwei verschiedene Fälle auftreten:

1. Enthält die Konstituente nach der Löschung der thematischen Rollen noch andere weitere Rollen, so wird die Konstituente aus der Menge entfernt und in seine Vaterphrase oder -SET mit den übrigen Rollen eingefügt.

2. Enthält die Konstituente nach der Löschung der thematichen Rollen keine weiteren Rollen mehr, so wird die Konstituente aus dem SET entfernt.

Je nach Typ des zu löschenden Modellelements kann es sein, dass das Modellelement noch Kindelemente besitzt. In dem Fall müssen die Kindelemente vor dem eigentlichen Element gelöscht werden. Dieser Prozess wird rekursiv für die Kindelemente wiederholt. Das genaue Löschverhalten hängt also vom Typ des zu löschenden Modellelements ab. Das Implementierungsverhalten wird in den folgenden Kapiteln besprochen. Eine Beispielspezifikation soll dabei das Verständnis für die Implementierung von Löschungen unterstützen, siehe Auszug 18.

```
1 [ {Boots Sneakers Sandals AND Mocassins}|FIN #are Shoes|FIC ].
2 [ Shoemaker|AG repair|ACT $black @Shoes|PAT ].
3 [ Shoemaker|AG produce|ACT $leathern @Shoes|OPUSP ].
4 [ @Shoemaker|POSS #has *20 Tools|HAB ].
```

Auszug 18: Ursprüngliche SAL_{E}-annotierte Spezifikation

Die Spezifikation wurde bereits in Unterunterabschnitt 4.4.1 besprochen. Die Überführung in das UML-Modell, siehe Abbildung 10 ist identisch zu Abbildung 22.

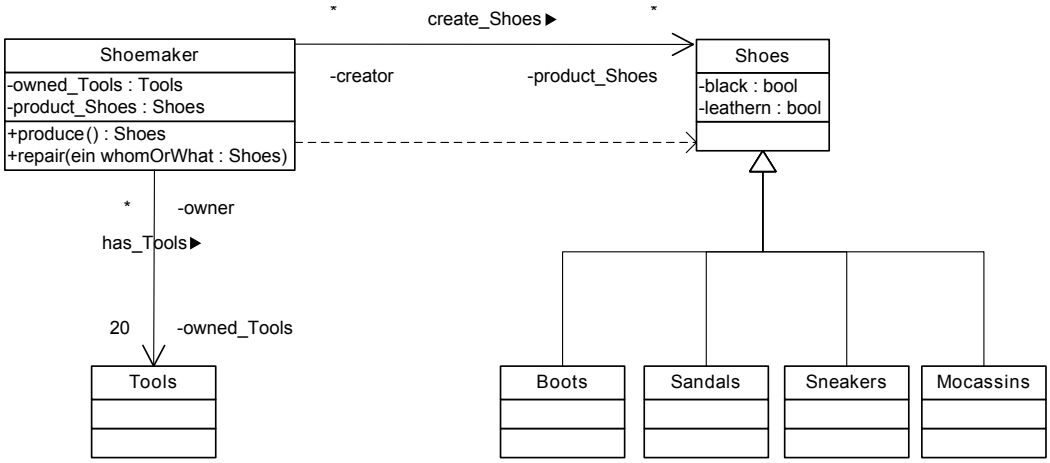

Abbildung 22: Initiales von SAL$_e$ MX erzeugtes Klassendiagramm.

Klassen löschen. Das Löschen einer Klasse aus dem strukturellen Klassendiagramm bedeutet das komplette Verschwinden aus der Struktur des Modells. Wird ein Element aus der Modellstruktur gelöscht muss jedes Vorkommen in der Spezifikation ebenfalls gelöscht werden. Würde die Konstituente in der Spezifikation bestehen dann gebe es dafür wieder ein abgebildetes Element in der Zielmenge, dem UML-Modell. Das Löschen einer Klasse führt zum Löschen aller *verfolgbaren* Konstituenten aus der Quellmenge, dem SAL_{E}-Graphen.

Enthält eine Klasse Kindelemente wie Attribute und Operationen, so werden die Kindelemente zuerst gelöscht bevor die Klasse gelöscht wird. Die Implementierung prüft die Klasse auf mögliche Kindelemente und löscht diese.

Wie eine Konstituente aus dem SAL_{E}-Graphen gelöscht wird, ist in der Einleitung ausführlich beschrieben. Die Löschung von Klassenelementen demonstrieren wir am vorgestellten Beispielmodell.

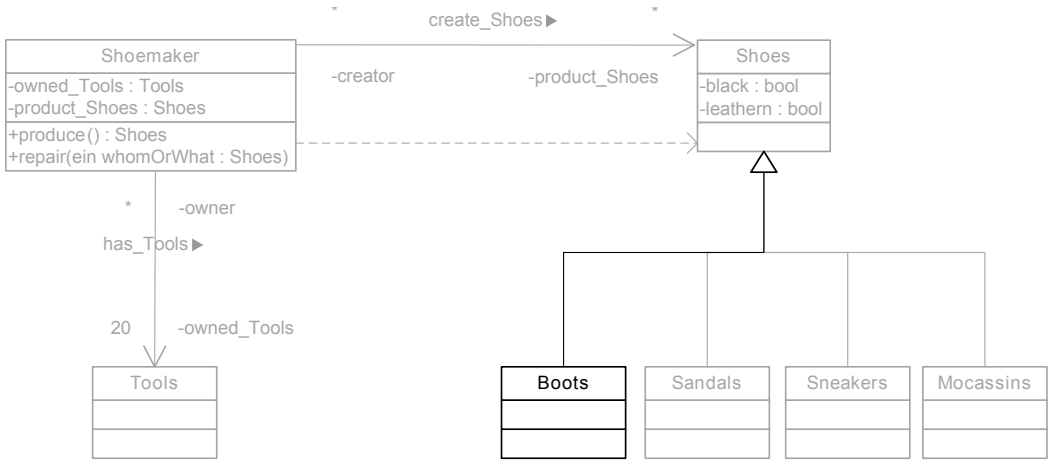

Abbildung 23: Lösche Klassen

Im Modell wollen wir die Klasse *Boots* löschen, siehe Abbildung 23. Der `Deletion`-Protokollknoten, siehe Auszug 19 beschreibt die Löschinstruktionen für REFS.

```
<!-- Delete class 'Boots' -->
<Deletion>
   <xmi:id>G-114</xmi:id>
</Deletion>
```

Auszug 19: Auszug aus dem Unterschiedsprotokoll: Lösche die Klasse Boots. Die Klasse wird über ihren eindeutigen Bezeichner `xmi:id` referenziert.

Attribute löschen. Attribute der Quellmenge werden in Eigenschaften vom Typ *Boolean* abgebildet. Das Löschen von Attributen der Zielmenge wird daher nur für Eigenschaften des Typs *Boolean* unterstützt. Alle anderen typisierten Eigenschaften modellieren die Navigierbarkeit von Assoziationen. Sie können nicht in die Quellmenge verfolgt werden. Sie werden automatisch erzeugt, um navigierbare Assoziationen darzustellen und haben keinen Quellkonstituenten.

Da Attribute keine Kindelemente besitzen, wird der Fall der rekursiven Kindlöschung nicht beachtet. Das Löschen von Attributen wird an Hand des folgenden Beispiels vertieft. Im Modell wollen wir ein Attribut löschen und die Auswirkungen auf die Spezifikation verdeutlichen.

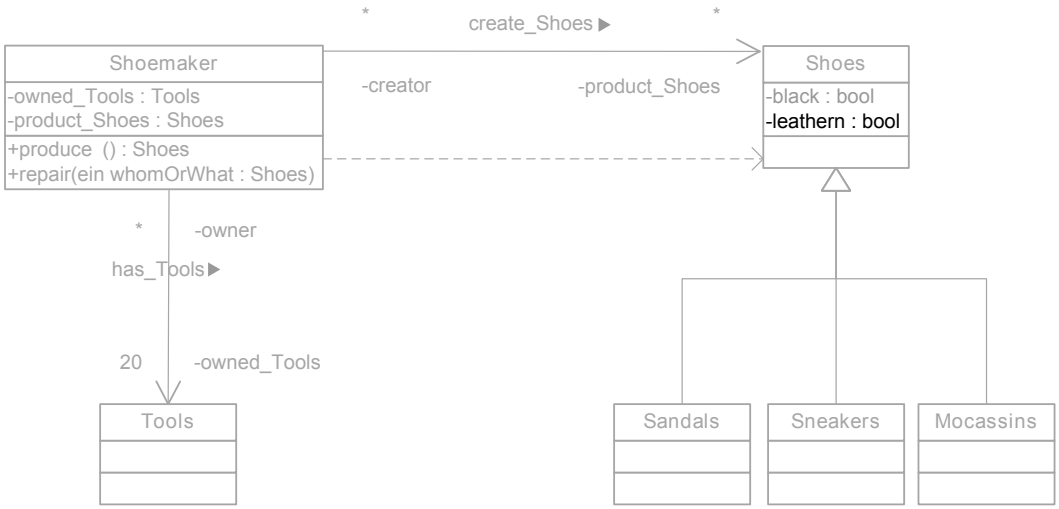

Abbildung 24: Lösche Attribut

Das Attribut *leathern* in der Klasse *Shoes* soll gelöscht werden, siehe Abbildung 24. Der `Deletion`-Protokollknoten beschreibt die Modelländerung, siehe Auszug 20.

```
<!-- Delete property 'leathern' in class 'Shoes' -->
<Deletion>
   <xmi:id>G-100</xmi:id>
</Deletion>
```

Auszug 20: Auszug aus dem Unterschiedsprotokoll: Lösche das Attribut *leathern* in Klasse *Shoes*. Das Attribut wird über den eindeutigen Bezeichner `xmi:id` referenziert.

Operationen löschen. Wird eine Operation im Modell gelöscht, muss die zugehörige Konstituente in der Quellmenge gefunden werden. Die Bestimmung der Konstituente ist nicht trivial. Die Konstituente kann einmal in einem Kontext einer Phrase mit verschiedenen Parametern vorkommen und eine Operation o1 im Modell erzeugen. Die selbe Konstituente kann durch eine Referenz in der SAL_{E}-Quellmenge in einer späteren Phrase mit anderen Parametern vorkommen und zu einer Operation o2 überführt werden. Strenge Beachtung bei der Löschung einer Operation sind nötig.

1. Die Signatur der Operation führt zur Auffindung der gesuchten Operation. Die Signatur besteht aus der Klasse in der sich die Operation befindet. Weiterhin sind die Parametertypen der Operation notwendig, um die genauen Konstituente ausfindig zu machen. Doppelte Operationen in der selben Klasse mit der selben Parameteranzahl werden von SAL_e MX nicht erzeugt.

2. Ist die Konstituente gefunden, darf sie nicht auf Anhieb gelöscht werden. Alle Sub-Rollen der thematischen Rolle `METHODROLE` im Kontext der inhabenden Phrase bzw. Menge werden gelöscht. Sollte die Konstituente keine weiteren Rollen mehr besitzen, kann sie gelöscht werden.

Außerdem muss beachtet werden, dass Operationen Kindelemente, nämlich ihre Parameter, besitzen. Die Löschung von Parametern wird im nächsten Abschnitt beschrieben.

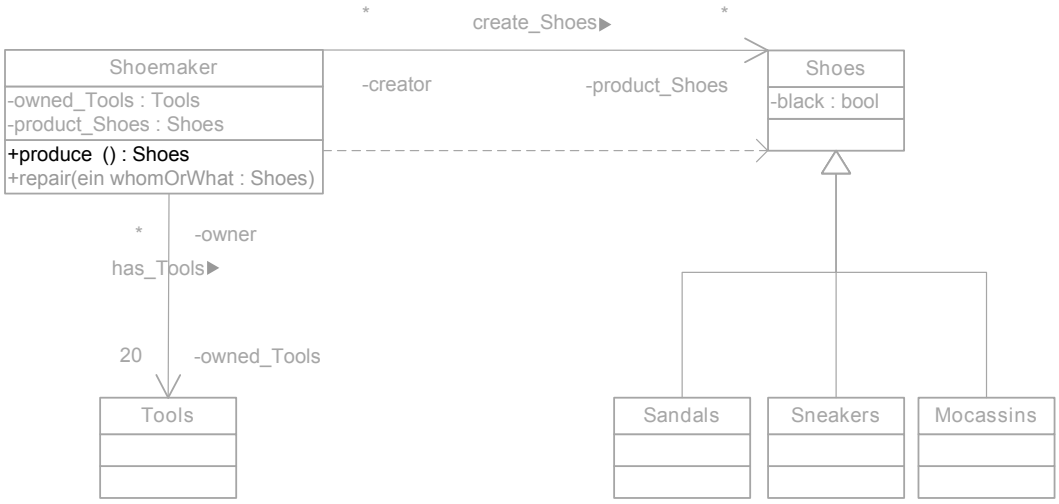

Abbildung 25: Lösche Operation

In Abbildung 25 wird die Operation *produce* gelöscht. Der Protokollknoten, siehe Auszug 21, beschreibt die Modelländerung und definiert die Löschinstruktionen für REFS.

```
<!-- Delete operation 'produce' in class 'Shoemaker' -->
<Deletion>
   <xmi:id>G-90</xmi:id>
</Deletion>
```

Auszug 21: Auszug aus dem Unterschiedsprotokoll: Lösche die Operation *produce* in Klasse *Shoemaker*.

Parameter löschen. Um den Parameter einer Operation zu löschen und auf die textuelle Spezifikation abzugleichen, ist eine genaue Feststellung der Konstituente notwendig. Die Bestimmung der Konstituente ist nicht trivial. Die Konstituente kann mehrfach in verschiedenen Phrasen durch Referenzierungen vorkommen. Um die Konstituente zu bestimmen wird, ähnlich wie beim Löschen einer Operation, die Operationssignatur bestimmt. Die Phrase in der der Parameter zusammen mit dem Operationsinhaber, der Operation selbst und den anderen Parametern vorkommt ist eindeutig bestimmt. Ist die Phrase bestimmt, wird der Parameter aus der Phrase entfernt.

Dabei werden alle thematischen Subrollen der Rolle `ADJUNCTROLE`, die in dem selben Kontext (`SETID`) wie die Operationssignatur vorkommen, gelöscht. Hat der Parameter dann keine weiteren thematischen Rollen mehr, wird die Konstituente gelöscht.

Da der Parameter keine Kindelemente enthalten kann, wird keine Prüfung auf Kindelemente durchgeführt. In folgendem Modell wird der Parameter einer Operation beispielhaft entfernt, um die Implementierung zu verdeutlichen.

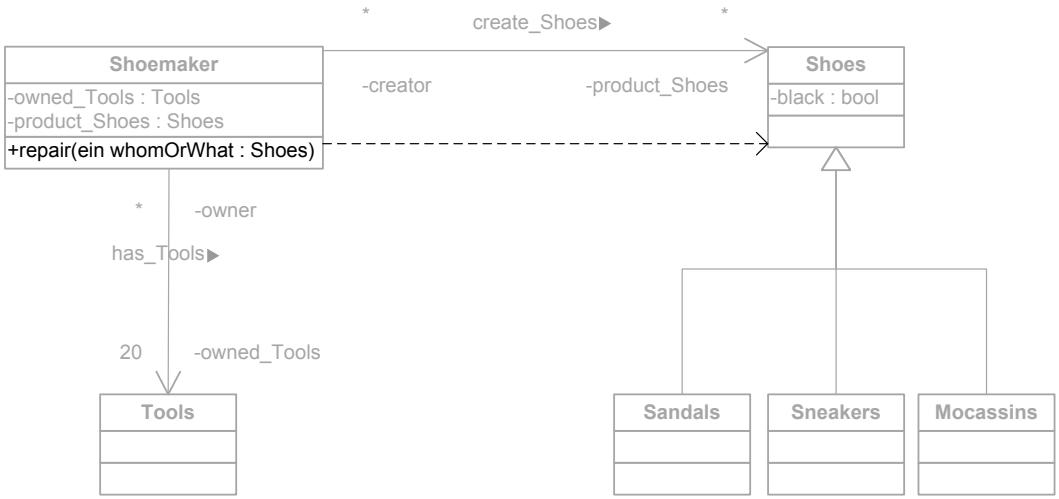

Abbildung 26: Lösche Parameter

Das Modell enthält eine Operation *repair* mit dem Parameter *whomOrWhat*, die gelöscht wird. Das Unterschiedsprotokoll beschreibt diese Änderung am Modell.

```
1 <!-- Delete parameter 'whomOrWhat' in operation 'repair' -->
2 <Deletion>
3    <xmi:id>G-74</xmi:id>
4 </Deletion>
```

Auszug 22: Auszug aus dem Unterschiedsprotokoll: Lösche den Parameter 'whomOrWhat' in Operation *produce* in Klasse *Shoemaker*. Der Parameter wird über den eindeutigen Bezeichner `xmi:id` referenziert.

Der `Deletion`-Protokollknoten definiert die Löschung des Parameters. REFS verarbeitet diesen Protokollknoten und koppelt die Änderung in die Spezifikation zurück. Das Ergebnis kann in Auszug 25 mit der originalen Spezifikation verglichen werden.

Assoziationen löschen. Assoziationen stellen binäre Verbindungen zwischen Objekten der Klassen dar. Es kann vorkommen, dass solche Verbindungen falsch modelliert wurden oder mehrfach deklariert wurden. Der Systemanalyst muss diese Assoziationen dann gegebenfalls löschen.

REFS sucht nach dem gemeinsamen Kontext der zwei verbundenen Klassen. Werden beide Klassen in einem gemeinsamen Kontext gefunden werden sie nach ihren thematischen Rollen durchsucht. Thematische Rollenpaare, die Assoziation erzeugen, sind `POSS-HAB`, `THE-THEII`, `QUAL-QUALII`, `OPUS-OBJECTROLE` und alle Verbindungen zwischen `STAT` und `OBJECTROLE`.

Die Implementierung durchsucht die Phrase in der beide Konstituenten mit demselben Kontext vorkommen und löscht die gefundenen Rollenpaare.

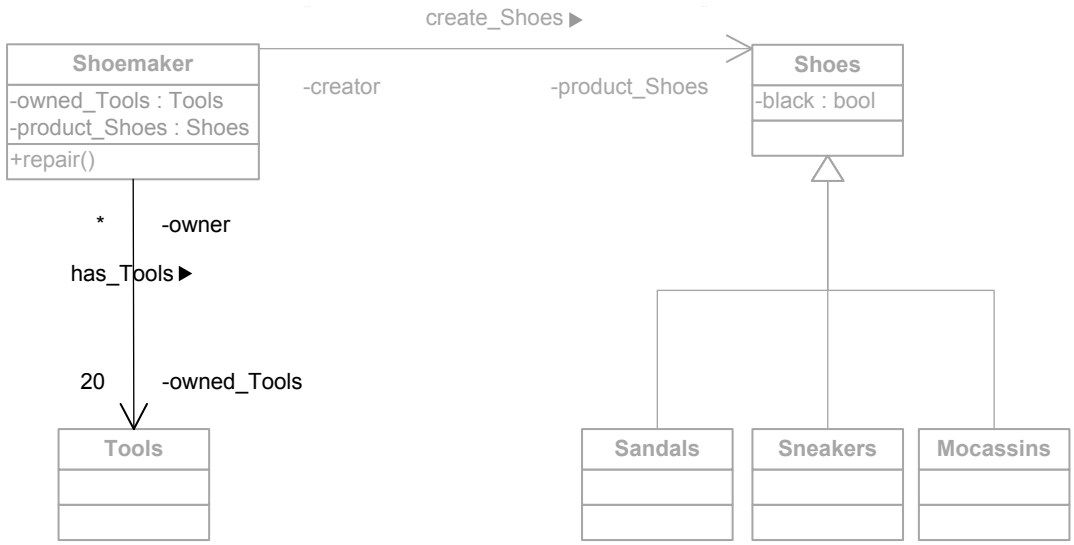

Abbildung 27: Lösche Assoziation

Im gezeigten Modell, siehe Abbildung 27, wollen wir die Verbindung zwischen dem Schuhmacher und seinen Werkzeugen löschen. Das Unterschiedsprotokollknoten fasst die Löschung in einem Protokollknoten zusammen, siehe Auszug 23.

```
<!-- Delete association 'has_Tools' -->
<Deletion>
   <xmi:id>G-78</xmi:id>
</Deletion>
```

Auszug 23: Auszug aus dem Unterschiedsprotokoll: Lösche die Assoziation zwischen *Shoemaker* und *Tools*.

REFS durchläuft das Protokoll. Die beiden Konstituenten der Klassen *Shoemaker* und *Tools* werden in einem gemeinsamen Kontext im SAL_{E}-Graphen gesucht. Sie kommen beide in der letzten Phrase des Dokuments bzw. Graphen vor. Die Phrase wird nach den

bekannten vordefinierten thematischen Rollen durchsucht um sie dann aus der Phrase zu löschen.

In diesem Fall wird das Rollenpaar `POSS-HAB` gefunden. Sie werden in einem anschließenden Schritt aus der Phrase gelöscht. Das Resultat kann mit Auszug 25 verglichen werden. Die beiden Konstituenten kommen dort nicht mehr vor, da die Rollen gelöscht wurden und es keine andere Verknüpfung durch Rollen mehr gab.

Vererbungsbeziehungen löschen. Über Vererbungsbeziehungen sind zwei Klassen miteinander verbunden. Eine Subklasse erweitert eine generalisierte Superklasse.

In der SAL$_E$-Annotation würde das die Auszeichnung zweier Konstituenten durch eines der thematischen Rollenpaare `FIN-FIC` oder `OMN-PARS` bedeuten. Eine Konstituente mit `FIN` bzw. `PARS` stellt die Unterklasse des Konstituenten mit `FIC` bzw. `OMN` dar. Wir wollen diese Art der Generalisierungsabbildung als direkte Generalisierung bezeichnen. Eine Außnahme bilden die thematischen Rollenpaare mit `CONT-CONTII` und `SUB-SUBII`. Hier werden zwei Konstituenten als gemeinsame Subklassen einer unbekannten Superklasse abgebildet. Diese Art der Abbildung von Vererbungen bezeichnen wir im Folgenden als indirekte Vererbung.

Wird eine Vererbungsbeziehung aus dem Modell gelöscht, bedeutet dies die Löschung der ausgezeichneten thematischen Rollen der betreffenden Konstituenten aus der SAL$_E$-Quellmenge. Im Fall einer direkten Vererbungsabbildung wird der gemeinsame Kontext der Sub- und Superklasse bestimmt, um dann die thematische Rolle der Konstituente für die Subklasse aus dem Graphen zu entfernen. Die Rolle der Konstituente für die Superklasse wird temporär beibehalten. Sollte die Klasse andere Subklassen enthalten, werden die Beziehungen nicht entfernt und bleiben im Modell erhalten. Findet REFS keine anderen Subklassen, wird auch die thematische Rolle für die Superklasse entfernt um dem Modell und der semantischen Bedeutung der Spezifikation gerecht zu werden. Im Folgenden betrachten wir ein Beispiel an unserem Modell.

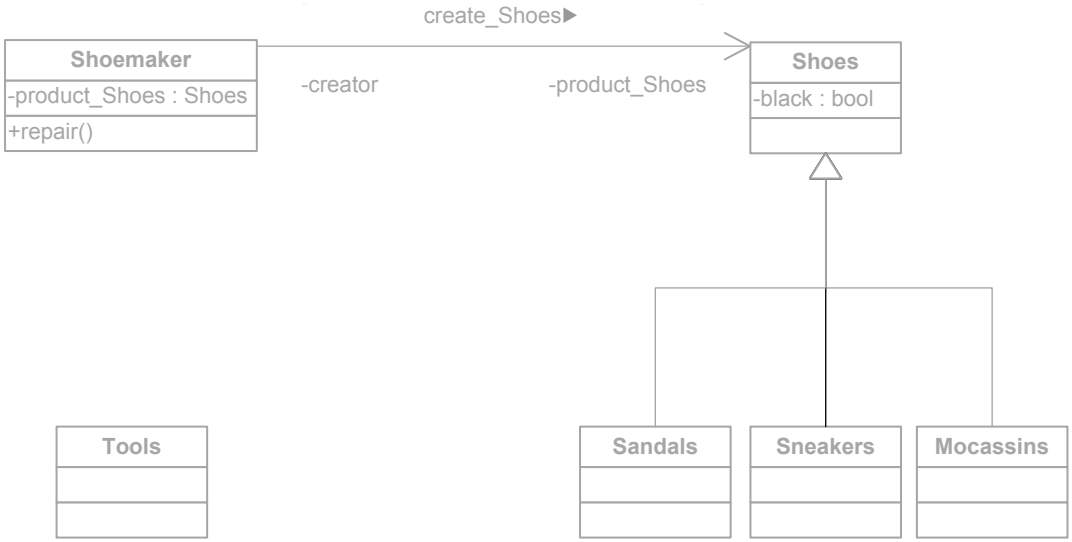

Abbildung 28: Lösche Vererbung

Die Vererbung zwischen der Subklasse *Boots* und seiner Superklasse *Shoes* wird gelöscht, siehe Abbildung 28.

REFS verarbeitet den `Deletion`-Protokollknoten, siehe Auszug 24, für die Assoziation und findet die Konstituenten *Sandals*, *Mocassins*, *Boots* und *Shoes* an einem gemeinsamen Kontextknoten. REFS sucht nach der Phrase, dem der Kontextknoten angehört.

```
<!-- Delete generalization from
       superclass 'Shoes' to subclass 'Sneakers' -->
<Deletion>
   <xmi:id>G-71</xmi:id>
</Deletion>
```

Auszug 24: Auszug aus dem Unterschiedsprotokoll: Lösche die Assoziation zwischen den Stiefel und Schuhen.

Ist die Phrase gefunden, werden die betreffenden Konstituenten nach den thematischen Rollenpaaren abgesucht, um sie dann zunächst aus der Konstituente der Subklasse zu entfernen. Da weitere Subklassen an der Superklasse verknüpft sind, wird die Rolle für die Superklasse beibehalten, um die restlichen, verbleibenden Generalisierungen beizubehalten.

Das Resultat der Löschung kann in Auszug 25, Zeile 1 betrachtet werden. Die Rolle ist für die Konstituente *Boots* entfernt worden. Da keine weiteren thematischen Rollen an der Konstituente hängen, wurde sie komplett aus der Phrase entfernt.

Die abgeglichene Spezifikation mit ihrer Annotation. Die ursprüngliche SAL_{E}-annotierte Spezifikation wurde um die gelöschten Modellelemente abgeglichen.

```
[ {Sandals AND Mocassins}|FIN #are Shoes|FIC ].
[  Shoemaker|AG repair|ACT ].
[ Shoemaker|AG ].
```

Auszug 25: Spezifikation nach Rückkopplung gelöschter Modellelemente

4.5. Dekompilieren des SALe-Graphen

Alle Änderungen von REFS geschehen an der Graphrepräsentation des SAL_E-annotierten Textes. Der Graph allerdings ist nicht lesbar und muss in eine textuelle Spezifikation ausgegeben werden. Der SAL_E-Dekompilierer gibt die serialisierte Version des Graphen als Text aus.

Der SAL_E-Dekompilierer kann den Graph wahlweise entweder in einen SAL_E-annotierten Text oder in einen rein natürlich-sprachlichen Text ohne SAL_E-Annotation ausgeben. Die Wahl der Ausgabe kann in der Transformationssteuerung (*.grs) gewählt werden. Die Ausgabe wird in eine Datei exportiert und gespeichert.

Der Dekompilierer sucht den Wurzelknoten *root* als Einstiegspunkt für die Ausgabe. Ist die Wurzel gefunden, wird die erste, mit der Wurzel verbundene Phrase, ausgegeben. Die Ausgabe beginnt mit der ersten Phrase. Gibt es keine erste Phrase, wird die weitere Ausgabe abgebrochen und der Dekompilierer beendet seine Aufgabe.

Zur Ausgabe benötigen wir einen Zähler um die serialisierten Konstituenten des Graphs in der korrekten Reihenfolge auszugeben und zum anderen einen anderen Zähler für eine wohlgeformte Einrückung des ausgegebenen Texts. Nach jeder vorkommenden inneren Phrase wird die Einrückung eines Tabulators um eins erhöht um innere Phrase hervorzuheben. Endet die innere Phrase wird der Zähler dekrementiert.

```
1  /* Outputs a given phrase. */
2  rule EmitPhrase ( phrase:ORIGINAL_PHRASE, poser:AUX_Counter, tab:AUX_Counter )
3  {
4     if
5     { phrase.VISITED == false; }
6
7     modify
8     {
9        /* Pretty formatting. */
10       exec ( EmitLineBreakAndTabulator (poser, tab) );
11
12       /* Emit the phrase. */
13       exec ( EmitComment (phrase, poser, tab) );
14       exec ( EmitWord (phrase, poser, tab) );
15       exec ( EmitSet (phrase, poser, tab) );
16       exec ( EmitMultiplicity (phrase, poser, tab) );
17       exec ( EmitAttribute (phrase, poser, tab) );
18       exec ( EmitReference (phrase, poser, tab) );
19       exec ( EmitInnerPhraseWithoutHead (phrase, poser, tab) );
20       exec ( EmitInnerPhraseWithHead (phrase, poser, tab) );
21       exec ( EmitPunctuation (phrase, poser) );
22    }
23 }
```

Auszug 26: Graphtransformationsregel: Sie gibt die Elemente einer Phrase aus.

Die Ausgabe einer Phrase wird durch die Transformationsregel in Auszug 26 ausge-

führt. Sie versucht an jeder gegebenen Zählerposition (*poser*), die jeweilige passende Konstituente auszugeben. Dabei wird jede Regel zur Ausgabe eines besonderen Konstituententyps aufgerufen. Nach dem Aufruf einer Regel wird auf die korrekte Position und den passenden Typ der übergebenen Konstituente getestet. Ist der Typ der Konstituente unpassend oder die Position falsch wird die nächste Regel in der Phrasenausgabe geschaltet. Dabei wird solange zur nächsten Regel weitergeschaltet bis eine der Regeln greift. Hat eine der Regeln gegriffen wird der Positionszähler um eins erhöht und der Name der Konstituente ausgegeben. Greift keine der Regeln mehr, so gibt es keine weiteren Konstituenten mehr und die Phrase ist beendet.

Weitere Phrasen werden rekursiv in der Go2NextPhrase-Regel übergeben und abgearbeitet, siehe Auszug 27. Die Transformationsregel akzeptiert eine bereits abgearbeitete Phrase als Eingabeargument. Die Regel sucht nach einer neuen, an der abgearbeiteten Phrase hängenden Phrase. Diese Phrase ist der neue Ausgangspunkt für die Transformationsregel.

Zuerst werden neue Positionszähler (*poser*) und Einrückungszähler (*tab*) angelegt, siehe Zeile 11 und 12, die beide auf den Anfangswert 1 initialisiert werden, siehe Zeilen 14 bis 15. In Zeile 20 und 21 geben die Regeln EmitLB und EmitCR eine öffnende eckige Klammer für die SAL_{E}-Syntax mit einem Zeileneinschub, für die wohlgeformte Einrückung, aus. Die anschließende Regel *EmitPhrase* gibt die einzelnen Phrasenelemente der übergebenen Phrase aus, wie sie vorher bereits besprochen wurden.

Folgende *EmitCR* und *EmitRB* fügen neue Zeileneinschübe ein und schließen die offene Phrase mit einer eckigen abschließenden Klammer ab, siehe Zeilen 24 bis 28. Bevor die Regel endet, werden die Zähler (*poser* und *tab*) gelöscht um Ressourcen zu schonen. Die Regel endet mit dem rekursiven Aufruf der eigenen Regel. Die im Regelauruf neue, bereits abgearbeitete Phrase wird an die *Go2NextPhrase*-Regel übergeben um sie dann wieder verarbeiten. Der Dekompilierer endet, wenn es keine Phrasen mehr gibt die verarbeitet werden können.

```
/* Process a given phrase and works recursively. */
rule Go2NextPhrase ( phrase:ORIGINAL_PHRASE )
{
   phrase -next:NEXT-> nextPhrase : ORIGINAL_PHRASE;

   if
   { next.VISITED == false; }

   modify
   {
      poser : AUX_Counter;
      tab : AUX_Counter;

      eval
      { tab.value = 1; }

      exec ( MarkNext (next) );

      // Emit phrase.
      exec ( EmitLB );
      exec ( EmitCR );
      exec ( EmitTabs( tab ) ); // Pretty printing.
      exec ( EmitPhrase ( nextPhrase, poser, tab ) );
      exec ( EmitCR );
      exec ( EmitRB );
      exec ( EmitDot );
      exec ( EmitCR );
      exec ( EmitCR );

      // Mark phrase if finished.
      exec ( MarkConstituent (nextPhrase) );

      // Remove AUX node, because it is not used any longer.
      exec ( RemoveAuxNode (poser) );
      exec ( RemoveAuxNode (tab) );

      // Proceed next phrase.
      exec ( Go2NextPhrase ( nextPhrase ) );
   }
}
```

Auszug 27: Arbeitet eine Phrase ab und läuft zur nächsten.

5. Evaluation

In diesem Abschnitt werden die Ergebnisse der Implementierung beschrieben. Eine Evaluierung der Ergebnisse ist in Unterabschnitt 5.1, Unterabschnitt 5.2, Unterabschnitt 5.3 und Unterabschnitt 5.4 zu finden.

5.1. Fallbeispiel 1 (Creations)

Das erste Fallbeispiel behandelt die Abgleichung neuer Modellelemente. Der ursprünglichen Spezifikation, siehe Auszug 28, werden neue Modellelemente hinzugefügt. Alle hinzugefügten Modellelemente wurden bereits in Unterunterabschnitt 4.4.1 ausführlich besprochen.

```
Boots Sneakers Sandals and Mocassins are Shoes.
Shoemaker repair black Shoes.
Shoemaker produce leathern Shoes.
Shoemaker has 20 Tools.
```

Auszug 28: Ursprüngliche Spezifikation ohne SAL_E-Annotation.

Die Modellrückkopplung durch REFS gleicht die Spezifikation ab und ergibt das neue abgeglichene Dokument in Auszug 29. Die Dokumente enthalten keine SAL_E-Annotierung. Beide Dokumente, die originale und die abgeglichene Spezifikation, können nun in einem Textbearbeitungsprogramm mit Vergleichsfunktion betrachtet werden. Wir wählen dafür Microsoft Office Word 2007, das eine integrierte Vergleichsfunktion anbietet.

```
Boots Sneakers Sandals and Mocassins are Shoes.
Shoemaker repair black Shoes.
Shoemaker produce leathern Shoes.
Shoemaker has 20 Tools.
There is/are Person.
There is/are Flip-Flop.
Shoes worn_by.
Shoes worn_by Person.
iron_made Tools.
Flip-Flop is/are Sandals.
The owner Person has_Shoes the owned_Shoes Shoes.
```

Auszug 29: Abgeglichene Spezifikation (Creations).

In Abbildung 29 sehen wir einen Bildausschnitt des Dokumentvergleichs. Die rechte Spalte zeigt die beiden ausgewählten Spezifikation übereinander. Das erste zeigt die Originalspezifikation. Im zweiten darunterliegenden Dokument sehen wir die von REFS abgeglichene Version der Spezifikation.

Die mittlere Spalte zeigt das Vergleichsdokument. Hier wurden beide Dokumente fusioniert. Der in schwarz gehaltene Textabschnitt zeigt die originale Spezifikation. Der violett und unterstrichene Textabschnitt zeigt die Änderungen gegenüber der ursprünglichen Spezifikation.

Hier wurden insgesamt sieben neue Phrasen eingefügt. Der Kunde und Analyst sehen alle Änderungen auf einen Blick. Die zusätzliche linke Spalte listet alle Änderungen auf, ohne die originale Spezifikation zu zeigen.

Die dritte Phrase wurde erzeugt als eine neue Operation *worn_by* in die Klasse *Shoes* des Modells eingefügt wurde. Der neue Parameter *Person* in der Operation *worn_by* der Klasse *Shoes* erzeugt die vierte Phrase. Hier zeigen sich Ansätze zur Verbesserung. Die

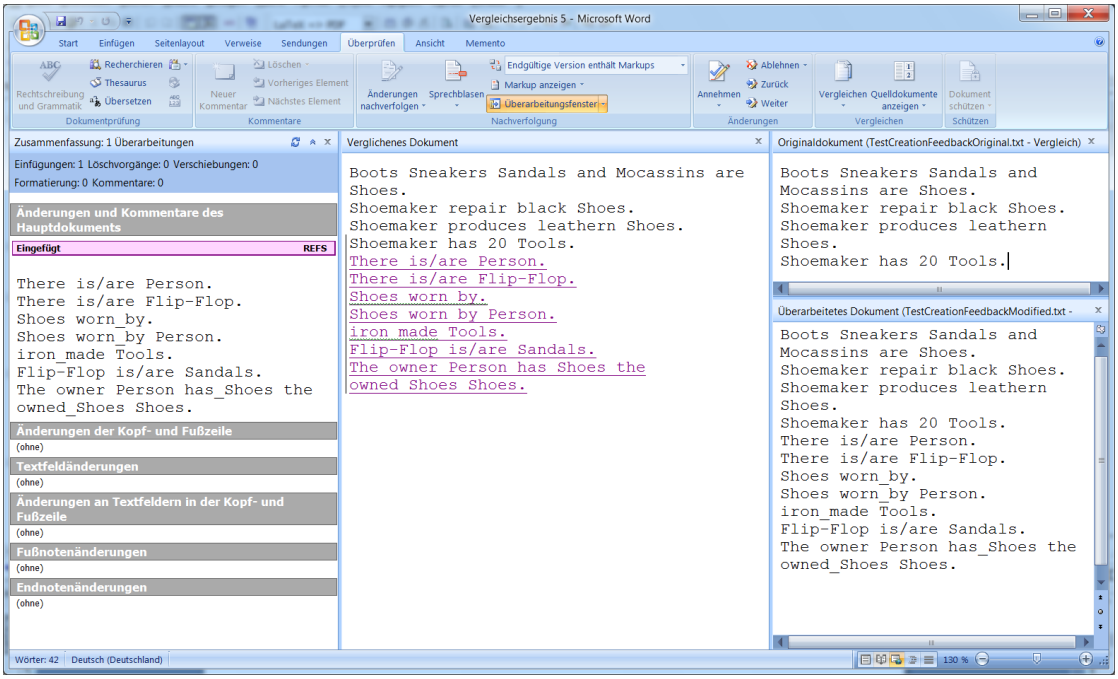

Abbildung 29: Ein Vergleich beider Dokumente (Creations).

dritte und vierte eingefügte Phrase sind sehr ähnlich. Die dritte Phrase ist redundant, da die vierte Phrase bereits die dritte Phrase aufsummiert.

Eine weitere Verbesserung für das Hinzufügen von Elemente ist in der sechsten eingefügten Phrase sichtbar. Die Generalisierung des Typs *Flip-Flop* und seines Supertyps *Sandals* beschreibt den Typ *Flip-Flop*. Die zweite eingefügte Phrase beschreibt, dass es einen *Flip-Flop* gibt. Die Phrase ist redundant, da die sechste Phrase den *Flip-Flop* bereits beschreibt. Das Auflösen von redundanten Phrasen ist in einen Nachbearbeitungsprozess verlagerbar.

5.2. Fallbeispiel 2 (Deletions)

Das zweite Fallbeispiel behandelt die Abgleichung gelöschter Modellelemente. Der ursprünglichen Spezifikation, siehe Auszug 30, werden Modellelemente gelöscht. Alle gelöschten Modellelemente wurden bereits in Abbildung 4.4.3 ausführlich besprochen.

```
Boots Sneakers Sandals and Mocassins are Shoes.
Shoemaker repair black Shoes.
Shoemaker produce leathern Shoes.
Shoemaker has 20 Tools.
```

Auszug 30: Ursprüngliche Spezifikation ohne SAL_E-Annotation.

Wieder gleichen wir mit REFS die Spezifikation ab und erhalten das neue Dokument in Auszug 31. Die Dokumente enthalten keine SAL_E-Annotierung. Beide Dokumente,

die originale und abgeglichene Spezifikation, können wieder in Microsoft Word mit der integrierten Vergleichsfunktion betrachtet werden.

```
Sandals and Mocassins are Shoes.
Shoemaker repair.
Shoemaker.
```

Auszug 31: Abgeglichene Spezifikation (Deletions).

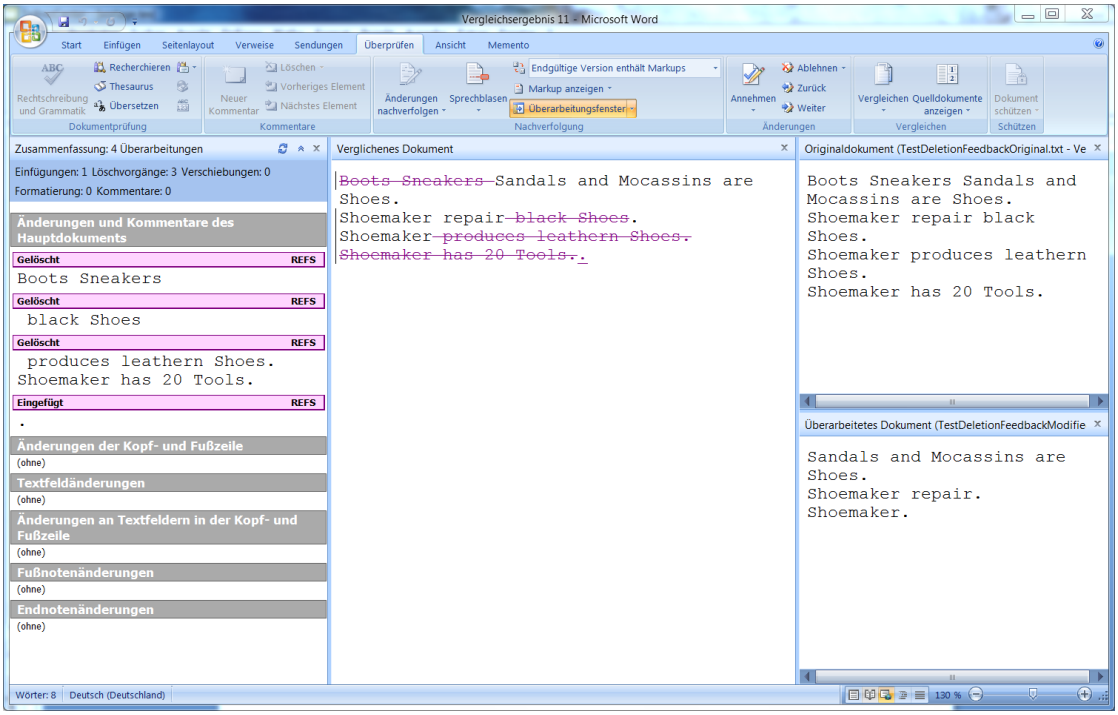

Abbildung 30: Ein Vergleich beider Dokumente

Insgesamt wurde eine komplette Phrase ausgelassen und mehrere Konstituenten gelöscht. Der Kunde und Analyst sehen alle Änderungen auf einen Blick. Die zusätzliche linke Spalte listet alle Änderungen auf, ohne die originale Spezifikation zu zeigen.

Aus der Menge in der ersten Phrase wurden die Konstituenten*Boots* und *Sneakers* entfernt. In der zweiten Phrase wurde die Konstituente *Shoes* entfernt, um die Löschung des Parameters *Shoes* in der Operation *repair* rückzukoppeln. Die Löschung der Konstituente hat auch die implizite Löschung ihres Attributs *black* zur Folge. Das Ergebnis der Rückkopplung zeigt, dass der *Shoemaker* nur noch repariert. Die Angabe, was genau repariert wird, wurde entfernt. Aus der nächsten Phrase wurden alle Konstituenten nach dem *Shoemaker* ausgelassen. Die Löschung der Operation *produce* hat zur Folge, dass auch der in der Operation enthaltene Parameter *Shoes* gelöscht wird. Damit wird auch sein Attribut *leathern* entfernt. Die letzte Phrase wurde komplett ausgelassen. Die navigierbare Assoziation wurde aus dem Modell entfernt. Damit wurden die semanti-

schen Rollen die für die Assoziation verantwortlich waren ebenfalls ausgelassen. Durch die Auslassung der Rollen in der Phrase hatten der *Shoemaker* und *Tools* keine weiteren an die Phrase anknüpfenden Rollen. Beide Konstituenten *Shoemaker* und *Tools* verlieren damit ihren Anspruch auf Existenz in der Phrase.

Die Semantik aus der Löschung der Modellelemente bleibt in der Spezifikation erhalten. Ähnlich wie in Unterabschnitt 5.1 angesprochen, treten hier Redundanzen auf, die durch einen Nachbearbeitugnsschritt vermieden werden können. Der *Shoemaker* tritt in der dritten Phrase als einzige Konstituente auf, obwohl in der vorherigen Phrase der *Shoemaker* Verwendung findet.

5.3. Fallbeispiel 3 (Updates)

Das nächste Fallbeispiel behandelt die Abgleichung geänderter Modellelemente. Der ursprünglichen Spezifikation, siehe Auszug 32, werden Modellelemente aktualisiert. Alle aktualisierten Modellelemente wurden bereits in Unterunterabschnitt 4.4.2 ausführlich besprochen.

```
Boots Sneakers and Mocassins are Shoes.
Shoemaker repair Shoes.
Shoemaker has 20 Tools.
```

Auszug 32: Ursprüngliche Spezifikation ohne SAL$_E$-Annotation.

Die Abgleichung der Spezifikation durch REFS ergibt das neue Dokument in Auszug 33. Die Dokumente enthalten keine SAL$_E$-Annotierung. Beide Dokumente, die originale und abgeglichene Spezifikation, können wieder in Microsoft Word mit der integrierten Vergleichsfunktion betrachtet werden.

```
Boots Sneakers and Mocassins are Shoes.
Shoemaker make Boots.
Shoemaker has 99 Tools.
```

Auszug 33: Abgeglichene Spezifikation (Updates).

Im Ähnlichkeitsvergleich mit Word wird eine Aktualisierung als Löschung des alten Elements und dem Hinzufügen eines neuen Elements interpretiert. Der Kunde und Analyst sehen die Änderungen auf einen Blick. Die zusätzliche linke Spalte listet alle Änderungen auf, ohne die Originalspezifikation zu zeigen.

Die erste Phrase bleibt unverändert, da hier keine Aktualisierungen am Modell vorgenommen wurden. Die zweite Phrase enthält zwei Aktualisierungen. Zum einen wird die Operation *repair* umbenannt in *make*, zum anderen wird der Parameter von *Shoes* auf *Boots* umtypisiert. In der letzten Phrase wird der Mengenwert von 20 *Tools* auf 99 erhöht.

Die Semantik und die Struktur bei der Aktualisierung von Modellelementen bleibt in der Spezifikation erhalten. Redundanzen wie sie in den vorherigen Kapiteln angesprochen wurden, entstehen bei Aktualisierungen nicht.

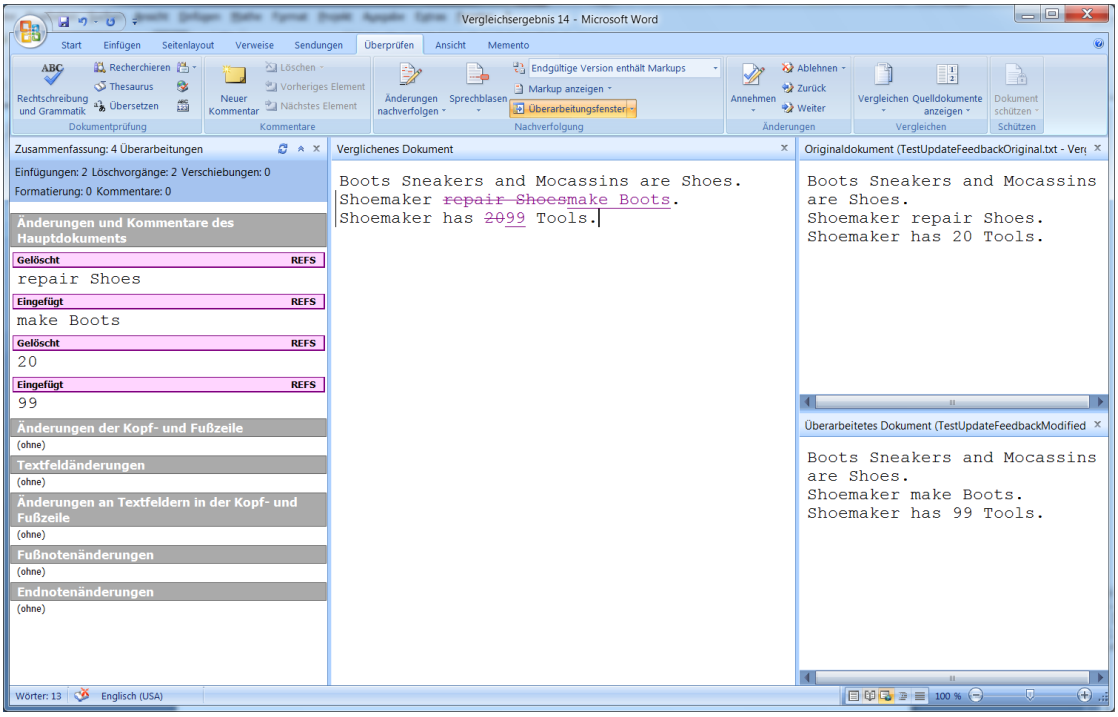

Abbildung 31: Ein Vergleich beider Dokumente

5.4. Fallbeispiel 4 (Kombiniert)

Das letzte Fallbeispiel behandelt die Abgleichung von Modellelementen in einer Kombination von Hinzufügungen, Aktualisierungen und Löschungen von Modellelementen. Der ursprünglichen Spezifikation, siehe Auszug 32, wird jeweils ein Modellelement hinzugefügt, gelöscht und aktualisiert. Die vorkommenden Änderungen der Modellelemente wurden bereits in den vorherigen Kapiteln ausführlich besprochen.

```
Boots Sneakers and Mocassins are Shoes.
Shoemaker repair Shoes.
Shoemaker has 20 Tools.
```

Auszug 34: Ursprüngliche Spezifikation ohne SAL_E-Annotierung.

Die Abgleichung der Spezifikation durch REFS ergibt das neue Dokument in Auszug 35. Die Dokumente enthalten keine SAL_E-Annotierung. Beide Dokumente, die originale und abgeglichene Spezifikation, können wieder in Microsoft Word mit der integrierten Vergleichsfunktion betrachtet werden.

```
Boots and Mocassins are Shoes.
Shoemaker repair Boots.
Shoemaker has 20 Tools.
Shoemaker produce.
```

Auszug 35: Abgeglichene Spezifikation (C/U/D).

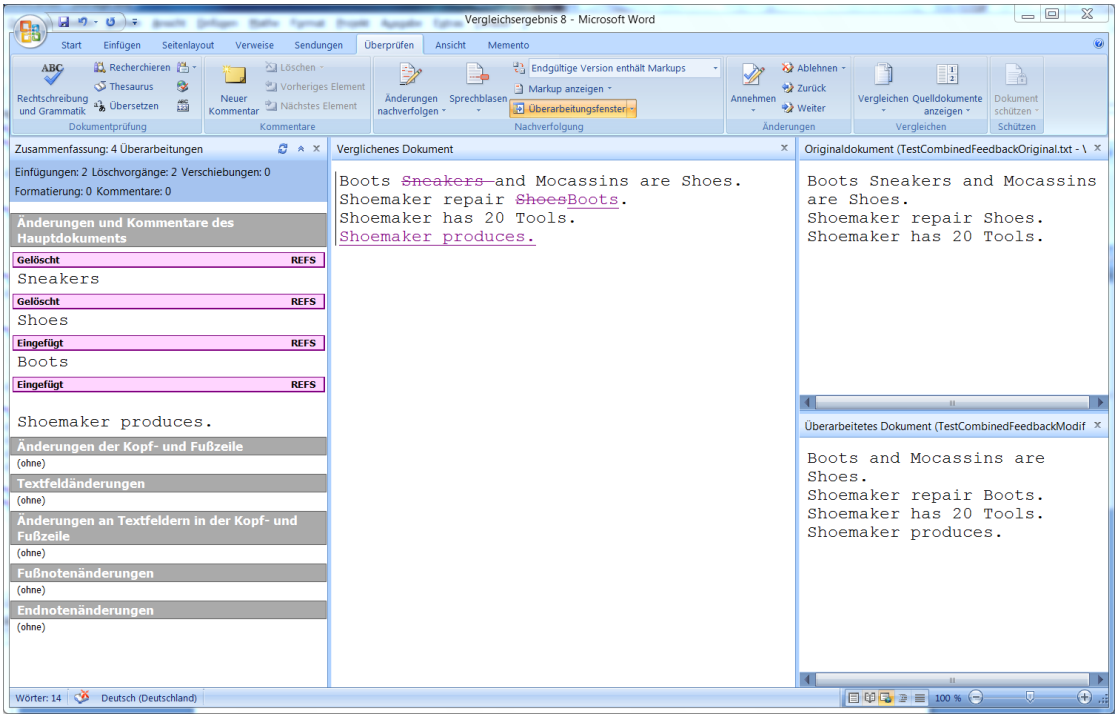

Abbildung 32: Ein Vergleich beider Dokumente

In der ersten Phrase wird die Konstituente *Sneakers* aus der Menge ausgelassen um die Löschung des zugehörigen Klassenelements im Modell wiederzugeben. Die zweite Phrase gibt die Umtypisierung des Parameters in der Operation *repair* von *Shoes* auf *Boots* an. Die letzte Phrase wurde neu hinzugefügt um die neue Operation *produce* in der Klasse *Shoemaker* zu reflektieren.

5.5. Fallbeispiel 5 (Das WHOIS Protokoll)

Ein Auszug des WHOIS Protokolls der Internet Engineering Task Force (IETF) [Dai04] dient als nächstes Beispiel für eine Evaluation. Das WHOIS Protokoll in der SAL_E-annotierten Version, siehe Auszug 36, liegt zusammen mit dem daraus erzeugten UML-Softwaremodell, auf der Homepage [KDGL] zum Runterladen bereit.

```
[ #A WHOIS_server|{AG(l),RECP(l)} listens|ACT(l) #on
  TCP_port_43|PAT(l) #for requests|HAB(l) #from
  WHOIS_client|DON(l) ].

[ [ #The @WHOIS_client|{AG(mr),DON(mr)} makes|ACT(mr) #a
    text_request|{HAB(mr)} #to #the
    @WHOIS_server|RECP(mr) ]|SUM(r)
  #then #the @WHOIS_server|{AG(r),DON(r)} replies|ACT(r)
  #with text_content|HAB(r) ].

[ #All @requests|PAT #are terminated|ACT #with
  { ASCII_CR AND #then ASCII_LF }|INSTP ].

[ [ #The response|{OMN(c), POTII} #might contain|{ACT(c),POTP} *more_than_one
  [ ^line|QUALII #of text|QUAL ]|PARS(c) ]|CAUP
  #so [ #the presence|ACT #of { @ASCII_CR OR @ASCII_LF }|PAT #characters ]|NOT(ni)
  #does not_indicate|ACT(ni)
  [ #the ^end|HAB #of #the @response|POSS ]|STIM(ni) ].

[ #The @WHOIS_server|{AG(cl),POSS} closes|ACT(cl)
  #the connection|{HAB,PAT(cl)}
  #as #soon #as [ #the output|AG #is finished|STAT ]|TEMP(cl) ].

[ #The $closed @connection|STIM #is #the
  indication|ACT #to #the @WHOIS_client|EXP
  #that [#the @response|PAT #has #been received|ACT ]|NOT ].
```

Auszug 36: Die SAL_E-annotierte Spezifikation des WHOIS Protokolls.

Das daraus erzeugte UML-Softwaremodell ist in Abbildung 33 abgebildet. Im Gegensatz zu den vorher abgebildeten UML-Modellen, wurde das Softwaremodell optisch nicht für den Druck angepasst, sondern liegt in der Version vor, wie sie von dem verwendeten UML-Werkzeug Altova UModel [Alt08] grafisch ausgegeben wird.

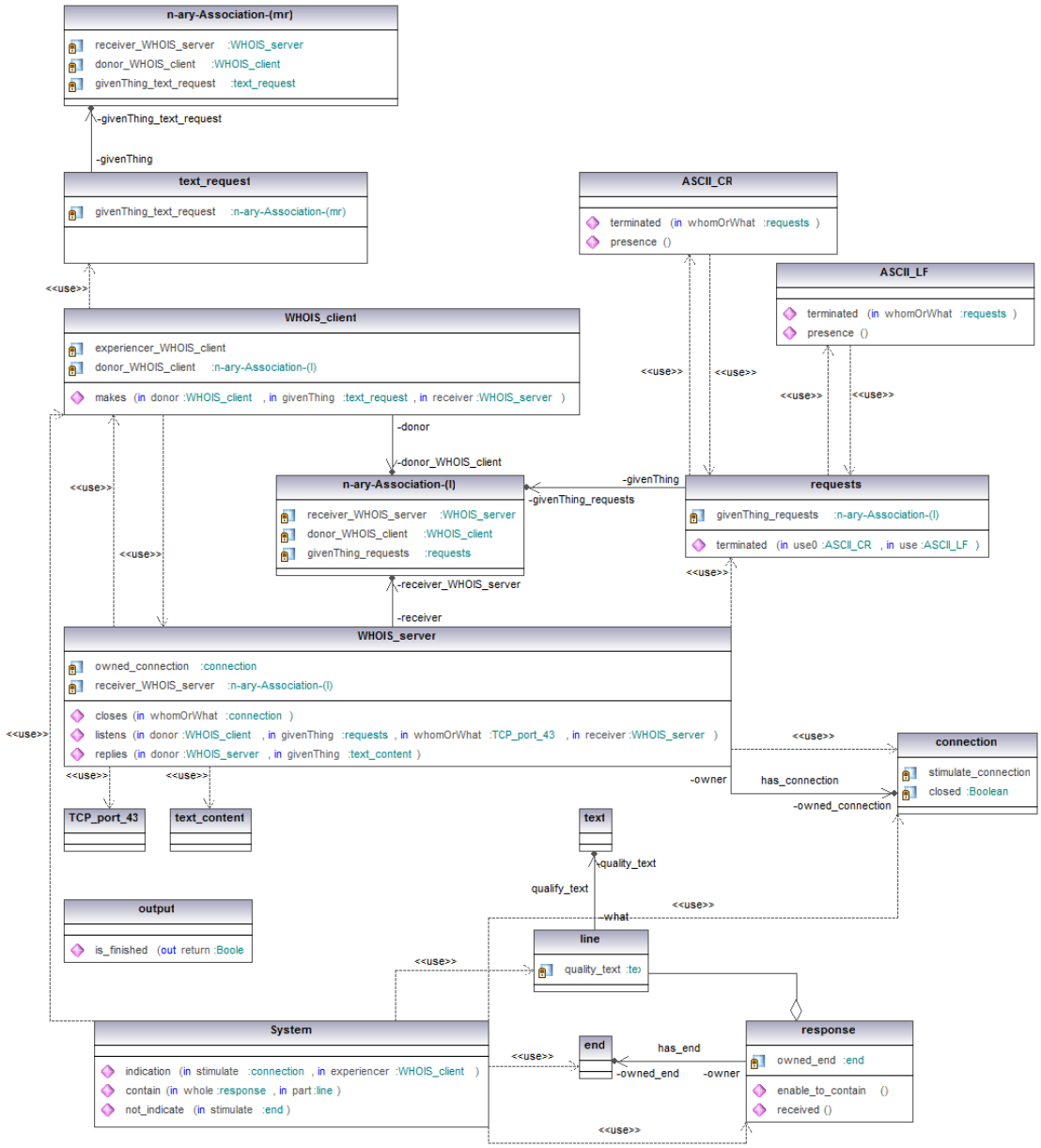

Abbildung 33: Das extrahierte Modell der annotierten WHOIS-Spezifikation.

Die Idee für die Rückkopplung der Modelländerung ist es, die Elemente des Softwaremodells so zu löschen, dass die Konstituenten der in den letzten beiden, in Auszug 36 vorkommenden, Phrasen entfernt werden. Dazu werden aus dem Klassendiagramm folgende Modellelemente entfernt:

- die Klasse *connection* und deren Kindelemente,
- die Klasse *output*,
- die Operation *received* der Klasse *response*,
- die Operation *closes* der Klasse *WHOIS_server*,
- die Operation *indication* der Klasse *operation*
- und die Assoziation *has_connection* zwischen den Klassen *WHOIS_server* und *connection*.

Die Änderungen werden im Unterschiedsprotokoll festgehalten und von REFS verarbeitet. Das Ergebnis der Rückkopplung der gelöschten Modellelemente wirkt sich auf die annotierte Spezifikation wie folgt aus.

```
1  [ #A WHOIS_server|{AG(l),RECP(l)} listens|ACT(l) #on
2    TCP_port_43|PAT(l) #for requests|HAB(l) #from
3    WHOIS_client|DON(l) ].
4
5  [ [ #The @WHOIS_client|{AG(mr),DON(mr)} makes|ACT(mr) #a
6      text_request|{HAB(mr)} #to #the
7      @WHOIS_server|RECP(mr) ]|SUM(r)
8    #then #the @WHOIS_server|{AG(r),DON(r)} replies|ACT(r)
9    #with text_content|HAB(r) ].
10
11 [ #All @requests|PAT #are terminated|ACT #with
12   { ASCII_CR AND #then ASCII_LF }|INSTP ].
13
14 [ [ #The response|{OMN(c), POTII} #might contain|{ACT(c),POTP} *more_than_one
15   [ ^line|QUALII #of text|QUAL ]|PARS(c) ]|CAUP
16   #so [ #the presence|ACT #of { @ASCII_CR OR @ASCII_LF }|PAT #characters ]|NOT(ni)
17   #does not_indicate|ACT(ni)
18   [ #the ^end|HAB #of #the @response|POSS ]|STIM(ni) ].
19
20 [ #The @WHOIS_server|AG(cl) #as #soon #as #the ].
21
22 [ #The #is #the #to #the @WHOIS_client|EXP #that ].
```

Auszug 37: Die SAL_{E}-annotierte Spezifikation des WHOIS Protokolls mit der Rückkopplung gelöschter Modellelemente.

Das Ergebnis der Rückkopplung zeigt; die Änderungen finden in den letzten beiden Phrasen statt. Die Konstituente *connection* sowie *output* wurde aus der kompletten Spezifikation ausgelassen. Weiterhin wurden die Konstituenten *received*, die in einem

gemeinsamen Kontext mit *response* vorkommt, aus der Spezifikation entfernt. Aus dem gemeinsamen Kontext mit *received* wurde *response* in der letzten Phrase entfernt.

Die ACT-Konstituenten *closes* und *indication* wurden ebenfalls aus den letzten beiden Phrase entfernt, da ihre Operationselemente aus dem Modell entfernt wurden und keine Verwendung mehr finden.

Zwar tauchen die redundanten Konstituenten *WHOIS_server* und *WHOIS_client* sowie mehrere Kommentare auf; sie beeinflussen das Ergebnis aber nicht. Die Semantik der letzten beiden Phrasen ist ohne Bedeutung und sie definieren keine neue Modelle. Werden redundante Vorkommnisse von Konstituenten entfernt, sind die letzten beiden Phrasen komplett sinnfrei und können komplett ausgelassen werden.

Lädt man nun beide Dokumente, die ursprüngliche und abgeglichene Version der Spezifikation, in das Textbearbeitungsprogramm und lässt es beide Spezifikationen vergleichen, so werden die Änderungen schnell sichtbar, siehe Abbildung 34. Zur besseren Darstellung der Abbildung, wurde beim Vergleich auf die ersten drei Phrasen des WHOIS Protokollauszugs verzichtet.

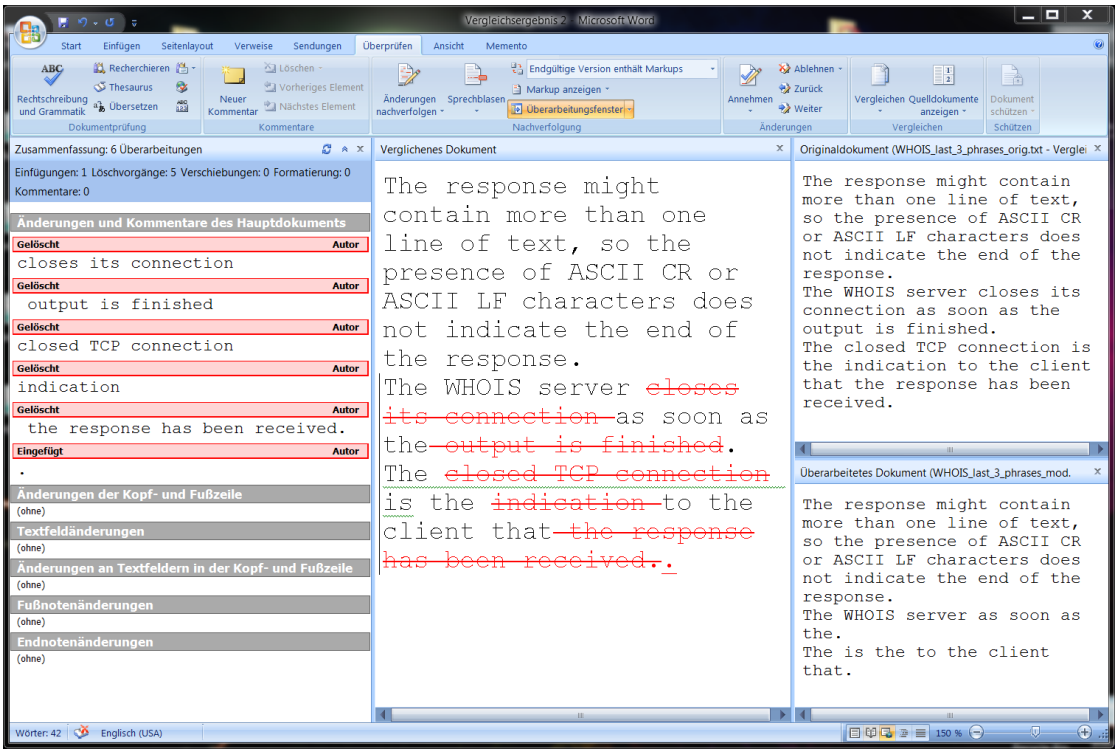

Abbildung 34: Automatisch angezeigte Unterschiede im WHOIS Protokoll.

5.6. Zusammenfassung

Tabelle 2: Erkennungsrate der Modellrückkopplungen in Unterabschnitt 5.1.

Fallbeispiel 1	Klassen	Attr.	Oper.	Param.	Assoz.	General.
Anzahl Creations im Modell / Anzahl Rückkopplungen im Text	2/2	1/1	1/1	1/1	1/1	1/1
Anzahl Updates im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Anzahl Deletions im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Restartefakte oder Redundanzen?	ja					

Tabelle 3: Erkennungsrate der Modellrückkopplungen in Unterabschnitt 5.2.

Fallbeispiel 2	Klassen	Attr.	Oper.	Param.	Assoz.	General.
Anzahl Creations im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Anzahl Updates im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Anzahl Deletions im Modell / Anzahl Rückkopplungen im Text	1/1	1/1	1/3	1/2	1/4	1/1
Restartefakte oder Redundanzen?	ja					

Tabelle 4: Erkennungsrate der Modellrückkopplungen in Unterabschnitt 5.3.

Fallbeispiel 3	Klassen	Attr.	Oper.	Param.	Assoz.	General.
Anzahl Creations im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Anzahl Updates im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Anzahl Deletions im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Restartefakte oder Redundanzen?	ja					

Tabelle 5: Erkennungsrate der Modellrückkopplungen in Unterabschnitt 5.4.

Fallbeispiel 4	Klassen	Attr.	Oper.	Param.	Assoz.	General.
Anzahl Creations im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	1/1	0/0	0/0	0/0
Anzahl Updates im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	1/1	0/0	0/0
Anzahl Deletions im Modell / Anzahl Rückkopplungen im Text	1/1	0/0	0/0	0/0	0/0	0/0
Restartefakte oder Redundanzen?	nein					

Tabelle 6: Erkennungsrate der Modellrückkopplungen in Unterabschnitt 5.5.

WHOIS Protokoll	Klassen	Attr.	Oper.	Param.	Assoz.	General.
Anzahl Creations im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Anzahl Updates im Modell / Anzahl Rückkopplungen im Text	0/0	0/0	0/0	0/0	0/0	0/0
Anzahl Deletions im Modell / Anzahl Rückkopplungen im Text	2/5	0/0	3/4	0/0	1/2	0/0
Restartefakte oder Redundanzen?	ja					

Mit den bisher gezeigten Spezifikation wurden die ersten Resultate der Modellrückkopplungskomponente REFS demonstriert. Die oben gezeigten Tabellen, spiegeln die Erkennungsrate von Modellrückkopplungen in die textuelle Spezifikation wider.

Es wird deutlich, dass `Deletion`s mehr Rückkopplungen zur Folge haben kann, als die Anzahl der Löschungen im Modell selbst. Tabelle 6 zeigt dies eindrucksvoll. Bei zwei gelöschten Klassenelementen aus dem Modell wirkt sich die Rückkopplung an fünf verschiedenen Stellen in der Spezifikation aus. Allerdings gibt es noch Schwächen, wie z. B. die auftretenden Redundanzen, die in einem Nachbearbeitungsprozess aufgelöst werden müssen.

Bei den Hinzufügungen neuer Modellelemente (`Creation`s) gibt es eine 1:1 Rückkopplungsrate. Für jedes neu definierte Modellelemente, gibt es genau eine entsprechende textuell erzeugte Reflektion in der Spezifikation. Dies führt aber auch dazu, dass redundante Konstituenten auftauchen, die ebenfalls in einem Nachbearbeitungsschritt aufgelöst werden müssen.

`Update`s führen nicht zu Redundanzen, da hier keine neuen Textstellen in der Spezifikation entstehen oder Textstellen ausgelassen werden, sondern lediglich zu Wertänderungen vorhandener Konstituenten führen.

Da die Struktur der Spezifikation bei der Rückkopplung erhalten bleibt, kann jeweils, die ursprüngliche und abgeglichene Version der Spezifikation, miteinander verglichen werden. Wir haben gezeigt, dass uns der reine Textvergleich von Textbearbeitungsprogrammen mit einer integrierten Vergleichsfunktion teilweise abgenommen wird. Weiterhin haben wir REFS bisher nur an kleinen Spezifikationen demonstrieren können. Eine Evaluation zu einer Spezifikation, wie sie in der Industrie vorkomment, ist sicherlich interessant.

Eine zusätzliche Evaluation mit einem Team aus Spezialisten scheint interessant, um von ihnen gemachte Änderungen mit REFS zu evaluieren und die Ergebnisse der Rückkopplung in der Spezifikation zu überprüfen.

6. Zusammenfassung und Ausblick

Die Rückkopplung von Modelländerungen in die zugehörige textuelle Spezifikation ist ein wichtiger Grundstein um Iterationen im teilweise automatisierten Anforderungsanalyseprozess zu unterstützen. Mit REFS gelingt es durch Rückverfolgbarkeit der modellierten Anforderungen eine Verknüpfung zwischen der Spezifikation und seinem erzeugten Modell herzustellen. Modelländerungen werden einfach und transparent in die ursprüngliche Spezifikation rückgekoppelt und eingefügt. Durch die Beibehaltung der linguistischen Struktur der Spezifikation werden Kunde und Analysten in ihrer Arbeit zusätzlich unterstützt. Sie können zwei unterschiedliche Versionen einer Spezifikation mit der selben Satzstruktur vergleichen. Änderungen zwischen der ursprünglichen Version und der korrigierten Version des Dokuments sind mit einer Vergleichsfunktion eines beliebigen Textbearbeitungsprogramms schnell und effektiv einsehbar.

Wir haben Schwächen an REFS gezeigt, die behoben werden müssen. An einigen Stellen traten redundante Vorkommnisse von Konstituenten auf, die in einem zusätzlichen Nachbearbeitungsschritt behoben werden können. Die Evaluierung sollte an einer zusätzlichen Real-Welt-Spezifikation, idealerweise aus der Industrie stammenden, Spezifikation getestet werden um die Qualität von REFS genauer zu evaluieren.

REFS wird in Zukunft zusammen mit SAL$_e$ MX und anderen ergänzenden Komponenten in ein Gesamtprodukt integriert, um es als anwendbare Lösung anzubieten.

A. Verwendete linguistische Strukturen

Tabelle 7: Linguistische Strukturen von SAL$_E$.

Linguistische Struktur	**Bedeutung**	**Abbildung zu UML**
AG *agens*	Eine agierende Person oder eine Sache die eine Aktion ausführt.	Klasse.
PAT *patiens*	Person oder Sache die durch eine Aktion betroffen ist oder an einer Aktion in einer Weise beteiligt ist.	Parameter einer Operation.
ACT (+AG +PAT) *actus*	Eine Aktion, ausgeführt durch AG an PAT.	Operation mit Parametern in einer Klasse.
OPUSP (+AG +ACT) *opusp*	Etwas das durch eine Aktion von AG erzeugt wurde.	Klasse die eine Operation besitzt mit einem Rückgabeparameter.
POSS (+HAB) *possessor* und *habitum*	POSS besitzt HAB.	Navigierbare Assoziation von POSS zu HAB.
FIN (+FIC) *fingens* und *fictum*	FIN nimmt die Rolle ein von agieren wie/mag/ist ein FIC.	Generalisierung wobei FIN die Sub- und FIC die Superklasse ist.

Glossar

Konstituente Eine Konstituente (auch Satzteil) ist ein (sprachliches) Element als Teil einer größeren Einheit. Der Begriff entstammt der amerikanischen strukturalistischen Linguistik. In SAL$_e$ MX werden Konstituenten eines natürlichsprachlichen Spezifikationstextes auf einzelne Graphknoten des SAL$_{\text{E}}$-Graphmodells abgebildet. . 4

Literatur

[Alt08] Altova. UModel - UML tool for software modeling and application development, 2008.

[BLO03] L. C. Briand, Y. Labiche, and L. O'Sullivan. Impact Analysis and Change Management of UML Models. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*, page 256, Washington, DC, USA, 2003. IEEE Computer Society.

[BW84] Victor R. Basili and David M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Trans. Software Eng.*, 10(6):728–738, 1984.

[CA07] Betty H. C. Cheng and Joanne M. Atlee. Research Directions in Requirements Engineering. In *FOSE '07: 2007 Future of Software Engineering*, pages 285–303, Washington, DC, USA, 2007. IEEE Computer Society.

[CBB+00] R. G.G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadowa, Torsten Stanienda, and Fernando Velez. *The Object Data Management Standard: ODMG 3.0*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[Che83] Peter P. Chen. English Sentence Structure and Entity-Relationship Diagrams. *Inf. Sci.*, 29(2-3):127–149, 1983.

[Dai04] L. Daigle. WHOIS Protocol Specification. Internet, September 2004. The Internet Engineering Task Force (IETF).

[DB09] Deva Kumar Deeptimahanti and Muhammad Ali Babar. An Automated Tool for Generating UML Models from Natural Language Requirements. *Automated Software Engineering, International Conference on*, 0:680–682, 2009.

[DS08] Deva Kumar Deeptimahanti and Ratna Sanyal. Static UML Model Generator from Analysis of Requirements (SUGAR). In *ASEA '08: Proceedings of the 2008 Advanced Software Engineering and Its Applications*, pages 77–84, Washington, DC, USA, 2008. IEEE Computer Society.

[Egy01] Alexander Egyed. Scalable Consistency Checking Between Diagrams-The ViewIntegra Approach. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 387, Washington, DC, USA, 2001. IEEE Computer Society.

[Fel98] Christaine Fellbaum. *WordNet: An Electronic Lexical Database*. The MIT Press, Cambridge, Massachusetts, 1998.

[GBG+06] Rubino Geiß, Gernot Veit Batz, Daniel Grund, Sebastian Hack, and Adam M. Szalkowski. Grgen: A fast spo-based graph rewriting tool. In A. Corradini,

H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *Graph Transformations - ICGT 2006*, Lecture Notes in Computer Science, pages 383 – 397. Springer, 2006. Natal, Brasil.

[GDG08] Tom Gelhausen, Bugra Derre, and Rubino Geiss. Customizing GrGen.NET for Model Transformation. In *3rd International Workshop on Graph and Model Transformation (GraMoT)*, Leipzig, Germany, May 2008. ACM.

[Gel10] Tom Gelhausen. *Modellextraktion aus natürlichen Sprachen*. PhD thesis, Institute for Program- and Datastructures (IPD) at Karlsruhe Institute of Technology (KIT), 2010.

[Got94] *An analysis of the requirements traceability problem*, April 1994.

[Gro07a] Object Management Group. Unified modeling language: Superstructure. Technical report, OMG Modeling and Metadata Specifications, November 2007.

[Gro07b] Object Management Group. Xml metadata interchange (xmi®) specification catalog. `http://www.omg.org/technology/documents/modeling_spec_catalog.htm`XMI, December 2007.

[GT07] Tom Gelhausen and Walter F. Tichy. Thematic Role based Generation of UML Models from Real World Requirements. In *First IEEE International Conference on Semantic Computing (ICSC 2007)*, volume 0, pages 282–289, Irvine, CA, USA, September 2007. IEEE Computer Society.

[KDGL] Sven J. Körner, Bugra Derre, Tom Gelhausen, and Mathias Landhaeusser. RECAA – the Requirements Engineering Complete Automation Approach.

[KGR06] Petr Kroha, Philipp Gerber, and Lars Rosenhainer. Towards Generation of Textual Requirements Descriptions from UML Models. In *Proceedings of the 9th International Conference Information Systems Implementation and Modelling ISIM2006*, pages 31 – 38. ISIM, April 2006.

[Kro00] P. Kroha. Preprocessing of Requirements Specification. In *Database and Expert Systems Applications*, volume 1873 of *Lecture Notes in Computer Science*, pages 675–684. Springer Berlin / Heidelberg, 2000.

[LRR97] Benoit Lavoie, Owen Rambow, and Ehud Reiter. Customizable descriptions of object-oriented models. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*, pages 253–256, Morristown, NJ, USA, 1997. Association for Computational Linguistics.

[LS81] Bennet P. Lientz and E. Burton Swanson. Problems in application software maintenance. *Commun. ACM*, 24(11):763–769, 1981.

[MAA08] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. Generating Natural Language specifications from UML class diagrams. *Requirements Engineering*, 13(1):1–18, January 2008.

[Ok10] Fatih Ok. Finden von UML Modelländerungen für das Anforderungsmanagement, 2010. Studienarbeit am Institut für Programmstrukturen und Datenorganisation (IPD) des Karlsruhe Institute of Technology (KIT).

[OLR01] Scott P. Overmyer, Benoit Lavoie, and Owen Rambow. Conceptual modeling through linguistic analysis using LIDA. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 401–410, Washington, DC, USA, 2001. IEEE Computer Society.

[Par09] Terence Parr. ANTLR Parser Generator. online, May 2009. Version 3.1.2.

[Pie78] Robert A. Pierce. A requirements tracing tool. In *Proceedings of the software quality assurance workshop on Functional and performance issues*, pages 53–60, New York, NY, USA, 1978. ACM.

[RCW07] Juergen Rilling, Philippe Charland, and René Witte. Traceability in Software Engineering – Past, Present and Future. Technical Report TR-74-211, IBM Technical Report, CASCON 2007 Workshop, October 25 2007.

[RD00] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Natural Language Processing. Cambridge University Press, 2000.

[Rup07] Chris Rupp. *Requirements-Engineering und Management*. Hanser, München, 2007.

[SS97] Ian Sommerville and Pete Sawyer. *Requirements Engineering. A Good Practice Guide*. John Wiley & Sons Inc, March 1997.

[Vol09] Volere. List of Requirement Engineering Tools, 2009.