Transformationen zwischen UML-Use-Case-Diagrammen und tabellarischen Darstellungen

Julia Pilarski
Fachgebiet Software Engineering
Leibniz Universität Hannover
julia.pilarski@arcor.de

Eric Knauss
Fachgebiet Software Engineering
Leibniz Universität Hannover
eric.knauss@inf.uni-hannover.de

Abstract: In den frühen Phasen eines Softwareprojekts steht die Modellierung in einem besonderen Spannungsfeld: Entweder sind die Modelle formal genug, um verifizieren zu können, dass sie richtig modelliert wurden, oder umgangssprachlich genug, damit beim Kunden validiert werden kann, ob das Richtige modelliert wurde. Aus diesem Grund eignen sich komplexere Modelle zur Szenario-Darstellung (z.B. UML-Sequenz-Diagramme) nicht so gut. Andererseits haben grafische Modelle den Vorteil, einen guten Überblick zu bieten. Transformationen zwischen textuellen Beschreibungen und grafischen Modellen können dieses Spannungsfeld auflösen, indem sie es erleichtern, grafische Modelle und natürliche Sprache parallel zu nutzen. Dieser Beitrag untersucht das Verhältnis zwischen den Use-Case-Diagrammen der UML und (typischerweise tabellarischen) natürlich-sprachlichen Use-Cases. Mit Hilfe eines Basismetamodells definieren wir die gemeinsamen Konzepte beider Darstellungen. Wir beschreiben entsprechende Transformationen und geben konkrete Beispiele.

1 Einleitung

Use-Cases (auch Anwendungsfälle genannt) setzen sich für die Anforderungsspezifikation immer weiter durch. Als Vorteil wird vor allem die Beschreibung von funktionalen Anforderungen im Kontext von Benutzerzielen gesehen. Auch die explizite Betrachtung von Ausnahmen ist eine der Stärken von Use-Cases.

Für die konkrete Ausgestaltung von Use-Cases gibt es verschiedene Vorschläge: Auf der einen Seite gibt es die modellierungsnahe Sicht (hier repräsentiert durch die UML [OMG07]). Use-Cases werden dabei in UML-Use-Case-Diagrammen benannt und die enthaltenen Erfolgs- und Sonderszenarien werden in Interaktionsdiagrammen spezifiziert. Die Vorteile sind hierbei, dass durch die definierte Semantik der Modelle Missverständnisse vermieden werden, sowie dass die Modelle durch Abstraktion eine gute Übersicht bilden können.

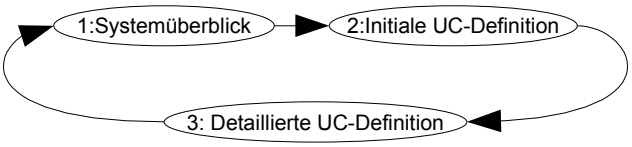

Abbildung 1. Effiziente Anwendungsfallspezifikation (nach [Bir06])

Dem steht auf der anderen Seite ein eher natürlich-sprachlicher Ansatz gegenüber [Coc01]. Hier werden die Szenarien als *User Stories* oder als Aufzählung in tabellarischen Vorlagen (im folgenden als *Template* bezeichnet) formuliert. Der Vorteil der natürlichen Sprache ist, dass Kunden ohne technischen Hintergrund einen Use-Case verstehen und kritisieren können. Dies ist eine wichtige Voraussetzung, um Use-Cases validieren zu können. Zudem hat man durch eine Menge von Formulierungsrichtlinien (z.B. [Rup06]) eine Basis geschaffen, um trotz natürlicher Sprache zu einer weitgehend eindeutigen Spezifikation zu kommen. Diese Richtlinien kann man zum Teil auch automatisch überprüfen, wie zum Beispiel in [Cri06, Kna07] gezeigt wurde. Als entscheidender Nachteil bleibt jedoch die mangelnde Übersichtlichkeit.

Hier wäre es schön, die Vorteile der UML einsetzen zu können. Abbildung 1 zeigt schematisch, wie man nach [Bir06] bei umfangreichen Anwendungsfallspezifikationen vorgehen sollte: UML-Use-Case Diagramme (1) lassen sich bei den ersten Gesprächen einsetzen, bei denen die Diskussion noch nicht ins Detail geht und ein Systemüberblick von größerer Bedeutung ist. Nachdem das Grobverhalten des Systems dokumentiert ist, können die Anforderungen in Form von textuellen Use-Case-Templates (bspw. nach [Coc01]) verfeinert werden (2). An dieser Stelle würde eine automatisierte Transformation von einem Diagramm zu einem Template die Überführung erleichtern. Anschließend folgt eine tiefere Analyse der Kundenwünsche. Dafür werden automatisch generierte Templates vervollständigt sowie neue textuelle Use-Cases angelegt (3). Erweiternd zu [Bir06] streben wir an, die vorgenommenen Änderungen wieder im Use-Case-Diagramm darstellen zu können (1). Voraussetzung dafür ist eine automatisierte Rücktransformation, da ansonsten der Aufwand für die Synchronisation und Sicherstellung der Konsistenz zu groß würde.

Wissensbasierte Ansätze im Requirements Engineering verfolgen zum Teil einen ähnlichen Kreislauf. In [FKM+01] wird die Verknüpfung mehrerer Szenarien genutzt, um eine Wissensbasis aufzubauen. Im Gegensatz dazu beschränken wir uns auf das weniger ergeizige Ziel, zwei nicht ganz deckungsgleiche Sichten auf Funktionale Anforderungen miteinander zu kombinieren. Dadurch stehen mit textuellen und grafischen Use-Case Repräsentationen zwei spezielle Modellierungssprachen mit ihren jeweiligen Stärken zur Verfügung.

Auf Basis der Vorarbeiten in [Pil07] präsentieren wir in diesem Beitrag ein Konzept, mit dem es möglich ist diesen Zyklus auf der Basis eines gemeinsamen Metamodells durchzuführen: Abschnitt 2 enthält eine systematische Analyse der gemeinsamen Konzepte

von UML-Use-Case-Diagrammen und textuellen Beschreibungen. Diese wird in einem Basismetamodell dargestellt. Nach der formalen Begriffsklärung beschreiben wir in Abschnitt 3 wie die vorgeschlagenen Transformationen auf dieser Grundlage realisiert werden können.

2 Ermittlung gemeinsamer Elemente

Eine Transformation zwischen zwei Modellen ist dann möglich, wenn die beiden zu transformierenden Modelle über eine Menge gemeinsamer Elemente verfügen. Die Ermittlung der Gemeinsamkeiten folgt aus dem Vergleich der Elemente beider Konzepte.

Der natürlich-sprachliche und der grafische Ansatz haben ursprünglich ähnliche Wurzeln und verfügen deshalb über eine Menge ähnlicher Elemente. Im Laufe der Entwicklung wurden beide Konzepte um neue Elemente bereichert. Infolgedessen ist es nötig zu untersuchen, inwieweit die Elemente beider Konzepte ähnlich geblieben sind und worin die Unterschiede liegen.

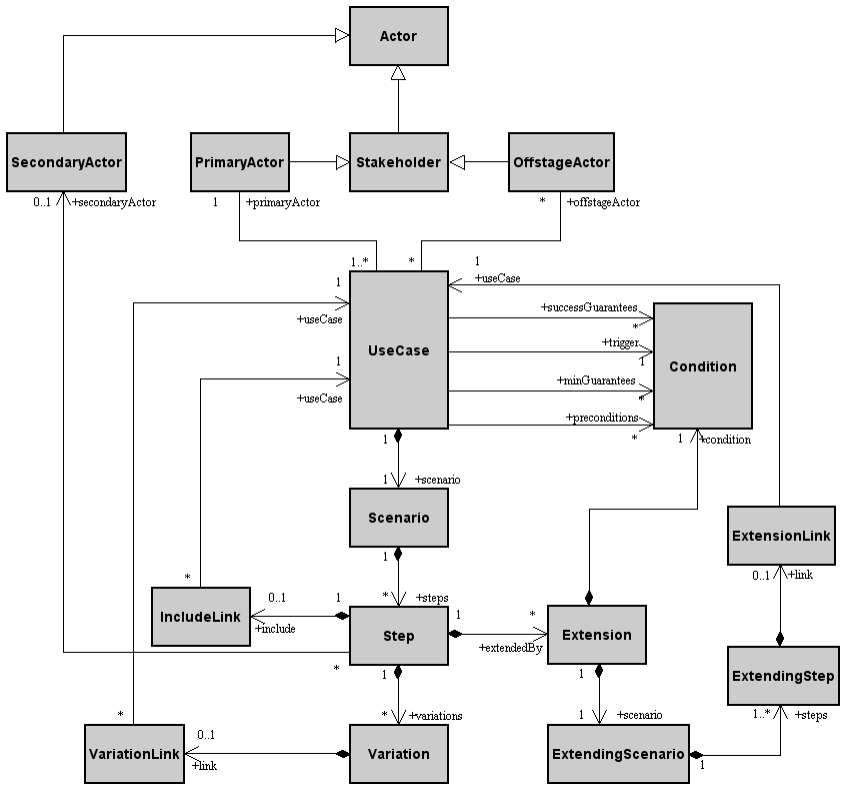

Abbildung 2: Use-Case-Metamodell

Als Ausgangspunkt der Vergleichsanalyse dient das Konzept tabellarischer Use-Cases nach [Coc1]. Im Weiteren werden Elemente dieses Konzeptes analysiert und in einem Metamodell abstrakt beschrieben. Als Notation dafür wird das UML-Klassendiagramm verwendet. Dieses Vorgehen bietet mehrere Vorteile. So wird angestrebt, eine allgemeine anwendungsunabhängige Definition tabellarischer Use-Cases festzulegen. Darüber hinaus liegt für die UML-Use-Case-Diagramme bereits ein Metamodell vor [OMG07].

Somit bildet die UML eine einheitliche sprachliche Basis, die einen Vergleich elementweise ermöglicht. Das Use-Case-Metamodell und das Use-Case-Diagramm-Metamodell werden in Abbildungen 2 und 3 gezeigt.

Das Use-Case-Metamodell wurde in Anlehnung an [Coc1] in [Lüb06, Pil07] erstellt. Ein *UseCase* ist hier das zentrale Element. Er hat eine Gruppe ihn auslösender Ereignisse und durch ihn zu gewährleistenden Garantien (*Condition*), sowie ein Szenario (*Scenario*), das aus einem oder mehreren Schritten (*Step*) besteht. Ein Schritt kann über einen *IncludeLink* auf einen anderen *UseCase* verweisen. Ein Schritt kann mehrere technische Variationen (*TechnologyVariation*) haben. Ein Schritt kann durch ein Erweiterungsszenario (*ExtendingScenario*) erweitert werden, das ebenfalls aus mehreren Schritten bestehen kann. Eine Erweiterung kann über einen *ExtensionLink* auf einen weiteren *UseCase*

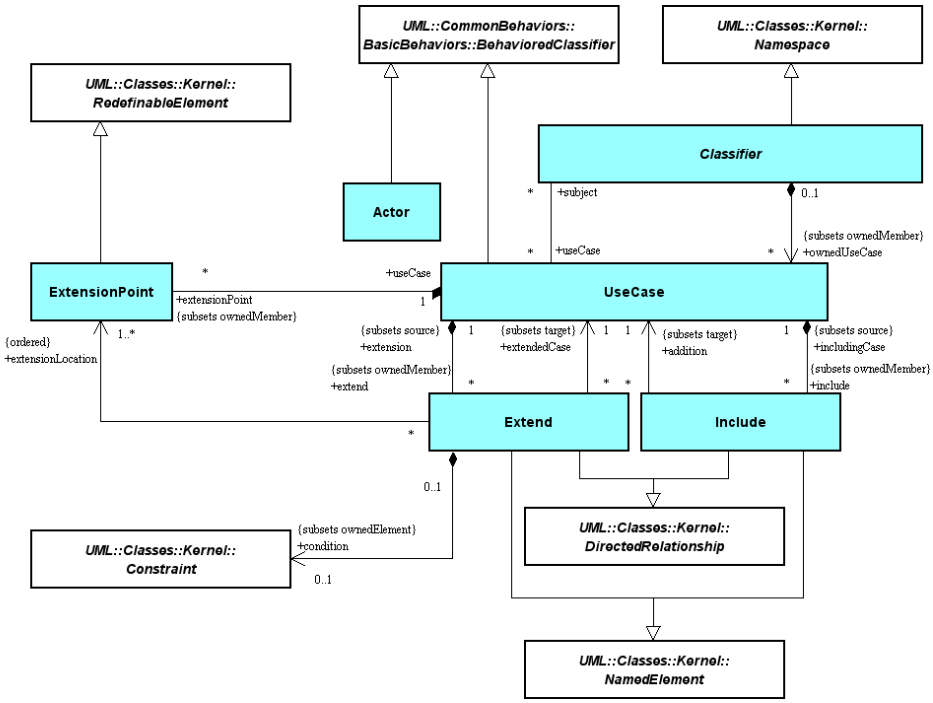

Abbildung 3: UML-Use-Case-Metamodell

zeigen. Ein *UseCase* hat eine assoziative Beziehung zu einem *Primärakteur* und mehreren *OffstageActors*.

Abbildung 3 zeigt das Use-Case-Diagramm-Metamodell, wie es in [OMG07] bereits definiert ist:

Das zentrale Element dieses Metamodells ist ebenfalls ein *UseCase*, der mehrere *«include»*- und *«extend»*-Beziehungen haben kann. Diese Beziehungen verweisen auf einen eingeschlossenen bzw. erweiterten *UseCase*. Eine Erweiterung kennt den Erweiterunspunkt (*ExtensionPoint*) des *UseCase*, in dem sie unter einer Bedingung (*Constraint*) eintritt. Als Classifier sind zwischen Akteuren (*Actor*) und *UseCases* Assoziationen, Generalisierungen und Spezialisierungen erlaubt.

Während der Analyse beider Metamodelle hat sich gezeigt, dass sich nicht alle Elemente direkt aufeinander abbilden lassen und infolgedessen ineinander nicht direkt übersetzt werden können. Als eine Abbildung wird eine Zuordnungsvorschrift bezeichnet, die jedem Element a ϵ A eindeutig ein Element f(a) ϵ B zuordnet [BSM+00]. Bereits bei dem ersten Betrachten fällt auf, dass die Anzahl der Elemente eines Template erheblich höher als die Anzahl der Diagramm-Elemente ist. Weiterhin stehen die Template-Elemente in komplizierteren Beziehungen zueinander. Einige Diagramm-Elemente können ausschließlich durch eine Gruppe von Elementen eines Template dargestellt werden.

So verfügt bspw. ein Use-Case im Use-Case-Metamodell über ein *Scenario* und Interaktionsschitte (*Step*). Im Use-Case-Diagramm-Metamodell ist lediglich ein Element *Use-*

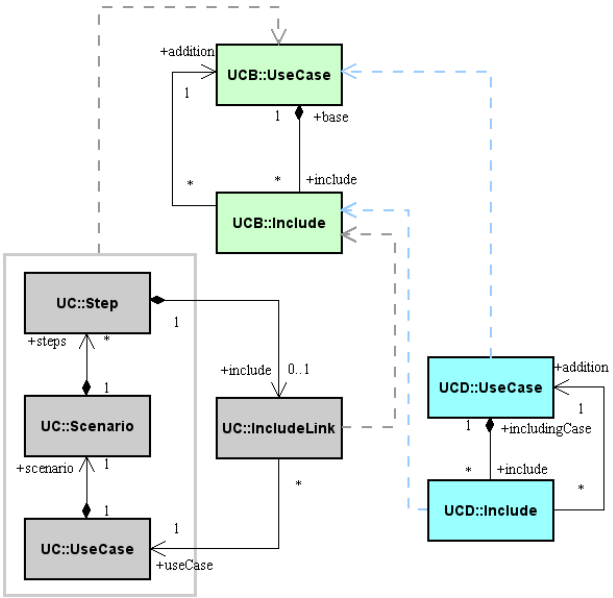

Abbildung 4. Include-Schnittmenge

Case zu finden. Eine *Include*-Beziehung in einem Diagramm kann aus mehreren Schritten bzw. *IncludeLinks* im Template bestehen. Ein ähnliches Verhalten stellt sich bei dem Vergleich der jeweiligen *Extend*-Beziehungen heraus. Dagegen lassen sich Technische Variationen aus dem Template im Diagramm am ehesten durch Vererbung von *UseCases* darstellen, auch weil für Templates in [Coc01] keine Generalisierung für Use-Cases vorgesehen ist (näher in [Pil07]).

Somit werden hier nicht Abbildungen, sondern Relationen zwischen Modellen betrachtet. Die Elemente bzw. ihre Beziehungen zu einander werden des Weiteren paarweise miteinander verglichen. Die gemeinsamen Elemente werden in Form eines Basismetamodells dokumentiert.

Das folgende Beispiel zeigt das allgemeine Prinzip des Vorgehens für die Ermittlung des gemeinsamen Elementes *UseCase* und seiner *«include»*-Beziehung.

Das Element *UseCase* in dem Use-Case-Metamodell enthält mehr Informationen als sein Diagramm-Verwandter. Zusammen mit einem Szenario und dessen Schritten wirkt er nach außen jedoch als ein Element. Diese Bindung ist in der Abbildung 4 durch ein Rechteck dargestellt. Zusammengenommen erfüllen diese Elemente inhaltlich die selbe Aufgabe wie ein Use-Case im Diagramm (Repräsentation eines Benutzer-Ziels). Daher kann *UseCase* als gemeinsames Element identifiziert und in das gemeinsame Basismetamodell übernommen werden.

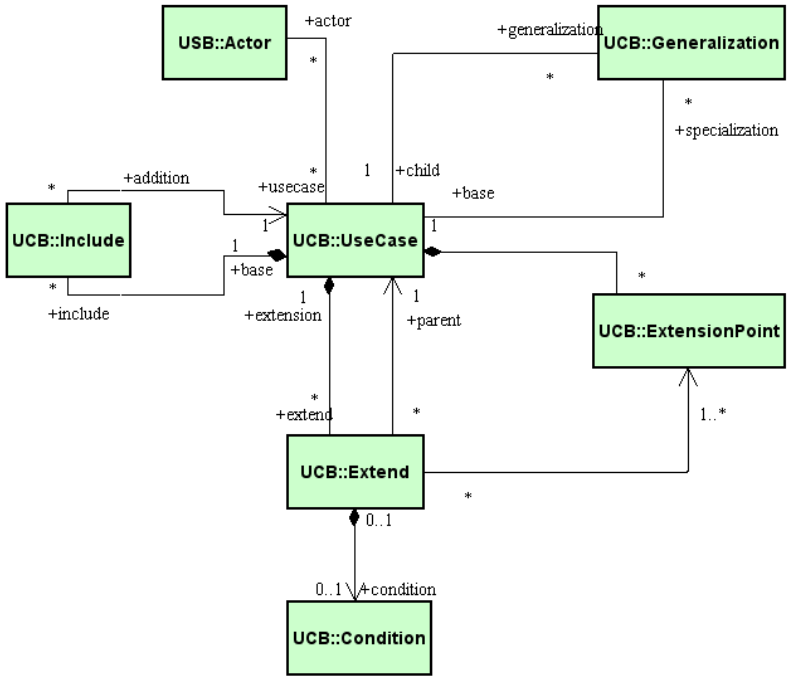

Abbildung 5. Use-Case Basismetamodell

Darüber hinaus weisen die beiden Metamodelle Verbindungen zu anderen *UseCases* auf. Das Element *UseCase* des Use-Case-Metamodells in Abbildung 2 hat dasselbe Verhalten zu dem Element *IncludeLink* wie das gleichnamige Element des UML-Use-Case-Diagramm-Metamodells zu dem Element *Include*. Ein Use-Case-Szenario kann aus mehreren Schritten bestehen. Ein Schritt kann über einen *IncludeLink* auf genau einen Use-Case zeigen. Ein Use-Case des UML-Use-Case-Diagramm-Metamodells kann ebenfalls mehrere *«include»*-Beziehungen haben. Jede dieser Beziehungen kann auf genau einen Use-Case verweisen.

Die gemeinsame Eigenschaft, durch eine Verlinkung von einem Use-Case auf einen weiteren Use-Case zu verweisen, wird ebenfalls in das Basismetamodell übernommen. Abbildung 4 zeigt das Ergebnis des Vergleiches. In der Abbildung werden folgende Kürzel verwendet: *UC* für das Use-Case-Metamodell, *UCD* für das Use-Case-Diagramm-Metamodell der UML, *UCB* für das Use-Case-Basismetamodell. Die gestrichelten Pfeile deuten an, welche Elemente Transformationen aufeinander abbilden müssen.

Zuletzt werden die einzelnen Elemente des Basismetamodells zusammengefügt. Das Ergebnis der Zusammensetzung stellt Abbildung 5 dar. Es enthält die Elemente, die sowohl für Use-Case Diagramme als auch für die Use-Case Beschreibungen relevant sind. Es fungiert als eine Art Schnittstellenbeschreibung für entsprechende Transformationen.

2.1 Darstellung von Use-Case-Hierarchien

Eines unserer Ziele ist es, grafische Use-Case Darstellungen aus den Use-Case Beschreibung abzuleiten. Ein Diagramm für eine komplette Spezifikation wird schnell unübersichtlich, vor allem, wenn es automatisch generiert wird. Darum bietet es sich an, für komplexe Systeme mehrere Diagramme zu bilden. Im Weiteren wird untersucht, nach welchem Prinzip Systeme im Diagramm gebildet werden und wie eine Gruppe von Templates sich einer Gruppe von Diagrammen zuordnen lässt. Unsere Lösung basiert auf der Ermittlung eines gemeinsamen Nenners – eines Ziels.

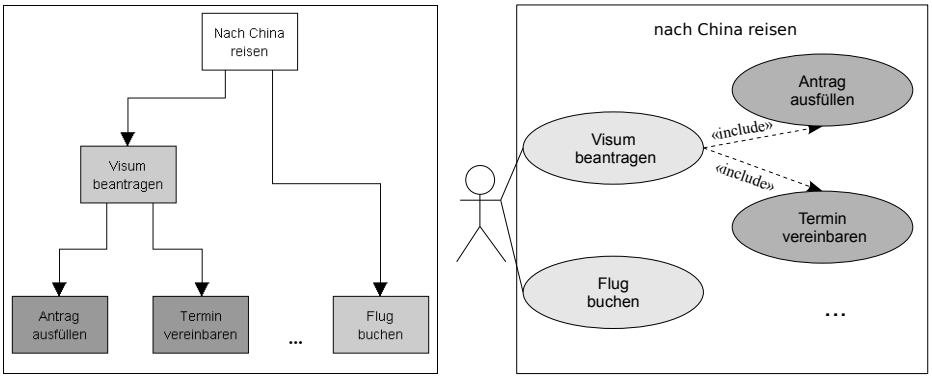

Abbildung 6. Hierarchische Erstellung der Diagramme

Das System in einem Diagramm ist diejenige Einheit, die das Verhalten, das durch die Use-Cases beschrieben wird, realisiert und anbietet [JRH+03]. Es umfasst mehrere Use-Cases. Ein Use-Case wird als Abkommen definiert, das das Systemverhalten beschreibt, mit der Absicht, ein bestimmtes fachliches Ziel (*Goal*) zu erreichen [Coc01]. Dies impliziert, dass eine Gruppe der in einem System vereinigten Use-Cases ein höheres fachliches Ziel (auch als *Geschäftsziel, Business Goal,* bezeichnet) auf dem Weg zu einem erwünschten Systemverhalten beschreibt. Die Teilsysteme werden zu einem ganzen System vereinigt, das das Hauptziel des Systemverhaltens repräsentiert. Die Idee der Zuordnung der Ziele gemäß ihrer Abstraktionsgrade ähnelt dem Konzept der Ebenen nach Cockburn [Coc01]. Neu ist jedoch die Abbildung übergeordneteter Ziele auf die Systemgrenzen ihrer Unterziele.

Basierend auf dem Konzept der Zielstrukturierung kann ein Projekt mit allen Unterzielen, die durch Use-Cases ausgedrückt werden, als eine Hierarchie betrachtet werden. Ein Use-Case kann im Diagramm als ein System dargestellt werden. Seine eingeschlossenen Use-Cases, deren erweiternde und eingeschlossene Use-Cases sowie Generalisierungen bilden den Inhalt des Systems. Solange ein Use-Case-Szenario auf einen eingeschlossenen Use-Case verweist, kann der einschließende Use-Case als ein System im Diagramm abgebildet werden. Ein Use-Case der untersten Abstraktionsebene, der über keine weiteren verlinkten Schritte verfügt, ist als System nicht mehr darstellbar. Das Projekt selbst wird im Diagramm ebenfalls als ein System dargestellt, das eine Übersicht über sämtliche Projektziele liefert. Abbildung 6 zeigt hierzu ein Beispiel: Eines der Ziele des Projektes *Urlaub* sei *nach China reisen*. Dieses Ziel kann durch das Gelingen der Unterziele *Flug buchen*, *Visum beantragen*, *Hotel buchen* und anderen erreicht werden. Im Diagramm wird das Ziel *nach China reisen* als System dargestellt. Die verlinkten Schritte des Use-Case-Szenarios *nach China reisen* werden zu den Use-Cases des Systems. Der Use-Case *Visum beantragen* kann ebenfalls in Form eines Systems dargestellt werden.

3 Transformation

Das Basismetamodell aus Abschnitt 2 zeigt die gemeinsamen Konzepte des UML-Use-Case-Metamodells und des Use-Case-Metamodells. Damit liefert es die Grundlage für unsere Transformationen und dokumentiert, welche Konzepte aufeinander abgebildet werden sollen. Dieser Abschnitt erläutert unser Transformationskonzept. In Abschnitt 4 wird ein Beispiel für den Einsatz der Transformationen gegeben.

Templates, wie bereits in Abbildung 2 gezeigt, beinhalten wesentlich mehr Information als Diagramme. Bei der Transformation dürfen diese zusätzlichen Informationen nicht überschrieben werden. Transformationen zwischen zwei Modellen sollten also in beide Richtungen (bidirektional) stattfinden, sowie relationale Beziehungen zwischen Modellen unterstützen. Unser Ansatz sieht einen Zwischenspeicher für die Transformation vor,

um die gemeinsamen Daten synchronisieren und die individuellen Daten wiederherstellen zu können.

Um diese Punkte gewährleisten zu können, wurde beschlossen als Transformationsspeicher ein vereinigtes Modell des Use-Case-Modells und des UML Use-Case-Diagramm-Modells zu bilden. Das vereinigte Modell beinhaltet alle Elemente beider Modelle [Pil07].

Transformationen finden zwischen dem vereinigten Modell und Teilmodellen statt, wie Abbildung 7 zeigt. Dabei werden die Elemente jeweils mit dem Zeitpunkt ihrer letzten Änderungen übermittelt. Neuere Elemente überschreiben ältere und der Zwischenspeicher verwaltet keine Historie. Damit wird auch nicht die Synchronisation konkurrierender Änderungen unterstützt, da wir davon ausgehen, das Benutzer entweder in der Diagramm- oder in der Template-Sicht arbeiten. Änderungen in einer Sicht sollen dann direkt in die andere übernommen werden.

Die Transformation kann in drei Schritte aufgeteilt werden: Im ersten Schritt werden die Daten der beiden Teilmodelle in das vereinigte Modell eingetragen. Dabei müssen die Daten untereinander synchronisiert werden. Anschließend werden im zweiten Schritt neue Diagramme im vereinigten Modell erstellt, bzw. die bestehenden aktualisiert. Im letzten Schritt wird aus dem vereinigten Modell wieder das Use-Case-Modell und das Use-Case-Diagramm-Modell generiert.

Die Verwendung eines vereinigten Modells als Transformationsspeicher bietet den Vorteil, dass die Beziehung zwischen vereinigtem Modell und den Teilmodellen (Use-Case-Modell, bzw. UML-Use-Case-Diagramm-Modell) jeweils eindeutig ist, da jedes Element aus einem Teilmodell einen Repräsentanten im vereinigten Modell hat. Bei einer Transformation zwischen den Teilmodellen wäre dies nicht der Fall (vergleiche Abschnitt 2).

Bei der Synchronisation werden die beiden Teilmodelle in das vereinigte Modell eingetragen und dabei synchronisiert. Die zu transformierenden Elemente sowie die Transformationsvorschriften lassen sich aus dem gemeinsamen Basismetamodell (Abschnitt 2) ableiten. Die folgenden Regeln werden bei der Transformation eingesetzt:

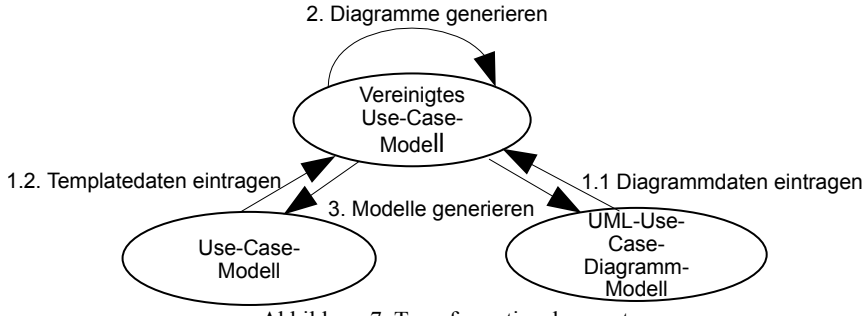

Abbildung 7. Transformationskonzept

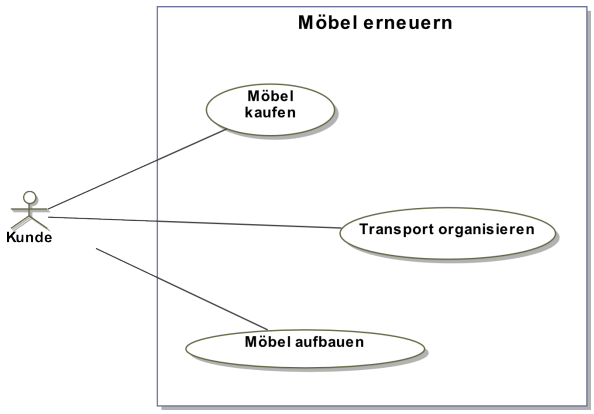

Abbildung 8. Use-Case-Diagramm: Möbel erneuern

1. Alle Elemente aus dem Use-Case-Diagramm-Modell werden in das vereinigte Modell eingetragen. Dies ist möglich, weil alle Elemente des Use-Case-Diagramm-Modells in dem vereinigten Modell zusammenhängend sind. Dadurch entsteht im vereinigten Modell ein vollständiges Bild des UML Use-Case-Diagramm-Modells. Die individuellen Elemente des Modells können als *sychronized* markiert werden.

2. Die Daten der Use-Case-Templates werden mit den Daten des vereinigten Modells abgeglichen. Elemente, die im vereinigten Modell nicht existieren, werden eingetragen und gegebenenfalls als *deleted* markiert. Die individuellen Elemente des Use-Case-Modells können in das vereinigte Modell direkt übernommen und als *sychronized* markiert werden.

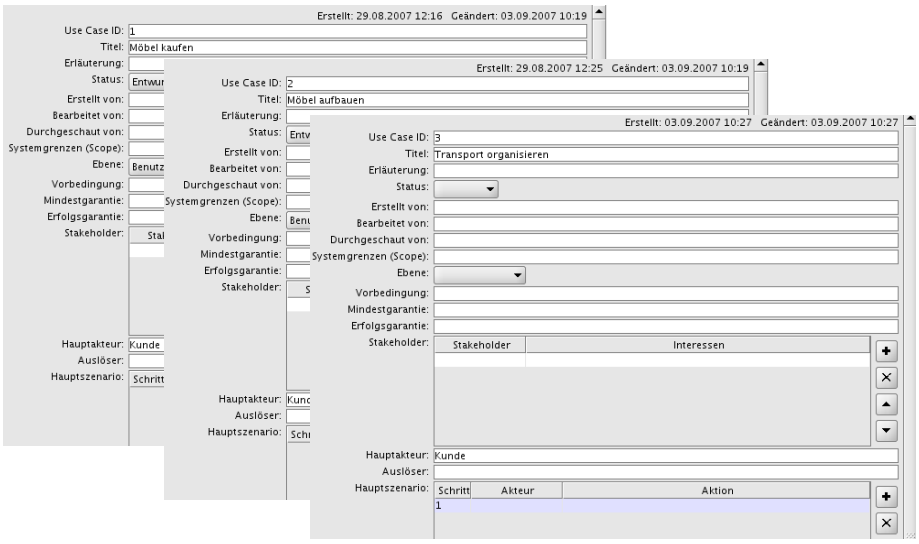

Abbildung 9. Initiale Use-Case-Definition

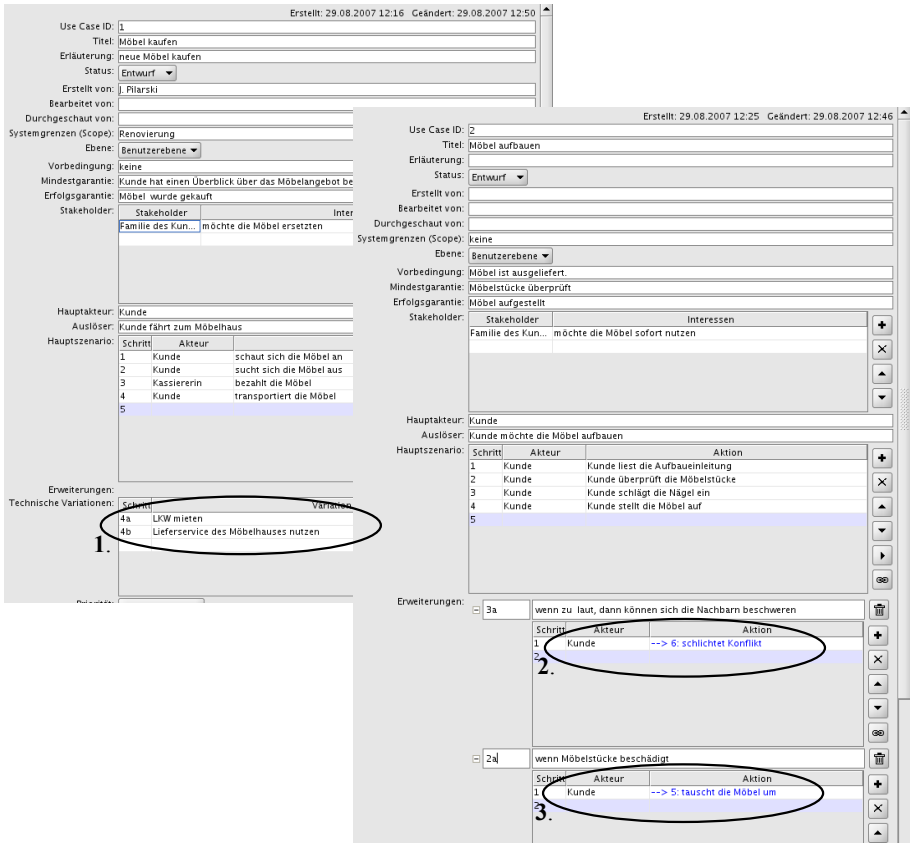

Abbildung 10. Ausschnitt einer detaillierten Use-Case Definition

3. Alle Elemente, die als *sychronized* markiert sind und deren Erstellungsdatum vor der letzten Transformation liegt, werden als *deleted* markiert. Anschließend werden alle *deleted*-Elemente gelöscht.

Damit ist die Synchronisation der Daten abgeschlossen.

Nachdem die Daten beider Modelle synchronisiert sind, müssen unter Umständen neue Diagramme erstellt bzw. aktualisiert werden. Anschließend können die Daten zurück transformiert werden. Dabei werden die Daten direkt aus dem vereinigten Modell in das Use-Case-Modell sowie Use-Case-Diagramm-Modell überführt. Dies ist ein einfacher Schritt, da eine eindeutige Zuordnung zwischen den Elementen des vereinigten Modells und den jeweiligen Teilmodellen existiert.

Dadurch, das wir immer nur wenige Use-Cases in einem Diagramm anzeigen, wird auch die Positionierung erleichtert. Neu generierte Use-Cases können in der Regel einfach an der ersten freien Stelle eingefügt werden: Auf der linken Seite, wenn sie einen Hauptak-

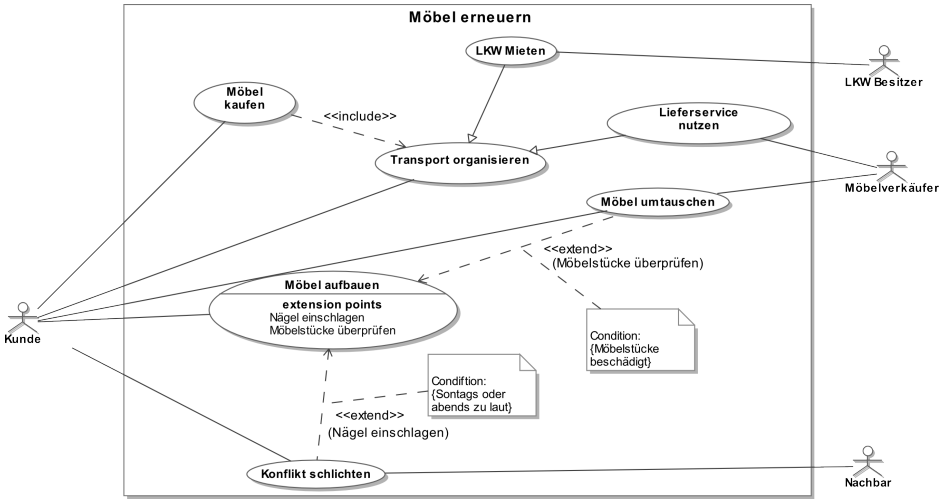

Abbildung 11. Use-Case-Diagramm: Möbel erneuern (vollständig)

teur besitzen, sonst rechts neben dem Use-Case durch den sie inkludiert sind. Benutzerdefinierte Positionen auf der Zeichenfläche gehören zu den Elementen, die von der Transformation nicht betroffen sind und bei einer Aktualisierung bestehen bleiben.

4 Beispiel

Die Vorzüge der Kombination von tabellarischen Use-Case-Notationen und einer graphischen Übersicht sowie ihre simultane Verwendung wurden im Allgemeinen in Abschnitt 1 beschrieben. Folgendes Beispiel erläutert den Kreislauf aus Abbildung 1 nach [Bir06] genauer.

Zu Beginn der Anforderungserhebung soll der Verlauf eines Möbeleinkaufs dokumentiert werden. Ein potentieller Kunde möchte die Möbel in seinem Haus erneuern. Sein Erfolgsszenario umfasst folgende Aktionen: *Transport* der Möbelstücke organisieren, *Möbel* in einem Einkaufszentrum *kaufen, Möbel aufbauen.* Diese Schritte sind im ersten Übersichtsdiagramm (Abbildung 8) dokumentiert.

Nachfolgend wird das Diagramm in Templates transformiert (siehe Abbildung 9). Für jeden Use-Case wird ein neues Template erstellt. Der Titel des jeweiligen Use-Case sowie der Name seines Hauptakteurs werden ins Template übernommen.

Im nächsten Schritt (Abbildung 10) wird der Inhalt der Templates verfeinert bzw. um weitere Informationen ergänzt:

Der Transport kann auf zwei Weisen stattfinden: entweder kümmert sich der Kunde um einen LKW, oder überlässt den Transport dem Möbelhaus (Ellipse 1). Diese Informationen sind als technische Variationen festgehalten. Der Use-Case *Möbel aufbauen* kann unter Bedingung *Möbelstücke beschädigt* durch den Use-Case *Möbel umtauschen* erweitert werden (Ellipse 3). Eine hohe Lautstärke beim Möbelaufbau kann in manchen Fällen Konflikte mit den Nachbarn auslösen. Das Szenario, wie diese beigeräumt werden können, wird im Use-Case *Konflikt schlichten* beschrieben (Ellipse 2).

Letztlich werden die tabellarischen Use-Cases in das Diagramm überführt. Abbildung 11 zeigt das Ergebnis dieser Transformation.

Dieses iterative Vorgehen hilft nicht nur eine Übersicht der Projektziele zu gewinnen, sondern auch alle beteiligte Akteure für weitere Gespräche zu ermitteln und ihre Zuordnung zu den entsprechenden Use-Cases zu dokumentieren.

5 Fazit

Dieser Beitrag präsentiert unseren Ansatz, die jeweiligen Stärken von UML-Use-Case-Diagrammen und natürlich-sprachlichen Use-Case-Beschreibungen in tabellarischen Templates miteinander zu verbinden. Dadurch wird es möglich, ständig beide Sichten auf die funktionalen Anforderungen zu haben. Details sieht man in den ausführlichen Beschreibungen, die Übersicht erhält man in den UML-Use-Case Diagrammen. Iterative Vorgehen, bei denen mehr als einmal zwischen beiden Darstellungen gewechselt wird (z.B. nach [Bir06]), werden so erleichtert: Aus einem initialen UML-Use-Case Diagramm können die Rümpfe von Use-Cases in Templates nach [Coc01] generiert werden. Änderungen an diesen textuellen Use-Case Beschreibungen können automatisch in die Use-Case-Diagramme zurücktransformiert werden. Der Vorteil dabei ist, das Änderungen immer in der jeweils angemesseneren Modellierungsart vorgenommen werden können: Struktur und Rollenzuteilung im UML-Use-Case Diagramm, lokale Details textuell über die Templates. Auf diese Weise werden zwei domänen-spezifische Sprachen für funktionale Anforderungen miteinander kombiniert.

Um dies zu erreichen, haben wir in Abschnitt 2 ein Basismetamodell verwendet, um die gemeinsamen Konzepte zu identifizieren. Am Beispiel der *Include*-Beziehung haben wir gezeigt, wie wir dabei vorgegangen sind. Auf Basis dieser konzeptionellen Schnittmenge von UML-Use-Cases und natürlich-sprachlichen Use-Cases haben wir Transformationen entwickelt. Uns war dabei wichtig, die beiden unterschiedlichen Sichten auf Use-Cases herauszuarbeiten und dabei die einschlägigen Metamodelle beizubehalten. Verfechter beider Ansätze können so von unserer Arbeit profitieren.

Abschnitt 3 und Abschnitt 4 geben den aktuellen Stand unserer Arbeit wieder. Zur Zeit evaluieren wir einen Prototypen dieser Transformationen. Dabei untersuchen wir

beispielsweise, welcher Ausschnitt des Use-Case-Modells je nach Menge und Abstraktion der Use Cases als UML-Diagramm die beste Übersichtlichkeit bietet.

Eine Evaluation im Einsatz (während einer Systemanalyse) steht noch aus. Dazu halten wir auch noch einige technische Erweiterungen für erforderlich. Als nächste Schritte wollen wir die bereits vorhandenen Transformationen kontinuierlich einsetzen, um UML-Diagramme und Use-Case-Tabellen synchron zu halten. Außerdem halten wir die Einführung von Transformationssprachen für sinnvoll, um unseren Ansatz flexibel zu halten.

Mit diesem Beitrag geben wir ein Beispiel für den Einsatz von Modellen und Transformationen in den frühen Phasen eines Softwareprojekts. Gerade das Spannungsfeld zwischen Verifizierbarkeit der Modelle und deren Lesbarkeit als Grundlage für die Validierung durch einen Kunden sind in diesen Phasen reizvoll. Wir laden Andere ein, unseren Ansatz in diesem Kontext zu diskutieren.

Literaturverzeichnis

[Bir06] Birk, A.: Erfahrungen mit Anwendungsfallspezifikation bei der Entwicklung großer IT-Systeme. Jahrestreffen der GI-Fachgruppe Requirements Engineering, München (2006) http://www.gi-ev.de/fachbereiche/softwaretechnik/re/pages/fg_treffen/2006/birk.pdf

[BSM+00] Bronstein, I.N., Semendjajew, K.A., Musiol, G., Mühlig, H.: Taschenbuch der Mathematik. Verla Harri Deutsch (2000).

[CH03] Krzysztof Czarnecki, Simon Helsen: Classification of Model Transformation Approaches. In online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim, (October 2003).

[Coc01] Cockburn, A.:Writing Effective Use Cases. Addison Wesley (2001).

[Cri06] Crisp, C.: Konzept und Werkzeug zur erfahrungsbasierten Erstellung von Use Cases., Masterarbeit, Hannover (2006).

[FKM+01] Fliedl, G., Kop, C., Mayerthaler, W., Mayr, H., Guidelines for NL-Based Requirements Specifications in NIBA, M. Bouzeghoub et al. (Eds.): NLDB 2000, LNCS 1959, pp. 251-264, 2001.Springer-Verlag Berlin Heidelberg (2001)

[JRH+03] Jeckle, M., Rupp, Ch., Hahn, J., Zengler, B., Queins, S.: UML 2 glasklar. Hanser Wissenschaft Muenchen (2003) .

[Kna07] Knauss, E.: Einsatz computergestützter Kritiken für Anforderungen. GI Softwaretechnik-Trends, Band 27, Heft 1, (2007)

[Lüb06] Lübke, D.: Transformation of Use Cases to EPC Models. Workshop EPK (2006)

[MCV05] Mens, T., Czarnecki, K., Van Gorp, P.: A Taxonomy of Model Transformations, http://drops.dagstuhl.de/opus/volltexte/2005/11/ (2005) .

[OMG07] Unified Modeling Language: Superstructure , Version 2.1.1 . Object Management Group, http://www.omg.org, 15.05.07 (2001).

[Pil07] Pilarski, J.: Konzept und Implementierung von Transformationen zwischen Use Case Diagrammen und tabellarischen Darstellungen, Bachelorarbeit, Hannover (2007).

[Ros99] Rosenberg, D.: Use Case Driven Object Modeling with UML : Addison Wesley Longman, Inc (1999).

[Rup06] Chris Rupp: Requirements-Engineering und Management, 4. Auflage. Hanser (2007)

[Stö05] Störrle, H. : UML 2 für Studenten. PEARSON Studium (2005).

[SV05] Stahl, Th., Völter, M.:Modellgetriebene Softwareentwicklung. dpunkt.verlag (2005).